Time Series Multivariate

FRE6871 & FRE7241, Spring 2023

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

May 12, 2023

The *Alpha* and *Beta* of Stock Returns

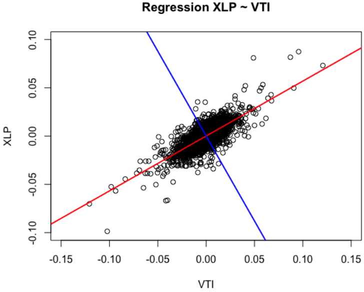
The daily stock returns $r_i - r_f$ in excess of the risk-free rate r_f, can be decomposed into *systematic* returns $\beta(r_m - r_f)$ (where $r_m - r_f$ are the excess market returns) plus *idiosyncratic* returns $\alpha + \varepsilon_i$ (which are uncorrelated to the market returns):

$$r_i - r_f = \alpha + \beta(r_m - r_f) + \varepsilon_i$$

The *alpha* α are the abnormal returns in excess of the risk premium, and ε_i are the regression residuals with zero mean.

The *idiosyncratic* risk (equal to ε_i) is uncorrelated to the *systematic* risk, and can be reduced through portfolio diversification.

```
> # Perform regression using formula
> retp <- na.omit(rutils::etfenv$returns[, c("XLP", "VTI")])
> riskfree <- 0.03/252
> retp <- (retp - riskfree)
> regmod <- lm(XLP ~ VTI, data=retp)
> regmodsum <- summary(regmod)
> # Get regression coefficients
> coef(regmodsum)
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.63e-05   8.26e-05    0.56    0.575
VTI         5.61e-01   6.77e-03   82.94    0.000
> # Get alpha and beta
> coef(regmodsum)[, 1]
(Intercept)         VTI
   4.63e-05    5.61e-01
```




```
> # Plot scatterplot of returns with aspect ratio 1
> plot(XLP ~ VTI, data=rutils::etfenv$returns, main="Regression XLP
+      xlim=c(-0.1, 0.1), ylim=c(-0.1, 0.1), pch=1, col="blue", asp=
> # Add regression line and perpendicular line
> abline(regmod, lwd=2, col="red")
> abline(a=0, b=-1/coef(regmodsum)[2, 1], lwd=2, col="blue")
```

The Statistical Significance of *Alpha* and *Beta*

The stock β is independent of the risk-free rate r_f:

$$\beta = \frac{\mathrm{Cov}(r_i, r_m)}{\mathrm{Var}(r_m)}$$

The *t*-statistic (*t*-value) is the ratio of the estimated value divided by its standard error.

The *p*-value is the probability of obtaining values exceeding the *t*-statistic, assuming the *null hypothesis* is true.

A small *p*-value means that the regression coefficients are very unlikely to be zero (given the data).

The *beta* β values of stock returns are very statistically significant, but the *alpha* α values are mostly not significant.

The *p*-value of the *Durbin-Watson* test is large, which indicates that the regression residuals are not autocorrelated.

In practice, the α, β, and the risk-free rate r_f, depend on the time interval of the data, so they're time dependent.

```
> # Get regression coefficients
> coef(regmodsum)
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.63e-05   8.26e-05    0.56    0.575
VTI         5.61e-01   6.77e-03   82.94    0.000
> # Calculate regression coefficients from scratch
> betav <- drop(cov(retp$XLP, retp$VTI)/var(retp$VTI))
> alpha <- drop(mean(retp$XLP) - betav*mean(retp$VTI))
> c(alpha, betav)
[1] 4.63e-05 5.61e-01
> # Calculate the residuals
> residuals <- (retp$XLP - (alpha + betav*retp$VTI))
> # Calculate the standard deviation of residuals
> nrows <- NROW(residuals)
> residsd <- sqrt(sum(residuals^2)/(nrows - 2))
> # Calculate the standard errors of beta and alpha
> sum2 <- sum((retp$VTI - mean(retp$VTI))^2)
> betasd <- residsd/sqrt(sum2)
> alphasd <- residsd*sqrt(1/nrows + mean(retp$VTI)^2/sum2)
> c(alphasd, betasd)
[1] 8.26e-05 6.77e-03
> # Perform the Durbin-Watson test of autocorrelation of residuals
> lmtest::dwtest(regmod)

Durbin-Watson test

data:  regmod
DW = 2, p-value = 1
alternative hypothesis: true autocorrelation is greater than 0
```

The *Alpha* and *Beta* of ETF Returns

The *beta* β values of ETF returns are very statistically significant, but the *alpha* α values are mostly not significant.

