CS146 Final Projects
Viet Hoang Tran Duong
Fall 2019

Executive Summary: CO_2 measurement modeling

Topic: CO_2 measurements of the world has been increasing steadily since the start of the industrial revolution in the 18th century. We attempt to build a statistical model to predict future CO_2 measurements to better prepare for future outcomes.

Data: Mauna Loa CO_2 measurements from ice core measurements.

Attempts: Create a statistical model to explain the data and forecast how the measurements will look like in 40 years, predict when the CO_2 level will be considered high risk (>450 ppm).

The outline of the projects is as followed:

Part 1: Data preprocessing

Step 1: Subtract the date data from the first date (1958-03-29) to have the Δt: time difference from the initial date. We do this because the model takes inputs as numeric values, not as date values. Hence, transforming the data would make it more convenient for running pystan.

Step 2: Visualize the data.

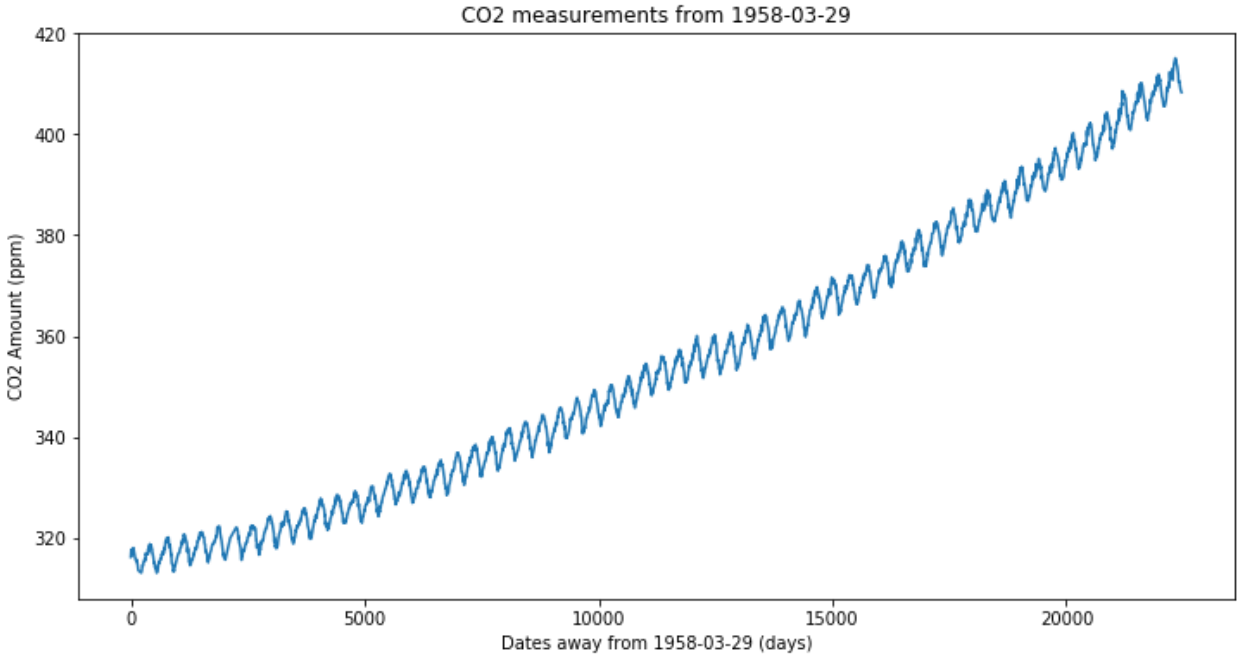


Fig. 1: CO_2 measurements from 1958-03-29

Step 3: Data partitioning

I subset the data into the first 70% data (2197 values) as the input for computing posterior distributions over the parameters using Stan, and the remaining latter 30% (942 values) to validate our prediction of future functions values.

Part 2: Model 1: The example model and its flaws

$$p(x_t|\theta) = N\left(c_0 + c_1 t + c_2 \cos\left(\frac{2\pi t}{365.25} + c_3\right), c_4^2\right)$$

- First, without any constraints on the parameters $c0, c1, c2, c3, c4$, the model diverges, with very correlated samples (based on the n_eff and Rhat values)

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
c0	310.31	1.4e-3	0.09	310.14	310.25	310.31	310.37	310.48	3593	1.0
c1	3.7e-3	1.6e-7	9.4e-6	3.7e-3	3.7e-3	3.7e-3	3.7e-3	3.7e-3	3658	1.0
c2	2.71	0.09	0.14	2.47	2.57	2.72	2.84	2.94	2	2.54
c3	2.92	2.07	2.92	6.6e-5	9.0e-4	2.9	5.85	5.88	2	218.23
c4	2.0	0.06	0.09	1.87	1.91	2.01	2.09	2.15	2	3.33

- With the pair plots between parameters are:

