Predicting temperature with time series models

December 6, 2021

Table of contents

1 Introduction

In our project we apply a bayesian approach to simple time series modelling and forecasting using Stan. We fit a model to historical hourly temperature data, and try to use it to predict the temperature of the next hour i.e. one time step ahead prediction. Our goal is to be able to get a reasonable prediction accuracy, and to get experience of bayesian time series analysis

time series modelling is a well studied field, and several established models exist ranging from simple to extremely complex. Because time series analysis is fairly new to our group, we will be looking at two simple models, namely the autoregressive model (AR) and the autoregressive moving average model (ARMA). Further after modelling the data using the two approaches we compare them in terms of their predictive accuracy and RMSE errors and use the LFO and LOO for further comparison.

The advantage of using a bayesian approach to these models, is that we get a distribution for the predicted values instead of only a point estimate. The posterior distribution for the prediction can then be used to quantify the uncertainty of the prediction, and to calculate credible intervals which are easy to interpret. Furthermore we can try to avoid overfitting by taking the uncertainty into account. This project primairily could be a stepping stone in our journey to integrate the bayesian element in statistical models and eventually machine learning models. Thereby, constructing more reliable models for commerical usage.

2 Problem and modeling idea

The problem that we specifically target in this project is trying to reduce the overfitting factor in existing statistical time series models AR and ARMA for the temperature time series data. The modelling idea as higlighted in the introduction is to predict a posterior distribution for the temperature value for the next time step instead of a single value which is obatined when apply AR and ARMA without the bayseian element, which ultimately helps us better in accounting for the uncertainity factor. Our task is further sub-divided in choosing some relevant domain specific priors and integerating the statistical models in the bayesian data analysis.

2.1 Problems unique to time series

Time series modelling has been in existing since a long period of time. Although, quite ancient for serially correlated data it is still very efficient in making predictions and forecasting general trends of variables or features with time along one axis. The first and foremost problem one encounters while creating a time series is to make sure that the data we use is stationary implying that its summary statistics such as mean and variance do not change over time. To overcome this two methods are commonly used. Firstly, we can use the dividing the data into parts and calculating their mean and variance, and if for each sub part the mean and variance is same then we can conclude that the data is stationary. Second commonly used method is ADF. The second task before modelling the time series data is to explore the data well through visualisations and test statistics to find trends or seasonality, because most of the time series models need additional coefficients to either acccount for seasonality or require removing the seasonal aspect from the data. After pre-processing depending on the data and auto correlation function ACF and PACF We try to figure out the number of lag terms used to predict the new value i.e. the parameter 'P'

and moving average term 'Q' which is used to predict the next value a.

3 Data description

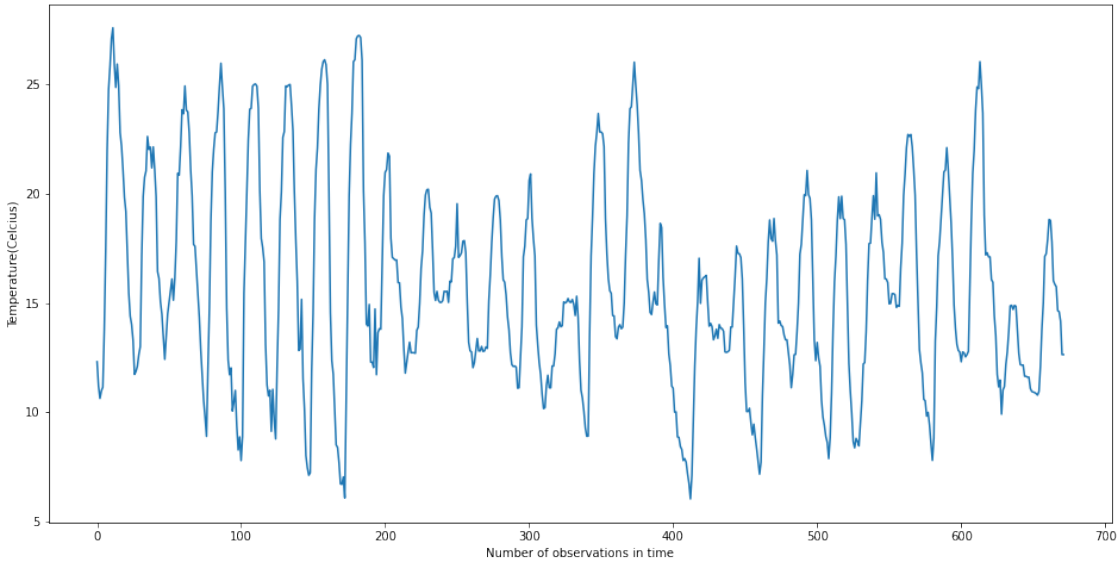
The dataset we were using was downloaded from Kaggle which has hourly weather statistics of Leeds, UK. The datapoints are available from 2006 to 2016 years and covers the following attributes:

- DateTime
- Summary
- Precip Type
- Temperature (C)
- Apparent Temperature (C)
- Humidity
- Wind Speed (km/h)
- Wind Bearing (degrees)
- Visibility (km)
- Loud Cover
- Pressure (millibars)
- Daily Summary

We are using a subset of this data, taking only the temperature data in celcius from the of the year 2013 and month September. We take just a month instead of the whole data because of the huge data size and availability of all datapoints in this month. Our analysis focuses on predicting the next hour's temperature given the previous temperature datapoints hence the time series temperature data is chosen along with the date time column.

Hence, the data comprises of two columns date-time and temperature(^oC) in which we have a total of 672 observation for the month of September 2013.

Below is the plot showing the variation of temperature against time.



