Testing

Contents

Data

Raw Political Data

```
#political shocks
raw_truths <- read.csv(here("data/political_data", "truths_new.csv"))
#raw_tweets <- read.csv(here("data/political_data", "tweets.csv"))
```

Raw Data

```
#market prices (loads and names them automatically)
#raw_ONEQ <- read.csv(here("data/market_data", "ONEQ.csv")) #USA
#raw_SMI <- read.csv(here("data/market_data", "SMI.csv")) #CH
#raw_VTHR <- read.csv(here("data/market_data", "VTHR.csv")) #USA
#raw_VTI <- read.csv(here("data/market_data", "VTI.csv")) #USA
#raw_DAX <- read.csv(here("data/market_data", "DAX.csv")) #DE
#raw_ASHR <- read.csv(here("data/market_data", "ASHR.csv")) #CHINA
raw_SPYy <- read.csv(here("data/market_data", "Spyqyahoo.csv")) #yahoo

#S&P500
data_loader(symbol="SPY")

#STOXX50
data_loader(symbol="VGK")




#CSI 300 (China)
data_loader(symbol="ASHR")
```

Daily Data

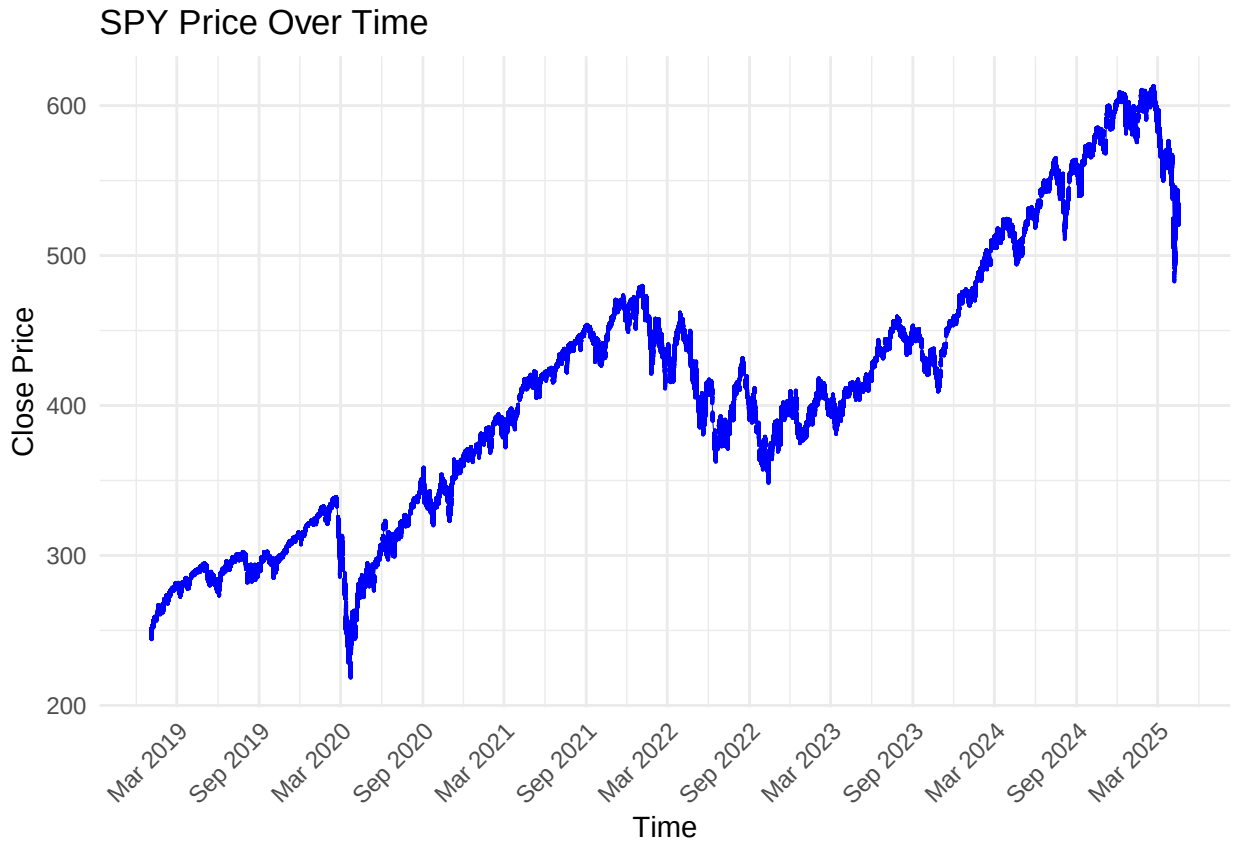
```
#political shocks


#market prices
day_SPY_0409 = filter(raw_SPY, str_detect(timestamp, "^2025-04-09")) #9th of april
day_SPY_0409$timestamp = as.POSIXct(day_SPY_0409$timestamp,
                                    format = "%Y-%m-%d %H:%M:%S", tz = "EST")

yahoo_ds0409 = filter(raw_SPYy, str_detect(Date, "^2025-04-09"))
yahoo_ds0409$Date = as.POSIXct(yahoo_ds0409$Date,
                                    format = "%Y-%m-%dT%H:%M:%S", tz = "UTC")
yahoo_ds0409$Date = with_tz(yahoo_ds0409$Date, "EST")


#extract a particular month
```

```
SPY_24_09 = month_selector(raw_SPY,2024,09) #november 2024


#extract a particular year
SPY_24 = year_selector(raw_SPY,2024) #2024
```