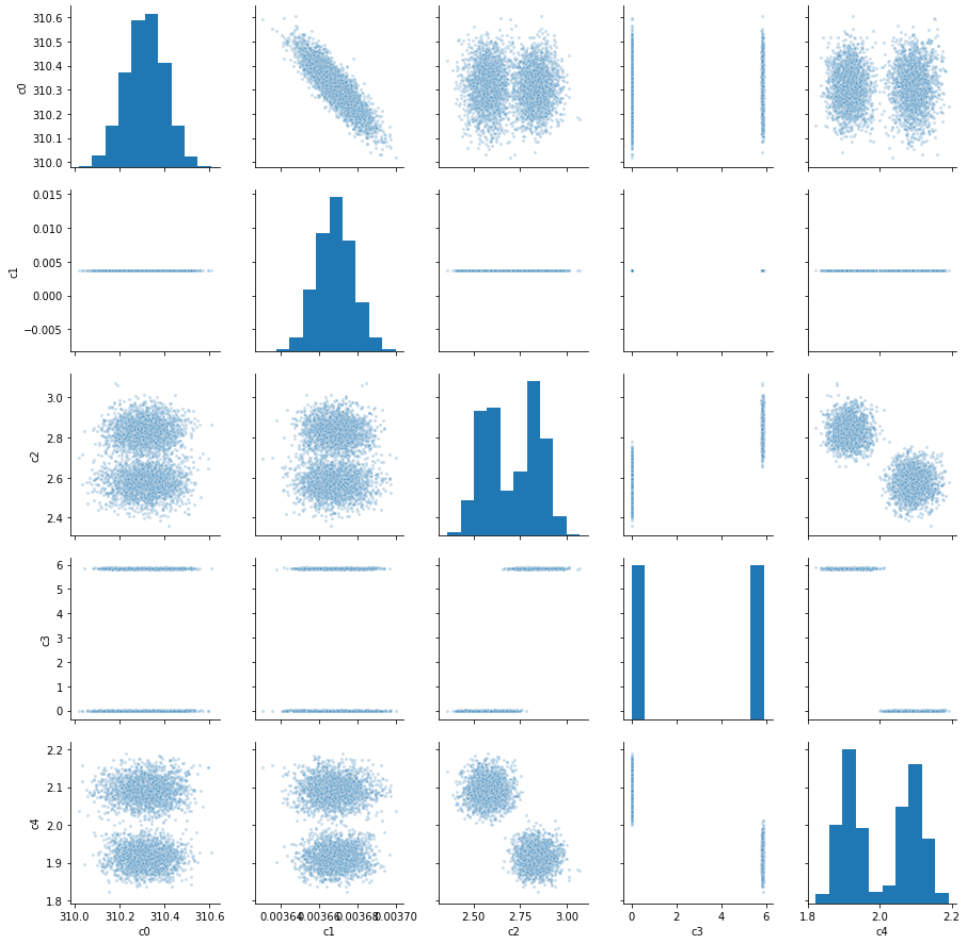


Fig 2: The pair plots visualize the relationship between the parameters plugging into the model.

- We can see that variable c_3 is causing the trouble as it is the clearest bimodal distributions. However, we realize that $\cos(x) = \cos(-x)$, which might be the reason for bimodal. As c_3 is

running freely and periodic in $[0,2\pi]$ Hence, we narrow the region for c_3 to be in $[0,\pi]$.

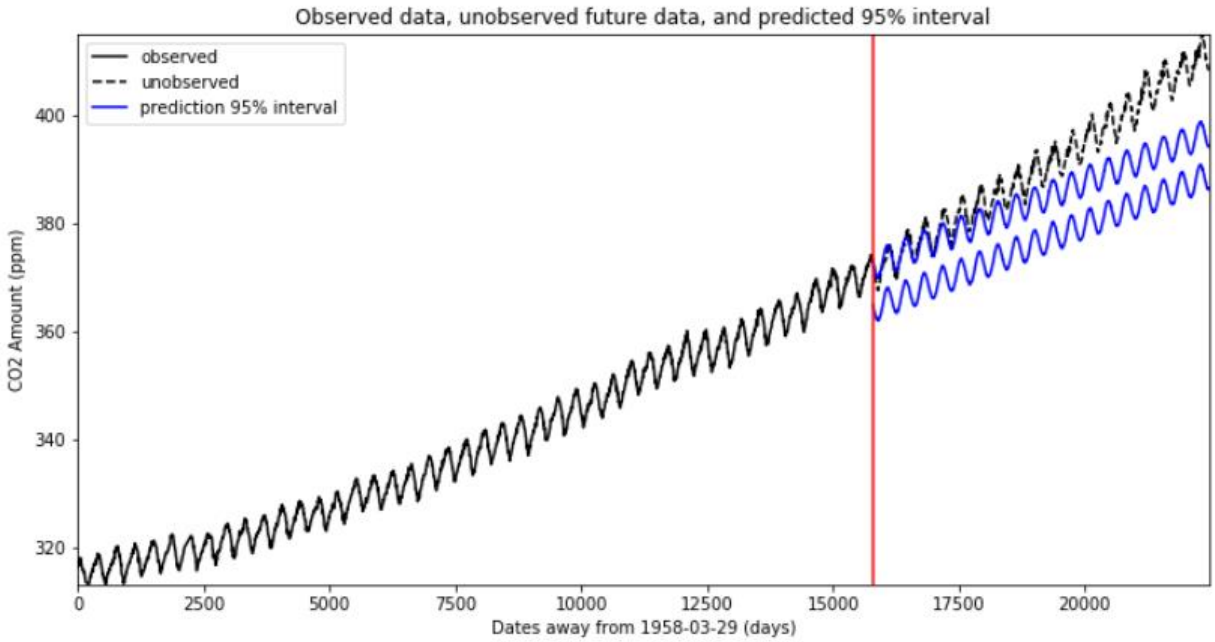


Fig 3: The observed data, unobserved data, and our predicted 95% interval.

Also, for the prediction, we can see that it is very far off, which might be because of the linearity. Hence, in the revised model, I will add a quadratic term.