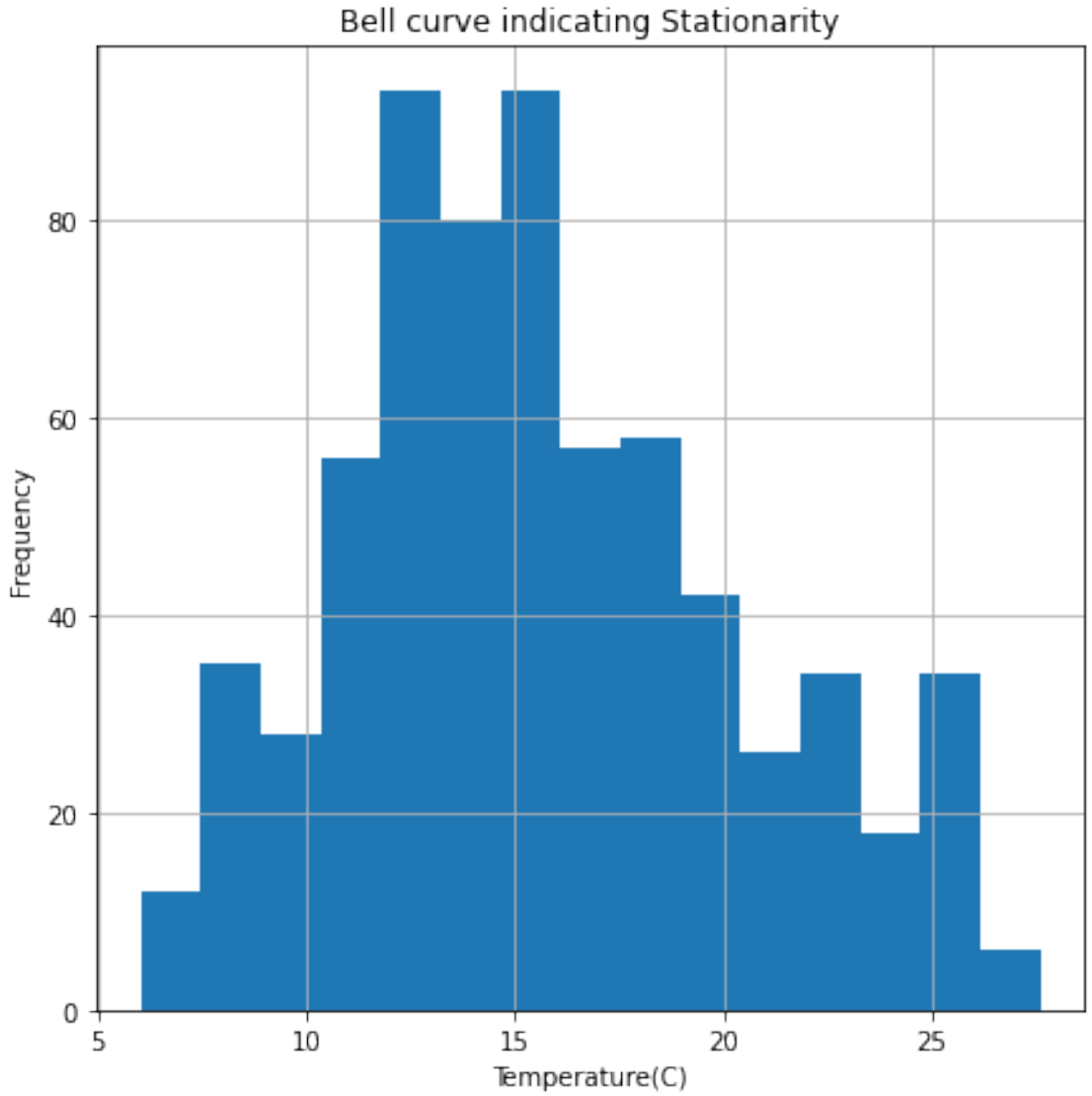
3.1 Testing Stationarity

The data needs to be stationary for the AR and ARMA model to work. So, we first do the stationarity check. In case the data comes out to be stationary we can use the data without any modifications and in case it does not come out to be stationary we modify the data to make it stationary and then proceed.

Below are various Stationarity Test performed:

3.1.1 Bell Curve test

For this we plot the histogram for the variable of interest to confirm whether it follows a bell curve or not if follows the bell curve then we have some assurance that the data might be stationary.



As we can see the bell curve forming, we can say that the data seems stationary.

3.1.2 Summary Statistics test

As a test for stationarity we divide the data into n parts where n can be any real number. Then, we calculate the mean and variance for these parts and if there means and variance almost lie in the same ball park we can me more certain that our data is stationary. If we divide the data into two parts, and check their mean and variance, we get the following:

	Parameter	First Division	Second Division
0	Mean	16.53	14.97
1	Variance	24.94	18.13

The mean and variance values are different for both the divided part, but are in the same ball-park. Hence, this backs up the fact that our data is stationary.

3.1.3 Augmented Dicky-Fuller Test

This test is a quick check to see in a given time series is stationary or not. In this test the null hypothesis is that the series can be represented by a unit root which says it is not stationary. Alternate hypothesis is then that the series is stationary. Running the `adfuller` function from the python package `statsmodels` produces the following output:

```
P Value:  0.01716789490823579
Test Statistics Value:  -3.2519476672986696
Critical Values:  1%: -3.440434903803
                  5%: -2.865989920612
                  10%: -2.569139761751
```

The test results obtained indicate that the test statistic value is -3.25 which is less than the critical values i.e.
'1%': -3.44, '10%': -2.56, '5%': -2.86

We don't need to look at the P value now, we can directly reject the null hypothesis and conclude that the time series is stationary.

4 Model description and priors

Now that our data is stationary we can directly jump to the models without making any modification to our data. Here we will present the two models that we use in our project.

4.1 Autoregressive model

Our first model will be a simple and common model used often in time-series analysis, called the autoregressive model. It is a regression model, where the assumption is that a future value can be generated from previous values. AR models vary in the amount of historical values they take into account for the next prediction. Here we use a AR(4) model, meaning that the prediction is based on the regression coefficients and four previous values. This means that a datapoint y at time t is assumed to be:

$$y_t \sim \text{normal}(\alpha + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \beta_3 y_{t-3} + \beta_4 y_{t-4}, \sigma)$$

