Project 1 - Application of Model Confidence Set for stocks data

Abhijeet Bhardwaj

May 7, 2022

Summary

The project aims at selecting best models with equal predictive ability (EPA) for different financial stocks data. Set of models with same but superior predictive ability are termed as Superior Set Models (SSM). The report identifies superior set models for five financial time series data. The report also studies extensively the application of the Model Confidence Set (MCS) package of R to identify SSM. Finally an analysis of the python code to run MCS package is also shown. Models with best predictive ability for each time series data are finally reported using the analysis done in R and python together. It is found that MCS package in R helps to identify SSM from 80 models for all 5 stocks however the process takes approximately 1.5hrs to run for a single stock data. The minimum lenght of forecast needed by MCS package is 6 for sampling bootstrap samples without error. I also find that the python code is correctly written but a small error exists in the simulation part of the code which is rectified and reported. The models predicted to be in SSM by python are the same as that predicted by R for all 5 stocks, however, R predicts a larger set of SSM models. Further, to make the process time efficient as number of bootstrap samples are decreased some high ranked models may get eliminated on different machines from SSM due to randomness in statistic generated using bootstrap samples. However, I do find that the python code is not too sensitive to Bootstrap size as R package. Throughout the report I use 'Tmax' statistics denoted in MCS package for both R and python.

1 Introduction

Every statistical analysis aims at identifying models with very high predictive ability. However, as quoted by Box, "all models are wrong some are useful", the target is to identify models that can predict the future as accurately as possible. In order to achieve the same the MCS package in R Bernardi & Catania (2018) provides a routine to identify the subset of models that give minimum forecast errors called Superior Set of Models (SSM). This study analyses negative log returns for five companies stocks which are calculated using closing value of each stock. The report refers reader to the package documentation of Bernardi & Catania (2018) for various acronyms and terms used. The five companies considered are Microsoft (MSFT), Google (GOOG), Adobe (ADB), Amazon(AMZN). American Express (AXP) and JP Morgan Chase & Co (JPM). Five yeas of data is analyzed.

As financial time-series are best described by volatility models I compare the performance of different volatility models with different parameter specifications to identify SSM. Initially a total of 160 model specifications were selected, however it was observed that the time taken to run the same is significantly very high even for a single stock data. Thus the process was terminated and models that were eliminated with very high frequency (for microsoft data as shown in Appendix Figure 1) were dropped for further analysis. Although not completely accurate for all other stocks, this exercise tried to over come the complete arbitrariness in reducing the number of model specifications. It was found that the 'iGarch' model and some family of GARCH sub-models were dropped very frequently.

The five base GARCH models selected are 'sGARCH', 'eGARCH', 'apARCH', 'csGARCH' and the family of GARCH sub-model 'NGARCH'. Each GARCH model have 8 different distributions which are ('norm', 'std', 'ged', 'snorm', 'sstd', 'sged', 'jsu', 'ghyp') which further have two types of mean models. In all there are a total of 80 models specifications to compare. Predictive ability of each model specification is evaluated using the VaR loss of a rolling forecast for each financial stock. The lenght of the forecast was initially suggested to be five, however, it was observed that the *boot.block* function of MCS package allows for a minimum forecast length to be 6 when the minimum number of block bootstrap length (k) is 3 as shown in small experiment in Figure 2. Please refer to the MCS Github location for details *https://github.com/cran/MCS/blob/master/R/MCS.R*. Also having too low forecast lowers the quality of the bootstrapped samples generated by the MCS package. Thus, the forecast length was kept at 10. In predicting the forecast values the 'ruGARCH' package Ghalanos (2022) estimate the model parameter by refitting the model at interval specified by the *refit.every* argument and then predicts one step ahead by using the same estimated parameters for the complete forecast length. To over come the model identifiability issues (see Figure 3) experiments were conducted to select the optimal value for the *refit.every* argument and was found to be 7. As the forecast length was low (10) taking high bootstraped samples does not bring in improvement as same samples would be re-sampled again and again. Thus the bootstrapped samples were reduced to B=1200 for all five stocks. The details of the functions used in R code for the analysis is presented next which is followed by the result tables.