Part 3: Model 2: Revision for the example model

$$p(x_t|\theta) = N\left(c_0 + c_1 t + c_5 t^2 + c_2 \cos\left(\frac{2\pi t}{365.25} + c_3\right), c_4^2\right)$$

- Changes:
  - Narrow $c_3 \in [0, \pi]$
  - Add quadratic term $c_5 t^2$
- Results: the model converges, with little correlated samples (based on the large n_eff values and Rhat values = 1).

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
c0	310.32	1.5e-3	0.09	310.13	310.25	310.32	310.38	310.5	3950	1.0
c1	3.7e-3	1.6e-7	1.0e-5	3.6e-3	3.7e-3	3.7e-3	3.7e-3	3.7e-3	3953	1.0
c2	2.57	1.3e-3	0.06	2.45	2.53	2.57	2.62	2.7	2263	1.0
c3	1.3e-3	2.4e-5	1.3e-3	2.3e-5	3.5e-4	8.5e-4	1.7e-3	4.9e-3	2935	1.0
c4	2.1	7.2e-4	0.03	2.03	2.07	2.09	2.12	2.16	2188	1.0

The prediction interval for unobserved data is:

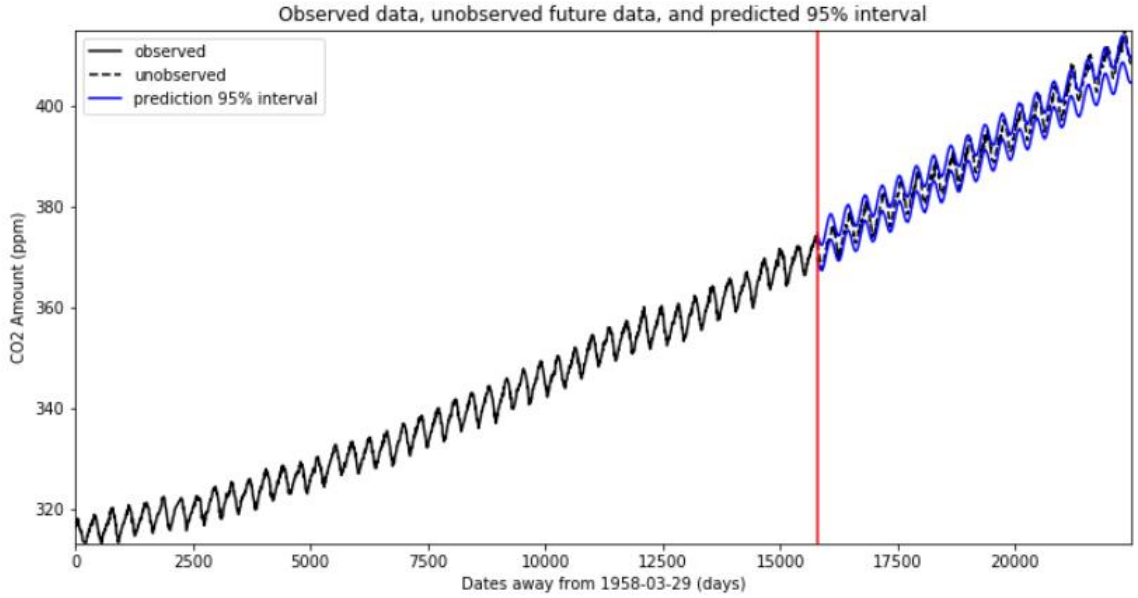


Fig 4: The observed data, unobserved data, and our predicted 95% interval

- We can see that this model fits the future unobserved data well; however, as the cosine function has a slightly curvy peak. When we look at the data, especially at the peaks, we observe that it is sharper than those of the predicted interval. Hence, I propose using the sawtooth wave function to recreate the sharp peaks.

Part 4: Model 3: Proposed model

$$p(x_t|\theta) = N\left(at^2 + bt + c + A\left(4 * \left|\left(\frac{t}{365.25} + \phi\right) mod\ 1 - 0.5\right| - 1\right), \sigma^2\right)$$

- Changes:
  - Using the sawtooth period function to achieve a sharp peak.
  - Modify a to be distributed by a log-normal prior. The value seen in the model will be log (a), denoted by "log_a."
- Results: the model converges, with little correlated samples (based on the large n_eff values and Rhat values = 1).

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
A	3.41	1.0e-3	0.04	3.34	3.38	3.41	3.44	3.48	1294	1.0
phi	0.93	2.8e-5	1.7e-3	0.93	0.93	0.93	0.93	0.94	3668	1.0
log_a	-16.23	3.6e-4	0.01	-16.25	-16.23	-16.23	-16.22	-16.2	1186	1.0
b	2.2e-3	5.5e-7	1.8e-5	2.2e-3	2.2e-3	2.2e-3	2.2e-3	2.3e-3	1114	1.0
c	314.2	1.9e-3	0.07	314.07	314.16	314.2	314.25	314.33	1151	1.0
sigma	0.98	3.0e-4	0.01	0.95	0.97	0.98	0.99	1.0	2412	1.0

- The prediction for future data is:
  (and the pair plots between the parameters)

  We can see that at the peaks, we have the sharp peaks as we desired, and the intervals seem a bit smaller compared to model 2 (the revised example model), which indicates a slightly better model fit.