Plots

Total

```
#SPY
ggplot(raw_SPY, aes(x = as.POSIXct(timestamp), y = close)) +
  geom_point(color = "blue", size = 0.01) +
  geom_line(aes(group=1), color="blue", linewidth=0.05) +
  labs(title = "SPY Price Over Time",
       x = "Time",
       y = "Close Price") +
  scale_x_datetime(date_labels = "%b %Y", date_breaks = "6 month") +
      theme_minimal() +
      theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Per Day

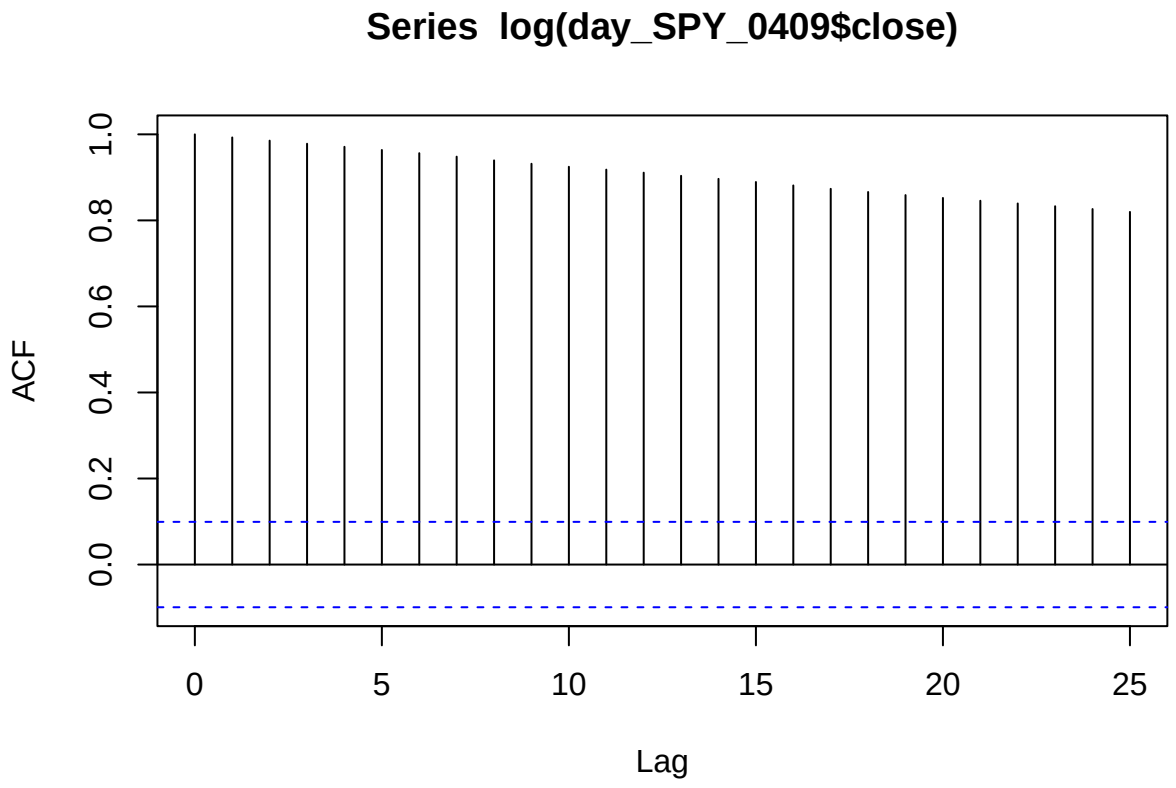
```
#SPY Source: alpha
ggplot(day_SPY_0409, aes(x = timestamp, y = close)) +
  geom_point(color = "blue", size = 0.01) +
  geom_line(aes(group=1)) +
  labs(title = "SPY alpha Price April 9th",
       x = "Time",
       y = "Close Price") +
   scale_x_datetime(date_labels = "%H:%M",
                    date_breaks = "60 min") +
   theme_minimal() +
   theme(axis.text.x = element_text(angle = 45, hjust = 1))
```


```
#SPY Source: yahoo
ggplot(yahoo_ds0409, aes(x = Date, y = Close)) +
  geom_point(color = "blue", size = 0.01) +
  geom_line(aes(group=1)) +
  labs(title = "SPY yahoo Price April 9th",
       x = "Time",
       y = "Close Price") +
   scale_x_datetime(date_labels = "%H:%M",
                    date_breaks = "60 min") +
   theme_minimal() +
   theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

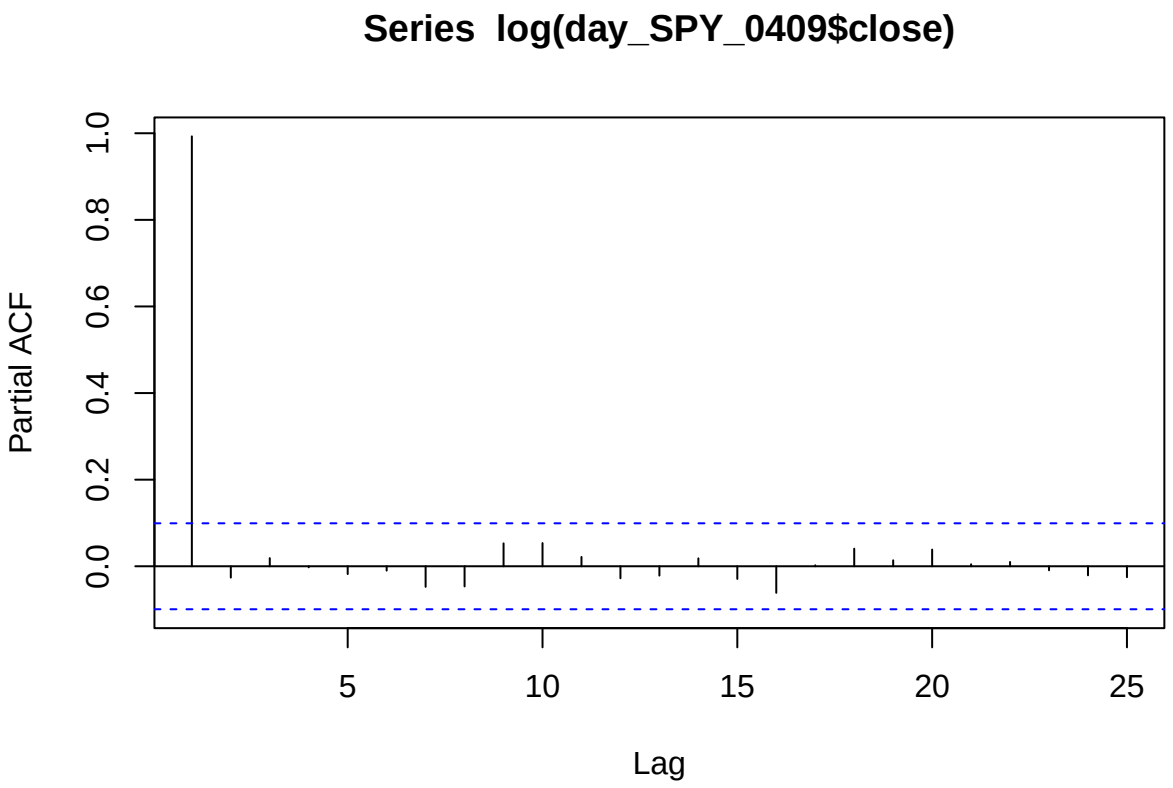
```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Time Series Analysis

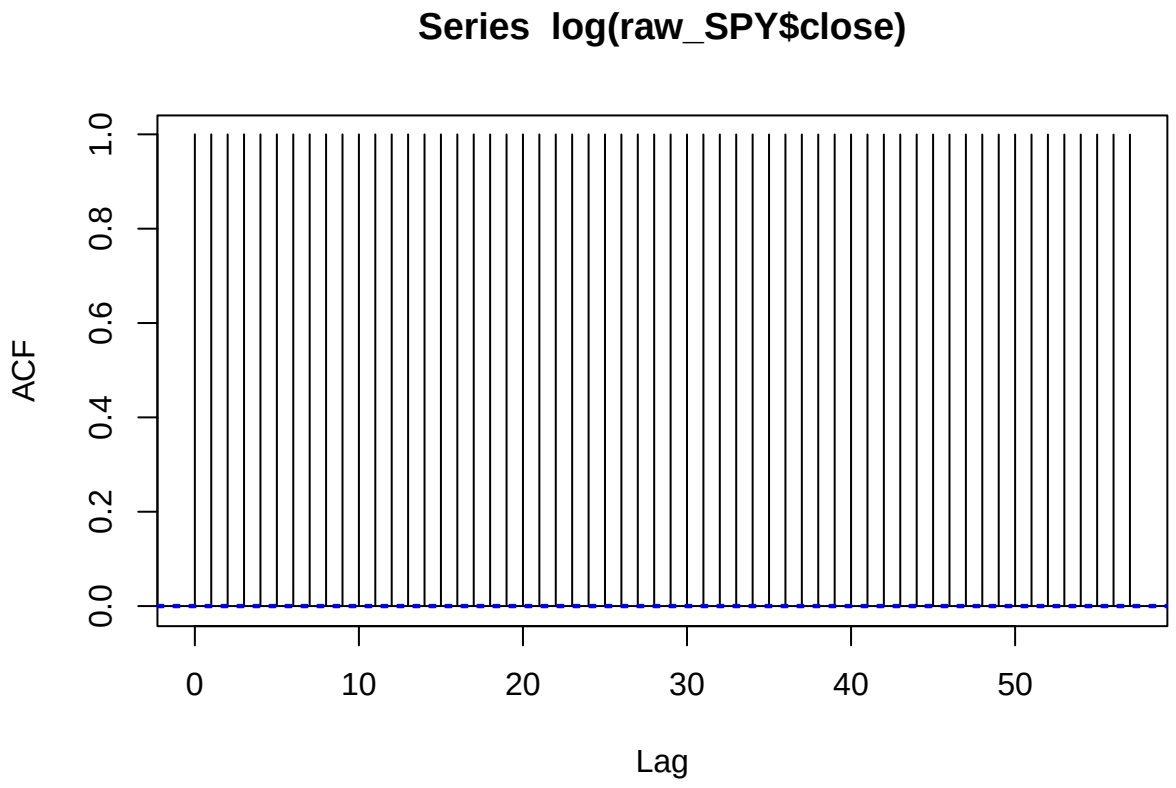
```
acf(log(day_SPY_0409$close))
```



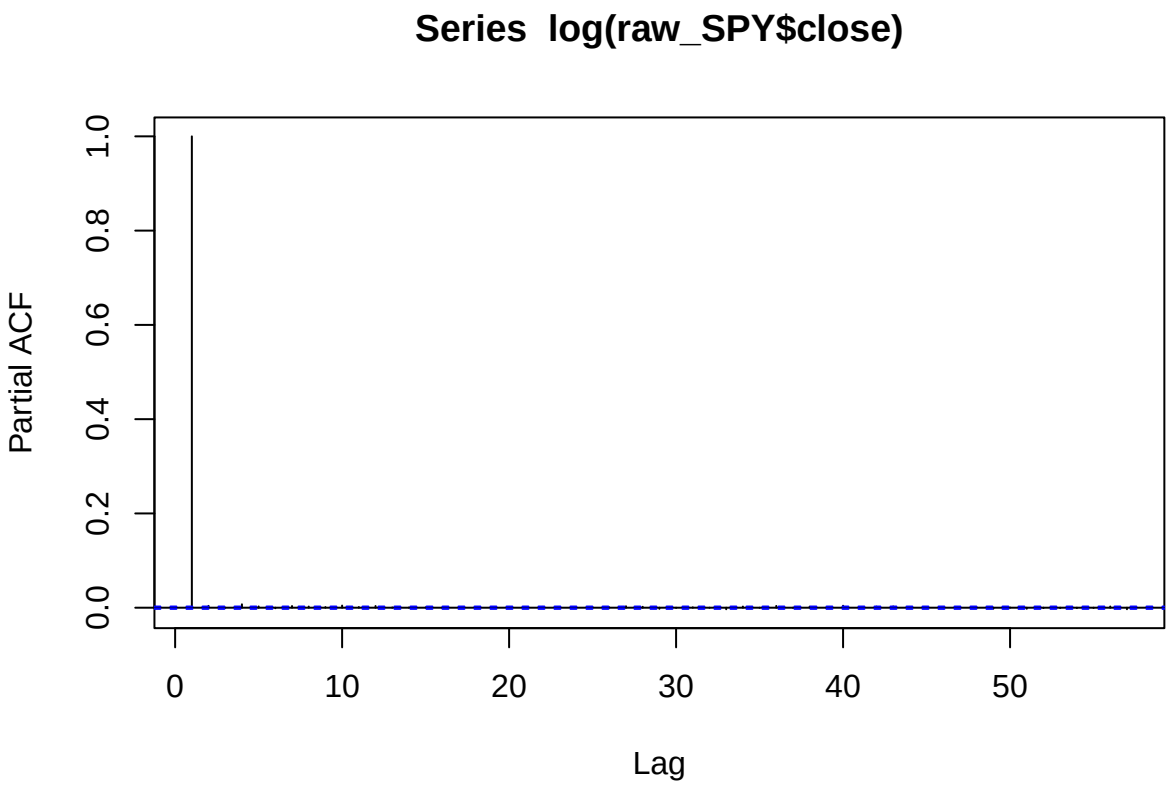