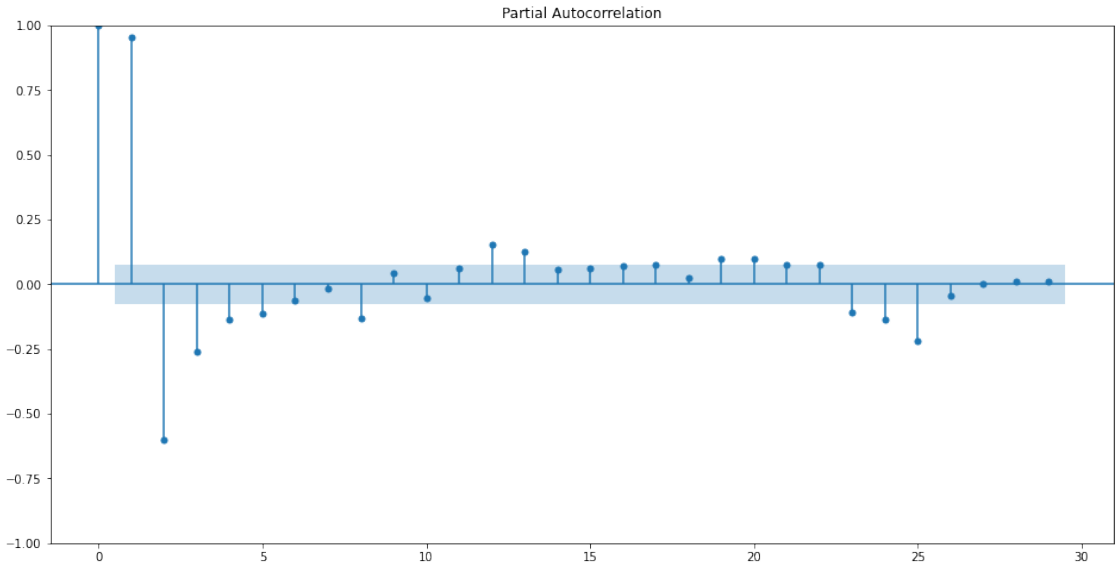
where α and β_i are the regression coefficients, y_{t-i} are the previous values, and σ is the noise term, assumed to be identically distributed for each parameter.

We have a lot of data, and we don't have specific domain knowledge here, so we will use weakly informative priors for the parameters. The error term σ is constrained to be positive, so an exponential distribution is sensible. Temperature is unlikely to double or triple from hour to hour, so the values these priors allow for the regression coefficients are well and beyond what we expect:

$$\begin{aligned} \alpha &\sim \text{normal}(0, 10) \\ \beta_i &\sim \text{normal}(0, 10) \\ \sigma &\sim \exp(1) \end{aligned}$$

The justification for using a AR(4) model comes from observing the partical autocorrelation plot for the data. This plot indicates what is the partial autocorrelation between values of the data at different lag values. Lag means how many points we are looking back at, so for example for points.y_n and y_{n-2} lag would be 2. Partial autocorrelation shows how much data at different lag values correlate without taking into account the effect of datapoints in between. So having a high partial autocorrelation at say lag 3, means that values in the data correlate with values 3 steps back, regardless of the data at steps 1 and 2.

Here is the partial autocorrelation plot for our data. On the y-axis is the correlation (-1,1) and on the x-axis are different lag values. Lag 0 is always 1, since data correlates with itself. The blue area is where the correlation is statistically insignificant, and can be assumed to be zero.



We want to include into our AR model as many degrees as there are correlating lags, so in our case we decide to take 4. Both the 4th and 5th lags have quite small correlation, so we take just up to 4.

4.2 Auto Regressive Moving Average(ARMA) Model

This second model seems like a natural extension to the auto regressive model from the fact that it considers the moving average component as well. ARMA model seems like a good choice for modelling the temperature data as ARMA is capable of modelling a weakly stationary stochastic time series. Similar to AR model this model uses the past datapoints to predict the future data points. The auto regressive component used i.e P = 4 and the moving average component used i.e. Q = 1. Moving average predicts the outputs using the series mean and previous errors which are not accounted for in the AR model. Therefore, a datapoint y and time t can be model using the following equation: From this we use the following mathematical formulation of ARMA model (4,1):

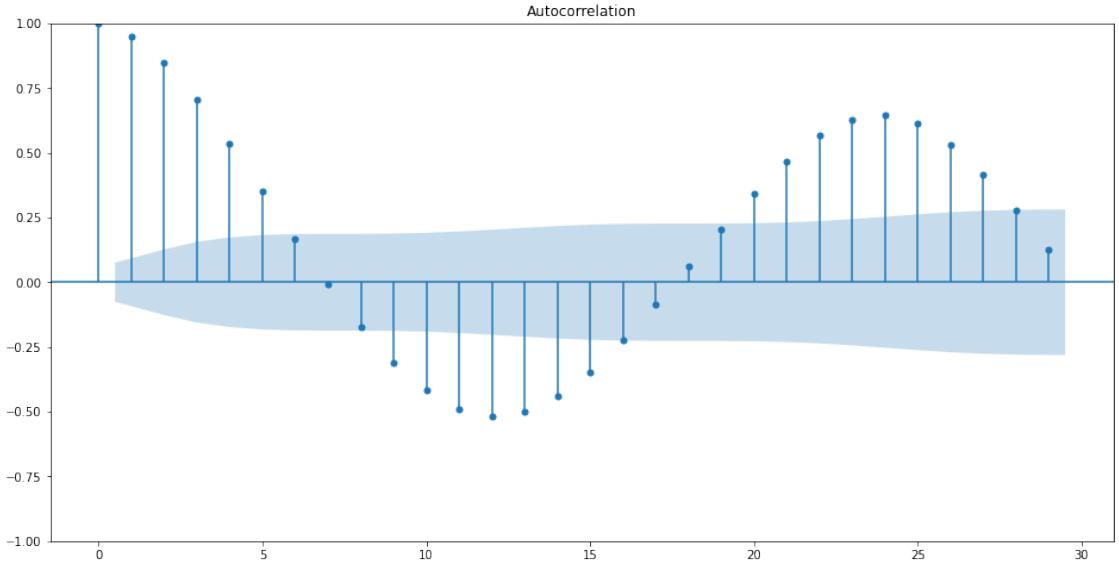
$$y_t \sim normal(\mu + \phi_1.y_{t-1} + \phi_2.y_{t-2} + \phi_3.y_{t-3} + \phi_4.y_{t-4} + .{}_{t-1},)$$

where μ is a constant, ϕ, θ are the regression coefficients, y_{t-1} is the previous value, is the error for each calculated value and σ is the noise term, assumed to be identically distributed for each parameter.

Using the same theory as above, we will be using weakly informative priors

$$\begin{aligned}
\mu &\sim \text{normal}(0, 1) \\
\phi &\sim \text{normal}(0, 2) \\
\theta &\sim \text{normal}(0, 2) \\
\sigma &\sim \text{cauchy}(0, 5) \\
\epsilon &\sim \text{normal}(0,)
\end{aligned}$$

The justification of P and Q values can be explained by the **ACF** and **PACF** figures. we take P as 4 is same as the mentioned above for AR model, which is that till lag 4 we have statistically significant correlation in the Partial Autocorrelation figure above. Here is the ACF plot for the data



Now, looking at the autocorrelation figure, we can notice a very gradual decrease and no sudden drop of lag. This basically tells us that just the MA (Moving-Average) model will not perform well when compared to AR (Auto-Regressive) model on our data. Hence, to reduce complexity in ARMA model we proceed by taking Q as 1, which is the coefficient for Moving Average.