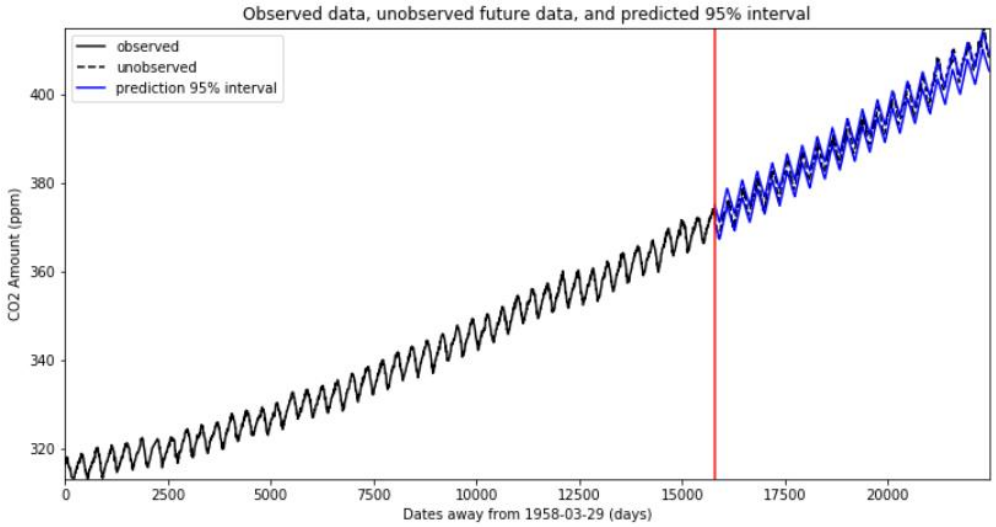


Fig 5: The observed data, unobserved data and our predicted 95% interval

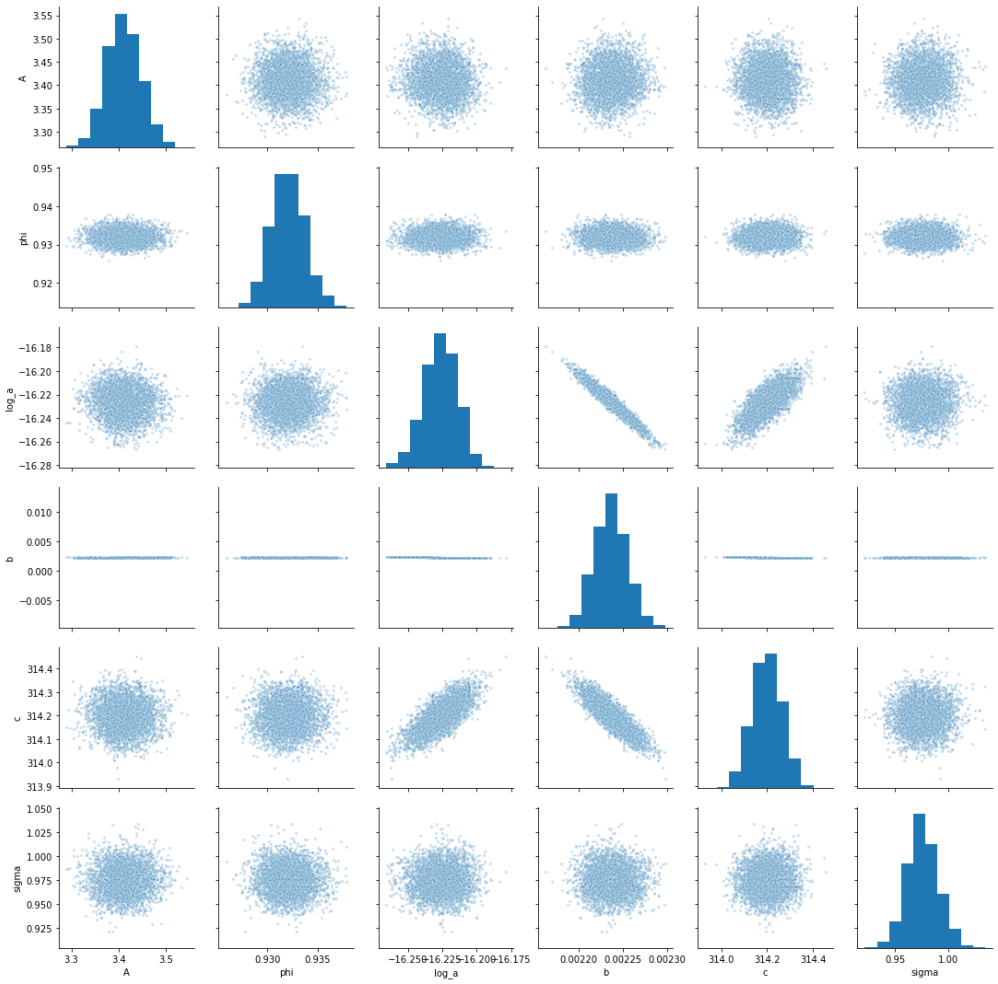


Fig 6: The pair plots visualize the relationship between the parameters plugging into the model.

Part 5: Model comparison:

Model 1 has many flaws: did not converge, and the linear component cannot describe the future data well.

Between model 2 (cosine periodic) and model 3 (sawtooth periodic):
Both models did an excellent job in predicting the future data: we see the prediction lines are very close to the unobserved data.

I prefer model 3 because:

- The shape of the peak of the data is sharp, which is more similar to the sawtooth than the cosine function.
- The prediction interval of model 3 is much narrower than of model 2: 100% of the intervals in model 3 is more restricted, which is a plus: indicates less uncertainty.
- Model 3 has lower predictive error: When we conduct mean squared errors on:
    - The mean of the samples at time t vs. the real value at time t:
      Model 3: error = 1.612156603044492
      Model 2: error = 2.1612224291754356
    - Every sample at the time t vs. the real value at time t:
      Model 3: error = 2.5857090803350777
      Model 2: error = 3.824635218251834

  Model 3 has more accurate predictions: model 3 both have a lower predictive error when comparing the real data with the mean of the samples or with all the samples values themselves.
  Furthermore, it has a much narrower interval (smaller Σ|97.5% interval – 2.5% interval|), which implies less uncertainty.
  Hence, model 3 will be the model we use for the remaining of this project.