```
pacf(log(day_SPY_0409$close))
```




```
acf(log(raw_SPY$close))
```




```
pacf(log(raw_SPY$close))
```




```
AR1 = arima(day_SPY_0409$close,c(1,0,0),method="ML")
AR2 = arima(day_SPY_0409$close,c(2,0,0),method="ML")
AR3 = arima(day_SPY_0409$close,c(3,0,0),method="ML")
table1 = export_summs(AR1,AR2,AR3, model.names = c("AR1","AR2","AR3"), digits = 4)
```

```
## Warning in FUN(X[[i]], ...): tidy() does not return p values for models of
## class data.frame; significance stars not printed.
## Warning in FUN(X[[i]], ...): tidy() does not return p values for models of
## class data.frame; significance stars not printed.
## Warning in FUN(X[[i]], ...): tidy() does not return p values for models of
## class data.frame; significance stars not printed.
```

```
huxtable::caption(table1) <- "AR Estimations"
huxtable::set_width(table1, 0.8)
```

```
AR1res = as.numeric(AR1$residuals)
AR1res_lagged <- lag(AR1res, 1)
iidcheck1 = lm(AR1res ~ AR1res_lagged)
AR2res = as.numeric(AR2$residuals)
AR2res_lagged <- lag(AR2res, 1)
iidcheck2 = lm(AR2res ~ AR2res_lagged)
AR3res = as.numeric(AR3$residuals)
AR3res_lagged <- lag(AR3res, 1)
iidcheck3 = lm(AR3res ~ AR3res_lagged)
```

Table 1: AR Estimations

	AR1	AR2	AR3
ar1	0.9983	1.0884	1.0919
	(0.0020)	(0.0504)	(0.0506)
intercept	517.4887	516.8318	517.3178
	(19.5350)	(18.7930)	(19.1239)
ar2		-0.0902	-0.1336
		(0.0505)	(0.0746)
ar3			0.0399
			(0.0506)
nobs	390	390	390
sigma	1.2932	1.2880	1.2869
logLik	-656.5286	-654.9411	-654.6302
AIC	1319.0572	1317.8822	1319.2604
BIC	1330.9556	1333.7468	1339.0912
nobs.1	390.0000	390.0000	390.0000

*** p < 0.001; ** p < 0.01; * p < 0.05.