5 Stan code for the models and running them

Here we list the full Stan code for both the AR and the ARMA models, and after then we specify how they were run and how the convergence behaved.

5.1 Stan code for the AR model

```
data {
    int<lower=0> P;    // Order of AR process
    int<lower=0> N;    // Numer of observations
    vector[N] y;       // Observations
    real predval;      // point estimate for LFO
}
parameters {
    real alpha;           // intercept term
    vector[P] beta;       // regression coefficients
    real<lower=0> sigma;  // noise term
}
transformed parameters {
    // mu is the models state at step t
    vector[N] mu;

    // Initial values
    mu[1:P] = y[1:P];
```

```
    for (t in (P + 1):N) {
        mu[t] = alpha;
        for (p in 1:P) {
            mu[t] += beta[p] * y[t-p];
        }
    }
}
model {
    // weakly informative priors
    alpha ~ normal(0, 1);
    beta ~ normal(0, 1);
    sigma ~ exponential(1);

    // Increment the log-posterior
    y ~ normal(mu, sigma);
}
generated quantities {
    // Generate posterior predictives for all values
    vector[N] y_pred;
    for (i in 1:N){
        y_pred[i] = normal_rng(mu[i], sigma);
    }

    // log likelihood point estimate for PSIS-LFO
    real log_lik_lfo;
    log_lik_lfo = normal_lpdf(predval | mu[N], sigma);

    // log_likelihood
    vector[N] log_lik;
    for (i in 1:N){
        log_lik[i] = normal_lpdf(y[i] | mu[i], sigma);
    }
}
```

5.2 Stan code for the Auto regressive moving average model(ARMA) with P = 4 , Q = 1

```
data {
    int P;
    int<lower=1> N;       // num observations
    vector[N] y;          // observed outputs
}
parameters {
    real mu;                // mean coeff
    vector[P] phi;          // autoregression coeff
    real theta;             // moving avg coeff
    real<lower=0> sigma;    // noise scale
```

```
}
transformed parameters {

    vector[N] nu;          // prediction for time t
    vector[N] err;         // error for time t

    nu[1] = mu + phi[1] * mu; // assume err[0] == 0
    err[1] = y[1] - nu[1];    // first error term

    nu[2] = mu + phi[1] * y[1] + phi[2] * mu +  theta*err[1];
    err[2] = y[2] - nu[2];

    nu[3] = mu + phi[1] * y[2] + phi[2]*y[1] + phi[3] * mu + theta*err[2];
    err[3] = y[3] - nu[3];

    nu[4] = mu + phi[1] * y[3] + phi[2] * y[2] + phi[3]*y[1] + phi[4] * mu +  theta*err[3];
    err[4] = y[4] - nu[4];

    for (t in 5:N){
      nu[t] = mu + phi[1]*y[t-1] + phi[2]*y[t-2] + phi[3]*y[t-3] + phi[4]*y[t-4] +
              theta*err[t-1];
      err[t] = y[t] - nu[t];
    }
}
model {
    mu ~ normal(0, 10);     // priors
    phi ~ normal(0, 10);
    theta ~ normal(0, 2);
    sigma ~ cauchy(0, 5);
    err ~ normal(0, sigma); // likelihood -log posterior
}
generated quantities{
    real y_pred;
    y_pred = normal_rng(mu + phi[1]*y[N] + phi[2]*y[N-1] + phi[3]*y[N-2]
                        + phi[4]*y[N-3] + theta*err[N],sigma);
}
```

5.3 Model running

When running the AR model the CmdStanPy defaults were used, which is 4000 samples and 1000 warmup iterations. This resulted in convergence for all parameters, with an $\hat{R}$ score of less than 1.05. No transitions ended in divergence or hit the maximum treedepth. The effective sample size for most parameters was between 1500 and 2000, which we deemed adequate.

For the ARMA model on the other hand default parameters proved to not result in convergence, and we had to increase the warmup iteration amount to 2000. After this all $\hat{R}$ scores were satisfactory and the chains had converged, but 15 transitions hit the maximum treedepth and 38 ended with divergence (out of 4000 samples). This hints that the sampling is having some

difficulties, and some draws have to be discarded in the process. This makes the sampling less optimal and hinders performance, but is not necessarily on its own terrible if its not a large number of transitions. For the ARMA model most sample sizes were between 1000 and 2000, which we deemed adequate.

6 Posterior predictive checks

Since we are interested in predicting the next values in the time series, the posterior checks are essentially what we look at in the **Predictive performance** section below, where we compare the models predictions to the actual data. That is the most sensible way to asses the quality of the posterior. Just looking at a histogram of values does not show the time component of the values, and would thus not show relevant information. We could inspect here either maximum or minimum values, or have a look at how large the percentual changes between consecutive values are in the predicted data, but both of these are faster and easier to check from the visual plot of the predictions.

7 Comparing the two models

To compare the models we will use the leave one out (LOO) measure as well as the leave future out (LFO) measure. LOO has as ready made implementation in the python Arviz package, but for LFO we had to try to implement our own version.

7.1 LOO

Since the LOO metric was used in the course, we won't explain it in more detail here. In essence it evaulates the models predictive performance by consecutively taking each value out of the data, and calculating what is the likelihood of seeing that specific real datapoint based on the model. When comparing two models, the one with a higher LOO score should theoretically be better.

Running the Arviz loo for the AR model produces the following:

```
         Estimate       SE
elpd_loo -1035.62    27.61
p_loo        8.41        -
------

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)      672  100.0%
 (0.5, 0.7]   (ok)          0    0.0%
   (0.7, 1]   (bad)         0    0.0%
   (1, Inf)   (very bad)    0    0.0%
```

And for the ARMA model:

```
         Estimate       SE
elpd_loo -1086.48    36.82
```

```
p_loo       20.84        -

There has been a warning during the calculation. Please check the results.
------

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)      670   99.7%
 (0.5, 0.7]   (ok)          0    0.0%
   (0.7, 1]   (bad)         1    0.1%
   (1, Inf)   (very bad)    1    0.1%
```

So

```
model    elpd_loo
AR       -1036
ARMA     -1086
```

As we see from the output, with the ARMA model we had some bad k values, which raises the possibility that the LOO values for ARMA is optimistic. However this does not change the ordering of the models, since the LOO value for AR is still higher. So in conclusion, based on the LOO estimates the AR model performs better.