As we decided to move forward with model 3, we rerun the model with 100% of the data we have. Here is the graph demonstrate the predicted CO_2 level in the next 40 years.

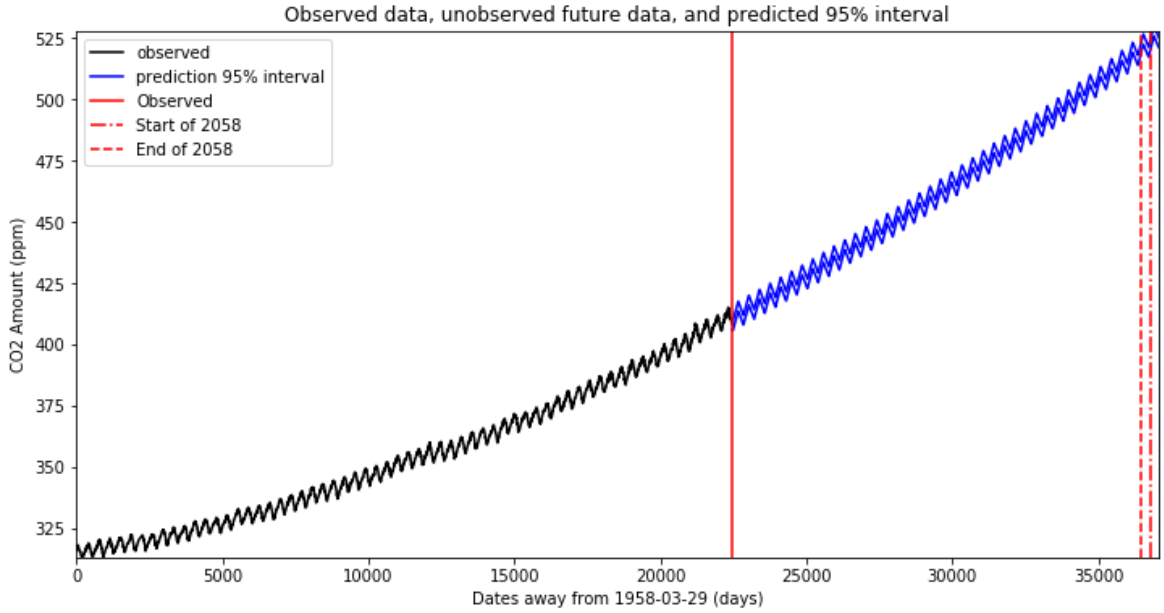


Fig 7: The observed data (current data) and our predicted 95% interval for future data.

Part 6: Predictions for the year 2058: the first measurement in 2058

Using our model, we have the predicted values for CO_2 Amount in the first measurement in 2058:

- With the mean at 519.5426720521242 (ppm)
- The median at 519.5288075969152 (ppm)
- Standard deviation at 1.073557430664101
- The 95% interval at: [517.4788034343148, 521.6107600881326]

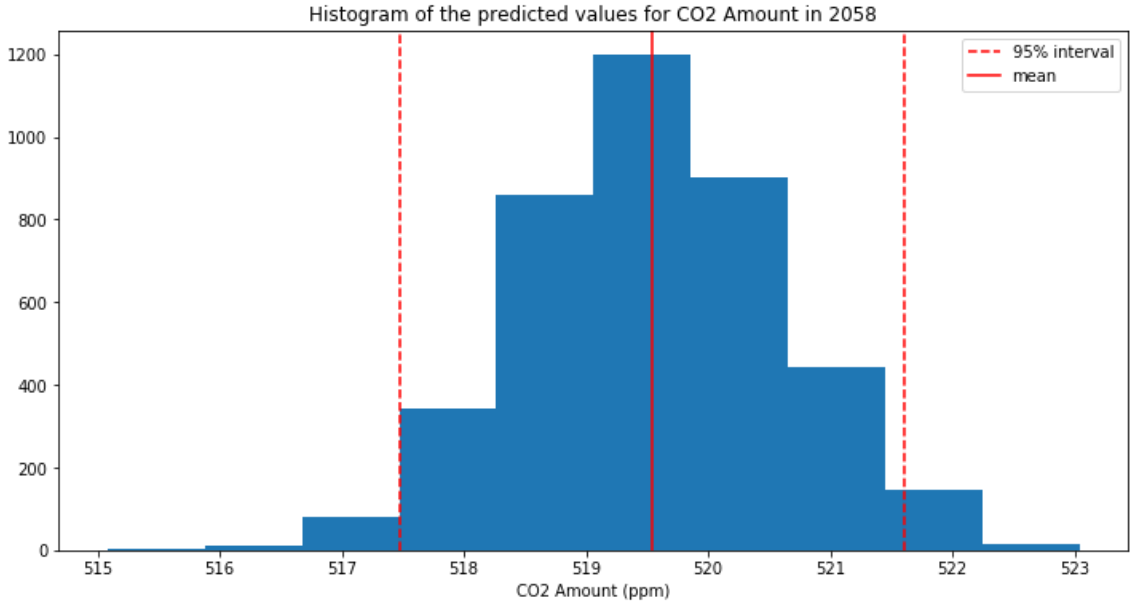


Fig 8: The histogram of the estimates of the CO_2 Amount in the 1st measurement in 2058

The graph here is drawn from 4000 samples data. There is a possibility that the samples are underestimating the values (which discussed later in part 8 and 9).