Some of the ETFs with significant *alpha* α values are the bond ETFs *IEF* and *TLT* (which have performed very well), and the natural resource ETFs *USO* and *DBC* (which have performed very poorly).

```
> retp <- rutils::etfenv$returns
> symbolv <- colnames(retp)
> symbolv <- symbolv[symbolv != "VTI"]
> # Perform regressions and collect statistics
> betam <- sapply(symbolv, function(symbol) {
+ # Specify regression formula
+   formulav <- as.formula(paste(symbol, "~ VTI"))
+ # Perform regression
+   regmod <- lm(formulav, data=retp)
+ # Get regression summary
+   regmodsum <- summary(regmod)
+ # Collect regression statistics
+   with(regmodsum,
+     c(beta=coefficients[2, 1],
+ pbeta=coefficients[2, 4],
+ alpha=coefficients[1, 1],
+ palpha=coefficients[1, 4],
+ pdw=lmtest::dwtest(regmod)$p.value))
+ })  # end sapply
> betam <- t(betam)
> # Sort by palpha
> betam <- betam[order(betam[, "palpha"]), ]
```

```
> betam
        beta      pbeta     alpha palpha      pdw
VEU   1.0010  0.00e+00 -2.55e-04 0.0117 9.99e-01
VXX  -2.7207  0.00e+00 -1.72e-03 0.0158 1.00e+00
XLF   1.3097  0.00e+00 -3.29e-04 0.0250 1.00e+00
USO   0.7143 3.00e-150 -7.09e-04 0.0355 4.84e-02
GLD   0.0546  4.02e-05  2.93e-04 0.0761 7.57e-01
IWD   0.9872  0.00e+00 -6.94e-05 0.1311 1.00e+00
VYM   0.8480  0.00e+00 -1.15e-04 0.1361 1.00e+00
XLK   1.0374  0.00e+00  1.19e-04 0.1478 1.00e+00
VLUE  1.0007  0.00e+00 -1.33e-04 0.1659 4.89e-01
IWF   0.9785  0.00e+00  6.43e-05 0.1708 1.00e+00
VNQ   1.1758  0.00e+00 -2.22e-04 0.2112 1.00e+00
XLP   0.5613  0.00e+00  9.85e-05 0.2331 9.95e-01
IVE   0.9735  0.00e+00 -5.67e-05 0.2389 1.00e+00
XLV   0.7133  0.00e+00  9.93e-05 0.2557 5.15e-01
AIEQ  1.0285  0.00e+00 -2.14e-04 0.2597 7.02e-01
EEM   1.2085  0.00e+00 -1.61e-04 0.2620 9.99e-01
IEF  -0.1131 4.80e-119  6.39e-05 0.2695 7.10e-01
TLT  -0.2604 2.32e-140  1.20e-04 0.3248 5.86e-01
VTV   0.9615  0.00e+00 -4.69e-05 0.3414 1.00e+00
IVW   0.9687  0.00e+00  4.19e-05 0.3451 1.00e+00
USMV  0.7455  0.00e+00  5.59e-05 0.4081 5.29e-01
DBC   0.4173 4.38e-188 -1.29e-04 0.4555 8.73e-01
SVXY  2.1591 3.41e-181 -5.71e-04 0.4643 2.19e-06
QUAL  0.9771  0.00e+00  3.27e-05 0.4738 9.88e-01
XLE   1.1674  0.00e+00 -9.69e-05 0.5978 2.25e-01
XLB   1.0664  0.00e+00 -5.55e-05 0.6014 9.99e-01
XLY   1.0231  0.00e+00  3.61e-05 0.6545 1.00e+00
XLU   0.6367  0.00e+00  5.00e-05 0.6905 9.94e-01
MTUM  0.9901  0.00e+00  2.99e-05 0.7776 7.43e-03
IWB   0.9804  0.00e+00  2.95e-06 0.8860 1.00e+00
XLI   0.9978  0.00e+00  2.35e-06 0.9754 1.00e+00
```

Capital Asset Pricing Model (*CAPM*)

The *CAPM* model states that the expected return for stock n: $\mathbb{E}[R_n]$ is proportional to its beta β_n times the expected excess return of the market $\mathbb{E}[R_m] - r_f$:

$$\mathbb{E}[R_n] = r_f + \beta_n(\mathbb{E}[R_m] - r_f)$$

The *CAPM* model states that if a stock has a higher beta then it's also expected to earn higher returns.

According to the *CAPM* model, assets are on average expected to earn only a *systematic* return proportional to their *systematic* risk.

The *CAPM* model is not a regression model.

The *CAPM* model depends on the choice of the risk-free rate r_f.

```
> library(PerformanceAnalytics)
> # Calculate XLP beta
> PerformanceAnalytics::CAPM.beta(Ra=retp$XLP, Rb=retp$VTI)
[1] 0.561
> # Or
> retxlp <- na.omit(retp[, c("XLP", "VTI")])
> betav <- drop(cov(retxlp$XLP, retxlp$VTI)/var(retxlp$VTI))
> betav
[1] 0.561
> # Calculate XLP alpha
> PerformanceAnalytics::CAPM.alpha(Ra=retp$XLP, Rb=retp$VTI)
[1] 9.85e-05
> # Or
> mean(retp$XLP - betav*retp$VTI)
[1] NA
> # Calculate XLP bull beta
> PerformanceAnalytics::CAPM.beta.bull(Ra=retp$XLP, Rb=retp$VTI)
[1] 0.562
> # Calculate XLP bear beta
> PerformanceAnalytics::CAPM.beta.bear(Ra=retp$XLP, Rb=retp$VTI)
[1] 0.577
```

The Security Market Line for ETFs

The *Security Market Line* (SML) represents the linear relationship between expected stock returns and their *systematic* risk β.

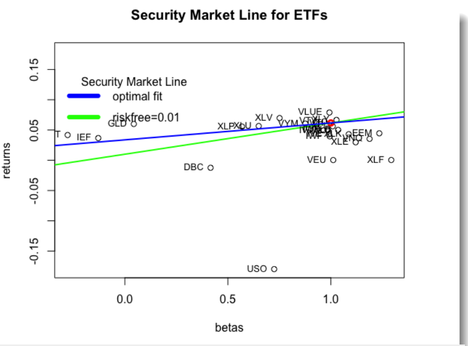
The *SML* depends on the choice of the risk-free rate r_f, with a steeper *SML* line for lower risk-free rates r_f.

All the different *SML* lines pass through the point $(\beta = 1, r = R_m)$ corresponding to the market, and they intersect the y-axis at the risk-free point $(\beta = 0, r = r_f)$.

A scatterplot of asset returns versus their β shows which assets earn a positive α, and which don't.

If an asset lies on the *SML*, then its returns are mostly *systematic*, and its α is equal to zero.

Assets above the *SML* have a positive α, and those below have a negative α.




```
> symbolv <- rownames(betam)
> betav <- betam[-match(c("VXX", "SVXY", "MTUM", "USMV", "QUAL"), s
> betav <- c(1, betav)
> names(betav)[1] <- "VTI"
> retsann <- sapply(retp[, names(betav)], PerformanceAnalytics::Ret
> # Plot scatterplot of returns vs betas
> minrets <- min(retsann)
> plot(retsann ~ betav, xlab="betas", ylab="returns",
+      ylim=c(minrets, -minrets), main="Security Market Line for ET
> retvti <- retsann["VTI"]
> points(x=1, y=retvti, col="red", lwd=3, pch=21)
> # Plot Security Market Line
> riskfree <- 0.01
> abline(a=riskfree, b=(retvti-riskfree), col="green", lwd=2)
```

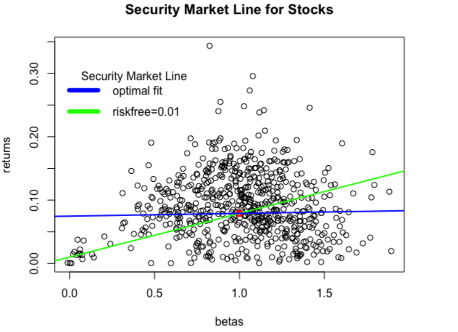
```
> # Add labels
> text(x=betav, y=retsann, labels=names(betav), pos=2, cex=0.8)
> # Find optimal risk-free rate by minimizing residuals
> rss <- function(riskfree) {
+   sum((retsann - riskfree - betav*(retvti-riskfree))^2)
+ }  # end rss
> optimrss <- optimize(rss, c(-1, 1))
> riskfree <- optimrss$minimum
> # Or simply
> retsadj <- (retsann - retvti*betav)
> betadj <- (1-betav)
> riskfree <- sum(retsadj*betadj)/sum(betadj^2)
> abline(a=riskfree, b=(retvti-riskfree), col="blue", lwd=2)
> legend(x="topleft", bty="n", title="Security Market Line",
+  legend=c("optimal fit", "riskfree=0.01"),
+  y.intersp=0.5, cex=1.0, lwd=6, lty=1, col=c("blue", "green"))
```

The Security Market Line for Stocks

The best fitting *Security Market Line* (SML) for stocks is almost flat, which shows that stocks with higher β don't earn higher returns.

