Aalto University

CS-E5710 - Bayesian Data Analysis

Predicting S&P 500 10-year real returns using Bayesian linear models

Contents

1 Introduction

Stock market data is considered highly stochastic, especially over shorter time periods. However, more extended periods, such as 10-year returns, exhibit some predictability. In the late 1980s, Robert Shiller (Campbell and Shiller, 1988) introduced the cyclically adjusted price-to-earnings ratio (CAPE-ratio). The CAPE ratio's formula

$$\text{CAPE}_t = \frac{P_t}{\frac{1}{120}\sum_{i=1}^{120} E_{t-i}} \tag{1}$$

includes the object's price, P_t, and trailing 10-year average returns, $\frac{1}{120}\sum_{i=1}^{120} E_{t-i}$. It was found that the CAPE ratio explains returns over more extended periods, especially 10-year-ahead returns, exceptionally well. Campbell and Shiller (1988) found that the CAPE ratio alone explains approximately 35.7 percent of the 10-year-ahead real returns of the S&P 500 index when using data from 1871 to 1987.

While Campbell and Shiller's (1988) observations were based on an in-sample study, more recent studies, such as Wang et al. (2021) and Davis et al. (2018), have conducted out-of-sample performance evaluations of the CAPE ratio. The studies show that the CAPE-ratio's predictive capabilities remain relatively high even in an out-of-sample study.

1.1 Problem Formulation

However, Wang et al. (2021) suggest that the CAPE-ratio's predictive capabilities have decreased since the 2000s. Davis et al. (2018) suggest that the 1990s dot-com bubble has made the historical and statistical relationship between CAPE and equity returns more unstable. Figure 1 illustrates how the sample mean of the CAPE ratio has nearly doubled since the 1990s.

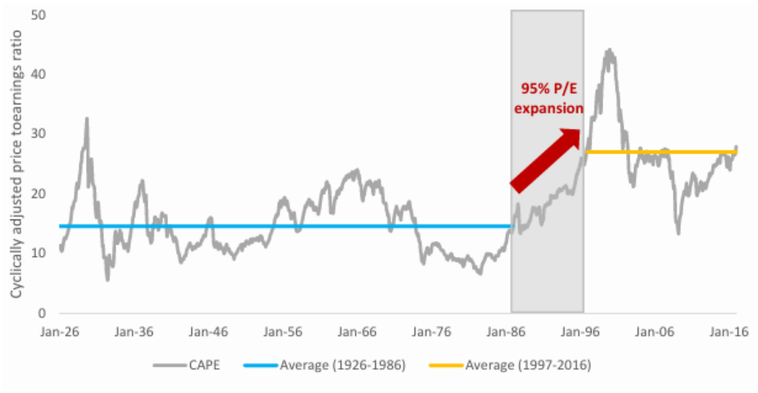


Figure 1: CAPE ratios expansion during the 2000s (Davis et al., 2018)

Moreover, Arnott et al (2017) show that CAPE, interest rates, and inflation levels are interlinked. Lower and more stable levels of inflation and interest rates seem to allow higher CAPE ratios (Appendix A.1). Additionally, the CAPE ratio is non-stationary, and its mean depends on the macroeconomic structure, which is time-dependent (Appendix A.2). The predictive power of the CAPE ratio relies on its mean-reversion feature. While relatively high CAPE values indicate lower 10-year-ahead returns and vice versa, the relationship depends on its varying mean, which it reverts back to.

These macroeconomic structural changes include changes in monetary policy, bank loan repayment periods, accounting regulations, and technology, which enable central banks to respond more quickly to emerging macroeconomic risks and, therefore, make the economy more stable. For example, explicit 2% inflation targets are a relatively recent innovation. The Federal Reserve formally adopted a 2% inflation goal in 2012 (Federal Reserve, 2012). Moreover, the ECB moved from a qualitative definition of below 2% in 1998 (ECB, 1998) to below, but close to 2% in 2003 (ECB, 2003), and only adopted an explicit symmetric 2% target in 2021 (ECB, 2021).

Lastly, these structural changes affect the relationship between CAPE and 10-year-ahead returns. The 2000s have allowed relatively higher CAPE ratios while keeping the 10-year-ahead annualized returns relatively high (Appendix A.3). This affects the statistical structure of the relationship between the CAPE ratio and 10-year-ahead real returns. Therefore, nowadays, predicting returns from past data using only CAPE to explain returns leads to underestimated returns.

1.2 Modelling

This study is conducted as an out-of-sample analysis with the target of predicting 10-year-ahead annualized real returns of the S&P 500 index. We use a rolling training and testing window to avoid look-ahead bias. Since the goal is to capture the structural change, explained previously, we use a testing window of 1/1990-9/2015 and, consequently, an initial training window of 12/1881-12/1979.

This study incorporates Bayesian data analytics into the CAPE-ratio-based linear models. We try to capture structural changes by implementing priors that are updated in three-year periods based on the trailing 30 years of data. These priors are intended to serve as weights for newer data, balancing the information from older data with information about current macroeconomic structures. Although we recognize that this procedure is not traditional in Bayesian data analytics, the aim is to simulate the prior knowledge of the relationship between the CAPE ratio and 10-year-ahead returns at a given time.

We also use Bayesian hierarchical models that group the data by 30-year periods. Since the literature suggests that multiple factors affect time-dependent macroeconomic structures, excessive numbers of variables in the regression model can create more noise than information. A hierarchical model separates these time-variant structures and allows partial pooling to regularize group-level variance. This should allow the model to capture the structural differences across periods while controlling their variance through partial pooling.

Lastly, for the pooled model, we perform a sensitivity analysis by varying the model priors based on the window size of the trailing data used to estimate them. By changing the window size, we can analyze the effect of including more recent structures. For the hierarchical model, we set the window size to a constant and vary the group-level standard deviation priors.

2 Data and Methodology

2.1 Data

This study uses a dataset from Robert Shiller's website (2025). The dataset includes all the relevant data, including 10-year-ahead annualized real returns of the S&P 500 and the CAPE ratio. The dataset is a time series, where each datapoint represents the month t, CAPE level, and 10-year-ahead annualized real returns from 1/1871 to 9/2025. The formula for 10-year-ahead annualized real returns is

$$R_{t+120} = \left(\frac{\mathrm{TR}_{t+120}}{\mathrm{TR}_t} \right)^{1/10} - 1, \tag{2}$$

where T_t is the real total return price, which includes inflation-adjusted returns of the S&P 500 and its dividend returns. Its mean is approximately 6.1 and its standard deviation is 5.2 percentage points.

The Shiller's dataset's CAPE ratio differs slightly from the previously mentioned CAPE ratio's formula (1). The formula for the CAPE ratio in Shiller's dataset includes S&P 500 index's inflation-adjusted price P_t^{real}, and the inflation-adjusted earnings E_{t-i}^{real}. Its mean is approximately 16.6, and its standard deviation is 6.6.

Additionally, we computed 30-year periods for the hierarchical model: 1/1870-12/1899, 1/1900-12/1929, 1/1930-12/1959, 1/1960-12/1989, and 1/1990-12/2019. Each period contains 360 data points except the first and last ones. The first period contains 217 data points, while the last period contains 309.

Figure 2 shows linear relationships between 10-year real returns and CAPE for every 30 years. The subplot shows that the different periods exhibit negative slopes with varying intercepts, with returns ranging approximately between 10 and 20 percent. The slopes seem to change only a little between periods, which we will take into consideration when we justify the priors used in Chapter 3.

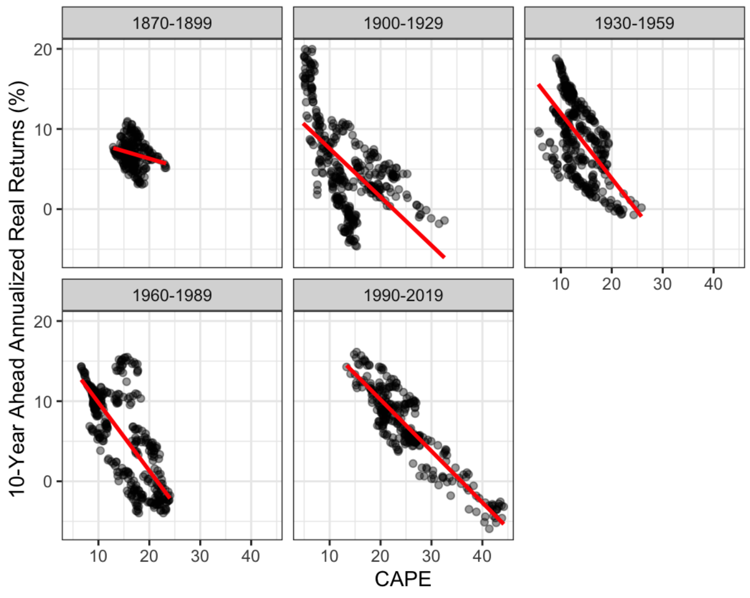


Figure 2: Linear relationships of the CAPE ratio and 10-year returns in different periods

2.2 Model Training and Testing

The models' predictive performance is evaluated out of sample, and they are compared using out-of-sample R^2, root mean squared error (rmse), and expected log pointwise predictive density (elpd) metrics. Since the data is a time series, we need to account for time when evaluating the models' out-of-sample performance. Therefore, we use time series cross-validation to split the data.

Figure 3 illustrates the first three training and testing folds. We incrementally add 6 months of data to the training set after testing the model on six months of testing data. Since the target is 10-year-ahead annualized real returns, we only have training data, for example, up to 12/1979, when we test the model from 1/1990 to 6/1990. Therefore, we must include a 10-year gap between the training and testing sets to avoid information leakage from the training data into the testing data. While the pooled model's training and testing procedure's source code is found from Appendix B.7, the partially pooled model's is found from Appendix B.8. Additionally, these algorithms use utility functions to compute the convergence diagnostics, posterior predictions and the expected log pointwise predictive densities (Appendix B.5).

We perform posterior predictive checks on the final models trained on the whole training sample to assess how well each model reproduces the observed data. Additionally, we examine convergence diagnostics for these final fits to ensure that the MCMC inference is reliable.

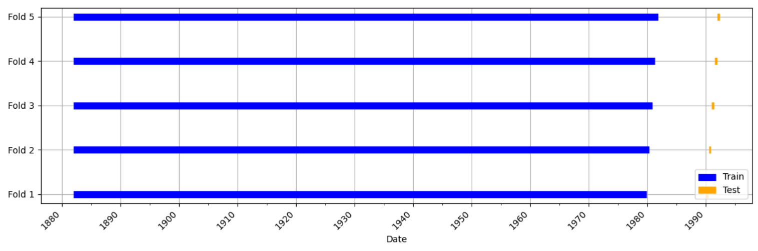


Figure 3: The first five folds for the training and testing procedure

Lastly, the same training and testing procedure is used in the studies by Davis et al. (2018) and Wang et al. (2021). Davis et al. (2018) evaluated the CAPE-based linear model's out-of-sample performance in predicting 10-year-ahead annualized real returns and compared it with their two-step VAR approach. Wang et al. (2021) evaluated the out-of-sample performance of their machine learning models and compared it with Davis et al.'s (2018) two-step VAR approach. Neither study implemented Bayesian data analytics in its model. This study uses Bayesian models to capture the structural changes by simulating prior knowledge across the training and testing procedures.