Part 7: The probability of CO_2 level reaching 450 ppm

Using the samples from our predicted values, we evaluate the probability of reaching 450 ppm by calculating among 4000 samples for a specific date, what proportion is more significant than 450 ppm.
These probabilities are dependent on the samples, which might not be fully reflective of future scenarios. Probability = 1 does not necessarily 100% happening; it only indicates that 100% of the samples satisfy the conditions.

We consider two thresholds: 0.5 and 1.0
The dates are only estimating and did not account for uncertainty. Uncertainty will be discussed in Part 8 &9 (shortcomings and recommendations).

- The first predicted date with the probability of reaching 450 ppm larger than 0.5 is "03-18-2034".
- The first date with the probability of reaching 450 ppm equal to 1.0 is "04-07-2035".
- The first date with the probability of everyday onward reaching 450 ppm larger than 0.5 is "11-17-2035".
- The first date with the probability of everyday onward reaching 450 ppm equal 1.0 is "11-29-2036".

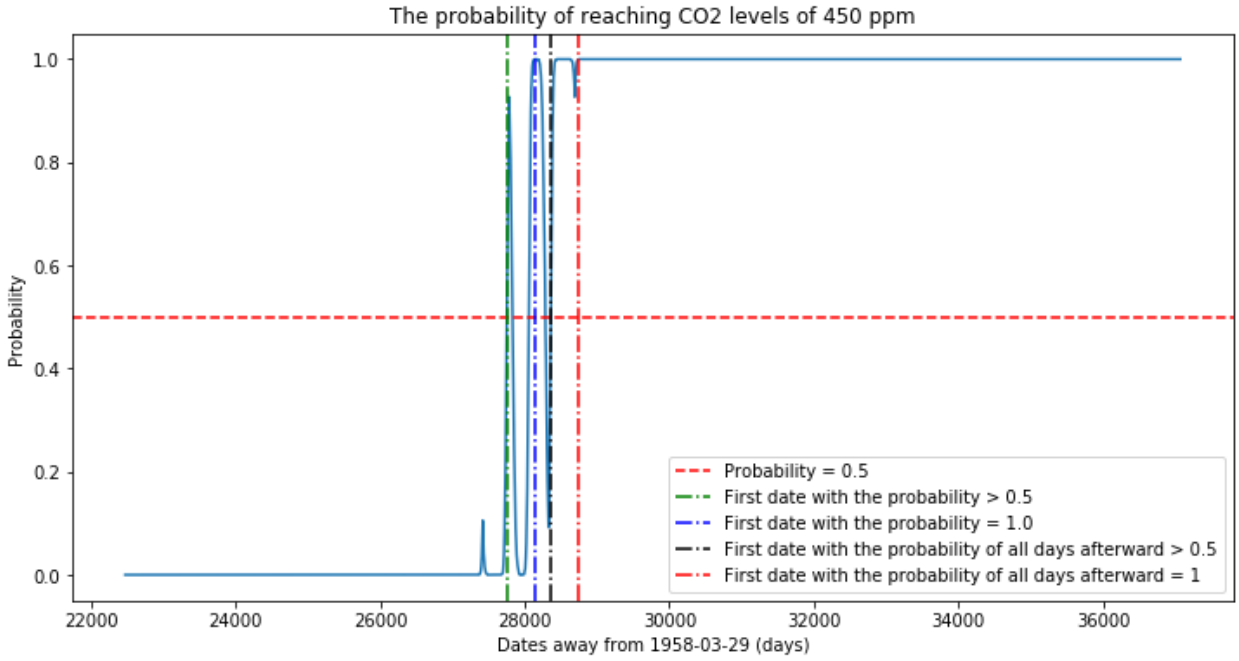


Fig 9: The probability that the estimates reaching levels of 450 ppm.

Part 8: Shortcomings:

Shortcomings

a. The process of designing the model
    i. Assumptions
    ii. Priors
    iii. Parameters
    iv. Observed vs. Unobserved variables
b. Test statistics

Part 8.a.: The process of designing the model

Part 8.a.i:

Assumption 1: Assuming the data has some periodic component by year (365.25). This assumption is reasonable because the year (contains the seasons), which directly affects the human. Also, this assumption helps the model because if the data is periodic by 1 year $\lambda = \frac{1}{365.25}$, it can be periodic by k years, with k is an integer ($\lambda = \frac{1}{365.25k}$), which might create divergences in stan sampling.

Assumption 2: Our model assumes that the increasing factor is quadratic. The reason is that we try to fit a polynomial factor as the increasing factor. The linear fails with model 1, which inspires us to use quadratic, which resulted in a very close fit model. We did not model higher-order polynomials or exponential because we suspect the growth rate is not that fast.

Assumption 3: We assume the noise factor is a Gaussian mean with a fixed standard deviation σ.

Assumption 4: All the priors' assumptions.

Part 8.a.ii. + 8.a.iii. + 8.a.iv.:

$A \sim N(0,5)$: the range of the periodic factor will be from $[-A, A]$. We choose $N(0,5)$ because the CO_2 measurements periodic fluctuation should not be too big. However, as we are unsure, we still keep a pretty big standard deviation.

For ϕ (the initial state), because we take modular 1, we need to rescale ϕ to [0,1] by $\frac{\arctan\frac{\phi_1}{\phi}}{2\pi} + 0.5$: $\arctan\frac{\phi_1}{\phi}$ will take values to $[-\pi, \pi]$, then the division and addition will transform this to [0,1].