2 Details of R code

The code is developed on R with specifications "R version 4.1.2 (2021-11-01)" nickname- "Bird Hippie". The packages used are *rugarch*, *MCS*, *readxl*→ for reading the stocks data in excel, *timetk*→ for converting the stocks data into the 'xts' format as required for input to MCS package, *parallel*→ to enable parallel computing functionality offered by rugarch and MCS, *this.path* → to identify the path where the R script is kept. The users are instructed to keep the excel file of stocks data (named 'STOCKS_FILE.xlsx') in the same folder location where the R script file is placed. Please also note that if there is a window defender pop up due to parallel command it could be canceled but if the code malfunctions then please allow the same.

The 'spec.comp' list keeps the specification for all 80 model specification. The 80 model specifications are generated using 5 base models with 8 distributions and 2 mean model types. The in-mean and not-in-mean model specification is generated by setting the *archm* parameter of the mean.model() function as 'True' or 'False' respectively. The function *apply_MCS_procedure* takes as input the stocks data, names of model specifications, and number of cores to parallelise the process, as inputs. The 'roll.comp' list stores the rolling forecast for each stock generated using *ugarchroll* function. The *VaR.comp* stores the value at risk forecasts from the rolling forecast for each specification as a data frame. Finally the 'Loss' matrix stores the VaR loss of the forecasted data which is provided as input to the *MCSprocedure* of MCS package. I use the 'Tmax' statistics and the alpha value of 0.1 to compare the models as explained in Bernardi & Catania (2018). The output of the *apply_MCS_procedure* function is an *S4* object in R which gives the description about SSM, where retained models are ranked as per the fit. *As I reduce the number of Bootstrap samples from 5000 to 3000 to speed up the process the SSM for some stocks differ on different machine due to randomness of bootstrap. It can be seen for the stocks of Amazon and American Express the result from R markdown file are slightly different what is reported here. I tried, the code on two different machines with B=1200 and B=3000.*

For B=1200 I found that the number of models in SSM differed more on different machines as compared to B=3000, thus in this report I use B=3000 for the R version mentioned in ReadMe file.

The helper function *get_model_counts* is used to generate the counts of models finally retained in SSM by taking the names of retained model specification and the names of all model specifications as inputs. Similarly, the helper function *get_distribution_counts* is used to generate the counts of distribution finally retained in SSM by taking the names of retained model specification and the names of all distributions as inputs. Finally the in-mean and not-in-mean counts are generated using the *get_inmean_counts* function which takes the names of retained model specification as input. Finally, the result tables are generated, where 'dataframe1' gives the result Table 1 where frequency of names of each model retained in SSM for each of 5 stocks are shown. Similarly 'dataframe2' gives the result Table 2 where frequency of names of each distribution retained in SSM for each of 5 stocks are shown. Further, the 'dataframe3' gives the result Table 3 which provide the frequency of in-mean and not-in-mean models retained by SSM. Lastly, the ranks of all retained models for each stocks is given in Table 4 which is generated from 'rslt_tab_comb' in the R code.

3 Analysis of the python code

In order to reproduce the results in python the project also test the python implementation of Model Confidence Set developed by Michael Gong. The text file provided runs well, however, there is just one issue in the final simulation. In the original file the authors mistakenly put the value of variable B (the number of bootstrapped sample) as 3 instead of 1000 which is actually the value of variable w, (block size for bootstrapping samples). The simulation data is generated using the *DataGeneratingProcess()* in python wherelse the Model Confidence Set is implemented by class *ModelConfidenceSet*. In order to check how well the python code matches the package in R the VaR loss ('Loss' matrix) calculated for each stock in R is fed as input to the *ModelConfidenceSet()* class of python. I observe that in R the *MCSprocedure* function returns a larger set of superior models whereas in python I observe high penalization and thus the final selected models are reduced. The SSM selected by python are also present in the SSM selected by R for the same alpha value (compare models in Tables 4 and Table 5 for each stock). However R have a larger set of models as opposed to that obtained in python. On inspecting the reason for that it is observed that the way in which t-statistics 'SQ' and 'R' are calculated in the functions *calculate_PvalR* or *calculate_PvalSQ* are quite large as to what is computed in R and thus small number of models are only selected. The t-statistics 'SQ' and 'R' corresponds to the first and the second t-statistics of Hansen *et al.* (2011). I present the results from python code in Table 5 where the model names selected by *ModelConfidenceSet()* class of python are presented for all stocks and are generated by 'MCS_final_models' dataframe in python. I find that all models presented by python code are present in the SSM predicted by R code thus affirming a good consistency however the python code gives a smaller SSM for each model as compared to R. Also the number of Bootstrapped samples affects the outcomes from R package with higher sensitivity than python, as python outputs do not change a lot by increasing the Bootstrap value from 1200-3000. The python code runs very fast as compared to R code. The '*' marked models in Table 4 are the ones that match with the output from python code.