7.2 LFO

This section of the LFO method and the code to implement it is in essence a summary of the loo vignette and the paper *Approximate leave-future-out cross-validation for Bayesian time series models* by Bürkner, Gabry & Vehtari.

Using LOO to assess time series models is not optimal, because of the time-dependency of both the model and the data. If the model has access to all datapoints exept one in the middle, what follows is essentially that future data ends up influencing previous data, which in the real world does not make sense. A better approach is to use the so called leave future out (LFO) metric, in which the model only has access to data up to a certain time point, and it is then allowed to predict the next points.

Using the MCMC method, we can use samples to estimate the likelihood for a datapoint y_{i+1} conditined only on previous datapoints $y_{1:i}$. So the model in use is fitted with only i datapoints, and then the likelihood for y_{i+1} is evaluated:

$$p(y_{i+1} \ | y_{1:i}) \approx \frac{1}{S} \sum_{s=1}^{S} p(y_{i+1} \ | \ y_{1:i}, \theta_{1:i}^{(s)})$$

Using this likelihood estimate, we can calculate the LFO-ELPD for the entire dataset. Usually some starting index L is defined so that we have some data to start with. The equation for the entire estimate is then:

$$\text{ELPD} = \sum_{i=L}^{N-1} \log p(y_{i+1} \ | y_{1:i})$$

Using this approach, the model has to be refit again for each index of i, since we are at each step feeding in a new datapoint to condition the model on. Model fitting is slow and heavy, so this is not optimal. Instead Bürkner et al. suggest using the pareto smoothed importance sampling (PSIS) presented in the paper Pareto Smoothed Importance Sampling

Importance sampling allows us to reduce the amount of refitting by approximating the predictive density for several points in the future using the smoothed importance weights w_i and the previously fitted model. If we have fitted the model at point a, then we increase i and get the next density by

$$p(y_{i+1} \mid y_{1:i}) \approx \frac{\sum_{s=1}^{S} w_i^{(s)} p(y_{i+1} \mid y_{1:i}, \theta_{1:a}^{(s)})}{\sum_{s=1}^{S} w_i^{(s)}}$$

So in essence we are keeping the model the same, but looking further and further into the future, and the likelihoods of the data. To get the smoothed weights, we first calculate the importance ratios r

$$r_i^{(s)} = \prod_{j \in (a+1):i} p(y_j \mid y_{1:(j-1)}, \theta_{1:a}^{(s)})$$

So we essentialy take the product of consecutive likelihoods beginning from the latest model fit point a. These are then smoothed with PSIS and used to estimate the predictive density. A more detailed derivation is prsented in the LFO paper.

The shape parameter k of the Pareto distribution in PSIS is evaluated, and then once that exceeds some boundary, the model is refitted, and the process continues. Each time the model is refitted, we get one exact ELPD value "for free" because for that next point we have the correct model. Here is a pseudocode of the procedure

```
L = number of observations to start with
N = total observations
a = L
out = zeros(N)
fit model with first L datapoints
out[L] = exact ELPD for prediction L+1

# loop over rest of values
for i in (L+1):(N-1)
    get log_likelihoods for values a:i
    calculate importance ratios
    do PSIS on ratios
    k = psis_k_value
    if k > limit:
        a = i
        refit model with 1:i values
        out[i] = exact ELPD for prediction i+1
    else:
        log_weights = psis_log_weigts
        out[i] = approx ELPD for i+1
```

```
return sum(out)
```

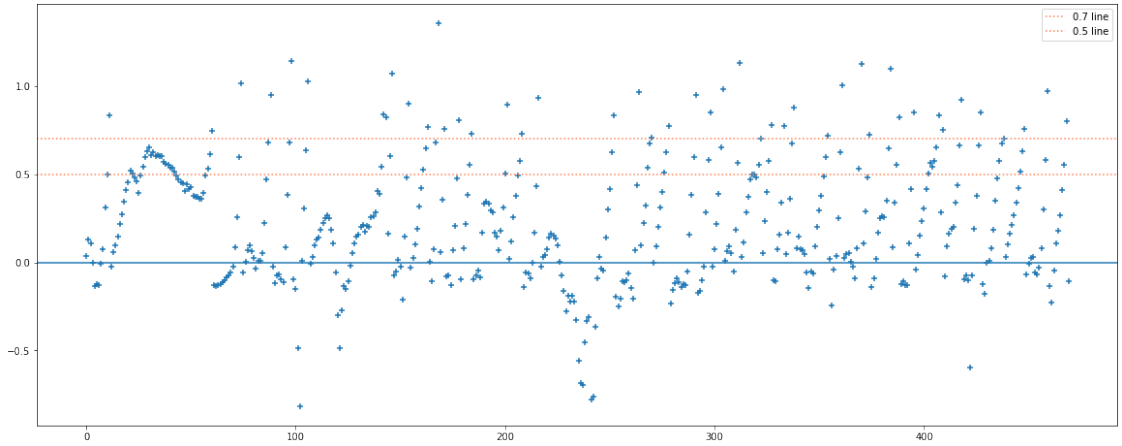
The python code we wrote is in the appendix.

Running the LFO for both models gives the following:

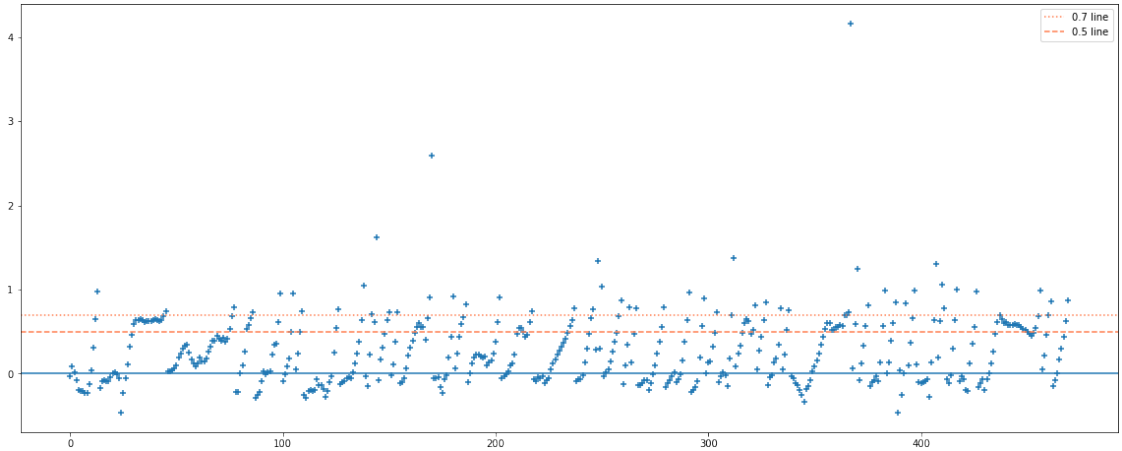
model	elpd_lfo
AR	-1930
ARMA	-1943

The LFO also suggests that the AR model performs better.