3 Models

3.1 Pooled Model

For observation i, denote the target variable 10-year returns as y_i and the explanatory variable CAPE as x_i. We assume that the returns are normally distributed, and specify a Gaussian linear model

$$\begin{aligned} y_i \mid \alpha, \beta, \sigma &\sim N(\mu_i, \sigma^2) \\ \mu_i &= \alpha + \beta x_i \\ \alpha &\sim N(\mu_\alpha^k, \sigma_\alpha^k) \\ \beta &\sim N(\mu_\beta^k, \sigma_\beta^k) \\ \sigma &\sim t_3(0, 5.2) \end{aligned} \tag{3}$$

The priors for intercept α and slope β are updated every three years based on trailing 30 years of data. The update is done using the regression coefficients of a linear regression model $y_{i \in k} = a + bx_{i \in k} + \varepsilon$, where k denotes the data from last 30 years. Parameters $\mu_{\alpha}^{k}, \sigma_{\alpha}^{k}$ and $\mu_{\beta}^{k}, \sigma_{\beta}^{k}$ are obtained as estimates and standard errors for a and b respectively. Since these priors are based entirely on observed past data, they can be classified as informative. The prior for standard deviation σ is Student's t-distribution with 3 degrees of freedom, location 0 and scale 5.2. This is the default brms prior for standard deviation. As a fairly heavy-tailed distribution, it does not restrict the parameter too much making it a weakly informative prior. The prior $t_3(0, 5.2)$ on σ has a standard deviation of roughly 9 percentage points. It places most mass on residual standard deviations of the same order as the historical variability of 10-year returns, while still allowing much larger values due to its heavy tails.

This data-driven prior specification procedure is meant to imitate an investor who regularly updates their views as new information arrives. While older observations still influence the posterior through the likelihood, the prior gives more weight to the most recent macroeconomic regime rather than the full historical sample. The three-year update frequency for the pooled priors is a compromise between allowing the priors to adapt to structural change and keeping the computation manageable. It would be difficult to set unbiased priors in an out-of-sample analysis, since we know the modern macroeconomic structure.

We perform sensitivity analysis for the pooled model by changing the prior specification window size. We repeat the pooled model with window sizes of 10 and 50 years. Shorter windows give more weight to recent regimes; however, they are less stable, since the priors are formed from smaller amount of data. Longer windows give more weight to longer-run averages, which makes the priors more stable, yet, make them slower to adapt to new regimes.

In the pooled model, all parameters are population-level parameters. This means that they do not vary between any possible subsets of the data. This is simple and computationally efficient, but may fail to account for shifts in economic dynamics that influence CAPE's effect on returns. Even if the relationship between CAPE and future real returns differs across the prediction window, a pooled model forces the same intercept and slope on all time periods.

3.2 Partially Pooled Model

To model the possible shift in CAPE's effect on long-term returns, we include time period as a categorical variable in the model and allow both the intercept and slope to vary between different periods. Using similar notation as with the pooled model and denoting the time period of observation i as $g(i)$, this model can be written as

$$\begin{aligned}
&y_i \mid \mu_i, \sigma \sim N(\mu_i, \sigma^2) \\
&\mu_i = \alpha_{g(i)} + \beta_{g(i)} x_i \\
&\begin{pmatrix} \alpha_{g(i)} \\ \beta_{g(i)} \end{pmatrix} \sim N(\begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \Sigma) \\
&\alpha \sim N(\mu_\alpha^k, \sigma_\alpha^k) \\
&\beta \sim N(\mu_\beta^k, \sigma_\beta^k) \\
&\Sigma = \begin{pmatrix} \tau_\alpha & 0 \\ 0 & \tau_\beta \end{pmatrix} R \begin{pmatrix} \tau_\alpha & 0 \\ 0 & \tau_\beta \end{pmatrix} \\
&\tau_\alpha \sim Exp(0.25) \\
&\tau_\beta \sim Exp(3) \\
&R \sim LKJ(2) \\
&\sigma \sim t_3(0, 5.2)
\end{aligned} \tag{4}$$

Here, the priors for global parameters α and β are obtained using the same trailing linear regression method as in pooled model. They represent the global means of time-period specific intercept $\alpha_{g(i)}$ and slope $\beta_{g(i)}$, which are drawn from multivariate normal distribution to model their possible correlation with each other. The correlation structure is governed by covariance matrix Σ, which can be decomposed into standard deviations τ_α and τ_β for intercept and slope, respectively, across different time periods, and correlation matrix R containing the actual correlation structure between slope and intercept of one time-period. The priors for standard deviations are exponential distributions with rate one, and the prior for correlation matrix is $LKJ(2)$. These priors, along with the same Student's t prior for σ as in pooled model, do not change in time.

We set the group-level slope standard deviation prior to $Exp(3)$ as a baseline for further sensitivity analysis. $Exp(3)$ allows the group-level slope to differ from the pooled slope approximately 0.33, which serves as a good regularized baseline. Moreover, we set the group-level intercept standard deviation prior to $Exp(0.25)$ for average regularization according to the plot in Appendix A.4. $Exp(0.25)$ allows the group-level intercept deviation to be 4 on average. For sensitivity analysis, we try lighter pooling and stronger pooling by changing the group-level standard deviation priors. Lastly, we keep the prior for the correlation matrix constant over this study, since $LKJ(2)$ is a weakly informative prior that mildly shrinks extreme correlations. Additionally, most of the pooling behaviour is driven by the group-level standard deviations, which we vary directly in the sensitivity analysis.

3.3 Brms-code

Brms-code for pooled and partially pooled model can be seen from figure 4. In addition to regression formula, fitting the model requires also the priors, but as the code for determining them is quite lengthy, it is not practical to include it here. Full code including the priors can be found in Appendix B.6.

```
formula <- bf(
    Real_Return_10Y ~ 1 + CAPE,
    family = "gaussian",
    center = FALSE
)
formula <- bf(
    Real_Return_10Y ~ 1 + CAPE + (1 + CAPE | Period) ,
    family = "gaussian",
    center = FALSE
)
```

Figure 4: Brms-code for pooled (above) and partially pooled model (below).

```
model <- brm(
    formula = formula,
    prior   = current_priors,
    data    = train,
    chains  = 3,
    iter    = 2000,
    warmup  = 1000,
    backend = "cmdstanr",
    cores   = cores,
    adapt_delta   = 0.99,
    max_treedepth = 15,
    seed    = 1
)
```

Figure 5: MCMC parameters used to fit the models.

Under the hood, brms uses Stan to fit the models. Stan estimates the model parameters with no-u-turn sampler (NUTS), which is a variant of Hamiltonian monte carlo (HMC). HMC relies on conservation of total energy to explore the parameter space.

Figure 5 shows the parameters passed to the sampler. The models were fitted using 3 parallel Markov chains. Running multiple chains is important for evaluating convergence, as if all the chains converge to same distribution the results are likely reliable. Default option is to run 4 chains, but one chain was dropped to decrease extremely long computation times. Each chain consisted of 2000 total iterations first 1000 of which were discarded as warmup to prevent including samples where sampler has not yet converged to the correct distribution.

The numerical integration in NUTS-algorithm creates inaccuracy, which affects the conservation of energy along the trajectory. If the energy has strayed too far from true value, a proposal sample is rejected. Parameter `adapt_delta` sets the target acceptance rate for a proposal and a high target rate of 0.99 forces the algorithm to use small stepsize to reduce inaccuracies, which also reduces number of divergences. Downside of using higher `adapt_delta` is computational cost.

NUTS builds a binary tree of possible steps to find the longest possible trajectory without u-turn to obtain samples without autocorrelation. Parameter `max_treedepth` controls the maximum depth of this tree, and value 15 allows for deeper search compared to default of 10. This reduces instances where maximum treedepth is exceeded and improves effective sample size. Again, downside is increased computational cost.

4 Results and Model Validation

Since there are previous studies made in considering CAPE in relation to 10-year annualized returns, it stands to reason that there would be some comparison to these studies. They were, however, made mostly if not entirely based on frequentist statistics, and including more metrics than CAPE, with different model specifications. Based on this, and the main goal of this project to practice Bayesian model building and evaluation, all of the analysis was done within and between our two model types.

In all of the models, inference at any given point is made only six months into the future, based on historical data up until ten years before. Because of this, running longer test windows results in multiple model fits, each with their own statistics and diagnostics. Although this is taken into account when discussing results, the last model fit is generally treated as the most important.

In short, any and all results discussed pertain to this six month predictive window specifically, and not to the models performance at predicting the whole test set at once.

4.1 Convergence diagnostics

For the basic evaluation of model convergence, we used the common ESS and $\hat{R}$ for both model types to evaluate the behavior of the MCMCs. For sampling, we used 3 chains of 2000 iterations including a warmup of 1000 iterations. Different instances of the convergence statistics come from fairly similar model fits, so we treat them as comparable when evaluating convergence.

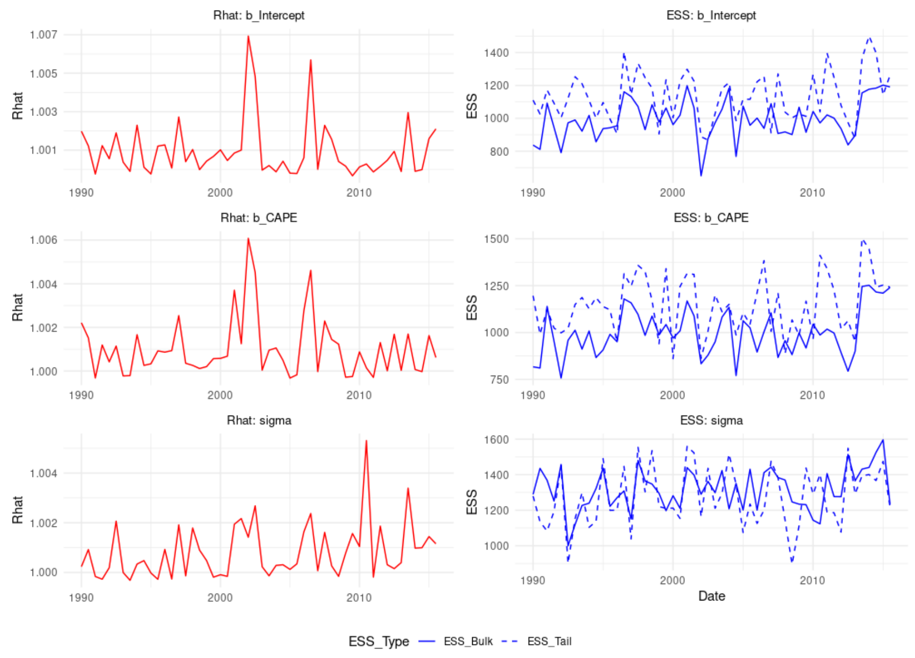


Figure 6: Rhat, ESS bulk and ESS tail for each parameter of the pooled model across entire testing window.

As we can see from the graphs in Figure 6, in the pooled models the MCMCs behave quite nicely. Values of $\hat{R}$ all stay under 1.01, which indicates excellent chain mixing, and consistent results. While effective sample size seems to vary somewhat, the mean of the ESS for both the bulk and the tail of the posterior remain at least 900 for all parameters. This is a very good result for our general sample size of 3000, and indicates that the results are reliable. We also found barely any divergences in the chains, less than five for each model. These results together indicate, that the modeling was most likely a success, and our results are good.

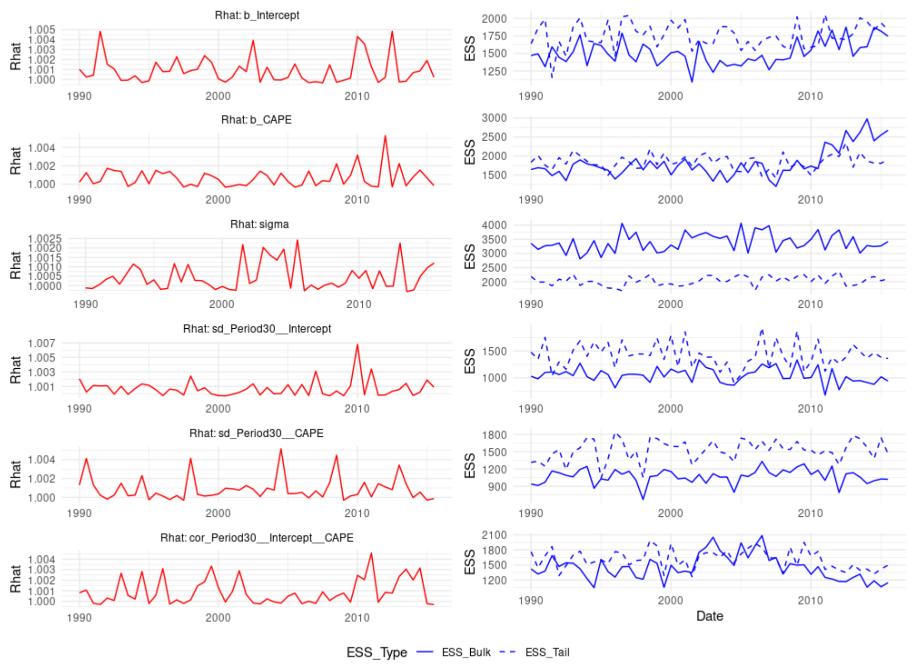


Figure 7: Rhat, ESS bulk and ESS tail for each population-level and group-level variance parameter of the partially pooled model across entire testing window.