This is called the *low beta anomaly*.

```
> # Load S&P500 constituent stock returns
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> retp <- returns[index(retvti), ]
> nrows <- NROW(retp)
> # Calculate stock betas
> betav <- sapply(retp, function(x) {
+   retp <- na.omit(cbind(x, retvti))
+   drop(cov(retp[, 1], retp[, 2])/var(retp[, 2]))
+ })  # end sapply
> mean(betav)
> # Calculate annual stock returns
> retsann <- retp
> retsann[1, ] <- 0
> retsann <- zoo::na.locf(retsann, na.rm=FALSE)
> retsann <- 252*sapply(retsann, sum)/nrows
> # Remove stocks with zero returns
> sum(retsann == 0)
> betav <- betav[retsann > 0]
> retsann <- retsann[retsann > 0]
> retvti <- 252*mean(retvti)
> # Plot scatterplot of returns vs betas
> plot(retsann ~ betav, xlab="betas", ylab="returns",
+      main="Security Market Line for Stocks")
> points(x=1, y=retvti, col="red", lwd=3, pch=21)
> # Plot Security Market Line
> riskfree <- 0.01
> abline(a=riskfree, b=(retvti-riskfree), col="green", lwd=2)
```




```
> # Find optimal risk-free rate by minimizing residuals
> retsadj <- (retsann - retvti*betav)
> betadj <- (1-betav)
> riskfree <- sum(retsadj*betadj)/sum(betadj^2)
> abline(a=riskfree, b=(retvti-riskfree), col="blue", lwd=2)
> legend(x="topleft", bty="n", title="Security Market Line",
+  legend=c("optimal fit", "riskfree=0.01"),
+  y.intersp=0.5, cex=1.0, lwd=6, lty=1, col=c("blue", "green"))
```

Beta-adjusted Performance Measurement

The *Treynor* ratio measures the excess returns per unit of the *systematic* risk *beta* β, and is equal to the excess returns (over a risk-free rate) divided by the β:

$$T_r = \frac{E[R - r_f]}{\beta}$$

The *Treynor* ratio is similar to the *Sharpe* ratio, with the difference that its denominator represents only *systematic* risk, not total risk.

The *Information* ratio is equal to the excess returns (over a benchmark) divided by the *tracking error* (standard deviation of excess returns):

$$I_r = \frac{E[R - R_b]}{\sqrt{\sum_{i=1}^{n} (R_i - R_{i,b})^2}}$$

The *Information* ratio measures the amount of outperformance versus the benchmark, and the consistency of outperformance.

```
> library(PerformanceAnalytics)
> # Calculate XLP Treynor ratio
> TreynorRatio(Ra=retp$XLP, Rb=retp$VTI)
[1] 0.101
> # Calculate XLP Information ratio
> InformationRatio(Ra=retp$XLP, Rb=retp$VTI)
[1] 0.017
```

CAPM Summary Statistics

`PerformanceAnalytics::table.CAPM()` calculates the *beta* β and *alpha* α values, the *Treynor* ratio, and other performance statistics.

```
> PerformanceAnalytics::table.CAPM(Ra=retp[, c("XLP", "XLF")],
+                           Rb=retp$VTI, scale=252)
                    XLP to VTI XLF to VTI
Alpha                   0.0001    -0.0003
Beta                    0.5613     1.3097
Beta+                   0.5620     1.3969
Beta-                   0.5770     1.3803
R-squared               0.5834     0.7067
Annualized Alpha        0.0251    -0.0797
Correlation             0.7638     0.8406
Correlation p-value     0.0000     0.0000
Tracking Error          0.1251     0.1741
Active Premium          0.0021    -0.0887
Information Ratio       0.0170    -0.5097
Treynor Ratio           0.1006    -0.0262
> capmstats <- table.CAPM(Ra=retp[, symbolv],
+         Rb=retp$VTI, scale=252)
> colnamev <- strsplit(colnames(capmstats), split=" ")
> colnamev <- do.call(cbind, colnamev)[1, ]
> colnames(capmstats) <- colnamev
> capmstats <- t(capmstats)
> capmstats <- capmstats[, -1]
> colnamev <- colnames(capmstats)
> whichv <- match(c("Annualized Alpha", "Information Ratio", "Treynor Rat
> colnamev[whichv] <- c("Alpha", "Information", "Treynor")
> colnames(capmstats) <- colnamev
> capmstats <- capmstats[order(capmstats[, "Alpha"], decreasing=TRUE), ]
> # Copy capmstats into etfenv and save to .RData file
> etfenv <- rutils::etfenv
> etfenv$capmstats <- capmstats
> save(etfenv, file="/Users/jerzy/Develop/lecture_slides/data/etf_data.RData")
```