For the quadratic component: b will be the main components for the increase, a will create the curve and increase when t gets larger, and c will represent the initial value at the beginning of the recording period. $at^2 + bt + x$

For a (quadratic term): we want a to be small (as the time component is huge), and the change of the CO_2 should not be too significant and center around a millionth. Hence, we model $\log(a) \sim N(-16, 6)$.

For b (linear term): we want b to be the primary increasing factor: $b \sim N(0,0.1)$.

For c (constant term): c is the initial state when we start recording the data. We use a broad term to demonstrate uncertainty and allow more freedom for the model. $c \sim N(300,100)$

σ (the noise): we are unsure, so we choose a broad range: $\sigma \sim Gamma(1, 0.3)$

As we believe the amount is increasing, we set the limit for all parameters to has lower $= 0$:
$A, \sigma, a, b, c \geq 0$

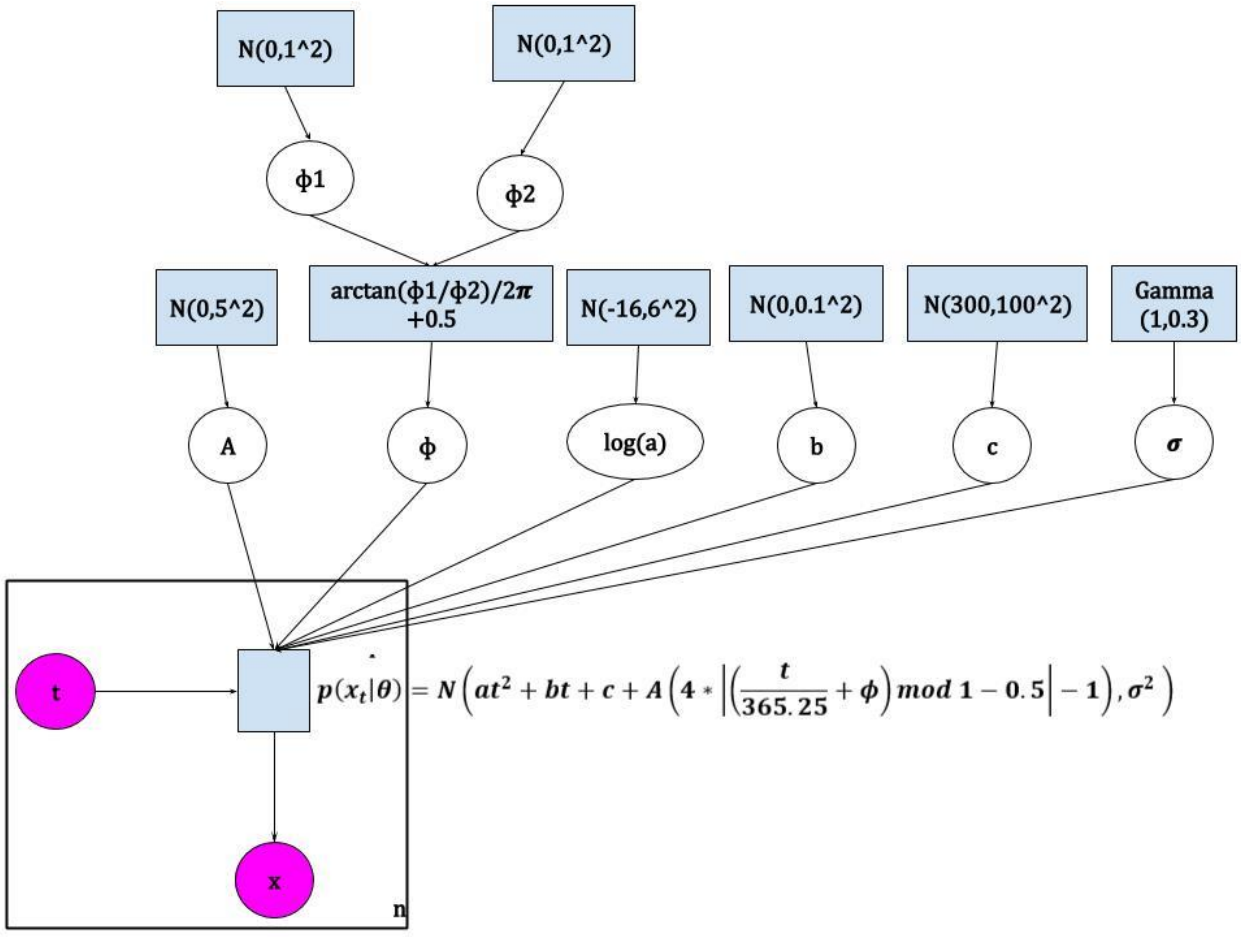


Fig 10: The factor graph of the sawtooth-quadratic model (model 3)

Part 8.b. test Statistics: As discussed above, we use the first 70% of the data (2197 values) as data for computing the posterior distributions over the parameters above using Stan. We use the other 30% of the data (the next 942 values) to test our prediction of future function values.

Our test statistic is the mean of future data: 942 last data points in the dataset.

The reason is that: we suspect the model, even though overall fit the future data very well (based on the plot), fails to predict the few anomalies in the original data fully. For example, when we zoom in the first 600 data points in the test future data, we can see at those peaks, our real data is slightly off compared to the predicted interval.

As the mean is sensitive to the outliers, we would expect the mean for the real future data to be in the upper or bottom percentile of the test statistics results of the sampled data.