The convergence in the hierarchical models is also good, as is shown in Figure 7. In the hierarchical model, the $\hat{R}$ of each of the parameters stay under 1.01 for all models, and the mean of ESS stays over 900. We also found very little divergences in the chains, less than six per model. These results indicate, that the hierarchical model results are also reliable.

Due to the requirements in processing power to run the full model, the proper convergence of the models was achieved by tweaking the MCMC hyperparameters, by trial and error, using a smaller subset of the data, and then applying it to the whole version.

4.2 Posterior Predictive Checks

The purpose of posterior predictive checking is to assess the model's capability to reproduce data similar to actual observations. If a model is well-specified, the original data should look plausible under the posterior predictive distribution.

Posterior predictive performance is assessed by creating replicated datasets from posterior predictive distribution and comparing them to observed data either visually or by using some test quantities to measure discrepancy between observed and replicated data. Here we conduct posterior predictive checks with visual inspection using the last model instance, as it contains most data and would be used to make future predictions.

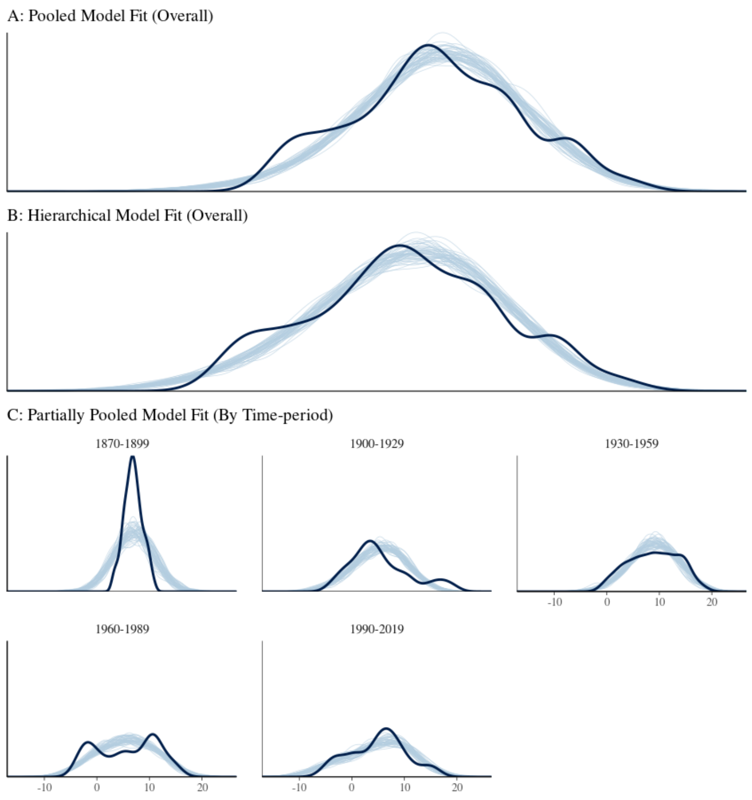


Figure 8: 50 posterior draws compared to observed data for pooled model (A), entire partially pooled model (B) and each time-period of the partially pooled model (C).

Figure 8 shows the distribution of observed returns overlaid with 50 draws from posterior predictive distribution for pooled and partially pooled models. We see that the draws from both models overall fit observed data decently but fail to capture some deviations and nuances. This especially evident from group-level posterior distributions of the partially pooled model. The regularization imposed by partial pooling structure pulls estimates towards global variance, which limits predictive performance especially in the first time period where variance was considerably lower. Another issue stems from predictive distribution for time period 1960-1989; model is not able to replicate the bimodal structure of actual returns. We experimented with different priors to mitigate these issues with little success.

It is likely that normal distribution is not ideal for capturing returns. In fact, stock returns generally exhibit fatter tails and more skewness as normal distribution would suggest. This means that in future studies it could be beneficial to use different observation family, for example Student's t for its fatter tails.

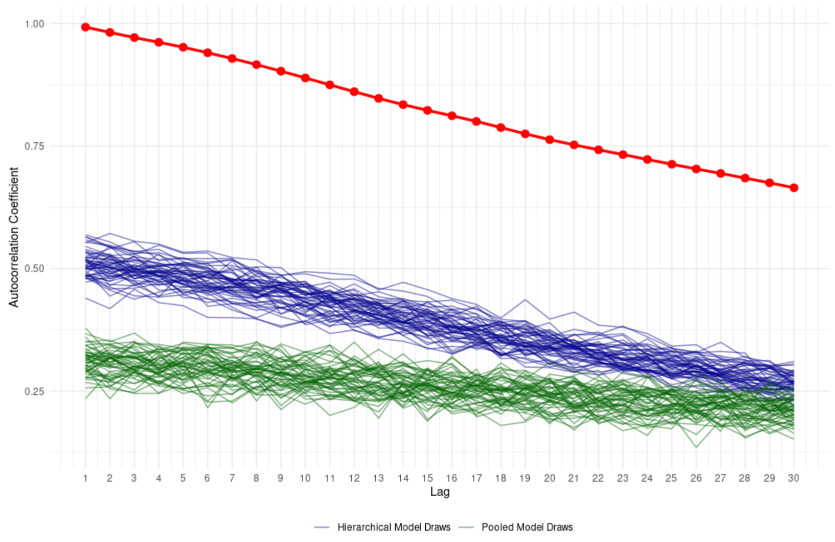


Figure 9: Autocorrelation of 50 posterior predictive draws from pooled and partially pooled model compared to autocorrelation of observed data.

Comparing overall distribution of observed data against posterior predictions is useful for checking general model assumptions like gaussian observations, but fails to account for temporal patterns in time series data. Even if the replicated density agrees perfectly with true values, the model can completely miss timing of the events.

We can test the models' ability to generate data with similar temporal structure as the observations by examining the autocorrelation function of posterior predictions. As can be seen from figure 9, the observed returns show significantly higher autocorrelation than replicated data from both models. This is to be expected, as the models only directly account for temporal dependence through 30-year grouping category in the case of partially pooled model, or not at all in the case of pooled model. That being said, including time period as categorical variable seems to considerably improve the correlation structure. While insufficient in explaining short-term shifts, the unique intercepts and slopes for each 30-year period allow the model to better account for correlations caused by large structural changes. This suggests that partial pooling using time period as category is justified and may create better predictions than only relying on CAPE to explain temporal changes.

We tried changing the period length to better account for short-term fluctuations, but 30 years was found to be a good middle ground between stability and adaptability. Future studies could further improve modeling by including autoregressive components to capture dependence on previous values.

Unimpressive posterior predictive check results indicate that the models are at least partially misspecified, which is unsurprising as stock returns are influenced by numerous factors making them practically stochastic. It is unlikely that we could substantially improve posterior predictive checks even with different observation family or grouping window, at least without increasing already large computational times.

4.3 Out-of-Sample Predictive Performance

While posterior predictive checks are valuable indicators of model quality, the practicality of the model lies in its predictive capabilities. As mentioned in Chapter 1.1 the predictive capabilities of the CAPE ratio have decreased since the 1990s. Figure 10 illustrates how the CAPE ratio-based pooled model underestimates the 10-year-ahead returns consistently, even when the prior gives more weight to more recent information. While the model cannot capture the structural change, its R^2 value is very high (0.8690). Moreover, its predictions are mostly very confident. This should be due to CAPE explaining 10-year-ahead returns well in historical data.

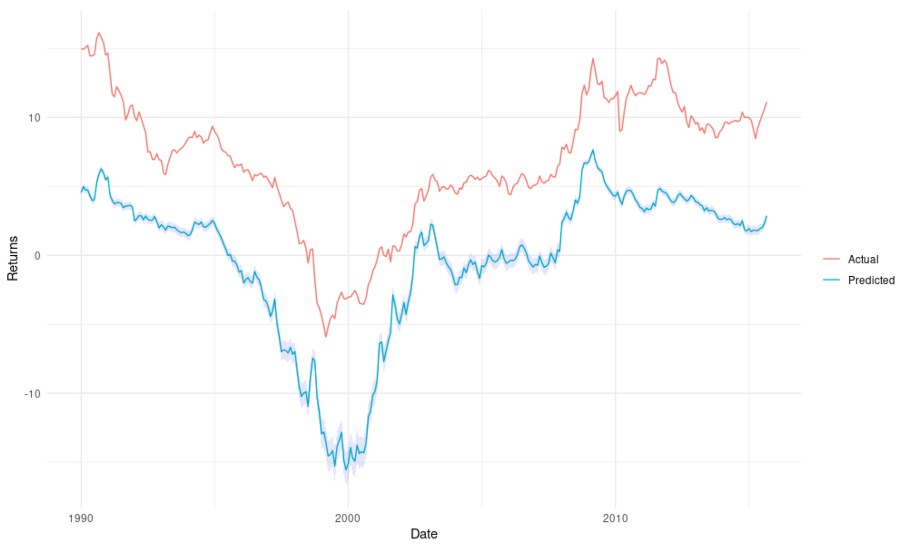


Figure 10: Predictions and 95% credible interval of the pooled model with a 30-year prior window compared to the actual value.

The partially pooled model (Figure 11) shows greater uncertainty in its predictions. This is due to its hierarchical structure, which introduces greater uncertainty for the period 1990-2000. The model has no training data during this period. Once the model is trained on the period 1990-2019, predictions become gradually more confident, as shown by shrinking confidence intervals from the early 2000s onward. Therefore, the model is a less confident pooled model up to 2000. When the partially pooled model includes training data from the 1990-2019 period group, the predictions jump from underestimating returns to overestimating them. The hierarchical model seems to capture the structural change that allows higher CAPE values combined with higher returns. Its R^2 value (0.461) decreases relative to the pooled model. Yet, its RMSE is approximately 1.929 lower than the pooled model's, indicating much closer predictions, while maintaining a high R^2 value even though the predictions jump in the early 2000s.

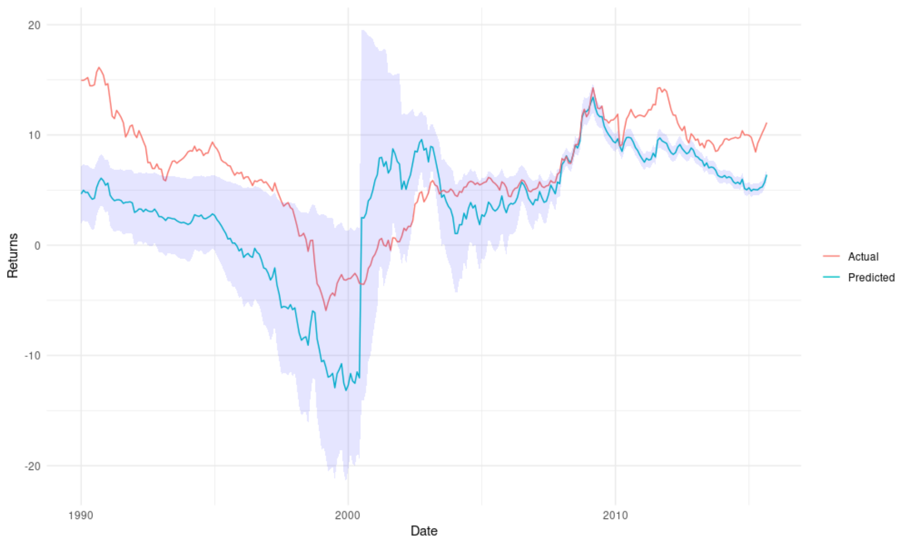


Figure 11: Predictions and 95% credible interval of the baseline partially pooled model compared to the actual value.