4 Results

I demonstrate the results in the following tables. In Table 1 we find that models like csGARCH and the family of GARCH submodel 'NGARCH' does a pretty well job in almost all the stocks. Similarly, from Table 2 I can see that 'snorm' distribution does a good job across all the stocks. Further from Table 3 I can see that Not In Mean configuration performs better as compared to In Mean configuration for all the stocks. Comparing Table 4 and Table 5 we can observe that the SSM predicted by python code are also the ones predicted by R, however R predicts higher number of models as compared to that produced in python implying higher penalization in python while calculating the t-statistics.

Table 1: Frequency of selected model by MCS package in R for each stocks

Model Names/Stocks	**MSFT**	**GOOG**	**AMZN**	**AXP**	**JPM**
sGARCH	2	0	0	7	3
eGARCH	0	0	0	0	0
apARCH	3	2	4	0	0
csGARCH	2	2	0	12	2
NGARCH	4	2	7	3	2

Table 2: Frequency of selected distributions for models by MCS package in R for each stocks

Distributions/Stocks	**MSFT**	**GOOG**	**AMZN**	**AXP**	**JPM**
norm	0	0	4	5	2
std	0	0	0	1	0
ged	0	0	0	0	0
snorm	8	6	4	6	5
sstd	1	0	1	4	0
sged	0	0	0	1	0
jsu	2	0	2	3	0
ghyp	0	0	0	2	0

Table 3: Frequency of selected In-Mean specification for SSM in R MCS package for each stocks

In Mean Config/Stocks	**MSFT**	**GOOG**	**AMZN**	**AXP**	**JPM**
In mean	4	3	6	10	2
Not in Mean	7	3	5	12	5

Table 4: Models selected by MCS package in R for each stocks. The '*' marked models are the ones that corresponds to the output from python code.