```
table2 = export_summs(iidcheck1,iidcheck2,iidcheck3,
             model.names = c("AR1 Residuals","AR2 Residuals","AR3 Residuals"),
             digits = 4)
huxtable::caption(table2) <- "Checking Residuals"
huxtable::set_width(table2, 0.8)
```

Volatility

JPR Formula

$$v_t = \frac{1}{N}\sum_{i=1}^{N}(\Delta p_{t,i})^2$$

where Δp_t is the difference in price (open - close)
and i represents every minute

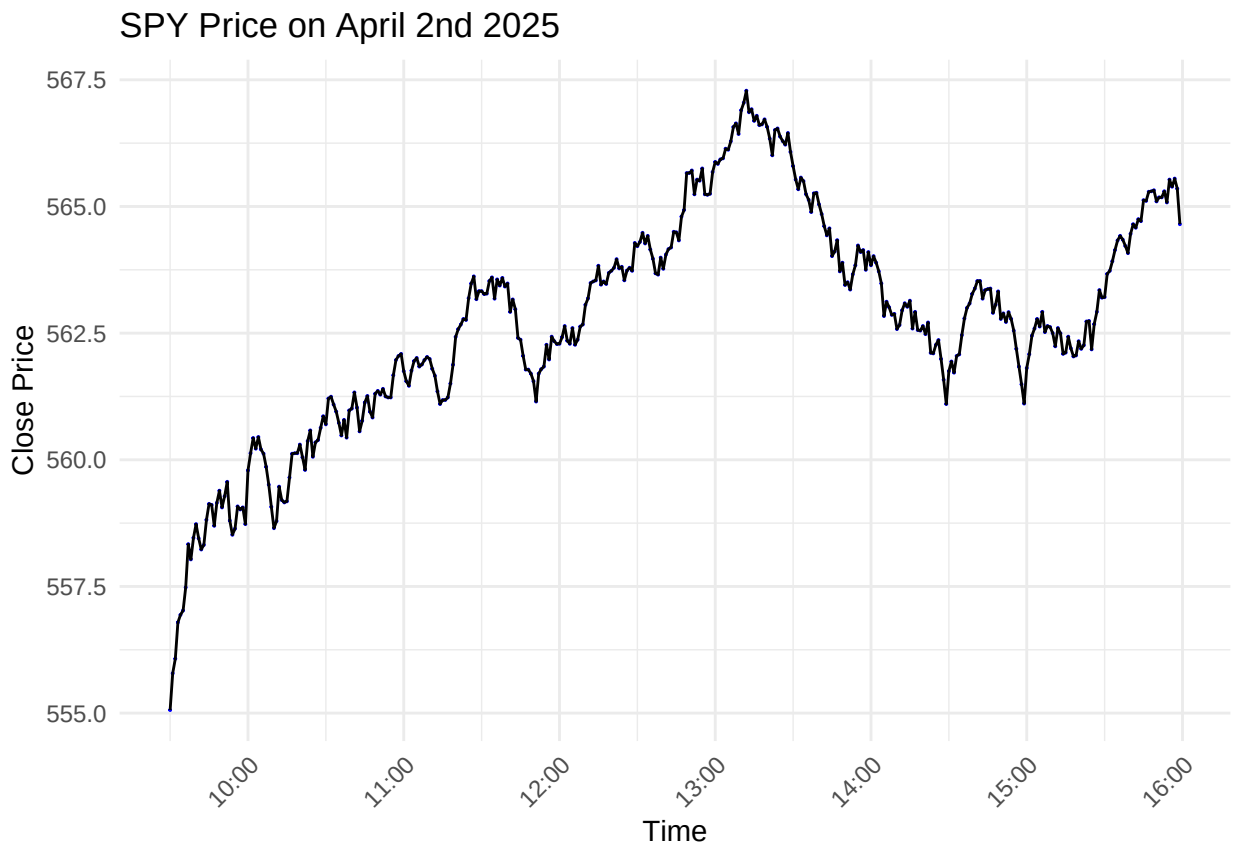
```
#extract a particular day
SPY_25_04_02 = day_selector(raw_SPY,2025,04,02) #april 2nd 2025
```

Table 2: Checking Residuals

	AR1 Residuals	AR2 Residuals	AR3 Residuals
(Intercept)	0.1102	0.1092	0.1135
	(0.0655)	(0.0655)	(0.0654)
AR1res_lagged	0.0799		
	(0.0506)		
AR2res_lagged		-0.0054	
		(0.0508)	
AR3res_lagged			-0.0078
			(0.0508)
N	389	389	389
R2	0.0064	0.0000	0.0001

*** p < 0.001; ** p < 0.01; * p < 0.05.

```
#let's plot it
price_plotter_day(SPY_25_04_02,"SPY Price on April 2nd 2025")
```




```
#quick
quickplot = function(x){price_plotter_day(day_selector(raw_SPY,2025,04,x),"SPY Price 2025")}
quickplot(9)
```


```
#realized volatility
delta_price = SPY_25_04_02$close - SPY_25_04_02$open
delta_price_sqr = delta_price^2
SPY_25_04_02 = cbind(day_selector(raw_SPY,2025,04,02),delta_price_sqr)
v_t = sum(delta_price_sqr) / length(delta_price)

#or is it like this??
#realized volatility method2
p_t = SPY_25_04_02$close
p_t_1 <- lag(p_t, 1)
delta_price2 = p_t_1 - p_t
v_t2 = sum((na.omit(delta_price2))^2) / length(na.omit(delta_price2))
```

```
#average per day (outputs scalar)
r.vol_day(SPY_25_04_02)
```

```
## [1] 2.58157e-07
```

```
#average per day for each day in a month (outputs vector of each day's realised volatility)
r.vol_month(SPY_24_09)
```

```
##  [1] 1.152006e-07 2.072106e-07 1.481908e-07 2.662110e-07 8.731839e-08
##  [6] 1.030985e-07 2.298372e-07 1.497054e-07 8.000296e-08 7.206636e-08
## [11] 1.000058e-07 4.548058e-07 9.703720e-08 7.208847e-08 5.626214e-08
## [16] 4.983626e-08 3.291504e-08 5.634374e-08 4.475474e-08 6.302773e-08
```

```
#for each hour in a day (outputs a vector of each hour's realised volatility)
r.vol_day_hour(SPY_25_04_02)
```

```
## [1] 5.063716e-07 2.771958e-07 2.077509e-07 1.979739e-07 1.978145e-07
## [6] 2.614280e-07 2.118669e-07
```

```
#for each hour in a day for each day in a month (outputs a matrix)
month_hour = r.vol_month_hour(SPY_24_09)
huxtable(head(data.frame(month_hour)))
```

X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	
98e-08	1.01e-07	4.04e-07	2.39e-07	7.39e-08	1.33e-07	1.2e-07	4.43e-08	2.31e-07	7.53e-08	1.93e-07	4
34e-07	2.03e-07	3.59e-07	2.53e-07	1.71e-07	1.86e-07	8.86e-08	8.2e-08	1.31e-07	7.85e-08	9.11e-08	1
66e-08	1.63e-07	2.48e-07	1.77e-07	5.63e-08	5.69e-08	5.66e-08	5.32e-08	9.8e-08	9.82e-08	4.02e-08	4
13e-08	1.02e-07	2.29e-07	8.9e-08	4.22e-08	6.32e-08	1.49e-07	2.97e-08	3.81e-08	3.48e-08	6.96e-08	2
32e-08	6.32e-08	1.35e-07	4.32e-08	2.94e-08	3e-08	1.19e-07	4.78e-08	3.56e-08	4.03e-08	3.44e-08	2
51e-08	4.9e-08	1.49e-07	2.39e-07	8.85e-08	3.28e-08	1.11e-07	1.04e-06	7.75e-08	7.38e-08	1.15e-08	3