4.4 Sensitivity analysis and model comparison

When doing sensitivity analysis on the pooled model, we changed the time window that was used to make priors every three years. A shorter window size emphasizes more recent data points in the model, and a larger window is more generalized. Although a 10 year prior window decreased the RMSE by 0.492 and increased ELPD by 63, it also decreased R^2 significantly, by 0.197 (see table 1). This is likely a result of overfitting. With the 50 year prior, R^2 stayed basically the same, but RMSE increased by 0.825 and ELPD decreased by 104. The selection of prior clearly influences predictive accuracy and reliability, especially at lower prior window sizes.

In the hierarchical model, changes in priors have basically no effect in RMSE, R^2 or ELPD. This implies that the model is robust and the results reliable.

Even though the inference results of the hierarchical model are largely affected by the fact that the class-specific hyperparameters are not active or not accurate for the first half of the testing set (see figure 11), the RMSE and ELPD of the inference are much better than in the pooled model, by 1.929 and 247.2 respectively, while still having a significant R^2 of 0.461. This, coupled with the differences in sensitivity regarding prior selection, makes the hierarchical model clearly a better and more well rounded choice for inference.

Table 1: Model Comparison Results

Model / Prior	RMSE	R^2	ELPD
Pooled:			
10 yr prior	6.861	0.6723	-1131
30 yr prior	7.353	0.8690	-1194
50 yr prior	8.178	0.8709	-1298
Hierarchical:			
weakly pooled	5.415	0.4663	-948.6
medium pooled	5.424	0.4611	-946.8
strongly pooled	5.456	0.4396	-948.6

5 Issues and Potential Improvements

Posterior predictive diagnostics revealed some flaws in model construction, mainly the models' inability to capture autocorrelations present in actual returns, and possibly misspecified gaussian observation family. Potential fixes include introducing dependency on past values using lagged returns as explanatory variables (autoregression) or using a more heavy-tailed distribution to govern returns. However, the impact of these changes would likely prove minor as predicting stock markets is extremely difficult.

Another possible concern is related to our prior selection; in Bayesian inference, priors should reflect prior domain knowledge before obtaining the data, and regularize results when data is noisy or sparse. However, our priors are entirely determined by parameters' estimated value and variance from past observations, essentially giving data "double weight". This method is justified as long as we make sure model does not see future data at any point through the priors, but it is not necessarily purely Bayesian. One could argue that this is approach somewhere between frequentist and Bayesian inference. With stronger domain knowledge we could potentially eliminate the need for data-driven priors.

6 Conclusions

We set to evaluate how well the CAPE ratio can predict the 10-year-ahead annualized real returns of the S&P 500 in an out-of-sample study, and weather a hierarchical model could capture the macroeconomic structural changes better than a simpler pooled linear model. The pooled CAPE model achieved a high R^2 metric; however, its predictions consistently underestimated the 10-year-ahead returns. In contrast, the partially pooled model achieved clearly lower RMSE and higher ELPD, indicating closer and more reliable forecasts. However, it came with the cost of lower R^2 and wider confidence intervals, especially in periods with no training data of the specific predicted period. The sensitivity analysis shows that the pooled model reacts strongly to the prior window length, while the hierarchical model stayed robust to reasonable changes in its priors.

These out-of-sample results support the earlier findings that indicate that the CAPE's relationship has changed since the 1990s. CAPE-based models that force a single slope and intercept across the historical data, tend to underestimate the more modern returns. However, the hierarchical model with partial pooling allows period specific coefficients, which adapt better to changes in macroeconomic regimes, while allowing regularization from the partial pooling.

Moreover, posterior predictive checks revealed that our Gaussian observation model and lack of autoregressive terms limit the model's capability to reproduce the heavy tails and autocorrelation of the actual returns. Additionally, this study relied on CAPE alone as a predictor and used data-driven priors, which simplifies the complexities of the economy and makes our study closer to empirical Bayes approach than purely subjective prior specification.

Future work should extend our study and use more heavy-tailed prior distributions, include autoregressive components, and perhaps implement more macroeconomic features, such as inflation ratio and interest rates.

7 Self-reflection

This project proved to be surprisingly time-consuming, even though some of us were familiar with the problem setting and previous research on this topic before this course. The temporal aspect of our chosen dataset created some challenges, as none of the practice problems during the course dealt with similar data. Some of the methods discussed in lectures, such as leave-one-out cross-validation, were inapplicable and we had to find suitable alternatives. Additionally, rolling forecast structure meant extremely long computational times, which made iterative analysis and trial-and-error approach slow.

Despite the challenges, or because of them, project definitely improved our understanding on all parts of the Bayesian workflow. Applying all the methods ourselves to a real world problem required much deeper understanding than practice exercises, which we sometimes admittedly answered without putting much effort to actually understand the underlying concepts. Especially role of posterior predictive checks and sensitivity analysis became much clearer during the project.

8 AI Usage

We declare that we have used generative AI to assist in coding and conceptualizing. All results from AI have been verified by us before being included in the report.

9 References

Arnott, R.D., Chaves, D.B. and Chow, T., 2017. King of the mountain: The Shiller P/E and macroeconomic conditions. The Journal of Portfolio Management, 43(5), pp.22–39.

Campbell, J.Y. and Shiller, R.J., 1988. *Stock Prices, Earnings and Expected Dividends*. Cambridge, Mass: National Bureau of Economic Research.

Davis, J., Aliaga-Díaz, R., Ahluwalia, H. and Tolani, R., 2018. Improving U.S. stock return forecasts: A "fair-value" CAPE approach. *The Journal of Portfolio Management*, 44(3), pp.43–55.

European Central Bank 1998, *A stability-oriented monetary policy strategy for the ESCB*, press release, 13 October, European Central Bank, Frankfurt am Main, viewed 24 November 2025, Available at: https://www.ecb.europa.eu/ecb-and-you/educational/shared/img/MP_0806_300dpi-textsheet.en.pdf

European Central Bank 2003, 'The outcome of the ECB's evaluation of its monetary policy strategy', *ECB Monthly Bulletin*, June, European Central Bank, Frankfurt am Main, viewed 24 November 2025, Available at: https://www.ecb.europa.eu/press/pr/date/2003/html/pr030508_2.en.html

European Central Bank 2021, *ECB's Governing Council approves its new monetary policy strategy*, press release, 8 July, European Central Bank, Frankfurt am Main, viewed 24 November 2025, Available at: https://www.ecb.europa.eu/press/pr/date/2021/html/ecb.pr210708~dc78cc4b0d.en.html

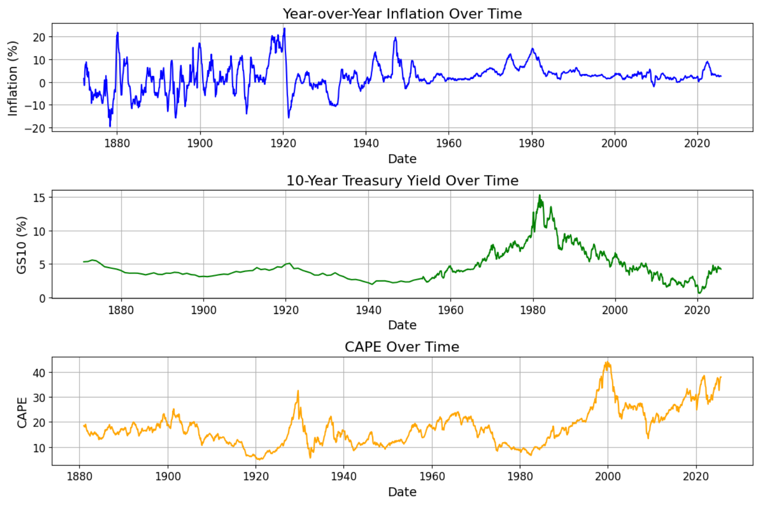
Federal Open Market Committee 2012, *Statement on Longer-Run Goals and Monetary Policy Strategy*, Board of Governors of the Federal Reserve System, Washington, DC, viewed 24 November 2025, Available at: https://www.federalreserve.gov/newsevents/pressreleases/monetary20120125c.htm

Shiller, R.J., 2025. Stock market data. Available at: https://shillerdata.com [Accessed 26 Sep. 2025].

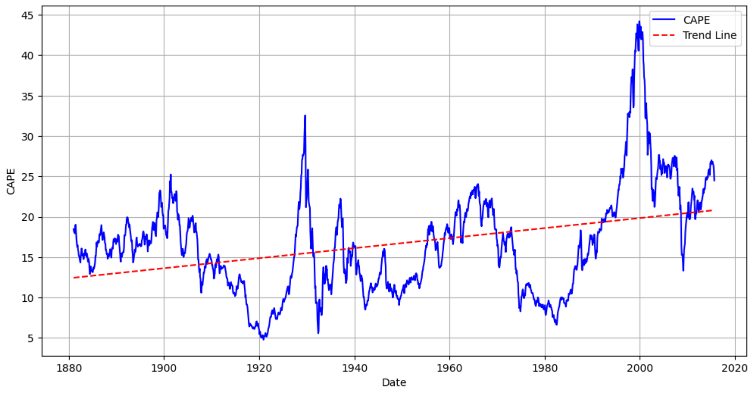
Wang, H., Ahluwalia, H.S., Aliaga-Díaz, R.A. and Davis, J.H., 2021. The best of both worlds: Forecasting US equity market returns using a hybrid machine learning–time series approach. *Journal of Financial Data Science*, 3(2), pp.9–20

A Appendices

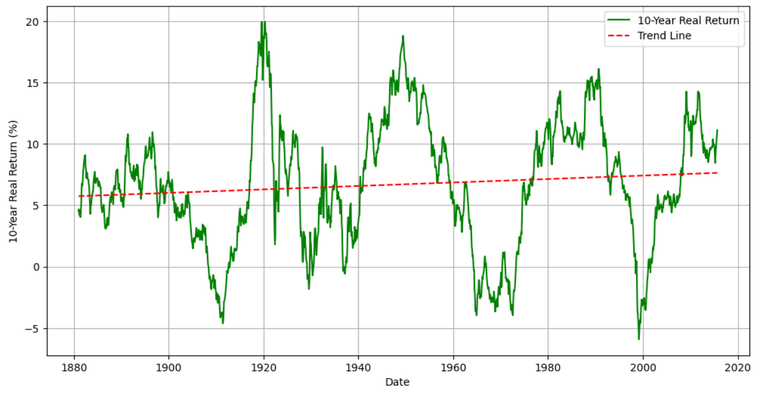
A.1 The levels of CAPE (1981-2025), inflation (1971-2025), and 10-year Treasury Yield (1971-2025) Over Time during



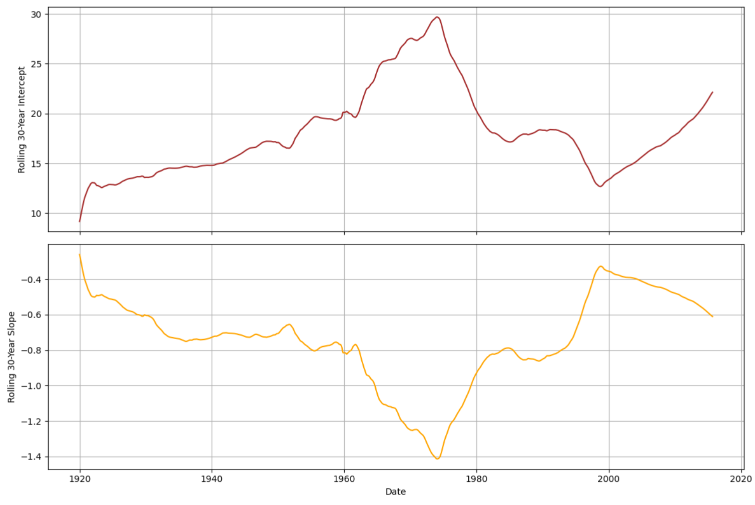
A.2 CAPE trend line over 1881-2025



A.3 The linear trend line of the 10-year-ahead annualized real returns S&P 500 index over 1881-2015