SelectedModel	Rank_M	v_M	MCS_M	Rank_R	v_R	MCS_R	Loss	stock
sGARCH-snorm-inmean-TRUE	7	0.177	0.987	11	13.812	0.000	2.32E-06	**MSFT**
sGARCH-snorm-inmean-FALSE	6	-0.505	1.000	10	8.983	0.000	2.30E-06	**MSFT**
apARCH-snorm-inmean-TRUE	9	0.673	0.840	2	1.463	0.600	2.36E-06	**MSFT**
apARCH-snorm-inmean-FALSE	10	1.095	0.626	4	2.039	0.165	2.35E-06	**MSFT**
apARCH-jsu-inmean-FALSE	11	1.947	0.109	6	2.681	0.007	2.40E-06	**MSFT**
csGARCH-snorm-inmean-TRUE	5	-0.522	1.000	9	7.371	0.000	2.30E-06	**MSFT**
csGARCH-snorm-inmean-FALSE	4	-1.123	1.000	8	5.904	0.000	2.28E-06	**MSFT**
NGARCH-snorm-inmean-TRUE	1	-4.819	1.000	3	1.901	0.266	2.27E-06	**MSFT**
***NGARCH-snorm-inmean-FALSE**	2	-3.195	1.000	1	-1.463	1.000	2.23E-06	**MSFT**
NGARCH-sstd-inmean-FALSE	8	0.641	0.851	7	2.977	0.000	2.32E-06	**MSFT**
NGARCH-jsu-inmean-FALSE	3	-2.783	1.000	5	2.278	0.105	2.30E-06	**MSFT**
***apARCH-snorm-inmean-TRUE**	2	-0.202	1.000	2	0.354	1.000	2.16E-06	**GOOG**
***apARCH-snorm-inmean-FALSE**	3	-0.154	1.000	4	0.804	0.829	2.16E-06	**GOOG**
***csGARCH-snorm-inmean-TRUE**	5	0.163	1.000	5	3.887	0.000	2.17E-06	**GOOG**
***csGARCH-snorm-inmean-FALSE**	1	-1.742	1.000	1	-0.354	1.000	2.15E-06	**GOOG**
NGARCH-snorm-inmean-TRUE	6	0.788	0.782	6	6.344	0.000	2.18E-06	**GOOG**
***NGARCH-snorm-inmean-FALSE**	4	0.145	1.000	3	0.643	0.940	2.17E-06	**GOOG**
apARCH-norm-inmean-TRUE	10	1.750	0.217	6	2.168	0.138	2.46E-06	**AMZN**
apARCH-norm-inmean-FALSE	9	1.639	0.260	4	2.099	0.182	2.45E-06	**AMZN**
apARCH-snorm-inmean-TRUE	11	2.038	0.112	8	2.783	0.000	2.47E-06	**AMZN**
apARCH-snorm-inmean-FALSE	8	1.584	0.270	5	2.153	0.158	2.44E-06	**AMZN**
NGARCH-norm-inmean-TRUE	4	-0.715	1.000	10	3.090	0.000	2.38E-06	**AMZN**
***NGARCH-norm-inmean-FALSE**	3	-1.622	1.000	9	3.037	0.000	2.36E-06	**AMZN**
NGARCH-snorm-inmean-TRUE	5	-0.633	1.000	11	3.902	0.000	2.39E-06	**AMZN**
***NGARCH-snorm-inmean-FALSE**	1	-2.590	1.000	2	0.362	1.000	2.35E-06	**AMZN**
NGARCH-sstd-inmean-TRUE	7	-0.114	1.000	7	2.529	0.017	2.40E-06	**AMZN**
***NGARCH-jsu-inmean-TRUE**	2	-1.680	1.000	1	-0.362	1.000	2.33E-06	**AMZN**
***NGARCH-jsu-inmean-FALSE**	6	-0.558	1.000	3	0.872	0.922	2.38E-06	**AMZN**
sGARCH-norm-inmean-TRUE	18	1.046	0.692	20	7.738	0.000	2.03E-06	**AXP**