```
#avg per day for each month of any dataset
#works for datasets with more than 1 year!
vol_SPY_daily = r.vol_daily(raw_SPY,merge=F)
head(vol_SPY_daily)
```

timestamp	r__vol__d
2019-01-02	4.76e-07
2019-01-03	6.03e-07
2019-01-04	3.84e-07
2019-01-07	2.57e-07
2019-01-08	2.09e-07
2019-01-09	2.16e-07

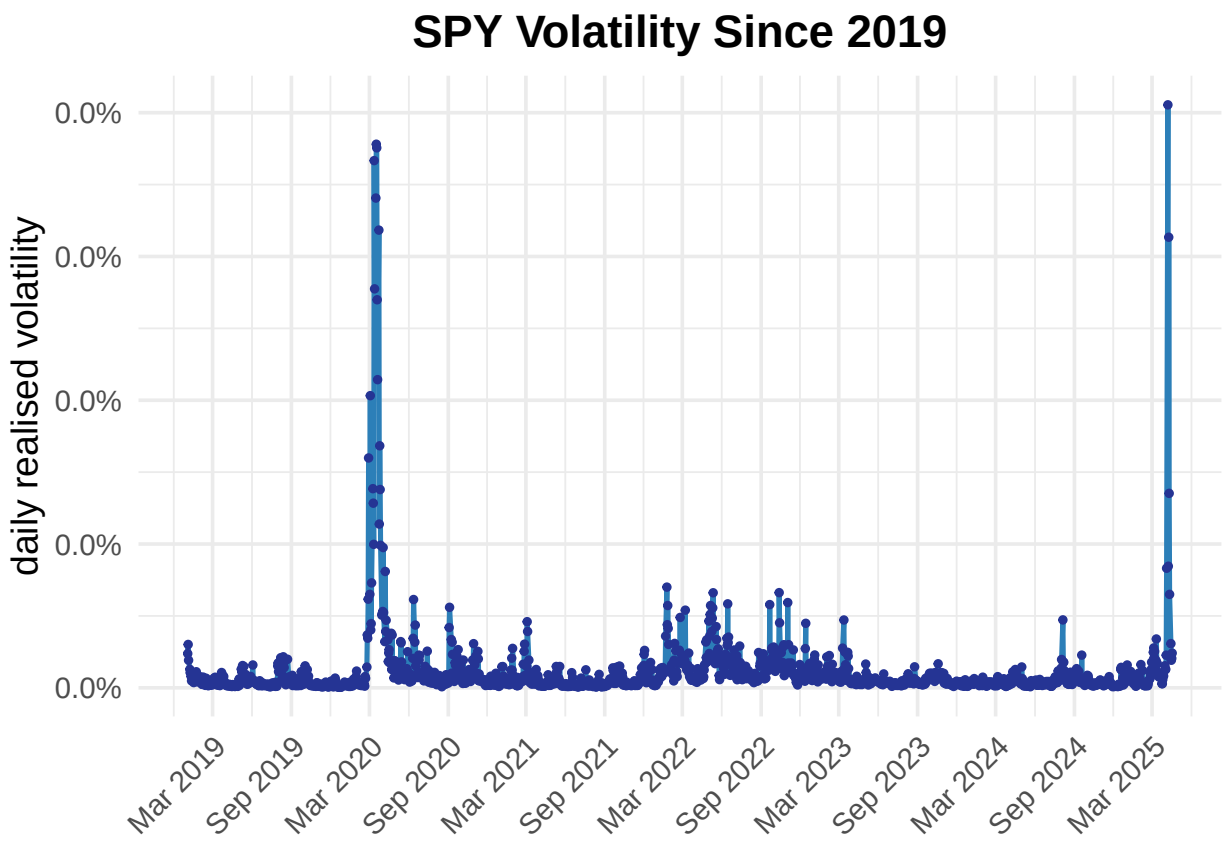
```
#can then filter out years, months, or days
vol_24d = year_selector(vol_SPY_daily,2024)
vol_24_08d = month_selector(vol_SPY_daily,2024,08)
vol_24_11_04d = day_selector(vol_SPY_daily,2024,11,04) #scalar


#avg per hour for each day of each month of any dataset
#works for datasets with more than 1 year!
```

```
vol_SPY_hourly = r.vol_hourly(raw_SPY,merge=F)
head(vol_SPY_hourly)
```

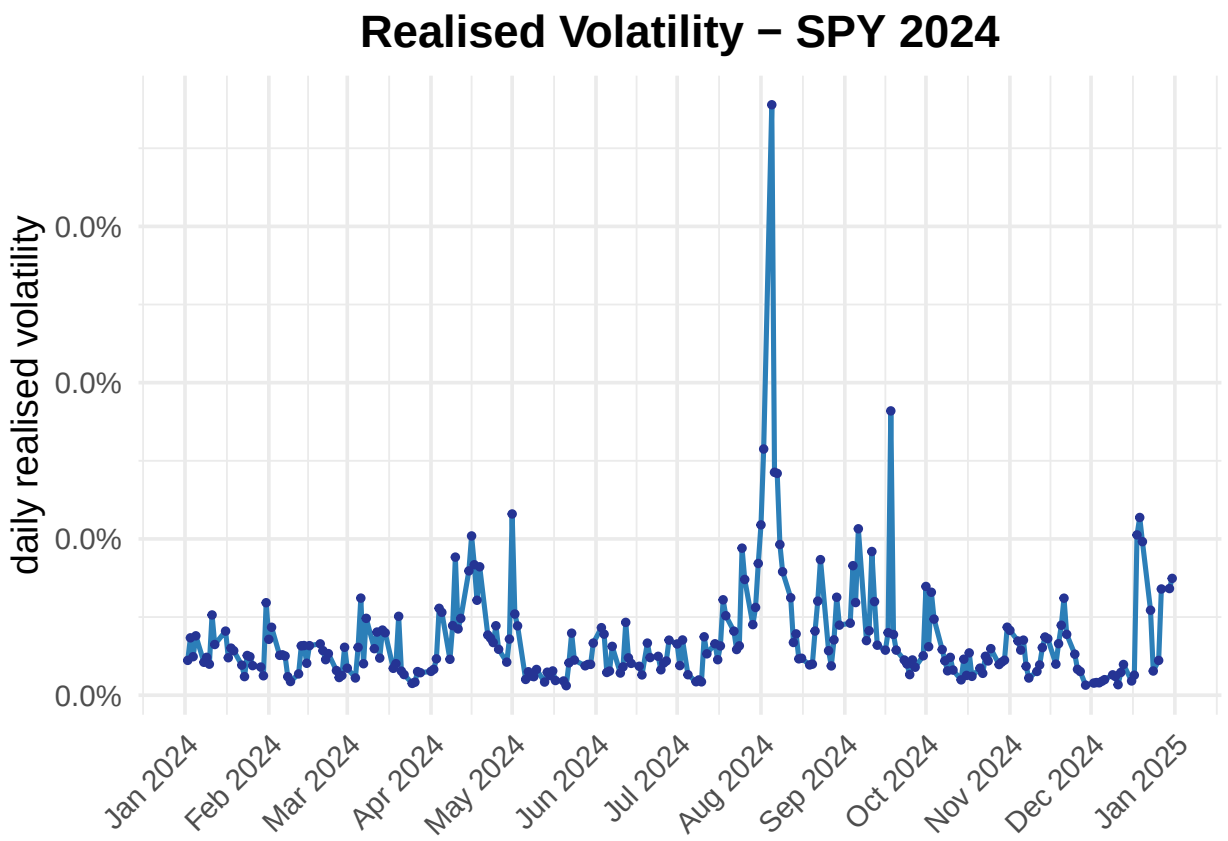
timestamp	r__vol__h
2019-01-02 09:00:00	5.59e-07
2019-01-02 10:00:00	6.51e-07
2019-01-02 11:00:00	5.83e-07
2019-01-02 12:00:00	2.98e-07
2019-01-02 13:00:00	2.94e-07
2019-01-02 14:00:00	3.17e-07