A.4 30-year rolling slope and intercept of the CAPE based linear regression



B Appendices (source code)

B.1 Computations for the raw dataset (Python)

```
import pandas as pd
# Load data
path1 =
 ↪  "/Users/kayttaja/Desktop/BDA_project/data/Shiller_data.xls"
shiller_df = pd.read_excel(path1, skiprows=7)

# Keep only CAPE, CPI, GS10, Date and 10 Year Annualized Stock
 ↪  Real Return
shiller_df = shiller_df[
    ["Date", "CAPE", "CPI", "GS10", "10 Year Annualized Stock
     ↪  Real Return"]
]
# Rename columns for easier access
shiller_df.columns = ["Date", "CAPE", "CPI", "GS10",
 ↪  "Real_Return_10Y"]
# Convert Date to datetime format
shiller_df["Date"] = shiller_df["Date"] * 100
shiller_df["Date"] = pd.to_datetime(shiller_df["Date"],
 ↪  format="%Y%m")
# Sort by Date
shiller_df =
 ↪  shiller_df.sort_values(by="Date").reset_index(drop=True)
shiller_df = shiller_df.set_index("Date")

# Compute year over year inflation
shiller_df["Inflation"] =
 ↪  shiller_df["CPI"].pct_change(periods=12) * 100
# Compute Real_Return_10Y to percent
shiller_df["Real_Return_10Y"] = shiller_df["Real_Return_10Y"] *
 ↪  100

# Save cleaned data
path = "/Users/kayttaja/Desktop/BDA_project/data/Shiller_clean⌋
 ↪  ed.csv"
shiller_df.to_csv(path, index=True)
```

B.2 Source code for plots A.1, A.2, A.3, and A.4 (Python)

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm

# Load the data
file_path = "/Users/kayttaja/Desktop/BDA_project/data/Shiller_⌋
↪  cleaned.csv"
df = pd.read_csv(file_path)
df["Date"] = pd.to_datetime(df["Date"])
df = df.set_index("Date")

# Plot time series of Inflation, GS10 and CAPE
plt.figure(figsize=(12, 8))
title_fs = 16
label_fs = 14
tick_fs = 12

plt.subplot(3, 1, 1)
plt.plot(df.index, df["Inflation"], color="blue")
plt.title("Year-over-Year Inflation Over Time",
↪  fontsize=title_fs)
plt.xlabel("Date", fontsize=label_fs)
plt.ylabel("Inflation (%)", fontsize=label_fs)
plt.grid()
plt.tick_params(axis="both", which="major", labelsize=tick_fs)

plt.subplot(3, 1, 2)
plt.plot(df.index, df["GS10"], color="green")
plt.title("10-Year Treasury Yield Over Time",
↪  fontsize=title_fs)
plt.xlabel("Date", fontsize=label_fs)
plt.ylabel("GS10 (%)", fontsize=label_fs)
plt.grid()
plt.tick_params(axis="both", which="major", labelsize=tick_fs)

plt.subplot(3, 1, 3)
plt.plot(df.index, df["CAPE"], color="orange")
plt.title("CAPE Over Time", fontsize=title_fs)
plt.xlabel("Date", fontsize=label_fs)
plt.ylabel("CAPE", fontsize=label_fs)
plt.grid()
plt.tick_params(axis="both", which="major", labelsize=tick_fs)

plt.tight_layout()
plt.show()


# Make a dataframe for regression
reg_df = df.dropna(subset=["CAPE", "Real_Return_10Y"])
# Let's plot a time series of CAPE and its trend line
plt.figure(figsize=(12, 6))
plt.plot(reg_df.index, reg_df["CAPE"], label="CAPE",
 ↪  color="blue")
# Trend line
z = np.polyfit(np.arange(len(reg_df)), reg_df["CAPE"], 1)
p = np.poly1d(z)
plt.plot(reg_df.index, p(np.arange(len(reg_df))), "r--",
 ↪  label="Trend Line")
plt.xlabel("Date")
plt.ylabel("CAPE")
plt.legend()
plt.grid()
plt.show()

# Let's plot a time series of 10-year real return and its trend
 ↪  line
plt.figure(figsize=(12, 6))
plt.plot(
    reg_df.index, reg_df["Real_Return_10Y"], label="10-Year
     ↪  Real Return", color="green"
)
# Trend line
z = np.polyfit(np.arange(len(reg_df)),
 ↪  reg_df["Real_Return_10Y"], 1)
p = np.poly1d(z)
plt.plot(reg_df.index, p(np.arange(len(reg_df))), "r--",
 ↪  label="Trend Line")
plt.xlabel("Date")
plt.ylabel("10-Year Real Return (%)")
plt.legend()
plt.grid()
plt.show()


# Rolling regression coefficients over time
window_years = 30
start_plot_date = pd.Timestamp("1920-01-01")

dates_rolling = []
slopes_rolling = []
intercepts_rolling = []

for end_date in reg_df.index:
    if end_date < start_plot_date:
        continue

    start_date = end_date - pd.DateOffset(years=window_years)
    sub = reg_df[(reg_df.index >= start_date) & (reg_df.index <
     ↪  end_date)].dropna(
        subset=["CAPE", "Real_Return_10Y"]
    )

    # skip if too few observations in the window
    if len(sub) < 10:
        continue

    X = sm.add_constant(sub["CAPE"])
    y = sub["Real_Return_10Y"]
    m_temp = sm.OLS(y, X).fit()

    dates_rolling.append(end_date)
    slopes_rolling.append(m_temp.params["CAPE"])
    intercepts_rolling.append(m_temp.params["const"])

# Plot as subplots
fig, axes = plt.subplots(2, 1, figsize=(12, 8), sharex=True)

# Intercepts
axes[0].plot(dates_rolling, intercepts_rolling, color="brown")
axes[0].set_ylabel(f"Rolling {window_years}-Year Intercept")
axes[0].grid(True)
```

```
# Slopes
axes[1].plot(dates_rolling, slopes_rolling, color="orange")
axes[1].set_ylabel(f"Rolling {window_years}-Year Slope")
axes[1].set_xlabel("Date")
axes[1].grid(True)

plt.tight_layout()
plt.show()
```

B.3 Source code for illustrating the model training and testing methodology (Figure 3) (Python)

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

train_start = pd.Timestamp("1881-12-01")
test_start = pd.Timestamp("1990-01-01")

step_size_months = 6
val_window_months = 6
gap_years = 10
gap_months = 1
n_folds = 5

rows = []
for fold in range(1, n_folds + 1):
    # validation window
    val_start = test_start + pd.DateOffset(months=(fold - 1) *
     ↪  step_size_months)
    val_end = val_start +
     ↪  pd.DateOffset(months=val_window_months - 1)

    # training window ends 10y + 1m before validation window
     ↪  end
    train_end = val_end - pd.DateOffset(years=gap_years,
     ↪  months=gap_months)

    rows.append(
        {
            "fold": fold,
            "train_start": train_start,
            "train_end": train_end,
            "val_start": val_start,
            "val_end": val_end,
        }
    )

splits_df = pd.DataFrame(rows)
```

```
# Create the plot
fig, ax = plt.subplots(figsize=(12, 4))

for i, row in splits_df.iterrows():
    y = row["fold"]

    # train segment
    ax.hlines(
        y,
        xmin=row["train_start"],
        xmax=row["train_end"],
        color="blue",
        linewidth=8,
        label="Train" if i == 0 else "",
    )

    # validation segment
    ax.hlines(
        y,
        xmin=row["val_start"],
        xmax=row["val_end"],
        color="orange",
        linewidth=8,
        label="Test" if i == 0 else "",
    )

ax.set_yticks(splits_df["fold"])
ax.set_yticklabels([f"Fold {f}" for f in splits_df["fold"]])

ax.xaxis.set_major_locator(mdates.YearLocator(base=10))
ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y"))
plt.setp(ax.get_xticklabels(), rotation=45, ha="right")

ax.set_xlabel("Date")
ax.legend(loc="lower right")
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

B.4 Source code for illustrating the regression line for different 30-year periods (Figure 2) (R)

```
rm(list=ls())
library(dplyr)
library(ggplot2)
library(broom)

file_path <- "~/Desktop/BDA/data/Shiller_cleaned.csv"
df <- read.csv(file_path, stringsAsFactors = FALSE) %>%
  mutate(Date = as.Date(Date),
         Year = lubridate::year(Date),
         Period30_start = 1990 + 30 * ((Year - 1990) %/% 30),
         Period30_end   = Period30_start + 29,
         Period30       = factor(paste0(Period30_start, "-",
         ↪  Period30_end))) %>%
  filter(complete.cases(.)) %>%
  arrange(Date)

summary(df)
mean(df$CAPE)
sd(df$CAPE)
sd(df$Real_Return_10Y)

# Plot regression lines by 30-year period

ggplot(df, aes(x = CAPE, y = Real_Return_10Y)) +
  geom_point(alpha = 0.4) +
  geom_smooth(method = "lm", se = FALSE, colour = "red") +
  facet_wrap(~ Period30) +
  labs(
       x = "CAPE",
       y = "10-Year Ahead Annualized Real Returns (%)") +
  theme_bw()

# Inspect regression coefficients by 30-year period

reg_30 <- df %>%
  filter(!is.na(Period30)) %>%
  group_by(Period30) %>%
  do(
    tidy(lm(Real_Return_10Y ~ CAPE, data = .))
  )

reg_30
```

B.5 Source code for the utility functions (R)

```
library(dplyr)
library(lubridate)
library(brms)

# Function for generating posterior predictions
generate_prediction <- function(model, newdata, prob = 0.95){
  posterior_draws <- posterior_epred(
    model,
    newdata = newdata,
    allow_new_levels = TRUE
  )
  pred_mean <- mean(posterior_draws)
  ci_lower <- quantile(posterior_draws, probs = (1 - prob) / 2)
  ci_upper <- quantile(posterior_draws, probs = 1 - (1 - prob)
   ↪  / 2)
  return (c(pred_mean, ci_lower, ci_upper))
}

# Store convergence diagnostics
convergence_diagnostics <- function(current_date, model){
  # Posterior parameter draws
  draws_array <- as_draws_array(model)

  # Automatically select parameter columns (exclude
   ↪  metadata/internal columns)
  param_cols <- setdiff(dimnames(draws_array)[[3]], c("lp__",
   ↪  "lprior"))

  # Initialize vectors to store diagnostics
  rhat_vals <- numeric(length(param_cols))
  ess_bulk_vals <- numeric(length(param_cols))
  ess_tail_vals <- numeric(length(param_cols))

  # Loop over parameters
  for (i in seq_along(param_cols)) {
    par <- param_cols[i]
    par_draws <- draws_array[,,par]  # iterations x chains
    rhat_vals[i] <- rhat(par_draws)
    ess_bulk_vals[i] <- ess_bulk(par_draws)
    ess_tail_vals[i] <- ess_tail(par_draws)
  }

  # Combine into data frame
  diag_df <- data.frame(
    Date      = current_date,
    Parameter = param_cols,
    Rhat      = rhat_vals,
    ESS_Bulk  = ess_bulk_vals,
    ESS_Tail  = ess_tail_vals
  )

  return (diag_df)
}

# Compute lpd
compute_log_pred_density_hier <- function(model, newdata) {
  log_lik_matrix <- log_lik(model, newdata = newdata,
   ↪  allow_new_levels = TRUE)
  lik_matrix     <- exp(log_lik_matrix)
  log(colMeans(lik_matrix))
}
```

B.6 Source code for building the priors (R)

```
library(dplyr)
library(lubridate)
library(brms)

