NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS
International College of Economics and Finance

Karapsin Danila

Gulomjonov Furkatjon

Takehome Exam, Topic 8

38.04.01 ECONOMICS

Master's Programme **'Financial Economics'**

Moscow 2024

Contents

a) - data loading

This section aims to show how did we load data needed for the project. The code which does the job is contained in the *data_load.R* file and duplicated below for convenience. The idea is simple: firstly, we save a data frame with all cryptocurrencies in the *crypto_df* object, then we are using *walk* function from *purrr* package to load every needed token, then the loaded data is saved in the loaded data folder in the rds format.

```
rm(list = ls())
gc()
library(dplyr)
library(crypto2)
library(purrr)

crypto_df <-
  crypto_list()%>%
    filter(id <= 10000)

walk(unique(crypto_df$id),
    ~ crypto_df%>%
       filter(id == .x)%>%
       crypto_history(convert = "USD")%>%
       saveRDS(paste0("loaded_data\\", .x, ".rds"))
)
```

b), c) - data cleaning + forecasting

This section provides an overview of the code from *predictions.R* file. Firstly, we consolidate each data frame loaded on the previous step into a single data frame using *map_dfr* function from *purrr* package. Then, we select only columns which will be needed further, keeping only those coins for which we have from 60 to 730 observations. Lastly, we compute log returns and also for every coin we are computing start of the prediction interval, which is equal to last_prediction_day + 1.

Note that *group_by(id)* in the code below ensures that filtering is correct, also because of that last_prediction_day + 1 is defined for every coin separately.

```
library(magrittr)
library(purrr)
library(forecast)
library(dplyr)

# loaded data concatenation + log_returns + some helpful vars
df <-
  list.files("loaded_data")%>%
    map_dfr(~.x%>%
              paste0("loaded_data\\", .)%>%
              readRDS()
    )%>%

  select(id, time_open, close)%>%
  group_by(id)%>%
  filter(time_open%>%
           length()%>%
           unique()%>%
           between(60, 730)
  )%>%
  arrange(id, time_open)%>%
  mutate(day_num = row_number())%>%
  mutate(last_day_num = max(day_num))%>%
  mutate(log_return = close%>%
                        `/`(lag(close, 1))%>%
                        log(),
         last_prediction_day = last_day_num/2
  )%>%
  filter(!is.na(log_return))%>%
  ungroup()%>%
  select(-close)
```

Here we are using expanding window approach to forecast log returns. The idea is simple, firstly, we start a *while* block. Inside that block for every iteration we are checking how many rows we still need to fill, if that number is equal to 0, then the job is done and we are breaking the *while* loop. Otherwise we continue. Then we use *filter* on df object, to keep only training period. In the end, we use *group_by(id)* to compute *auto.arima()* one

step ahead forecast, from which we are taking a middle point. After that, made prediction are added to *predictions_df* object and we are increasing last_prediction_day by 1 for those coins for which made predictions on the current step.

```
# expanding window, auto.arima on every step
predictions_df <- data.frame(id = numeric(),
                             day_num = numeric(),
                             predicted_log_return = numeric()
                  )

while(TRUE){

    rows_left <- df%>%filter(day_num > last_prediction_day)%>%nrow()
    if(rows_left == 0){break}
    print(paste0("rows left: ", rows_left))

    current_predictions <-
      df%>%
        filter(day_num < last_prediction_day + 1)%>%
        group_by(id)%>%
        summarise(day_num = max(day_num) + 1,
                  predicted_log_return = log_return%>%
                                         auto.arima()%>%
                                         forecast(h = 1)%$%
                                         mean
        )%>%
        ungroup()

      predictions_df%<>%
        rbind(current_predictions)

    df%<>%
      mutate(last_prediction_day = ifelse(id %in% current_predictions$id,
                                          last_prediction_day + 1,
                                          last_prediction_day)
      )
}
```

After previous step is finished, we are adding predictions to the initial data frame and saving it.

```
# add predicted values and save df for further usage
df%<>%
  left_join(predictions_df, by = c("day_num"="day_num", "id"="id"))

saveRDS(df, "data_with_predictions.rds")
```

d) - prediction intervals

The code below is from the *predictions.R* file and it computes all the required prediction intervals, except the "FACI" intervals mentioned in the task, for which there is no function in the package. The data frame with computed prediction intervals is then saved in the rds format. Note again that we use *group_by(id)* to compute all the intervals for each coin separately.

```
rm(list = ls())
library(magrittr)
library(purrr)
library(forecast)
library(dplyr)
library(AdaptiveConformal)

df <- readRDS("data_with_predictions.rds")

intervals_df <-
df%>%
  filter(!is.na(predicted_log_return))%>%
  group_by(id)%>%
  summarise(
            ################################################
            # alpha 5%
            intervals_AgACI5 =
            aci(method = "AgACI")%>%
            update(newY = log_return,
                   newpredictions = predicted_log_return
            )%$%
            intervals,

            # no FACI in the package :(
            intervals_dtACI5 =
            aci(method = "DtACI")%>%
            update(newY = log_return,
                   newpredictions = predicted_log_return
            )%$%
            intervals,

            intervals_SF_OGD5 =
            aci(method = "SF-OGD")%>%
            update(newY = log_return,
                   newpredictions = predicted_log_return,
                   parameters=list(
                   gamma = max(abs(log_return-predicted_log_return))/sqrt(3)
                     )
            )%$%
```

```
        intervals,

        intervals_SAOCP5 =
        aci(method = "SAOCP")%>%
        update(newY = log_return,
               newpredictions = predicted_log_return,
               parameters=list(
               D = max(abs(log_return-predicted_log_return))/sqrt(3)
                 )
               )%$%
        intervals,

        ###################################################
        # alpha 1%
        intervals_AgACI1 =
        aci(method = "AgACI", alpha = 0.99)%>%
        update(newY = log_return,
               newpredictions = predicted_log_return
        )%$%
        intervals,

        # no FACI in the package :(
        intervals_dtACI1 =
        aci(method = "DtACI", alpha = 0.99)%>%
        update(newY = log_return,
               newpredictions = predicted_log_return
        )%$%
        intervals,

        intervals_SF_OGD1 =
        aci(method = "SF-OGD", alpha = 0.99)%>%
        update(newY = log_return,
               newpredictions = predicted_log_return,
               parameters=list(
               gamma = max(abs(log_return-predicted_log_return))/sqrt(3)
                 )
        )%$%
        intervals,

        intervals_SAOCP1 =
        aci(method = "SAOCP", alpha = 0.99)%>%
        update(newY = log_return,
               newpredictions = predicted_log_return,
               parameters=list(
               D = max(abs(log_return-predicted_log_return))/sqrt(3)
                 )
```

```
            )%$%
            intervals
  )

# we can do that since the order of days is preserved
intervals_df$day_num <-
  df%>%
  filter(!is.na(predicted_log_return))%$%
  day_num

intervals_df$log_return <-
  df%>%
  filter(!is.na(predicted_log_return))%$%
  log_return

saveRDS(intervals_df, "VaR_data.rds")
```