We can also visualize the k values for the LFO. Here is the plot for the AR model:



And here the same for the ARMA model:



From the k plots we can observe the expected behaviour of the LFO method, where the k values gradually get worse, and once they exeed the limit set for them, they drop and start rising again. In both cases the amount of refits the model had to do was rather high. The plot for the ARMA model is a bit distorted, since there were some values with $>> 1$ values.

8 Predictive performance

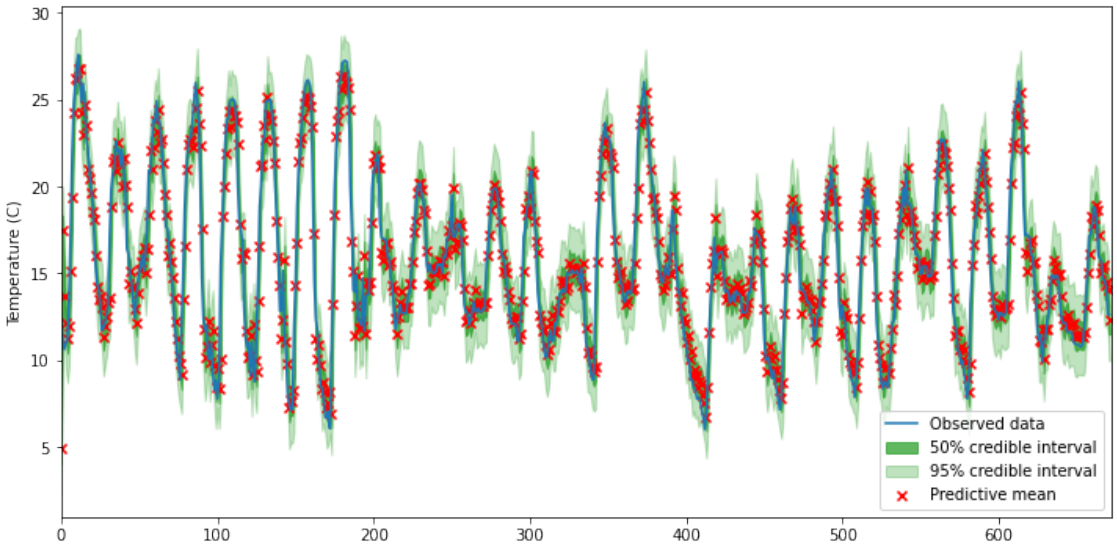
The section in the Stan code for predicting datapoints and the way to visualize the posterior intervals in python has been influenced by this blog post from Jack Walton.

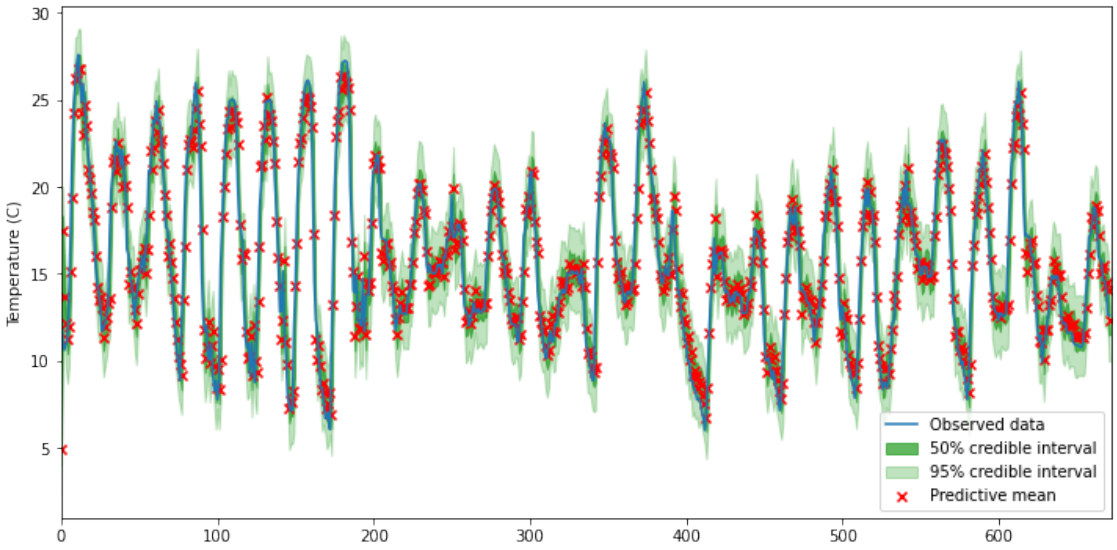
To assess how well our model manages to predict the 1-step forecast with our dataset, we calculate the root mean squared error (RMSE) between the predictions of the model and the actual datapoints. This value will give us a numerical score of the quality of the predictions.

model	RMSE
AR	1.1176
ARMA	1.1896

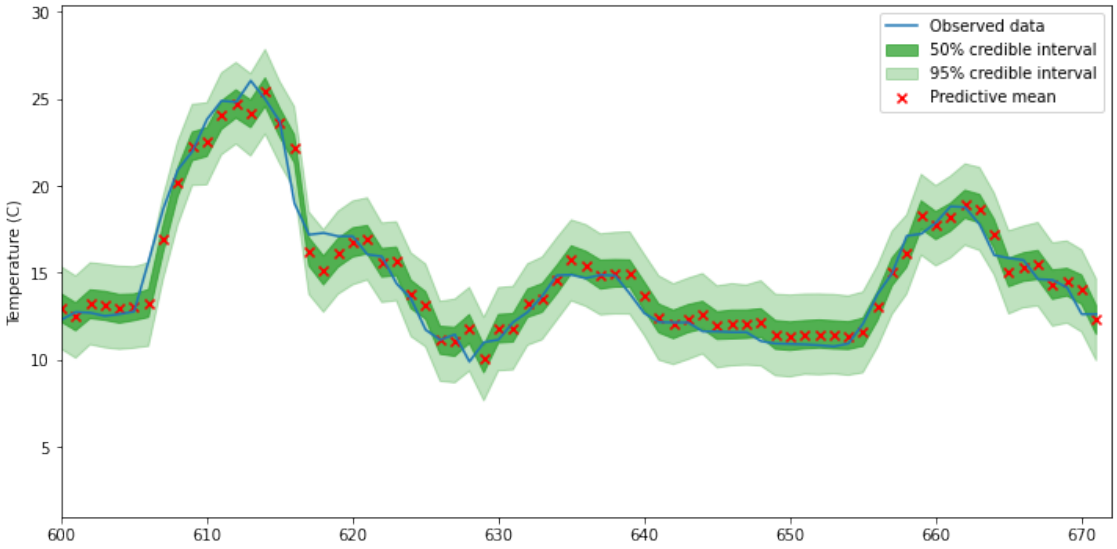
The RMSE score tells that on average the AR model's prediction is 0.12 degrees off, and the ARMA model is 0.19 degrees of. Here again, AR seems to be the better model.