# Pooled priors
build_priors_from_window <- function(df, train_end,
↪  window_months = 360) {
  prior_start <- train_end %m-% months(window_months)

  prior_window <- df %>%
    filter(Date > prior_start, Date <= train_end)

  if (nrow(prior_window) < 50) {
    stop("Not enough data in prior_window to estimate priors.")
  }

  lm_fit <- lm(Real_Return_10Y ~ CAPE, data = prior_window)
  summ   <- summary(lm_fit)
  beta_tab <- summ$coefficients

  intercept_mean <- beta_tab["(Intercept)", "Estimate"]
  intercept_sd   <- beta_tab["(Intercept)", "Std. Error"]

  slope_mean <- beta_tab["CAPE", "Estimate"]
  slope_sd   <- beta_tab["CAPE", "Std. Error"]

  # Build the prior strings
  slope_prior_str     <- paste0("normal(", slope_mean, ", ",
  ↪  slope_sd, ")")
  intercept_prior_str <- paste0("normal(", intercept_mean, ",
  ↪  ", intercept_sd, ")")
  priors <- c(
    do.call(prior,list(slope_prior_str, class = "b", coef =
    ↪  "CAPE")),
    do.call(prior,list(intercept_prior_str, class = "b", coef =
    ↪  "Intercept"))
    )

  priors
}
```

```
# Hierarchical priors
hierarchical_priors_from_window_uncentered <- function(df,
 ↪  train_end, window_months = 360) {
  prior_start <- train_end %m-% months(window_months)

  prior_window <- df %>%
    dplyr::filter(Date > prior_start, Date <= train_end)

  if (nrow(prior_window) < 50) {
    stop("Not enough data in prior_window to estimate priors.")
  }

  lm_fit   <- lm(Real_Return_10Y ~ CAPE, data = prior_window)
  summ     <- summary(lm_fit)
  beta_tab <- summ$coefficients

  intercept_mean <- beta_tab["(Intercept)", "Estimate"]
  intercept_sd   <- beta_tab["(Intercept)", "Std. Error"]
  slope_mean     <- beta_tab["CAPE",        "Estimate"]
  slope_sd       <- beta_tab["CAPE",        "Std. Error"]

  slope_sd      <- slope_sd * 2
  intercept_sd  <- intercept_sd * 2

  slope_prior_str     <- paste0("normal(", slope_mean,     ",
   ↪  ", slope_sd,     ")")
  intercept_prior_str <- paste0("normal(", intercept_mean, ",
   ↪  ", intercept_sd, ")")

  priors <- c(
    do.call(prior,list(slope_prior_str, class = "b", coef =
     ↪  "CAPE")),
    do.call(prior,list(intercept_prior_str, class = "b", coef =
     ↪  "Intercept")),
    prior(exponential(0.25), class = "sd", group = "Period30",
     ↪  coef = "Intercept"),
    prior(exponential(1),   class = "sd", group = "Period30",
     ↪  coef = "CAPE"),
    prior(lkj(2), class = "cor", group = "Period30")
  )

  priors
}
```

B.7 Source code for the pooled model (R)

```
rm(list = ls())
library(lubridate)
library(ggplot2)
library(dplyr)
library(brms)
library(purrr)
library(parallel)
library(posterior)
library(tidyr)
library(cmdstanr)
setwd("C:/Users/Käyttäjä/Desktop/BDA_project/models")
source("C:/Users/Kyttj/Desktop/BDA_project/models/Functions.R")
set_cmdstan_path("~/cmdstan/cmdstan-2.37.0")

cores <- max(1, parallel::detectCores() - 1)

### Preprocess data ###
file_path <- "C:/Users/Kyttj/Desktop/BDA_project/data/Shiller_⌋
↪  cleaned.csv"
df <- read.csv(file_path, stringsAsFactors = FALSE) %>%
  mutate(Date = as.Date(Date)) %>%
  filter(complete.cases(.)) %>%
  arrange(Date)

train_start <- as.Date("1881-01-01")
test_start <- as.Date("1990-01-01")
test_end <- as.Date("2015-09-01")

# Lists to store the results and diagnostics
predictions <- c()
actuals <- c()
lowers <- c()
uppers <- c()
lpds <- c()
dates_vec <- as.Date(character())
diagnostics <- list()

# Generate test dates
test_dates <- seq(from = test_start, to = test_end, by =
↪  "month")
```

```r
n_iter <- length(test_dates)
step_size_months <- 6L


### Define the model ###
formula <- bf(
  Real_Return_10Y ~ 1 + CAPE,
  family = "gaussian",
  center = FALSE
)


### Rolling Forecast Loop With Model Updates ###
# Define the updating schedule for the priors
prior_step_years <- 3L

prior_update_dates <- seq(
  from = test_start,
  to = test_end,
  by = paste0(prior_step_years, " years")
)

# List to store priors per block
priors_used <- vector("list", length(prior_update_dates))


### The main loop over different prior blocks ###
block_index <- 0

for (k in seq_along(prior_update_dates)) {

  prior_date <- as.Date(prior_update_dates[k])

  # Training end for the first test date in this block
  train_end <- as.Date(prior_date) %m-% years(10) %m-%
   ↪  months(1)
  train <- df %>%
    filter(Date >= train_start, Date <= train_end)

  current_priors <- build_priors_from_window(
    df = df,
    train_end = train_end,
    window_months = 120
  )

  # Save the priors to a list
  priors_used[[k]] <- list(
    block = k,
    prior_date = prior_date,
    train_end = train_end,
    priors = current_priors
  )

  # The model for the current block
  model <- brm(
    formula = formula,
    prior = current_priors,
    data = train,
    chains = 3,
    iter = 2000,
    warmup = 1000,
    backend = "cmdstanr",
    cores = cores,
    adapt_delta = 0.99,
    max_treedepth = 15,
    seed = 1
  )


  # Block boundaries
  block_start <- as.Date(prior_date)
  block_end <- if (k < length(prior_update_dates)) {
    as.Date(prior_update_dates[k + 1]) %m-% months(1)
  } else {
    test_end
  }

  block_dates <- seq(
    from = block_start,
    to = block_end,
    by = paste0(step_size_months, " months")
  )
```

```
# Inner loop for the current block
for (current_date in block_dates) {

  current_date <- as.Date(current_date)

  block_index <- block_index + 1
  cat("\nBLOCK", k, "STEP", block_index,"DATE",
   ↪  as.character(current_date), "\n")

  # Define a 6-month test set window
  window_end <- min(current_date %m+% months(step_size_months
   ↪  - 1L),block_end)

  test_sample <- df %>%
    filter(Date >= current_date, Date <= window_end)

  if (nrow(test_sample) == 0) next  # just in case

  # Predictions
  posterior_draws <- posterior_epred(
    model,
    newdata = test_sample
  )

  # Store the results
  y_pred_vec <- colMeans(posterior_draws)
  ci_lower_vec <- apply(posterior_draws, 2, quantile, probs =
   ↪  0.025)
  ci_upper_vec <- apply(posterior_draws, 2, quantile, probs =
   ↪  0.975)
  y_true_vec <- test_sample$Real_Return_10Y

  # Log predictive density for each testing date
  lpd_vec <- compute_log_pred_density_hier(model,
   ↪  test_sample)

  # Store all results to the lists
  predictions <- c(predictions, as.numeric(y_pred_vec))
  actuals <- c(actuals, as.numeric(y_true_vec))
  lowers <- c(lowers, ci_lower_vec)
```

```
    uppers <- c(uppers, ci_upper_vec)
    dates_vec <- c(dates_vec, test_sample$Date)
    lpds <- c(lpds, lpd_vec)

    # Store the convergence diagnostics for the testing dates
    diag_df <- convergence_diagnostics(current_date, model)
    diag_df$Block <- k
    diag_df$Step  <- block_index
    diagnostics[[block_index]] <- diag_df

    # Update the training end date
    next_train_end <- window_end %m-% years(10) %m-% months(1)

    if (next_train_end > train_end) {
      train_end <- next_train_end
      train <- df %>%
        filter(Date >= train_start, Date <= train_end)

      model <- update(
        model,
        newdata = train,
        recompile = FALSE)
    }
  }
  # Save the current block's model
  saveRDS(
    model,
    file = paste0(
      "Pooled_50yr_block_",
      sprintf("%02d", k), "_",
      format(block_end, "%Y-%m-%d"),
      ".rds"
    )
  )
}

# Save the results
results_df <- data.frame(
  Date = dates_vec,
  Predicted = predictions,
  Actual = actuals,
  Upper = uppers,
  Lower = lowers,
  Lpds = lpds
)

###########
# Import data
# Set the wanted results folder
setwd("C:/Users/Kyttj/Desktop/BDA_project/results/Pooled_50yr_
↪  prior")

# Forecast results
results_df <- read.csv("results_forecast.csv", stringsAsFactors
↪  = FALSE)
results_df$Date <- as.Date(results_df$Date)   # convert back to
↪  Date

# Converge diagnostics
diagnostics_df <- read.csv("diagnostics_forecast.csv",
↪  stringsAsFactors = FALSE)
diagnostics_df$Date <- as.Date(diagnostics_df$Date)

# Priors used in each block
priors_df <- read.csv("priors_used.csv", stringsAsFactors =
↪  FALSE)
priors_df$prior_date <- as.Date(priors_df$prior_date)
priors_df$train_end  <- as.Date(priors_df$train_end)

# The last fitted model
library(rstan)
model <- readRDS("Pooled_50yr_block_09_2015-09-01.rds")
###########


### Inspect results ###
overall_rmse <- sqrt(mean((results_df$Actual -
↪  results_df$Predicted)^2))
r_squared <- cor(results_df$Actual, results_df$Predicted)^2
total_elpd <- sum(lpds)

cat("Overall RMSE:", overall_rmse, "\n")
cat("Overall R-squared:", r_squared, "\n")
cat("Overall ELPD:", total_elpd, "\n")

ggplot(results_df, aes(x = Date)) +
  geom_line(aes(y = Predicted, colour = "Predicted")) +
  geom_line(aes(y = Actual,    colour = "Actual")) +
  geom_ribbon(aes(ymin = Lower, ymax = Upper),alpha = 0.1) +
  labs(
    title = "Predicted vs Actual Returns",
    x = "Date",
    y = "Returns",
    colour = ""
  ) +
  theme_minimal()


### Inspect diagnostics ###
summary(model)

param_diag_df <- bind_rows(diagnostics)

ggplot(param_diag_df, aes(x = Date, y = Rhat)) +
  geom_line() +
  facet_wrap(~ Parameter, scales = "free_y") +
  labs(
    title = "Rhat over time per parameter",
    x = "Date",
    y = "Rhat"
  ) +
  theme_minimal()

ess_long <- param_diag_df %>%
  pivot_longer(
    cols = c(ESS_Bulk, ESS_Tail),
    names_to = "ESS_Type",
    values_to = "ESS_Value"
  )

ggplot(ess_long, aes(x = Date, y = ESS_Value, linetype =
 ↪  ESS_Type)) +
  geom_line() +
  facet_wrap(~ Parameter, scales = "free_y") +
  labs(
    title = "ESS (Bulk and Tail) over time per parameter",
    x = "Date",
    y = "ESS",
    linetype = "Type"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")

mcmc_plot(model, type="trace")

# Posterior predictive checks
pp_check(model)
pp_check(model, type = "scatter_avg")
pp_check(model, type = "intervals")
pp_check(model, type = "error_hist")
pp_check(model, type = "error_scatter")



### Save the diagnostics and results ###
diagnostics_df <- bind_rows(diagnostics)

# Results and diagnostics as csv
write.csv(results_df, "results_forecast.csv", row.names =
 ↪  FALSE)
write.csv(diagnostics_df, "diagnostics_forecast.csv", row.names
 ↪  = FALSE)

# Save the model summary
smry <- summary(model)

# Save the printed version
capture.output(
  print(smry),
  file = "model_summary.txt"
)

# Save the model priors
# Priors_used into a single data frame
```

```
priors_df <- map_dfr(priors_used, function(x) {
  if (is.null(x)) return(NULL)

  p_df <- as.data.frame(x$priors)

  p_df$block <- x$block
  p_df$prior_date <- x$prior_date
  p_df$train_end  <- x$train_end

  p_df
})