e) - backtesting

The code in *uc_cc_report.R* file allows to create a table which provided below. The code itself is not provided in that report because of its ugliness (but it is availiable in the mentioned file). "+" indicates that both tests allowed to reject H0, "uc" or "cc" inidicate that we were able to reject the null only with one test, and, finally, "-" means that we can not reject the null with both tests.

id	0.5_AgACI	2.5_AgACI	0.5_DtACI	2.5_DtACI	0.5_SF-OGD	2.5_SF-OGD	0.5_SAOCP	2.5_SAOCP
22	cc	cc	cc	cc	+	+	-	-
26	NaN	NaN	NaN	NaN	+	+	NaN	-
1612	NaN	+	NaN	+	+	-	NaN	-
1956	-	-	-	-	+	+	NaN	-
2997	-	cc	-	cc	+	cc	NaN	+
3415	-	-	-	-	+	+	-	-
3906	-	uc	-	+	+	uc	-	-
4100	uc	-	uc	-	+	+	-	-
4474	cc	cc	-	-	+	+	-	-
4513	cc	-	cc	-	+	-	-	-
4596	-	cc	-	cc	+	-	-	+
4790	-	-	-	-	NaN	-	NaN	NaN
4883	-	+	-	uc	-	uc	NaN	+
5066	-	-	-	-	+	+	-	-
5145	+	-	+	-	+	+	-	-
5602	-	-	-	-	+	-	NaN	-
6224	-	-	-	-	+	-	-	-
6235	NaN	+	NaN	+	+	+	-	+
6252	NaN	NaN	NaN	NaN	+	+	NaN	NaN
6377	cc	cc	cc	cc	+	-	-	uc
6759	-	-	-	-	+	+	-	+
6780	NaN	+	NaN	+	+	+	-	+
6827	cc	cc	cc	cc	+	+	-	+
6845	-	-	-	uc	+	-	-	+
6961	NaN	+	NaN	+	+	cc	NaN	+
7719	NaN	NaN	NaN	NaN	+	+	NaN	-
7760	+	cc	+	cc	NaN	-	NaN	NaN
7961	-	cc	-	cc	+	cc	-	-
7962	-	-	-	-	-	-	NaN	uc
7963	cc	-	cc	cc	+	cc	-	-
8067	NaN	-	NaN	-	+	+	-	-
8108	-	-	-	uc	+	+	-	uc
8209	-	-	-	-	cc	cc	NaN	-
8266	-	+	NaN	uc	+	+	-	+
8370	+	cc	+	cc	+	-	-	+
8573	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8624	-	+	-	uc	+	-	-	+
8724	NaN	NaN	NaN	NaN	+	+	NaN	-
8977	-	uc	-	-	+	+	-	-
9052	-	uc	-	+	+	cc	-	+
9063	-	-	-	-	-	-	NaN	-
9069	NaN	+	NaN	+	+	+	-	+
9184	-	-	-	-	-	-	NaN	-
9261	-	-	-	-	+	+	NaN	-
9404	-	-	-	-	+	+	NaN	-
9427	-	-	-	uc	+	-	-	-
9486	-	+	-	+	cc	-	-	+
9531	-	cc	-	-	+	cc	NaN	+
9785	-	+	NaN	+	+	cc	-	+
9808	NaN	NaN	NaN	+	+	+	NaN	-

9900	NaN	-	NaN	NaN	+	+	NaN	-

Note that some cells contain "NaN". Sometimes, VaRTest is not working on our data. We investigated source code from the github of the rugarch package and found that this happens when we have no log returns which are below computed VaR (or when we have to few of them). Obviously, in such a case we can not test anything because lack of observations. So we have just "NaN" in that case.

f) - saving the results

For some reason we need to save the results in a very specific format. The code below does exactly that.

```
library(dplyr)
library(purrr)

final_df <-
readRDS("data_with_predictions.rds")%>%
  select(id, time_open, day_num, log_return, predicted_log_return)%>%
  left_join(readRDS("VaR_data.rds")%>%
              select(-log_return),
            by = c("id" = "id", "day_num" = "day_num")
  )

final_df$id%>%
  unique()%>%
  map(~final_df%>%
        filter(id == .x)
  )%>%
  saveRDS("final_result.rds")
```