```
#can then filter out years, months, or days
vol_24h = year_selector(vol_SPY_hourly,2024)
vol_24_08h = month_selector(vol_SPY_hourly,2024,08)
vol_24_11_04h = day_selector(vol_SPY_hourly,2024,11,04) #vector
```

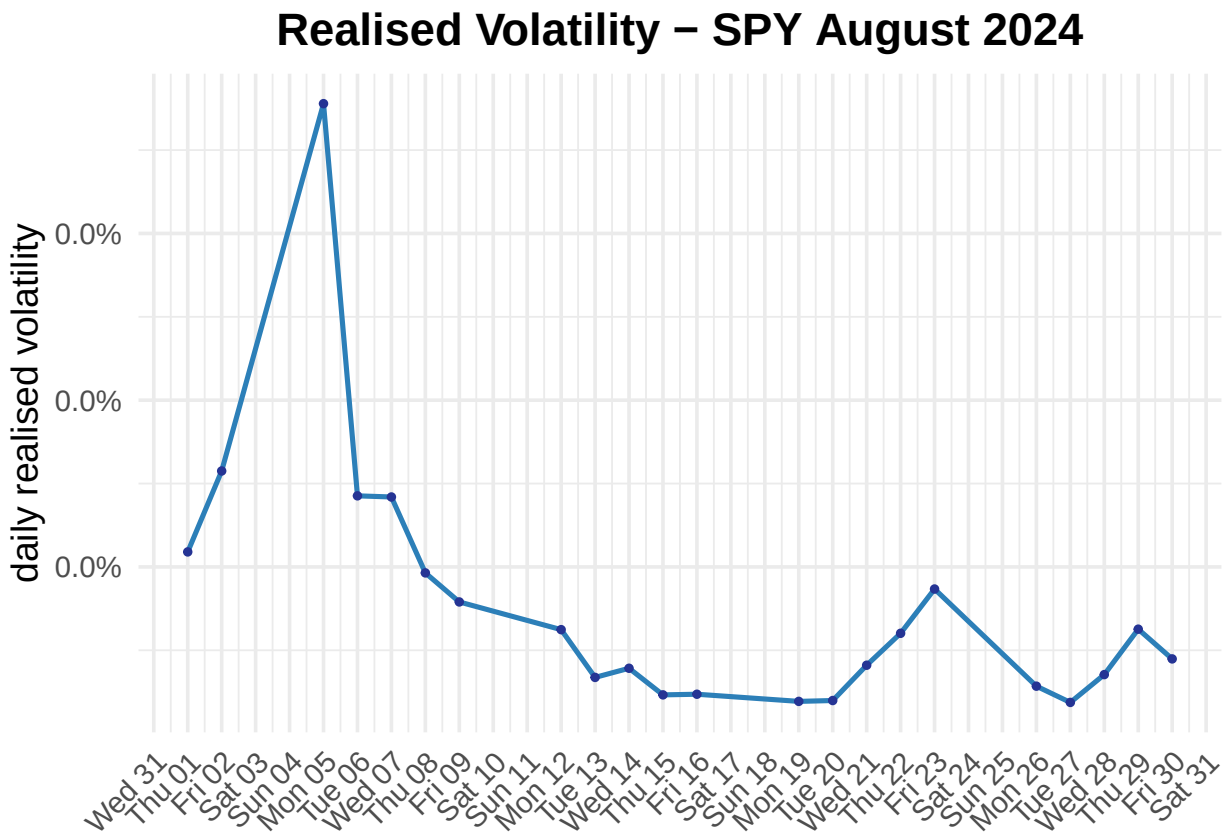
```
#avg per day volatility all time
dvol_plotter(vol_SPY_daily,breaks="yearly",
             title="SPY Volatility Since 2019")
```



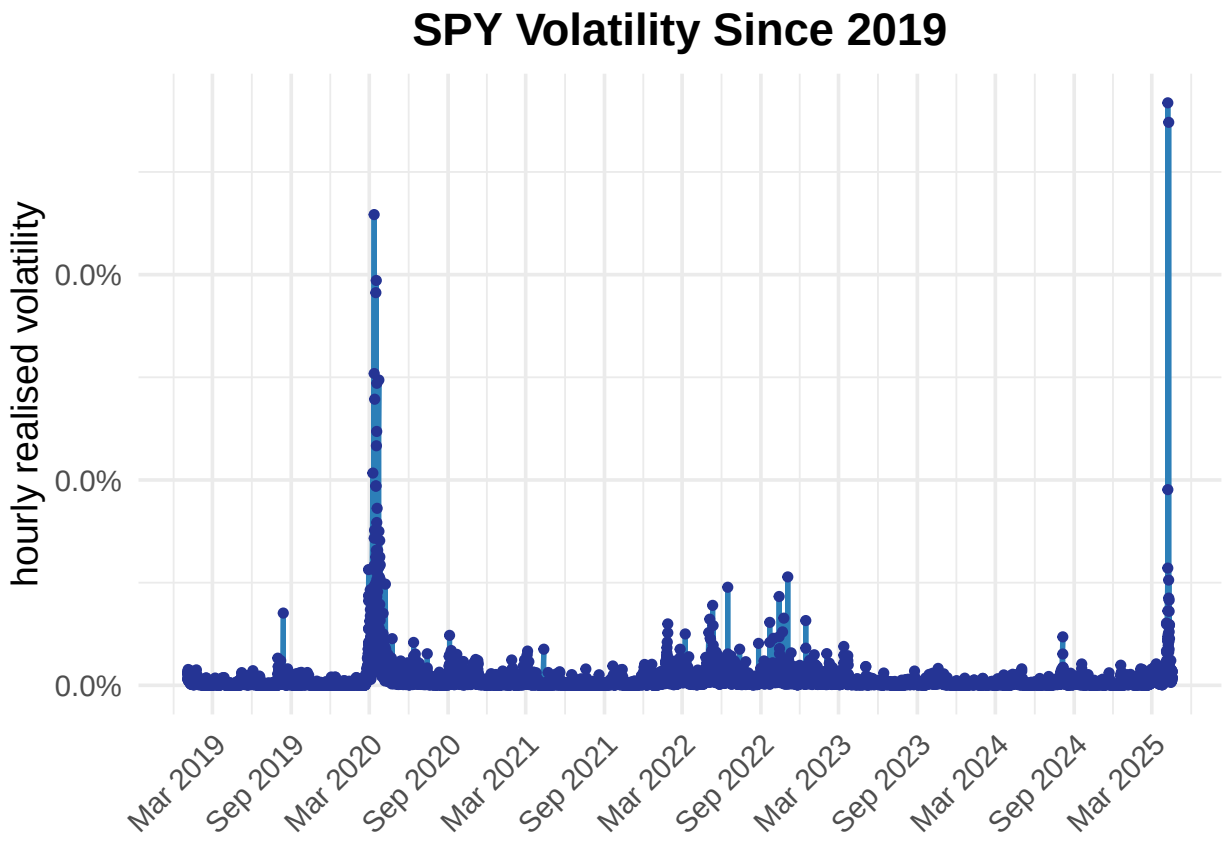
```
#avg per day volatility in a year
dvol_plotter(vol_24d,breaks="monthly",
             title="Realised Volatility - SPY 2024")
```



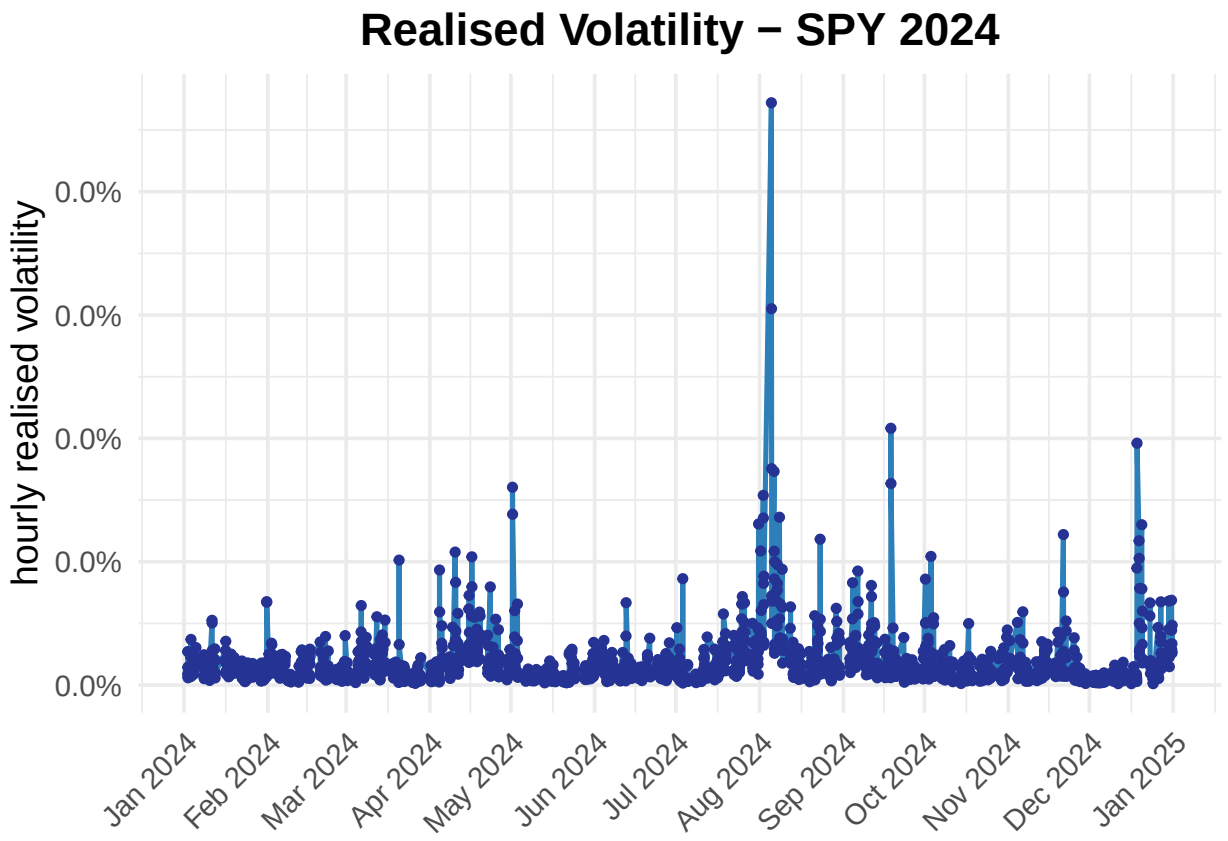

