Postshock

Qiyang Wang

2025-05-28

Contents

Nike data

Import the data form NIKE

```r
nike<-read.csv("NKE_quarterly_valuation_measures.csv",header = TRUE)
nike1<-nike[,-c(1,2)]
A<-matrix(NA,nrow=ncol(nike1),ncol=3)
for (i in 1:ncol(nike1)) {
  A[i,1]<-colnames(nike1)[i]
  A[i,2]<-nike1[2,i]
  A[i,3]<-nike1[8,i]

}
A<-as.data.frame(A)
colnames(A)<-c("date","ev","ev.rev")
A$ev.rev<-as.numeric(A$ev.rev)
A$ev<-as.numeric(gsub(",","",A$ev))
A$date<-gsub("X","",A$date)
A$revenue<-A$ev/A$ev.rev
b<-c()
i=1
for (i in 1:(nrow(A)-1)) {
  b[i]<-((A[i,4]-A[i+1,4])*100/A[i+1,4])/100
}
b[nrow(A)]<-0
A$per<-b
A$per<-round(A$per,3)
```

```r
Nike.f<-A
Nike.f<-Nike.f[1:which(Nike.f$date=="02.28.2001", arr.ind=TRUE),]
```

import the dbw and QL loss function

plot function may be used in the arime part

Starting the arime part testing

Function4: SynthPrediction

SynthPrediction Function — Key Enhancements

Compared to the original, single-path implementation, this version adds:

1. **User-specified ARIMA order**
   - Introduces `arima_order`. When non-`NULL`, the function calls `forecast::Arima()` with the user's `(p,d,q)` (and optionally `(P,D,Q)`), bypassing the default `auto.arima()` search.
2. **SARIMA (seasonal) support**
   - Adds `seasonal`, `seasonal_order` and `seasonal_period` parameters.
   - If `seasonal = TRUE` and no period is provided, automatically detects frequency via `forecast::findfrequency()`.
   - Honors user-supplied seasonal orders when fitting with `Arima()`.
3. **Flexible auto-search controls**
   - Exposes `stepwise` and `approximation` arguments in the `auto.arima()` call, letting users trade off speed vs. accuracy.
   - Wraps the call in `tryCatch()` so that any fit failure falls back to a simple `Arima(1,1,1)` rather than crashing.
4. **Robust covariate lag & NA handling**
   - After creating lagged regressors (`lag.xts()`), automatically drops leading `NA` rows via `na.omit()`, guaranteeing that the dependent series and `xreg` remain time-aligned.
5. **Character or numeric shock times**
   - Accepts both date strings and index positions in `shock_time_vec`, converting either form into integer indices without manual user intervention.
6. **Fixed-effect extraction from donors**
   - Uses `lmtest::coeftest()` on each donor's fit to pull out the coefficient on the "post_shock_indicator" regressor, storing these in `omega_star_hat_vec`.
7. **Distance-based weighting integration**
   - Calls the custom `dbw()` function to compute optimal donor weights `w_hat`, then forms the aggregate shock effect `omega_star_hat = w_hat %*% omega_star_hat_vec`.
8. **Unified multi-step forecasting & adjustment**

- Fits the target series with the same ARIMA/SARIMA logic.
- Generates an unadjusted forecast (via `predict()` or `forecast::forecast()`), then simply adds `omega_star_hat` to produce the "adjusted" forecast.

9. **Structured output and plotting**
   - Returns a list containing both the donor weight vector and two forecast objects (`unadjusted_pred`, `adjusted_pred`).
   - Includes a `plots` flag to trigger `plot_maker_synthprediction()`, producing side-by-side time-series comparisons.
10. **Verbose console feedback**
    - Prints donor count, shock times, weights, MSE, and forecast comparisons via `cat()`, aiding interactive debugging and auditability.

Next Steps

- **This week**: Test the function thoroughly using simulated data to ensure all branches work correctly.
- **Next week**: If you have time, I'd like to schedule a meeting to review results and discuss the plan for further development.