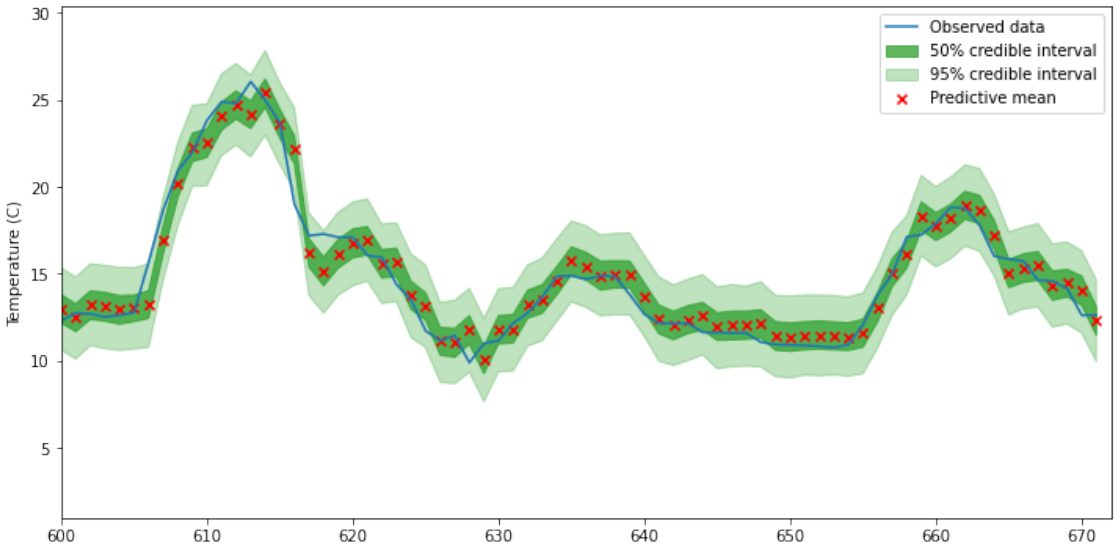
Additionally we plot the predictions and their 50% and 95% credible intervals to visually inspect how close the predicted distribution matches the original data. Here are the predictive densities for the entire data set first for the AR model, and then for the ARMA model:





Looking at the whole data, it is a bit hard to see details so we will show the last values up from index 600:





Visually both models seem to be performing quite well, and its not really possible to tell which one would be better. The plots look identical. An interesting detail is how the uncertainty of the model is smaller in those intervals where there is either a strong down- or upward trend. This shows that the model has learned that trends tend to keep moving in the direction they are going, whereas stationary trends where the temperature does not change are harder to predict, since it could start changing up or down.

9 Prior sensitivity analysis

9.1 AR model

For the AR model 5 different types of priors were tested namely the **weakly informative priors**, **improper flat priors** a mixture of **weakly informative and improper flat priors** and some randomly chosen priors. After carrying out the $\hat{R}$ convergence analysis and checking the changes in posterior we reach a staganant conclusion. The $\hat{R}$ value obtained for each set of tested priors is "1" with no significant change in the posterior.

The purpose of priors is to guide us to the correct posterior i.e. infering the values of estimated paramters and data in model is also trying to do the same thing, therefore when we use weakly informative priors, we try to add some prior information to our model, when we use strongly informative priors our goal is to add significant prior information in the model but sometimes the data or priors can overwhelm each other therefore it looks like our model is overwhelmed by the data. This might be due to the large sample size and weakly informative priors.

The $\hat{R}$ convergence with different priors is summarized in the table below:

α	β		$\hat{R}$ *value*	*Prior type*
$normal(0,1)$	$normal(0,1)$	$exponential(1)$	1	*weakly informative*
–	–	–	1	*improper flat priors*
$normal(0,1)$	–	–	1	*weakly informative and flat*
$normal(0,100)$	$normal(0,100)$	$exponential(10)$	1	*uninformative*
$cauchy(10,100)$	$cauchy(0,10)$	$gamma(10)$	1	*random*

9.2 ARMA model

For ARMA model we again tested 5 different priors to see how sensitive the $\hat{R}$ convergence is in each case. We can see that using Improper flat prior gives us no convergence with a $\hat{R}$ of 2.34. After trying a combination of weakly informative and improper prior we proceed to reducing the standard deviation and increasing the mean for our 3 parameters (μ, ϕ, θ) and we notice that the $\hat{R}$ value decreases to 1.6. When weakly informative prior is desired, lower parameter values are to be used. Hence, We next try out a weakly informative prior with zero mean and relevant standard deviation value, we can see the $\hat{R}$ to be almost similar with a value of 1.53.

For σ sampling we stick to Cauchy distribution with a zero mean as we know that mean does not exist for a Cauchy prior as according to Ghosh et al. and a small value of standard deviation which works well. We also try a random Cauchy only priors for all parameters and notice convergence even in less chains. We take somewhat relevant values for Cauchy prior as per our knowledge.

Overall, we can see that our $\hat{R}$ value is pretty sensitive to the priors we choose for our 4 parameters. We don't play around with the ϵ prior and let it be same for all cases which is $(0, \sigma)$.