sGARCH-norm-inmean-FALSE	16	0.570	0.920	21	9.398	0.000	2.02E-06	**AXP**
sGARCH-snorm-inmean-TRUE	14	0.198	1.000	11	3.434	0.000	2.01E-06	**AXP**
sGARCH-snorm-inmean-FALSE	11	-0.208	1.000	6	1.250	0.787	2.00E-06	**AXP**
sGARCH-sstd-inmean-TRUE	19	1.505	0.361	7	2.515	0.007	2.03E-06	**AXP**
sGARCH-sstd-inmean-FALSE	20	1.726	0.193	19	7.128	0.000	2.04E-06	**AXP**
sGARCH-jsu-inmean-TRUE	22	1.798	0.165	10	3.228	0.000	2.04E-06	**AXP**
csGARCH-norm-inmean-TRUE	17	1.001	0.725	14	5.306	0.000	2.02E-06	**AXP**
csGARCH-norm-inmean-FALSE	3	-1.411	1.000	8	2.517	0.007	1.98E-06	**AXP**
csGARCH-std-inmean-FALSE	21	1.786	0.165	22	29.368	0.000	2.05E-06	**AXP**
csGARCH-snorm-inmean-TRUE	7	-0.726	1.000	18	6.508	0.000	1.99E-06	**AXP**
***csGARCH-snorm-inmean-FALSE**	1	-2.285	1.000	1	-0.309	1.000	1.96E-06	**AXP**
***csGARCH-sstd-inmean-TRUE**	4	-1.373	1.000	2	0.309	1.000	1.97E-06	**AXP**
***csGARCH-sstd-inmean-FALSE**	2	-1.491	1.000	3	0.319	1.000	1.97E-06	**AXP**
csGARCH-sged-inmean-FALSE	12	-0.086	1.000	15	6.201	0.000	2.00E-06	**AXP**
csGARCH-jsu-inmean-TRUE	13	0.173	1.000	12	3.798	0.000	2.00E-06	**AXP**
csGARCH-jsu-inmean-FALSE	6	-0.754	1.000	16	6.222	0.000	1.99E-06	**AXP**
***csGARCH-ghyp-inmean-TRUE**	5	-0.758	1.000	9	3.096	0.000	1.98E-06	**AXP**
csGARCH-ghyp-inmean-FALSE	10	-0.273	1.000	13	4.082	0.000	2.00E-06	**AXP**
NGARCH-norm-inmean-FALSE	15	0.299	0.976	17	6.378	0.000	2.01E-06	**AXP**
NGARCH-snorm-inmean-TRUE	8	-0.588	1.000	4	0.979	1.000	1.99E-06	**AXP**
NGARCH-snorm-inmean-FALSE	9	-0.473	1.000	5	1.005	1.000	1.99E-06	**AXP**
sGARCH-norm-inmean-FALSE	4	-0.140	1.000	7	20.235	0.000	1.98E-06	**JMP**
sGARCH-snorm-inmean-TRUE	7	1.584	0.304	6	6.442	0.000	1.98E-06	**JMP**
***sGARCH-snorm-inmean-FALSE**	1	-2.949	1.000	2	0.159	1.000	1.97E-06	**JMP**
***csGARCH-snorm-inmean-TRUE**	3	-0.304	1.000	3	1.488	1.000	1.98E-06	**JMP**
***csGARCH-snorm-inmean-FALSE**	2	-2.446	1.000	1	-0.159	1.000	1.97E-06	**JMP**
NGARCH-norm-inmean-FALSE	6	1.495	0.339	5	2.255	0.031	1.99E-06	**JMP**
NGARCH-snorm-inmean-FALSE	5	1.326	0.445	4	2.225	0.031	1.99E-06	**JMP**