# Save the priors to a csv
write.csv(priors_df, "priors_used.csv", row.names = FALSE)
```

B.8 Source code for the partially pooled model (R)

```
rm(list = ls())
library(lubridate)
library(ggplot2)
library(dplyr)
library(brms)
library(purrr)
library(parallel)
library(posterior)
library(tidyr)
library(cmdstanr)
setwd("C:/Users/Käyttäjä/Desktop/BDA_project/models")
source("C:/Users/Kyttj/Desktop/BDA_project/models/Functions.R")
set_cmdstan_path("~/cmdstan/cmdstan-2.37.0")

cores <- max(1, parallel::detectCores() - 1)

### Preprocess data ###

file_path <- "C:/Users/Kyttj/Desktop/BDA_project/data/Shiller_⌋
 ↪  cleaned.csv"
df <- read.csv(file_path, stringsAsFactors = FALSE) %>%
  mutate(Date = as.Date(Date),
         Year = lubridate::year(Date),
         Period30_start = 1990 + 30 * ((Year - 1990) %/% 30),
         Period30_end = Period30_start + 29,
         Period30 = factor(paste0(Period30_start, "-",
          ↪  Period30_end))) %>%
  filter(complete.cases(.)) %>%
  arrange(Date)

train_start <- as.Date("1881-01-01")
test_start <- as.Date("1990-01-01")
test_end <- as.Date("2015-09-01")

# Lists to store the results and diagnostics
predictions <- c()
actuals <- c()
lowers <- c()
uppers <- c()
lpds <- c()
```

```
dates_vec <- as.Date(character())
diagnostics <- list()

# Generate monthly test dates
test_dates <- seq(from = test_start, to = test_end, by =
 ↪  "month")
n_iter <- length(test_dates)
step_size_months <- 6L

### Define the model ###
formula <- bf(
  Real_Return_10Y ~ 1 + CAPE + (1 + CAPE | Period30),
  family = "gaussian",
  center = FALSE
)


### Rolling Forecast Loop With Model Updates ###
# Define the updating schedule for the priors
prior_step_years <- 3L

prior_update_dates <- seq(
  from = test_start,
  to = test_end,
  by = paste0(prior_step_years, " years")
)

# List to store priors per block
priors_used <- vector("list", length(prior_update_dates))


### The main loop over different prior blocks ###
block_index <- 0

for (k in seq_along(prior_update_dates)) {

  prior_date <- as.Date(prior_update_dates[k])

  # Training end for the first test date in this block
  train_end <- as.Date(prior_date) %m-% years(10) %m-%
   ↪  months(1)
```

```
train <- df %>%
  filter(Date >= train_start, Date <= train_end)

current_priors <- hierarchical_priors_from_window_uncentered(
  df = df,
  train_end = train_end,
  window_months = 360
)

# Save the priors to a list
priors_used[[k]] <- list(
  block = k,
  prior_date = prior_date,
  train_end = train_end,
  priors = current_priors
)

# The model for the current block
model <- brm(
  formula = formula,
  prior = current_priors,
  data = train,
  chains = 3,
  iter = 2000,
  warmup = 1000,
  backend = "cmdstanr",
  cores = cores,
  adapt_delta = 0.99,
  max_treedepth = 15,
  seed = 1
)


# Block boundaries
block_start <- as.Date(prior_date)
block_end <- if (k < length(prior_update_dates)) {
  as.Date(prior_update_dates[k + 1]) %m-% months(1)
} else {
  test_end
}
```

```
block_dates <- seq(
  from = block_start,
  to = block_end,
  by = paste0(step_size_months, " months")
)

# Inner loop for the current block
for (current_date in block_dates) {

  current_date <- as.Date(current_date)

  block_index <- block_index + 1
  cat("\nBLOCK", k, "STEP", block_index,"DATE",
   ↪  as.character(current_date), "\n")

  # Define a 6-month test set window
  window_end <- min(current_date %m+% months(step_size_months
   ↪  - 1L),block_end)

  test_sample <- df %>%
    filter(Date >= current_date, Date <= window_end)

  if (nrow(test_sample) == 0) next  # just in case

  # Predictions
  posterior_draws <- posterior_epred(
    model,
    newdata = test_sample,
    allow_new_levels = TRUE
  )

  # Store the results
  y_pred_vec <- colMeans(posterior_draws)
  ci_lower_vec <- apply(posterior_draws, 2, quantile, probs =
   ↪  0.025)
  ci_upper_vec <- apply(posterior_draws, 2, quantile, probs =
   ↪  0.975)
  y_true_vec <- test_sample$Real_Return_10Y

  # Log predictive density for each testing date
  lpd_vec <- compute_log_pred_density_hier(model,
   ↪  test_sample)

  # Store all results to the lists
  predictions <- c(predictions, as.numeric(y_pred_vec))
  actuals <- c(actuals, as.numeric(y_true_vec))
  lowers <- c(lowers, ci_lower_vec)
  uppers <- c(uppers, ci_upper_vec)
  dates_vec <- c(dates_vec, test_sample$Date)
  lpds <- c(lpds, lpd_vec)

  # Store the convergence diagnostics for the testing dates
  diag_df <- convergence_diagnostics(current_date, model)
  diag_df$Block <- k
  diag_df$Step <- block_index
  diagnostics[[block_index]] <- diag_df

  # Update the training end date
  next_train_end <- window_end %m-% years(10) %m-% months(1)

  if (next_train_end > train_end) {
    train_end <- next_train_end
    train <- df %>%
      filter(Date >= train_start, Date <= train_end)

    model <- update(
      model,
      newdata = train,
      recompile = FALSE)
  }
}
# Save the current block's model
saveRDS(
  model,
  file = paste0(
    "Hier_med_block_",
    sprintf("%02d", k), "_",
    format(block_end, "%Y-%m-%d"),
    ".rds"
  )
)
```

```
}

# Save the results
results_df <- data.frame(
  Date      = dates_vec,
  Predicted = predictions,
  Actual    = actuals,
  Upper     = uppers,
  Lower     = lowers,
  Lpds      = lpds
)

###########
# Import data
# Set the wanted results folder
setwd("C:/Users/Kyttj/Desktop/BDA_project/results/Hier_medium_
 ↪  pool")

# Forecast results
results_df <- read.csv("results_forecast.csv", stringsAsFactors
 ↪  = FALSE)
results_df$Date <- as.Date(results_df$Date)   # convert back to
 ↪  Date

# Convergence diagnostics
diagnostics_df <- read.csv("diagnostics_forecast.csv",
 ↪  stringsAsFactors = FALSE)
diagnostics_df$Date <- as.Date(diagnostics_df$Date)

# Priors used in each block
priors_df <- read.csv("priors_used.csv", stringsAsFactors =
 ↪  FALSE)
priors_df$prior_date <- as.Date(priors_df$prior_date)
priors_df$train_end  <- as.Date(priors_df$train_end)

# The last fitted model
library(rstan)
model <- readRDS("Hier_med_block_09_2015-09-01.rds")
###########
```

```
### Inspect results ###

overall_rmse <- sqrt(mean((results_df$Actual -
↪  results_df$Predicted)^2))
r_squared    <- cor(results_df$Actual, results_df$Predicted)^2
total_elpd   <- sum(lpds)

cat("Overall RMSE:", overall_rmse, "\n")
cat("Overall R-squared:", r_squared, "\n")
cat("Overall ELPD:", total_elpd, "\n")

ggplot(results_df, aes(x = Date)) +
  geom_line(aes(y = Predicted, colour = "Predicted")) +
  geom_line(aes(y = Actual,    colour = "Actual")) +
  geom_ribbon(aes(ymin = Lower, ymax = Upper),alpha = 0.1) +
  labs(
    title = "Predicted vs Actual Returns",
    x = "Date",
    y = "Returns",
    colour = ""
  ) +
  theme_minimal()


### Inspect diagnostics ###

summary(model)

param_diag_df <- bind_rows(diagnostics)

ggplot(param_diag_df, aes(x = Date, y = Rhat)) +
  geom_line() +
  facet_wrap(~ Parameter, scales = "free_y") +
  labs(
    title = "Rhat over time per parameter",
    x = "Date",
    y = "Rhat"
  ) +
  theme_minimal()

ess_long <- param_diag_df %>%
  pivot_longer(
    cols = c(ESS_Bulk, ESS_Tail),
    names_to = "ESS_Type",
    values_to = "ESS_Value"
  )

ggplot(ess_long, aes(x = Date, y = ESS_Value, linetype =
↪  ESS_Type)) +
  geom_line() +
  facet_wrap(~ Parameter, scales = "free_y") +
  labs(
    title = "ESS (Bulk and Tail) over time per parameter",
    x = "Date",
    y = "ESS",
    linetype = "Type"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")

mcmc_plot(model, type="trace")

# Posterior predictive checks
pp_check(model)
pp_check(model, type = "scatter_avg")
pp_check(model, type = "intervals")
pp_check(model, type = "error_hist")
pp_check(model, type = "error_scatter")
pp_check(model, type = "dens_overlay_grouped", group =
↪  "Period30")
summary(model)$fixed
summary(model)$random
summary(model)$spec_pars


### Save the diagnostics and results ###
diagnostics_df <- bind_rows(diagnostics)

# Results and diagnostics as csv
write.csv(results_df, "results_forecast.csv", row.names =
↪  FALSE)
```

```
write.csv(diagnostics_df, "diagnostics_forecast.csv", row.names
 ↪  = FALSE)

# Save the model summary
smry <- summary(model)

# Save the printed version
capture.output(
  print(smry),
  file = "model_summary.txt"
)

# Save the model priors
# Priors_used into a single data frame
priors_df <- map_dfr(priors_used, function(x) {
  if (is.null(x)) return(NULL)

  p_df <- as.data.frame(x$priors)

  p_df$block      <- x$block
  p_df$prior_date <- x$prior_date
  p_df$train_end  <- x$train_end

  p_df
})

# Save the priors to a csv
write.csv(priors_df, "priors_used.csv", row.names = FALSE)
```

B.9 Source code for posterior predictive checks (R)

```
    model_pooled <- readRDS("Results/Pooled_30yr_prior/Pooled_⌋
     ↪  weakinf_block_09_2015-09-01.rds")
model_hierarchical <- readRDS("Results/Hier_medium_pool/Pooled⌋
 ↪  _weakinf_block_09_2015-09-01.rds")


p_pooled_dens <- pp_check(model_pooled, type = "dens_overlay",
 ↪  ndraws = 50) +
  labs(title = "A: Pooled Model Fit (Overall)", subtitle =
   ↪  NULL, x = NULL) +
  theme(legend.position = "none",
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank())

p_hier_dens <- pp_check(model_hierarchical, type =
↪  "dens_overlay", ndraws = 50) +
  labs(title = "B: Hierarchical Model Fit (Overall)", subtitle
   ↪  = NULL, x = NULL) +
  theme(legend.position = "none",
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank())

# C. Hierarchical Model Grouped Fit (Bottom)
p_hier_grouped <- pp_check(model_hierarchical, type =
↪  "dens_overlay_grouped", group = "Period30", ndraws = 50) +
  labs(title = "C: Partially Pooled Model Fit (By
   ↪  Time-period)", subtitle = NULL) +
  theme(legend.position = "none")


p_legend_base <- pp_check(model_hierarchical, type =
↪  "dens_overlay") +
  theme(legend.position = "bottom")

get_legend <- function(a_ggplot){
  tmp <- ggplot_gtable(ggplot_build(a_ggplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) ==
   ↪  "guide-box")
  if (length(leg) > 0) tmp$grobs[[leg]] else NULL
}

common_legend <- get_legend(p_legend_base)


# Stack the three main plots first
plots_stacked <- grid.arrange(
  p_pooled_dens,
  p_hier_dens,
  p_hier_grouped,
  ncol = 1,
  heights = c(1.5, 1.5, 3)
```