```
#avg per day volatility in a month
dvol_plotter(vol_24_08d,breaks="daily",
             title="Realised Volatility - SPY August 2024")
```



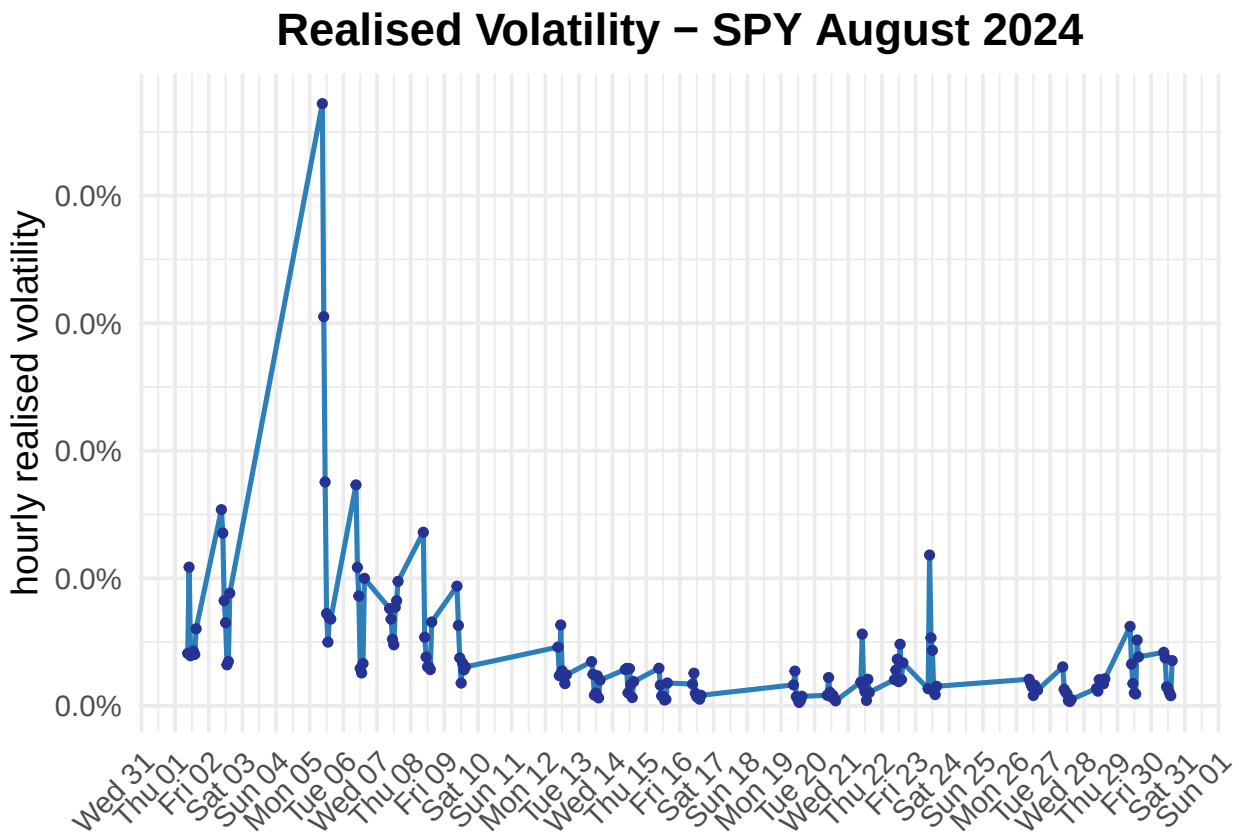