Table 5: Models selected by *ModelConfidenceSet()* class of python by Michael Gong. All selected models also present in SSM of R

MSFT	GOOG	AMZN	AXP	JPM
NGARCH.snorm.inmean.FALSE	csGARCH.snorm.inmean.TRUE	NGARCH.jsu.inmean.FALSE	csGARCH.ghyp.inmean.TRUE	csGARCH.snorm.inmean.TRUE
	apARCH.snorm.inmean.FALSE	NGARCH.norm.inmean.FALSE	csGARCH.sstd.inmean.FALSE	sGARCH.snorm.inmean.FALSE
	NGARCH.snorm.inmean.FALSE	NGARCH.snorm.inmean.FALSE	csGARCH.sstd.inmean.TRUE	csGARCH.snorm.inmean.FALSE
	apARCH.snorm.inmean.TRUE	NGARCH.jsu.inmean.TRUE	csGARCH.snorm.inmean.FALSE	
	csGARCH.snorm.inmean.FALSE			

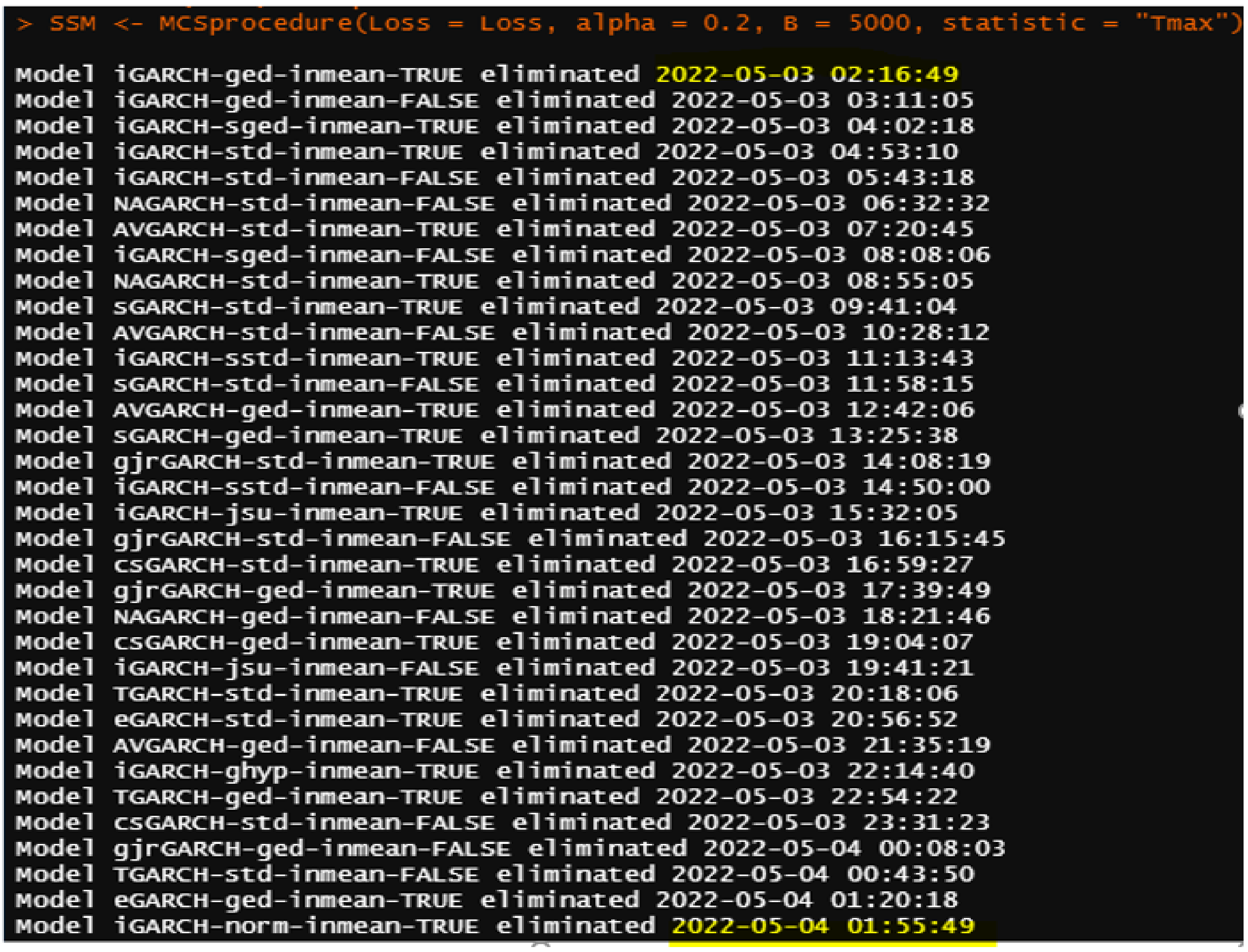
References

Bernardi, Mauro, & Catania, Leopoldo. 2018. The model confidence set package for R. *International Journal of Computational Economics and Econometrics*, **8**(2), 144–158.

Ghalanos, Alexios. 2022. *rugarch: Univariate GARCH models.* R package version 1.4-8.

Hansen, Peter R, Lunde, Asger, & Nason, James M. 2011. The model confidence set. *Econometrica*, **79**(2), 453–497.