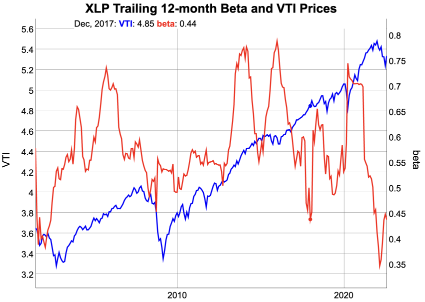
```
> rutils::etfenv$capmstats[, c("Beta", "Alpha", "Information"
        Beta   Alpha Information Treynor
GLD   0.0546  0.0767      0.0580  1.1643
TLT  -0.2604  0.0307     -0.1907 -0.0042
XLK   1.0374  0.0303      0.3156  0.0802
XLV   0.7133  0.0253      0.0850  0.0895
XLP   0.5613  0.0251      0.0170  0.1006
IWF   0.9785  0.0163      0.2859  0.0708
IEF  -0.1131  0.0162     -0.2162 -0.0503
USMV  0.7455  0.0142     -0.1018  0.1123
XLU   0.6367  0.0127     -0.0804  0.0656
IVW   0.9687  0.0106      0.1757  0.0651
XLY   1.0231  0.0091      0.0682  0.0591
QUAL  0.9771  0.0083      0.1934  0.0773
MTUM  0.9901  0.0076      0.0441  0.0805
IWB   0.9804  0.0007     -0.0090  0.0552
XLI   0.9978  0.0006     -0.0384  0.0512
VTI   1.0000  0.0000         NaN  0.0543
VTV   0.9615 -0.0117     -0.2747  0.0355
XLB   1.0664 -0.0139     -0.1653  0.0325
IVE   0.9735 -0.0142     -0.3237  0.0379
IWD   0.9872 -0.0173     -0.3926  0.0347
XLE   1.1674 -0.0241     -0.2025  0.0107
VYM   0.8480 -0.0285     -0.4553  0.0225
DBC   0.4173 -0.0321     -0.3314 -0.0585
VLUE  1.0007 -0.0331     -0.5112  0.0213
EEM   1.2085 -0.0398     -0.2938  0.0049
AIEQ  1.0285 -0.0526     -0.5460 -0.0083
VNQ   1.1758 -0.0544     -0.3634 -0.0160
VEU   1.0010 -0.0624     -0.6876 -0.0263
XLF   1.3097 -0.0797     -0.5097 -0.0262
SVXY  2.1591 -0.1340         NaN     NaN
USO   0.7143 -0.1636     -0.6534 -0.2595
VXX  -2.7207 -0.3518         NaN     NaN
```

Trailing Stock Beta Over Time

The trailing beta of *XLP* versus *VTI* changes over time, with lower beta in periods of stock selloffs.

The function `roll_reg()` from package *HighFreq* performs trailing regressions in C++ (*RcppArmadillo*), so it's therefore much faster than equivalent R code.

```
> # Calculate XLP and VTI returns
> retp <- na.omit(rutils::etfenv$returns[, c("XLP", "VTI")])
> # Calculate monthly end points
> endd <- xts::endpoints(retp, on="months")[-1]
> # Calculate start points from look-back interval
> look_back <- 12  # Look back 12 months
> startp <- c(rep(1, look_back), endd[1:(NROW(endd)-look_back)])
> head(cbind(endd, startp), look_back+2)
> # Calculate trailing beta regressions every month in R
> formulav <- XLP ~ VTI  # Specify regression formula
> betar <- sapply(1:NROW(endd), FUN=function(tday) {
+     datav <- retp[startp[tday]:endd[tday], ]
+     # coef(lm(formulav, data=datav))[2]
+     drop(cov(datav$XLP, datav$VTI)/var(datav$VTI))
+   })  # end sapply
> # Calculate trailing betas using RcppArmadillo
> controlv <- HighFreq::param_reg()
> reg_stats <- HighFreq::roll_reg(respv=retp$XLP, predm=retp$VTI,
+   startp=(startp-1), endp=(endd-1), controlv=controlv)
> betav <- reg_stats[, 2]
> all.equal(betav, betar)
> # Compare the speed of RcppArmadillo with R code
> library(microbenchmark)
> summary(microbenchmark(
+   Rcpp=HighFreq::roll_reg(respv=retp$XLP, predm=retp$VTI, startp=(startp-1), endp=(endd-1), controlv=controlv),
+   Rcode=sapply(1:NROW(endd), FUN=function(tday) {
+     datav <- retp[startp[tday]:endd[tday], ]
+     drop(cov(datav$XLP, datav$VTI)/var(datav$VTI))
+   }),
+   times=10))[, c(1, 4, 5)]  # end microbenchmark summary
```



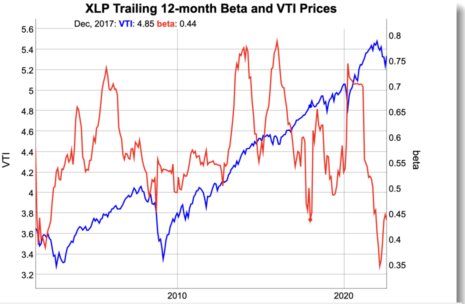

```
> # dygraph plot of trailing XLP beta and VTI prices
> datev <- zoo::index(retp[endd, ])
> pricev <- rutils::etfenv$prices$VTI[datev]
> datav <- cbind(pricev, betav)
> colnames(datav)[2] <- "beta"
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav, main="XLP Trailing 12-month Beta and VTI
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue") %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=500)
```

Trailing Stock Beta Over Time

The trailing beta of *XLP* versus *VTI* changes over time, with lower beta in periods of stock selloffs.

The function `roll_reg()` from package *HighFreq* performs trailing regressions in C++ (*RcppArmadillo*), so it's therefore much faster than equivalent R code.

```
> # Calculate XLP and VTI returns
> retp <- na.omit(rutils::etfenv$returns[, c("XLP", "VTI")])
> # Calculate monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")[-1]
> # Calculate start points from look-back interval
> look_back <- 12  # Look back 12 months
> startp <- c(rep(1, look_back), endd[1:(NROW(endd)-look_back)])
> head(cbind(endd, startp), look_back+2)
> # Calculate trailing beta regressions every month in R
> formulav <- XLP ~ VTI  # Specify regression formula
> betar <- sapply(1:NROW(endd), FUN=function(tday) {
+     datav <- retp[startp[tday]:endd[tday], ]
+     # coef(lm(formulav, data=datav))[2]
+     drop(cov(datav$XLP, datav$VTI)/var(datav$VTI))
+   })  # end sapply
> # Calculate trailing betas using RcppArmadillo
> controlv <- HighFreq::param_reg()
> reg_stats <- HighFreq::roll_reg(respv=retp$XLP, predm=retp$VTI,
+   startp=(startp-1), endp=(endd-1), controlv=controlv)
> betav <- reg_stats[, 2]
> all.equal(betav, betar)
> # Compare the speed of RcppArmadillo with R code
> library(microbenchmark)
> summary(microbenchmark(
+   Rcpp=HighFreq::roll_reg(respv=retp$XLP, predm=retp$VTI, startp=(startp-1), endp=(endd-1), controlv=controlv),
+   Rcode=sapply(1:NROW(endd), FUN=function(tday) {
+     datav <- retp[startp[tday]:endd[tday], ]
+     drop(cov(datav$XLP, datav$VTI)/var(datav$VTI))
+   }),
+   times=10))[, c(1, 4, 5)]  # end microbenchmark summary
```




```
> # dygraph plot of trailing XLP beta and VTI prices
> datev <- zoo::index(retp[endd, ])
> pricev <- log(rutils::etfenv$prices$VTI[datev])
> datav <- cbind(pricev, betav)
> colnames(datav)[2] <- "beta"
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav, main="XLP Trailing 12-month Beta and VTI
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=500)
```