```
)

plot_combined_ppc_acf <- function(model_pooled,
 ↪  model_hierarchical, n_draws = 50, lag_max = 10) {

  y_raw <- model_pooled$data$Real_Return_10Y
  y <- y_raw[!is.na(y_raw)]


  acf_y <- acf(y, lag.max = lag_max, plot = FALSE)
  df_acf_y <- data.frame(
    lag = acf_y$lag[-1],
    acf = acf_y$acf[-1],
    type = "Observed"
  )

  calculate_acf_draws <- function(model, model_type) {
    message(paste("Generating", n_draws, "draws for",
     ↪  model_type, "..."))
    y_rep_raw <- posterior_predict(model, ndraws = n_draws)
    acf_reps <- list()

    for (i in 1:n_draws) {
      y_rep_clean <- y_rep_raw[i, ][!is.na(y_rep_raw[i, ])]
      if (length(y_rep_clean) < 2) next

      acf_rep <- acf(y_rep_clean, lag.max = lag_max, plot =
       ↪  FALSE)

      acf_reps[[i]] <- data.frame(
        lag = acf_rep$lag[-1],
        acf = acf_rep$acf[-1],
        draw = i,
        Model = model_type
      )
    }
    return(bind_rows(acf_reps))
  }

  # Calculate draws for both models
  df_pooled_acf <- calculate_acf_draws(model_pooled, "Pooled
   ↪  Model Draws")
  df_hier_acf <- calculate_acf_draws(model_hierarchical,
   ↪  "Hierarchical Model Draws")

  # Combine draw dataframes
  df_acf_yrep <- bind_rows(df_pooled_acf, df_hier_acf)

  if(nrow(df_acf_yrep) == 0) {
    stop("Failed to produce valid ACF for predictive draws.")
  }

  p <- ggplot() +
    # Draw the replicated ACFs (thin lines for both models)
    geom_line(data = df_acf_yrep, aes(x = lag, y = acf, group =
     ↪  interaction(draw, Model), color = Model),
              alpha = 0.5) +

    # Draw the observed ACF (thick red line)
    geom_line(data = df_acf_y, aes(x = lag, y = acf),
              color = "red", linewidth = 1.2) +
    geom_point(data = df_acf_y, aes(x = lag, y = acf), color =
     ↪  "red", size = 3) +

    # Define colors for the models (Observed is fixed to red)
    scale_color_manual(
      name = "",
      values = c("Pooled Model Draws" = "darkgreen",
       ↪  "Hierarchical Model Draws" = "darkblue")
    ) +
    scale_x_continuous(breaks = 1:lag_max) +

    labs(
      x = "Lag",
      y = "Autocorrelation Coefficient"
    ) +
    theme_minimal() +
    theme(legend.position = "bottom")

  return(p)
}
```

```
plot_combined_ppc_acf(model_pooled, model_hierarchical, n_draws
↪  = 50, lag_max = 20)
```

B.10 Source code for figures 6, 7, 10 and 11

```
    #####################
#### PREDICTIONS ####
#####################

pooled_30yr_prior_res <- read.csv(
  "Results/Pooled_30yr_prior/results_forecast.csv",
  stringsAsFactors = FALSE
) %>%
  mutate(Date = as.Date(Date))

hierarchical_medium_pool_res <- read.csv(
  "Results/Hier_medium_pool/results_forecast.csv",
  stringsAsFactors = FALSE
) %>%
  mutate(Date = as.Date(Date))

results_df <- hierarchical_medium_pool_res

# Plot the results
ggplot(results_df, aes(x = Date)) +
  geom_line(aes(y = Predicted, colour = "Predicted")) +
  geom_line(aes(y = Actual,    colour = "Actual")) +
  geom_ribbon(aes(ymin = Lower, ymax = Upper), fill = "blue",
   ↪  alpha = 0.1) +
  labs(
    x = "Date",
    y = "Returns",
    colour = ""
  ) +
  theme_minimal()

overall_rmse <- sqrt(mean((results_df$Actual -
↪  results_df$Predicted)^2))
cat("Overall RMSE:", overall_rmse, "\n")
```

```
#####################
#### DIAGNOSTICS ####
#####################

pooled_30yr_prior_diag <- read.csv(
  "Results/Pooled_30yr_prior/diagnostics_forecast.csv",
  stringsAsFactors = FALSE
) %>%
  mutate(Date = as.Date(Date))

diagnostics_df <- pooled_30yr_prior_diag


# --- 2. Prepare ESS Data (Long format) ---
ess_long <- diagnostics_df %>%
  pivot_longer(
    cols = c(ESS_Bulk, ESS_Tail),
    names_to = "ESS_Type",
    values_to = "ESS_Value"
  )

target_params <- c("b_Intercept", "b_CAPE", "sigma")

# Filter the data to include only the target parameters
diagnostics_filtered <- diagnostics_df %>%
  filter(Parameter %in% target_params)

ess_filtered <- ess_long %>%
  filter(Parameter %in% target_params)


# Function to create the Rhat plot for a given parameter
create_rhat_plot <- function(df, param_name) {
  df_param <- filter(df, Parameter == param_name)

  ggplot(df_param, aes(x = Date, y = Rhat)) +
    geom_line(color = "red") +
    labs(
      title = paste0("Rhat: ", param_name),
      x = NULL,
      y = "Rhat"
    ) +
    theme_minimal() +
    theme(plot.title = element_text(size = 10, hjust = 0.5))
}

create_ess_plot <- function(df_ess, param_name) {
  df_param <- filter(df_ess, Parameter == param_name)

  ggplot(df_param, aes(x = Date, y = ESS_Value, linetype =
   ↪  ESS_Type)) +
    geom_line(color = "blue") +
    scale_linetype_manual(values = c("ESS_Bulk" = "solid",
     ↪  "ESS_Tail" = "dashed")) +
    labs(
      title = paste0("ESS: ", param_name),
      x = "Date",
      y = "ESS"
    ) +
    theme_minimal() +
    theme(
      legend.position = "bottom",
      plot.title = element_text(size = 10, hjust = 0.5)
    )
}

p_rhat_intercept <- create_rhat_plot(diagnostics_filtered,
 ↪  "b_Intercept")
p_rhat_cape <- create_rhat_plot(diagnostics_filtered, "b_CAPE")
p_rhat_sigma <- create_rhat_plot(diagnostics_filtered, "sigma")

p_ess_intercept_base <- create_ess_plot(ess_filtered,
 ↪  "b_Intercept")
p_ess_cape_base <- create_ess_plot(ess_filtered, "b_CAPE")
p_ess_sigma_base <- create_ess_plot(ess_filtered, "sigma")

# Extract legend from one plot
get_legend <- function(a_ggplot){
  tmp <- ggplot_gtable(ggplot_build(a_ggplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) ==
   ↪  "guide-box")
  if (length(leg) > 0) tmp$grobs[[leg]] else NULL
```

```
}
combined_legend <- get_legend(p_ess_intercept_base)

p_ess_intercept <- p_ess_intercept_base + theme(legend.position
 ↪  = "none", axis.title.x = element_blank())
p_ess_cape <- p_ess_cape_base + theme(legend.position = "none",
 ↪  axis.title.x = element_blank())
p_ess_sigma <- p_ess_sigma_base + theme(legend.position =
 ↪  "none")


# Arrange the 6 plots first
plots_6_grid <- grid.arrange(
  p_rhat_intercept, p_ess_intercept,
  p_rhat_cape, p_ess_cape,
  p_rhat_sigma, p_ess_sigma,
  ncol = 2,
  layout_matrix = rbind(c(1, 2), c(3, 4), c(5, 6))
)

# Arrange the 6 plots and the legend vertically
grid.arrange(
  plots_6_grid,
  combined_legend,
  ncol = 1,
  heights = c(10, 1) # Plots take 10 units of height, legend
   ↪  takes 1
)



hierarchical_medium_pool_diag <- read.csv(
  "Results/Hier_medium_pool/diagnostics_forecast.csv",
  stringsAsFactors = FALSE
) %>%
  mutate(Date = as.Date(Date))

diagnostics_df <- hierarchical_medium_pool_diag

# Define the 6 key parameters to plot
key_params <- c(
  "b_Intercept", "b_CAPE", "sigma",
  "sd_Period30__Intercept",
  "sd_Period30__CAPE",
  "cor_Period30__Intercept__CAPE"
)

diagnostics_filtered <- diagnostics_df %>%
  filter(Parameter %in% key_params) %>%
  # Optionally: Ensure the plotting order is fixed
  mutate(Parameter = factor(Parameter, levels = key_params))

# Prepare ESS Data (Long format)
ess_long <- diagnostics_df %>%
  pivot_longer(
    cols = c(ESS_Bulk, ESS_Tail),
    names_to = "ESS_Type",
    values_to = "ESS_Value"
  )

# Function to create the Rhat plot (Column 1)
create_rhat_plot <- function(df, param_name, y_label = "Rhat",
↪  x_label = NULL) {
  df_param <- filter(df, Parameter == param_name)

  ggplot(df_param, aes(x = Date, y = Rhat)) +
    geom_line(color = "red") +
    labs(
      title = paste("Rhat:", param_name),
      x = x_label,
      y = y_label
    ) +
    theme_minimal() +
    theme(plot.title = element_text(size = 9, hjust = 0.5))
}

# Function to create the ESS plot (Column 2)
create_ess_plot <- function(df_ess, param_name, y_label = "ESS
↪  Value", x_label = "Date") {
  df_param <- filter(df_ess, Parameter == param_name)
  ggplot(df_param, aes(x = Date, y = ESS_Value, linetype =
   ↪  ESS_Type)) +
    geom_line(color = "blue") +
    scale_linetype_manual(values = c("ESS_Bulk" = "solid",
     ↪  "ESS_Tail" = "dashed")) +
    labs(
      title = paste("ESS:", param_name),
      x = x_label,
      y = y_label
    ) +
    theme_minimal() +
    theme(
      legend.position = "none",
      plot.title = element_text(size = 9, hjust = 0.5)
    )
}

rhat_plots <- list()
ess_plots_base <- list()
N <- length(key_params)

for (i in 1:N) {
  param_name <- key_params[i]

  # Y-axis label only for the first column (Rhat) and ESS plots
  rhat_y_label <- "Rhat"
  ess_y_label <- "ESS"

  # X-axis label only for the last row (i=N) of the second
   ↪  column (ESS)
  x_label <- if (i == N) "Date" else NULL

  rhat_plots[[i]] <- create_rhat_plot(
    diagnostics_filtered, param_name,
    y_label = rhat_y_label,
    x_label = NULL # Rhat column never gets X-label
  )

  ess_plots_base[[i]] <- create_ess_plot(
    ess_long, param_name,
    y_label = ess_y_label,
    x_label = x_label
  )
}


final_plot_list <- list()
for(i in 1:N) {
  final_plot_list[[2*i - 1]] <- rhat_plots[[i]]
  final_plot_list[[2*i]] <- ess_plots_base[[i]]
}

# Remove titles from all ESS plots for a cleaner look
final_plot_list <- lapply(final_plot_list, function(p) {
  if (grepl("ESS:", p$labels$title)) {
    p <- p + labs(title = NULL)
  }
  return(p)
})


# Function to extract the legend (unchanged)
get_legend <- function(a_ggplot){
  tmp <- ggplot_gtable(ggplot_build(a_ggplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) ==
   ↪  "guide-box")
# Use the first ESS plot to generate the legend
p_ess_with_legend <- ess_plots_base[[1]] +
 ↪  theme(legend.position = "bottom")
combined_legend <- get_legend(p_ess_with_legend)

# Generate the layout matrix (6 rows, 2 columns)
layout <- matrix(1:12, ncol = 2, byrow = TRUE)

# Arrange the 12 plots
plots_12_grid <- grid.arrange(
  grobs = final_plot_list,
  ncol = 2,
  layout_matrix = layout
)

# Final arrangement with legend
```

```
grid.arrange(
  plots_12_grid,
  combined_legend,
  ncol = 1,
  heights = c(15, 1)
)
```