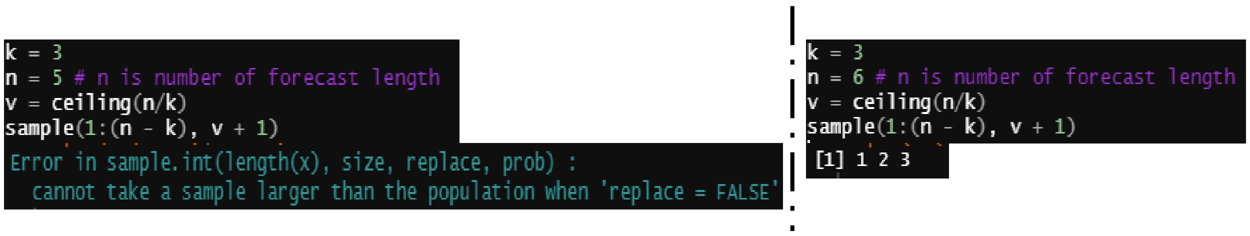
5 Appendix



```
> SSM <- MCSprocedure(Loss = Loss, alpha = 0.2, B = 5000, statistic = "Tmax")

Model iGARCH-ged-inmean-TRUE eliminated 2022-05-03 02:16:49
Model iGARCH-ged-inmean-FALSE eliminated 2022-05-03 03:11:05
Model iGARCH-sged-inmean-TRUE eliminated 2022-05-03 04:02:18
Model iGARCH-std-inmean-TRUE eliminated 2022-05-03 04:53:10
Model iGARCH-std-inmean-FALSE eliminated 2022-05-03 05:43:18
Model NAGARCH-std-inmean-FALSE eliminated 2022-05-03 06:32:32
Model AVGARCH-std-inmean-TRUE eliminated 2022-05-03 07:20:45
Model iGARCH-sged-inmean-FALSE eliminated 2022-05-03 08:08:06
Model NAGARCH-std-inmean-TRUE eliminated 2022-05-03 08:55:05
Model sGARCH-std-inmean-TRUE eliminated 2022-05-03 09:41:04
Model AVGARCH-std-inmean-FALSE eliminated 2022-05-03 10:28:12
Model iGARCH-sstd-inmean-TRUE eliminated 2022-05-03 11:13:43
Model sGARCH-std-inmean-FALSE eliminated 2022-05-03 11:58:15
Model AVGARCH-ged-inmean-TRUE eliminated 2022-05-03 12:42:06
Model sGARCH-ged-inmean-TRUE eliminated 2022-05-03 13:25:38
Model gjrGARCH-std-inmean-TRUE eliminated 2022-05-03 14:08:19
Model iGARCH-sstd-inmean-FALSE eliminated 2022-05-03 14:50:00
Model iGARCH-jsu-inmean-TRUE eliminated 2022-05-03 15:32:05
Model gjrGARCH-std-inmean-FALSE eliminated 2022-05-03 16:15:45
Model csGARCH-std-inmean-TRUE eliminated 2022-05-03 16:59:27
Model gjrGARCH-ged-inmean-TRUE eliminated 2022-05-03 17:39:49
Model NAGARCH-ged-inmean-FALSE eliminated 2022-05-03 18:21:46
Model csGARCH-ged-inmean-TRUE eliminated 2022-05-03 19:04:07
Model iGARCH-jsu-inmean-FALSE eliminated 2022-05-03 19:41:21
Model TGARCH-std-inmean-TRUE eliminated 2022-05-03 20:18:06
Model eGARCH-std-inmean-TRUE eliminated 2022-05-03 20:56:52
Model AVGARCH-ged-inmean-FALSE eliminated 2022-05-03 21:35:19
Model iGARCH-ghyp-inmean-TRUE eliminated 2022-05-03 22:14:40
Model TGARCH-ged-inmean-TRUE eliminated 2022-05-03 22:54:22
Model csGARCH-std-inmean-FALSE eliminated 2022-05-03 23:31:23
Model gjrGARCH-ged-inmean-FALSE eliminated 2022-05-04 00:08:03
Model TGARCH-std-inmean-FALSE eliminated 2022-05-04 00:43:50
Model eGARCH-ged-inmean-TRUE eliminated 2022-05-04 01:20:18
Model iGARCH-norm-inmean-TRUE eliminated 2022-05-04 01:55:49
```

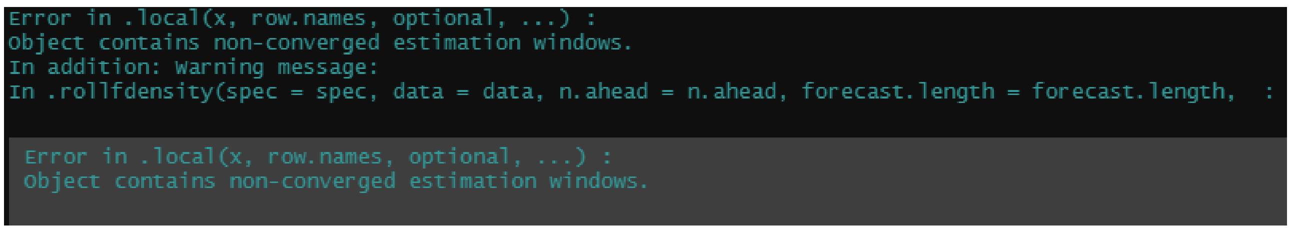
Figure 1: Terminating the experiment with 160 model specifications to identify most frequently eliminated models



```
k = 3
n = 5 # n is number of forecast length
v = ceiling(n/k)
sample(1:(n - k), v + 1)
 Error in sample.int(length(x), size, replace, prob) :
   cannot take a sample larger than the population when 'replace = FALSE'
```

```
k = 3
n = 6 # n is number of forecast length
v = ceiling(n/k)
sample(1:(n - k), v + 1)
 [1] 1 2 3
```

Figure 2: Error in keeping forecast length lower than 6. Tried changing k but minimum permissible length was found to be 6



```
Error in .local(x, row.names, optional, ...) :
object contains non-converged estimation windows.
In addition: Warning message:
In .rollfdensity(spec = spec, data = data, n.ahead = n.ahead, forecast.length = forecast.length,  :

 Error in .local(x, row.names, optional, ...) :
 object contains non-converged estimation windows.
```

Figure 3: Model convergence issues with low values of *refit.every* argument in ‘ugarchroll’ function