The $\hat{R}$ convergence for ARMA with different priors is summarized in the table below:

μ	ϕ	θ	σ	$\hat{R}$ *value*	*Prior type*
–	–	–	–	2.34	*Improper Flat*
$normal(0,100)$	–	–	$cauchy(0,5)$	2.30	*Weakly and Improper prior*
$normal(100,15)$	$normal(100,15)$	$normal(100,15)$	$cauchy(0,5)$	1.60	*Uninformative*
$normal(0,10)$	$normal(0,10)$	$normal(0,2)$	$cauchy(0,5)$	1.53	*Weakly Informative*
$cauchy(10,100)$	$cauchy(0,10)$	$cauchy(0,10)$	$cauchy(0,5)$	1.00	*Random Unimformative*

10 Issues and potential improvements

Both the AR and the ARMA model assume that the time series they are modelling is stationary. We chose our dataset to such that it was stationary to be able to use these models, but in practice this is not optimal. Temperature in general is not always stationary, because it might have strong seasonal trends, especially in northern or southern countries.

To simplify our modelling task we also only looked at 1-step ahead predictions, so trying to predict the next hours temperature based on the previous values. If we were to extend this project, it would be interesting to compare these same models but try different ranges of prediction, to see how many hours into the future we can still attain reasonable prediction accuracy.

11 Conclusions

Both the AR and the ARMA model proved to be capable to achieve reasonable accuracy on our chosen dataset. We managed to attain a prediction accuracy that was on average 0.12 degrees off from the actual temperature for that hour. The AR model even though simpler in design managed to outperform the more complex ARMA model, although not with a large margin. This shows a point about modelling in general, that one should begin with simple models and only move on to more complex ones if the simple models fail.

In addition to being more complex, the ARMA model also took longer to compute and needed more iterations from the sampler to reach convergence.

12 Self reflection

Going into this we knew that we wanted to do something with time series. Since the topic was fairly new to us and had not been presented much in the course, we decided to stick with 2 relatively simple models and only a 1 dimensional dataset. This allowed us to focus on the models and how to run them and assess their performance. The Stan manual proved valuable here, as there were good example implementations there to build upon.

We would say that we met our goal of learning basics of how time series models are implemented in Stan, and how the bayesian approach to these models differs from the traditional one. In addition to this, we learned a ton about time series modelling in general while researching material for this project. The concept of stationarity in time series, how autocorrelation and partial autocorrelation influence model choice and LFO were mostly new. Models like the autoregressive integrated moving average (ARIMA) and stochastic volatility models were something we found interesting and considered doing, but rejected for sake of simplicity

Because we selected this sort of prediction task, we did not get much experience of building highly custom case dependent models or comparing hierarchical models to separate models, as we have done during course. Also our project did not have a clear question or goal of analysis that we would have set out to solve as there is in some topics. But regardless of this we found this topic to be motivating and educational.

13 Appendix

Python code for LFO.

```
def fit_model(fulldata, end):
    """
        data: the full data in question
        end: up to which index we fit the model
    """
    data = fulldata[:end]
    model = CmdStanModel(stan_file='simple_ar.stan')
    input_data = dict(
        P = 4,
```

```
        N = len(data),
        y = data,
        predval = fulldata[end+1] # predict next value
    )
    mcmc = model.sample(data=input_data)
    return (model, mcmc)

def next_estimate(fulldata, end, model, mcmc, pred):
    """ Rerun CmdStanPy models generate_quantities with new data
            without refitting the model. In essence get the
            log_likelihood of a new point.
    """
    # first recreate input data with new point to predict
    data = fulldata[:end]
    input_data = dict(
        P = 4,
        N = len(data),
        y = data,
        predval = fulldata[pred] # predict next value
    )
    # then generate quantities and return 1D log_likelihood
    m = model.generate_quantities(data=input_data, mcmc_sample=mcmc)
    log_lik = m.stan_variable('log_lik_lfo')
    return log_lik[:, 0]


def psis_lfo_cv(data, L, tau=0.7):
    """ Calculate 1-step ahead LFO for a model

    data:  temperature data we are fitting
    L:     where to start LFO calculation
    tau:   threshold for k values for refitting
    """
    N = len(data)
    out = np.zeros(N)
    k_values = []  # store k values here

    # first fit the model with first L observations
    i_star = L
    model, mcmc = fit_model(data, i_star)

    # calculate exact ELPD value by taking mean of log likelihood
    log_lik = mcmc.stan_variable('log_lik_lfo')
    exact_elpd = np.log(np.mean(np.exp(log_lik)))
    out[L] = exact_elpd

    log_likes = []
    for i in range(L+1, N-1):
```

```
            print(i)
            # get next log likelihood
            log_lik = next_estimate(data, i_star, model, mcmc, i+1)
            log_likes.append(log_lik)

            # get weights by multiplying (sum of log) observations
            ratios = np.sum( np.stack(log_likes, axis=1), axis=1)
            # make psis object and check k value, decide if to refit
            log_weights, k = psislw(ratios)
            k_values.append(k)
            if k > tau:
                # need to refit the model
                i_star = i
                model, mcmc = fit_model(data, i_star)
                # get the exact elpd value again
                log_lik = mcmc.stan_variable('log_lik_lfo')
                exact_elpd = np.log(np.mean(np.exp(log_lik)))
                out[i] = exact_elpd
                log_likes = []  # reset
            else:
                # use approximation for PSIS with log sum of log weights
                out[i] = np.log(np.sum(np.exp( log_weights + log_lik )))

    return np.sum(out), k_values
```

We used R code as well for doing prior sensitivity analysis for both AR and ARMA model.

```
library(aaltobda)
library(rstan)
library(bayesplot)
library(loo)
data =read.csv("Leeds_Temp_2013-09.csv")

stan_data <- list(
P = 4,
N = length(data$Temperature..C.),
y = data$Temperature..C.
)
model_ar <- stan(file = 'AR_V1.stan', data = stan_data)

monitor(model_ar)
ext = extract(model_ar)

loglik_ar = extract_log_lik(model_ar, merge_chains = FALSE)
var_r_eff = relative_eff(exp(loglik_ar))
loo_ar_var = loo(loglik_ar, r_eff = var_r_eff, type = "lfo")
print(loo_ar_var)
```

```r
stan_data <- list(
P = 4,
N = length(data$Temperature..C.),
y = data$Temperature..C.
)
model_arma <- stan(file = 'ARMA_V1.stan', data = stan_data)

monitor(model_arma)
ext = extract(model_arma)

plot(data$Temperature..C.,type="l", col = 'blue')

lines(ext$nu[1,],type="l", col = 'red')

for (i in 2:4)
  {
    lines(ext$nu[i,],type = "l",col="red")
  }

loglik_arma = extract_log_lik(model_arma, merge_chains = FALSE)
var_r_eff = relative_eff(exp(loglik_arma))
loo_arma_var = loo(loglik_arma, r_eff = var_r_eff, type  ="lfo")
print(loo_arma_var)
```