Recursive Trailing Stock Beta

The trailing beta β of a stock with returns r_t with respect to a stock index with returns R_t can be updated using these recursive formulas with the weight decay factor λ:

$$\bar{r}_t = \lambda \bar{r}_{t-1} + (1 - \lambda) r_t$$

$$\bar{R}_t = \lambda \bar{R}_{t-1} + (1 - \lambda) R_t$$

$$\sigma^2_t = \lambda \sigma^2_{t-1} + (1 - \lambda)(R_t - \bar{R}_t)^2$$

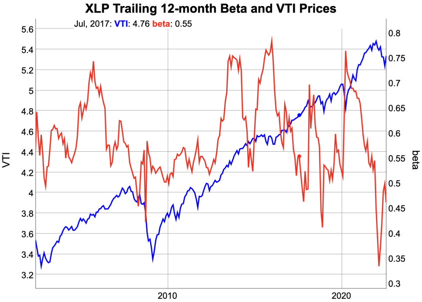
$$\operatorname{cov}_t = \lambda \operatorname{cov}_{t-1} + (1 - \lambda)(r_t - \bar{r}_t)(R_t - \bar{R}_t)$$

$$\beta_t = \frac{\operatorname{cov}_t}{\sigma^2_t}$$

The parameter λ determines the rate of decay of the weight of past returns. If λ is close to 1 then the decay is weak and past returns have a greater weight, and the trailing mean values have a stronger dependence on past returns. This is equivalent to a long look-back interval. And vice versa if λ is close to 0.

The function `HighFreq::run_covar()` calculates the trailing variances, covariances, and means of two *time series*.

```
> # Calculate the trailing betas
> lambda <- 0.99
> covarv <- HighFreq::run_covar(retp, lambda)
> betav <- covarv[, 1]/covarv[, 3]
```




```
> # dygraph plot of trailing XLP beta and VTI prices
> datav <- cbind(pricev, betav[endd])[-(1:11)] # Remove warmup peri
> colnames(datav)[2] <- "beta"
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav, main="XLP Trailing 12-month Beta and VTI
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=500)
```

Principal Components of *S&P500* Stock Constituents

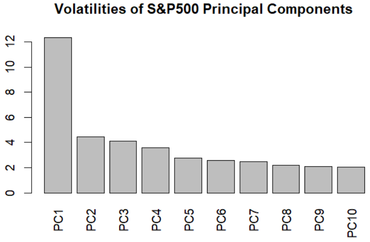
The *PCA* standard deviations are the volatilities of the *principal component* time series.

The original time series of returns can be calculated approximately from the first few *principal components* with the largest standard deviations.

The *Kaiser-Guttman* rule uses only *principal components* with *variance* greater than 1.

Another rule of thumb is to use the *principal components* with the largest standard deviations which sum up to 80% of the total variance of returns.

```
> # Load S&P500 constituent stock prices
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_prices.RData'
> # Calculate stock prices and percentage returns
> pricets <- zoo::na.locf(pricets, na.rm=FALSE)
> pricets <- zoo::na.locf(pricets, fromLast=TRUE)
> retp <- rutils::diffit(log(pricev))
> # Standardize (de-mean and scale) the returns
> retp <- lapply(retp, function(x) {(x - mean(x))/sd(x)})
> retp <- rutils::do_call(cbind, retp)
> # Perform principal component analysis PCA
> pcad <- prcomp(retp, scale=TRUE)
> # Find number of components with variance greater than 2
> ncomp <- which(pcad$sdev^2 < 2)[1]
```



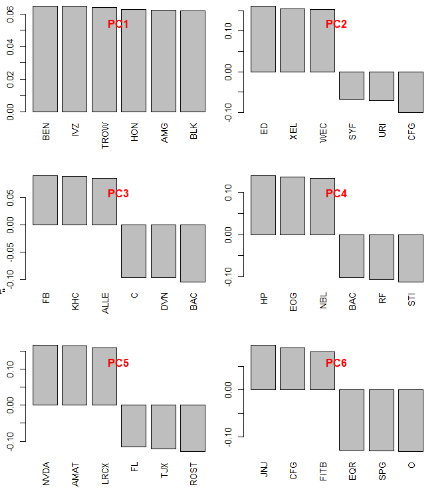

```
> # Plot standard deviations of principal components
> barplot(pcad$sdev[1:ncomp],
+   names.arg=colnames(pcad$rotation[, 1:ncomp]),
+   las=3, xlab="", ylab="",
+   main="Volatilities of S&P500 Principal Components")
```

S&P500 Principal Component Loadings

Principal component loadings are the weights of *principal component* portfolios.

The *principal component* portfolios have mutually orthogonal returns represent the different orthogonal modes of the return variance.

```
> # Calculate principal component loadings (weights)
> # Plot barplots with PCA weights in multiple panels
> ncomps <- 6
> par(mfrow=c(ncomps/2, 2))
> par(mar=c(4, 2, 2, 1), oma=c(0, 0, 0, 0))
> # First principal component weights
> weightv <- sort(pcad$rotation[, 1], decreasing=TRUE)
> barplot(weightv[1:6], las=3, xlab="", ylab="", main="")
> title(paste0("PC", 1), line=-2.0, col.main="red")
> for (ordern in 2:ncomps) {
+   weightv <- sort(pcad$rotation[, ordern], decreasing=TRUE)
+   barplot(weightv[c(1:3, 498:500)], las=3, xlab="", ylab="", main=
+   title(paste0("PC", ordern), line=-2.0, col.main="red")
+ }  # end for
```

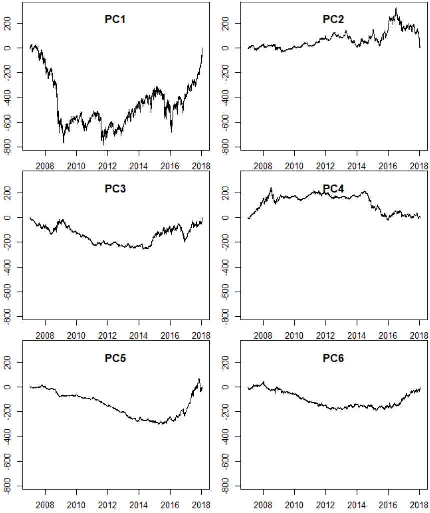


S&P500 Principal Component Time Series

The time series of the *principal components* can be calculated by multiplying the loadings (weights) times the original data.