```
#hourly volatility all time
hvol_plotter(vol_SPY_hourly,breaks="yearly",
             title="SPY Volatility Since 2019")
```



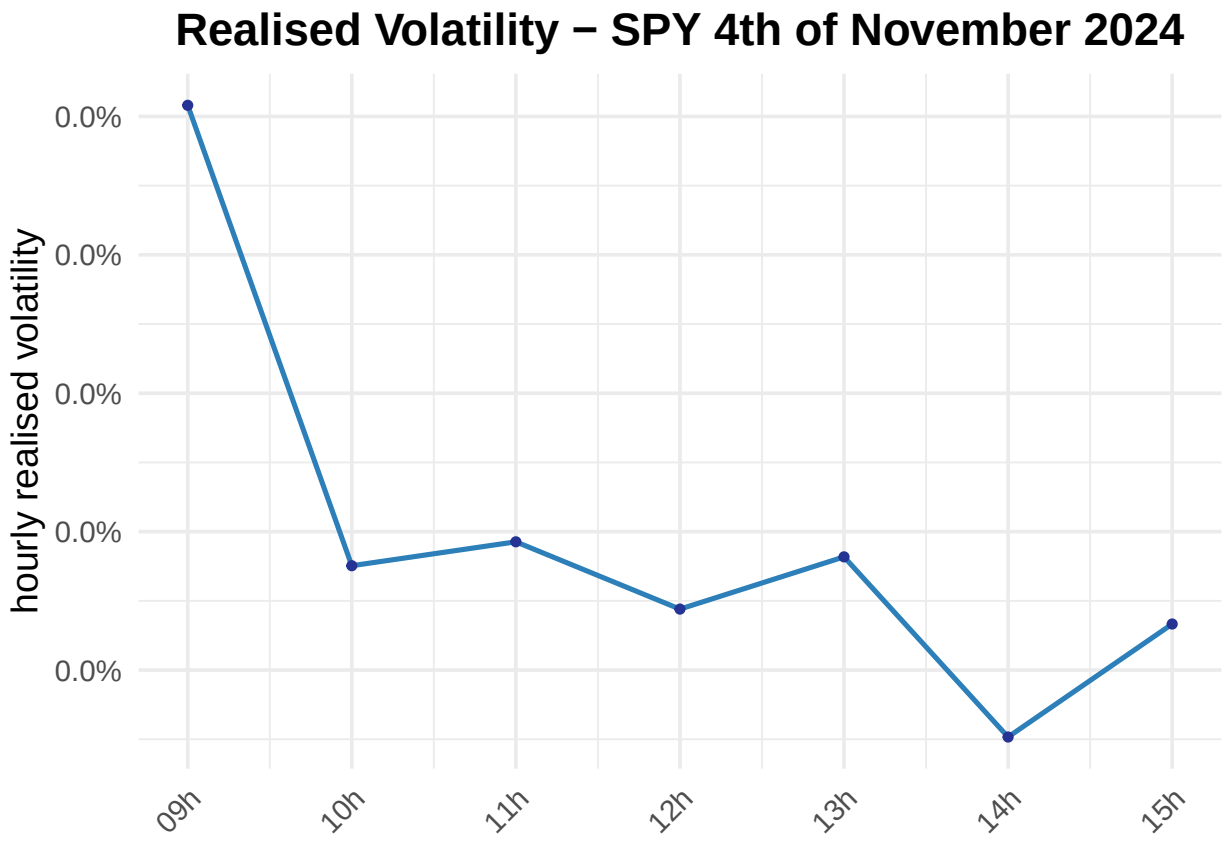

```
#hourly volatility in a year
hvol_plotter(vol_24h,breaks="monthly",
             title="Realised Volatility - SPY 2024")
```




```
#hourly volatility in a month
hvol_plotter(vol_24_08h,breaks="daily",
             title="Realised Volatility - SPY August 2024")
```




```
#hourly volatility in a day
hvol_plotter(vol_24_11_04h,breaks="hourly",
             title="Realised Volatility - SPY 4th of November 2024")
```



Garman and Klass (1980) Formula

Note that this formula uses open-high-low-close information. \ This model is based on the assumption that price returns follow a Wiener process with zero drift and constant infinitesimal variance. It's constructed by minimizing the variance of a quadratic estimator subject to the constraints of price and time symmetry and scale invariance of volatility. Source: https://assets.bbhub.io/professional/sites/10/intraday_volatility-3.pdf

$$V_{ohlc} = 0.5[log(H) - log(L)]^2 - [2log(2) - 1][log(C) - log(O)]^2$$

```
#extract a particular day
day_SPY_0402 = day_selector(raw_SPY,2025,04,02) #april 2nd 2025

#variables
C = day_SPY_0402$close
O = day_SPY_0402$open
H = day_SPY_0402$high
L = day_SPY_0402$low

#realized volatility
V_ohlc = 0.5*(log(H)-log(L))^2 - (2*log(2)-1)*(log(C)-log(O))^2
v_ohlc = sqrt(V_ohlc)
day_SPY_0402 = cbind(day_selector(raw_SPY,2025,04,02),V_ohlc)
avg = sum(V_ohlc) / length(V_ohlc)
```

Tweets

```
#cleanup
truthsbackup <- truths_processer(raw_truths)
truths = as.data.table(truthsbackup)
names(truths)[names(truths) == 'date_time_parsed'] <- 'timestamp'
truths <- truths[order(truths$timestamp, decreasing=T), ]

#count by hour
truth_count = truths[, .N, by=.(year(timestamp), month(timestamp),
                                day(timestamp), hour(timestamp))]

#fix timestamp
truth_count$timestamp = as.POSIXct(sprintf("%04d-%02d-%02d %02d:00:00",
                         truth_count$year, truth_count$month, truth_count$day,
                         truth_count$hour), format = "%Y-%m-%d %H:00:00")

#remove useless columns and reorder by oldest first
truth_count = select(truth_count, timestamp, N)
truth_count = truth_count[ order(truth_count$timestamp , decreasing = F ),]
head(truth_count)
```

timestamp	N
2022-02-14 10:00:00	1
2022-03-04 09:00:00	1
2022-03-12 20:00:00	1
2022-03-14 13:00:00	1
2022-03-14 15:00:00	1
2022-03-14 22:00:00	1

ARMA-X (hourly)

```
#merge by matching the hours
#SPY_prices = select(raw_SPY, timestamp, close, volume)
#SPY_prices$timestamp <- as.POSIXct(raw_SPY$timestamp)
#SPY_prices$timestamp_hour <- as.POSIXct(format(SPY_prices$timestamp,
#                                               "%Y-%m-%d %H:00:00"))
#prices dont make sense cuz not hourly
#armax_price = merge(SPY_prices, truth_count, by.x = "timestamp_hour",
#                    by.y = "timestamp", all.x = TRUE)


#merge by matching the hours
```

```r
#NOTE: this ignores tweets made outside trading hours!!
SPY_volatility = r.vol_hourly(raw_SPY,merge=F)
#since tweet database not full yet (combine tweets & truths)
SPY_volatility = filter(SPY_volatility,
                        year(SPY_volatility$timestamp) >= 2022 &
                        month(SPY_volatility$timestamp) >= 03)
colnames(SPY_volatility)[1] <- "timestamp_hour"

#take all relevant for armax
armax_vol = merge(SPY_volatility, truth_count, by.x = "timestamp_hour",
                  by.y = "timestamp", all.x = T)
#NA tweets means no tweets
armax_vol$N[is.na(armax_vol$N)] = 0
```