Higher order *principal components* are gradually less volatile.

```
> # Calculate principal component time series
> retpca <- xts(retp %*% pcad$rotation[, 1:ncomps], order.by=datev)
> round(cov(retpca), 3)
> pcacum <- cumsum(retpca)
> # Plot principal component time series in multiple panels
> par(mfrow=c(ncomps/2, 2))
> par(mar=c(2, 2, 0, 1), oma=c(0, 0, 0, 0))
> rangev <- range(pcacum)
> for (ordern in 1:ncomps) {
+   plot.zoo(pcacum[, ordern], ylim=rangev, xlab="", ylab="")
+   title(paste0("PC", ordern), line=-2.0)
+ }  # end for
```



S&P500 Factor Model From *Principal Components*

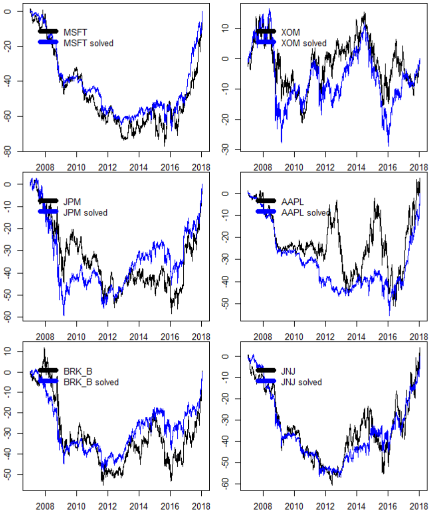
By inverting the *PCA* analysis, the *S&P500* constituent returns can be calculated from the first k *principal components* under a *factor model*:

$$\mathbf{r}_i = \alpha_i + \sum_{j=1}^{k} \beta_{ji} \, \mathbf{F}_j + \varepsilon_i$$

The *principal components* are interpreted as *market factors*: $\mathbf{F}_j = \mathbf{pc}_j$.

The *market betas* are the inverse of the *principal component loadings*: $\beta_{ji} = w_{ij}$.

The ε_i are the *idiosyncratic* returns, which should be mutually independent and uncorrelated to the *market factor* returns.




```
> # Invert principal component time series
> invmat <- solve(pcad$rotation)
> all.equal(invmat, t(pcad$rotation))
> solved <- retpca %*% invmat[1:ncomps, ]
> solved <- xts::xts(solved, datev)
> solved <- cumsum(solved)
> retc <- cumsum(retp)
> # Plot the solved returns
> symbolv <- c("MSFT", "XOM", "JPM", "AAPL", "BRK_B", "JNJ")
> for (symbol in symbolv) {
+   plot.zoo(cbind(retc[, symbol], solved[, symbol]),
+     plot.type="single", col=c("black", "blue"), xlab="", ylab="")
+   legend(x="topleft", bty="n",
+    legend=paste0(symbol, c("", " solved")),
+    title=NULL, inset=0.05, cex=1.0, lwd=6,
+    lty=1, col=c("black", "blue"))
+ }  # end for
```

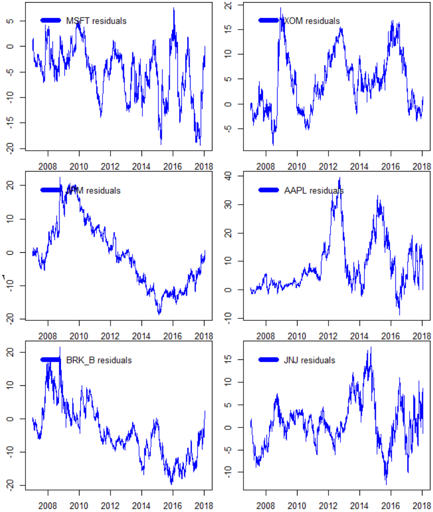
S&P500 Factor Model Residuals

The original time series of returns can be calculated exactly from the time series of all the *principal components*, by inverting the loadings matrix.

The original time series of returns can be calculated approximately from just the first few *principal components*, which demonstrates that *PCA* is a form of *dimension reduction*.

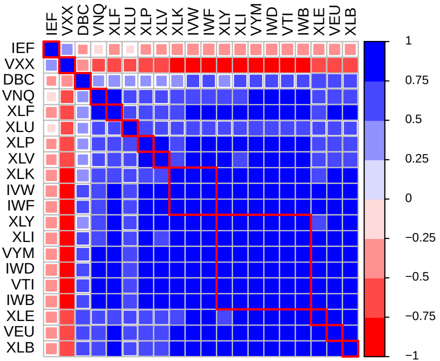
The function solve() solves systems of linear equations, and also inverts square matrices.

```
> # Perform ADF unit root tests on original series and residuals
> sapply(symbolv, function(symbol) {
+   c(series=tseries::adf.test(retc[, symbol])$p.value,
+     resid=tseries::adf.test(retc[, symbol] - solved[, symbol])$p.v
+ })  # end sapply
> # Plot the residuals
> for (symbol in symbolv) {
+   plot.zoo(retc[, symbol] - solved[, symbol],
+     plot.type="single", col="blue", xlab="", ylab="")
+   legend(x="topleft", bty="n", legend=paste(symbol, "residuals"),
+    title=NULL, inset=0.05, cex=1.0, lwd=6, lty=1, col="blue")
+ }  # end for
> # Perform ADF unit root test on principal component time series
> retpca <- xts(retp %*% pcad$rotation, order.by=datev)
> pcacum <- cumsum(retpca)
> adf_pvalues <- sapply(1:NCOL(pcacum), function(ordern)
+   tseries::adf.test(pcacum[, ordern])$p.value)
> # AdF unit root test on stationary time series
> tseries::adf.test(rnorm(1e5))
```



Correlation and Factor Analysis

```
> ### Perform pair-wise correlation analysis
> # Calculate correlation matrix
> cormat <- cor(retp)
> colnames(cormat) <- colnames(retp)
> rownames(cormat) <- colnames(retp)
> # Reorder correlation matrix based on clusters
> # Calculate permutation vector
> library(corrplot)
> ordern <- corrMatOrder(cormat, order="hclust",
+         hclust.method="complete")
> # Apply permutation vector
> cormat <- cormat[ordern, ordern]
> # Plot the correlation matrix
> colorv <- colorRampPalette(c("red", "white", "blue"))
> corrplot(cormat, tl.col="black", tl.cex=0.8,
+     method="square", col=colorv(8),
+     cl.offset=0.75, cl.cex=0.7,
+     cl.align.text="l", cl.ratio=0.25)
> # draw rectangles on the correlation matrix plot
> corrRect.hclust(cormat, k=NROW(cormat) %/% 2,
+           method="complete", col="red")
```



Hierarchical Clustering Analysis

The function `as.dist()` converts a matrix representing the *distance* (dissimilarity) between elements, into a list of class `"dist"`.

For example, `as.dist()` converts `(1-correlation)` to distance.

The function `hclust()` recursively combines elements into clusters based on their mutual *distance*.

First `hclust()` combines individual elements that are closest to each other.

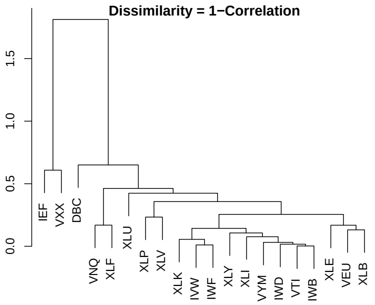
Then it combines elements to the closest clusters, then clusters with other clusters, until all elements are combined into one cluster.

This process of recursive clustering can be represented as a *dendrogram* (tree diagram).

Branches of a *dendrogram* represent clusters.

Neighboring branches contain elements that are close to each other (have small distance).

Neighboring branches combine into larger branches, that then combine with their closest branches, etc.

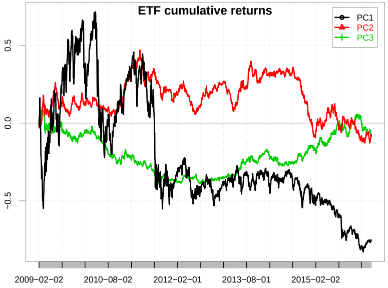



```
> # Convert correlation matrix into distance object
> distancev <- as.dist(1-cormat)
> # Perform hierarchical clustering analysis
> cluster <- hclust(distancev)
> plot(cluster, ann=FALSE, xlab="", ylab="")
> title("Dendrogram representing hierarchical clustering
+ \nwith dissimilarity = 1-correlation", line=-0.5)
```

depr: *Principal Component* Returns Time Series

```
> # PC returns from rotation and scaled returns
> retsc <- apply(retp, 2, scale)
> retpca <- retsc %*% pcad$rotation
> # "x" matrix contains time series of PC returns
> dim(pcad$x)
> class(pcad$x)
> head(pcad$x[, 1:3], 3)
> # Convert PC matrix to xts and rescale to decimals
> retpca <- xts(pcad$x/100, order.by=zoo::index(retp))
```

```
> chart.CumReturns(
+   retpca[, 1:3], lwd=2, ylab="",
+   legend.loc="topright", main="")
> # Add title
> title(main="ETF cumulative returns", line=-1)
```

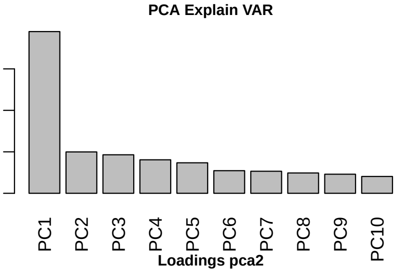


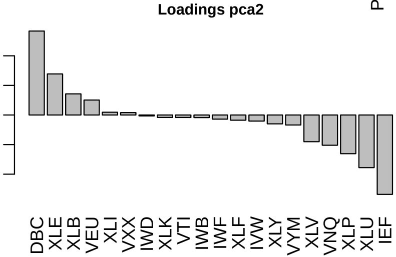
depr: *Principal Component* Returns Analysis

```
> # Calculate PC correlation matrix
> cormat <- cor(retpca)
> colnames(cormat) <- colnames(retpca)
> rownames(cormat) <- colnames(retpca)
> cormat[1:3, 1:3]
> table.CAPM(Ra=retpca[, 1:3], Rb=retp$VTI, scale=252)
```

depr: Principal Component Analysis

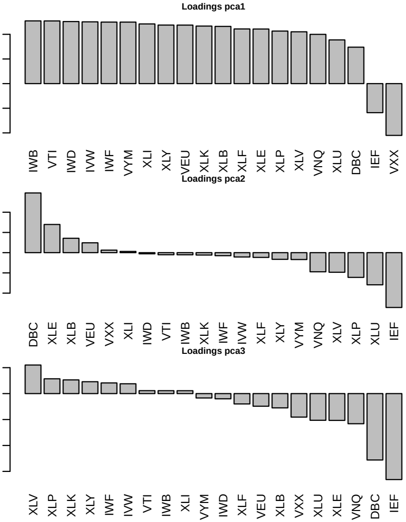
```
> ### Perform principal component analysis PCA
> retp <- na.omit(rutils::etfenv$returns)
> pcad <- prcomp(retp, center=TRUE, scale=TRUE)
> barplot(pcad$sdev[1:10],
+   names.arg=colnames(pcad$rotation)[1:10],
+   las=3, ylab="STDEV", xlab="PCVec",
+   main="PCA Explain VAR")
> # Show first three principal component loadings
> head(pcad$rotation[,1:3], 3)
> # Permute second principal component loadings by size
> pca2 <- as.matrix(
+   pcad$rotation[order(pcad$rotation[, 2],
+   decreasing=TRUE), 2])
> colnames(pca2) <- "pca2"
> head(pca2, 3)
> # The option las=3 rotates the names.arg labels
> barplot(as.vector(pca2),
+   names.arg=rownames(pca2),
+   las=3, ylab="Loadings",
+   xlab="Symbol", main="Loadings pca2")
```





depr: Principal Component Vectors

```
> # Get list of principal component vectors
> pca_vecs <- lapply(1:3, function(ordern) {
+   pca_vec <- as.matrix(
+     pcad$rotation[
+     order(pcad$rotation[, ordern],
+     decreasing=TRUE), ordern])
+   colnames(pca_vec) <- paste0("pca", ordern)
+   pca_vec
+ })  # end lapply
> names(pca_vecs) <- c("pca1", "pca2", "pca3")
> # The option las=3 rotates the names.arg labels
> for (ordern in 1:3) {
+   barplot(as.vector(pca_vecs[[ordern]]),
+   names.arg=rownames(pca_vecs[[ordern]]),
+   las=3, xlab="", ylab="",
+   main=paste("Loadings",
+     colnames(pca_vecs[[ordern]])))
+ }  # end for
```



depr: Package *factorAnalytics*

The package *factorAnalytics* performs estimation and risk analysis of linear factor models for portfolio asset returns.

```
> library(factorAnalytics)  # Load package "factorAnalytics"
> # Get documentation for package "factorAnalytics"
> packageDescription("factorAnalytics")  # Get short description
> help(package="factorAnalytics")  # Load help page
```

```
> # List all objects in "factorAnalytics"
> ls("package:factorAnalytics")
>
> # List all datasets in "factorAnalytics"
> # data(package="factorAnalytics")
>
> # Remove factorAnalytics from search path
> detach("package:factorAnalytics")
```

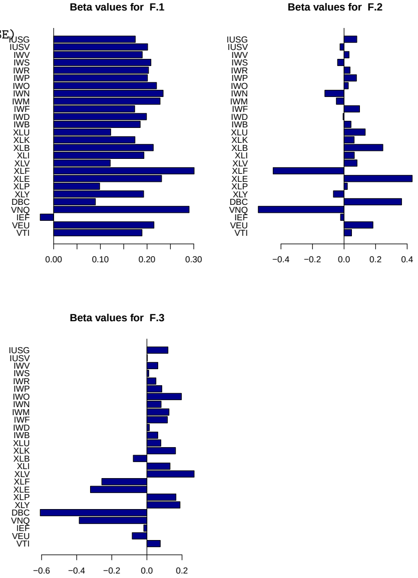
depr: Fitting Factor Models Using PCA

```
> library(factorAnalytics)
> # Fit a three-factor model using PCA
> factpca <- fitSfm(rutils::etfenv$returns, k=3)
> head(factpca$loadings, 3)  # Factor loadings
> # Factor realizations (time series)
> head(factpca$factors)
> # Residuals from regression
> factpca$residuals[1:3, 1:3]
```

```
> factpca$alpha  # Estimated alphas
> factpca$r2  # R-squared regression
> # Covariance matrix estimated by factor model
> factpca$Omega[1:3, 4:6]
```

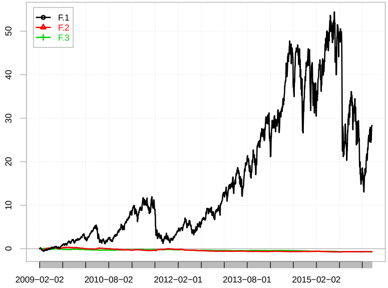
depr: Factor Loadings

```
> plot(factpca, which.plot.group=3, n.max=30, loop=FALSE)
> # ?plot.sfm
```



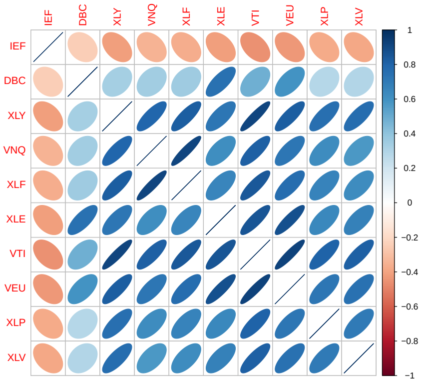
depr: Time Series of Factors

```
> library(PortfolioAnalytics)
> # Plot factor cumulative returns
> chart.CumReturns(factpca$factors,
+     lwd=2, ylab="", legend.loc="topleft", main="")
>
> # Plot time series of factor returns
> # Plot(factpca, which.plot.group=2,
> #   loop=FALSE)
```



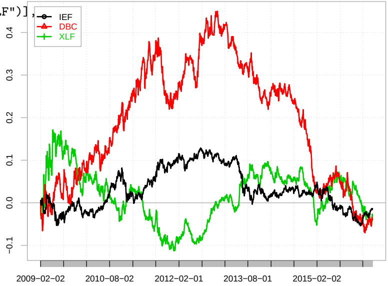
depr: Asset Correlations

```
> # Asset correlations "hclust" hierarchical clustering
> plot(factpca, which.plot.group=7, loop=FALSE,
+      order="hclust", method="ellipse")
```



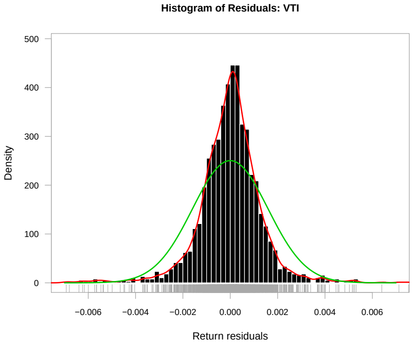
depr: Time Series of Residuals

```
> library(PortfolioAnalytics)
> # Plot residual cumulative returns
> chart.CumReturns(factpca$residuals[, c("IEF", "DBC", "XLF")],
+   lwd=2, ylab="", legend.loc="topleft", main="")
```



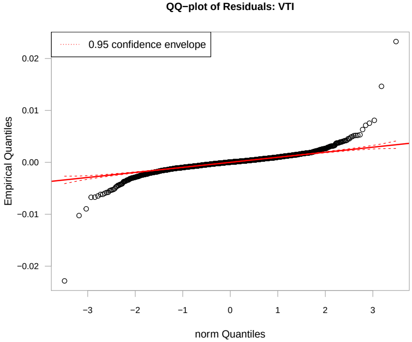
depr: Residual Returns Histogram

```
> library(PortfolioAnalytics)
> # Plot residual histogram with normal curve
> plot(factpca, asset.name="VTI",
+      which.plot.single=8,
+      plot.single=TRUE, loop=FALSE,
+      xlim=c(-0.007, 0.007))
```



depr: Residual Returns and the Q-Q Plot

```
> # Residual Q-Q plot
> plot(factpca, asset.name="VTI",
+      which.plot.single=9,
+      plot.single=TRUE, loop=FALSE)
```



depr: Autocorrelation of Residuals

```
> # SACF and PACF of residuals
> plot(factpca, asset.name="VTI",
+      which.plot.single=5,
+      plot.single=TRUE, loop=FALSE)
```