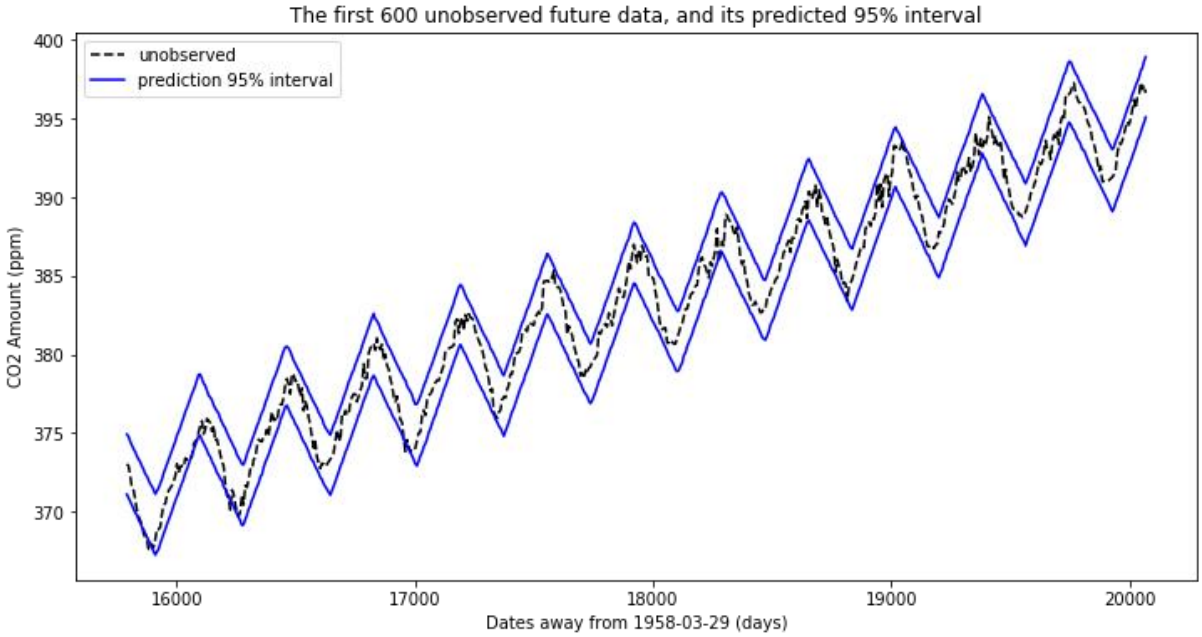


Fig 11: The zoom-in version for the first 600 datapoints to investigate the performance of the model

We generate 4000 sampled datasets (each dataset is an estimate for the 30% last unobserved data)

We then computed the mean of those 4000 datasets and compared to the mean of the real unobserved data.

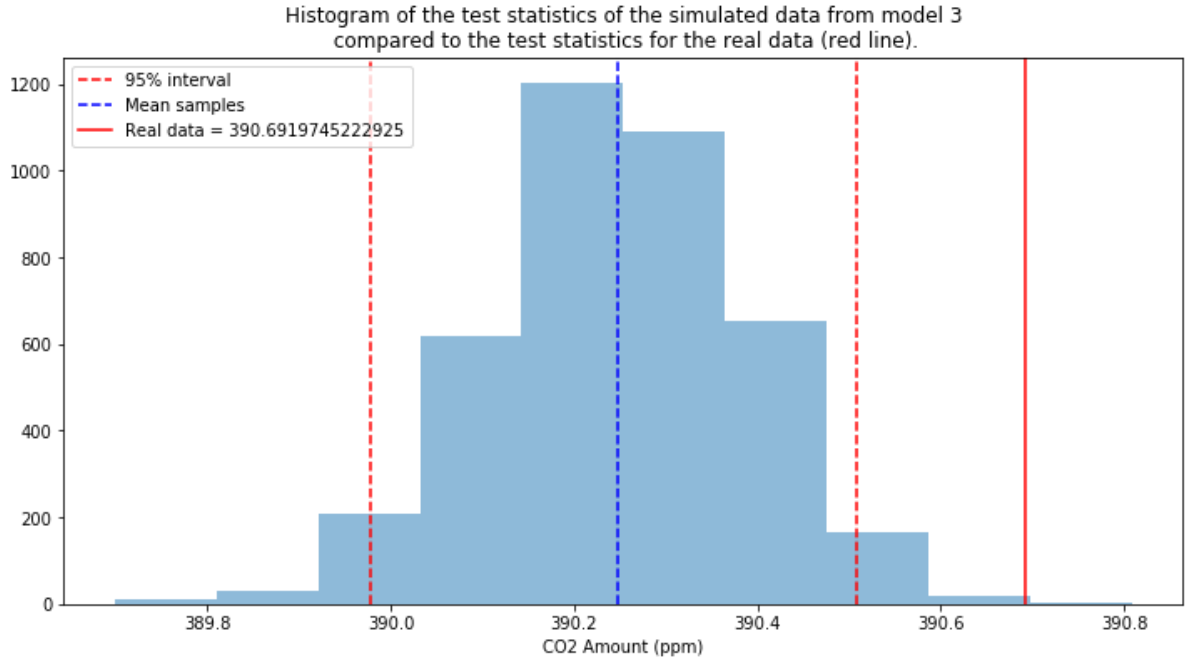


Fig 12: The histogram of the test statistics simulated from model 3.

The percentage that samples result larger than the real statistic value: 0.1 %

Hence, the p-value: 0.999: which is more significant than 0.95, meaning that the model failed to sufficiently predict the unobserved data with respect to this test statistics.

Looking at the histogram, we can infer that the predictive results by our model tend to underestimate the real data. Model 2 faced the same failure in this test statistic (p-value = 0.9928).

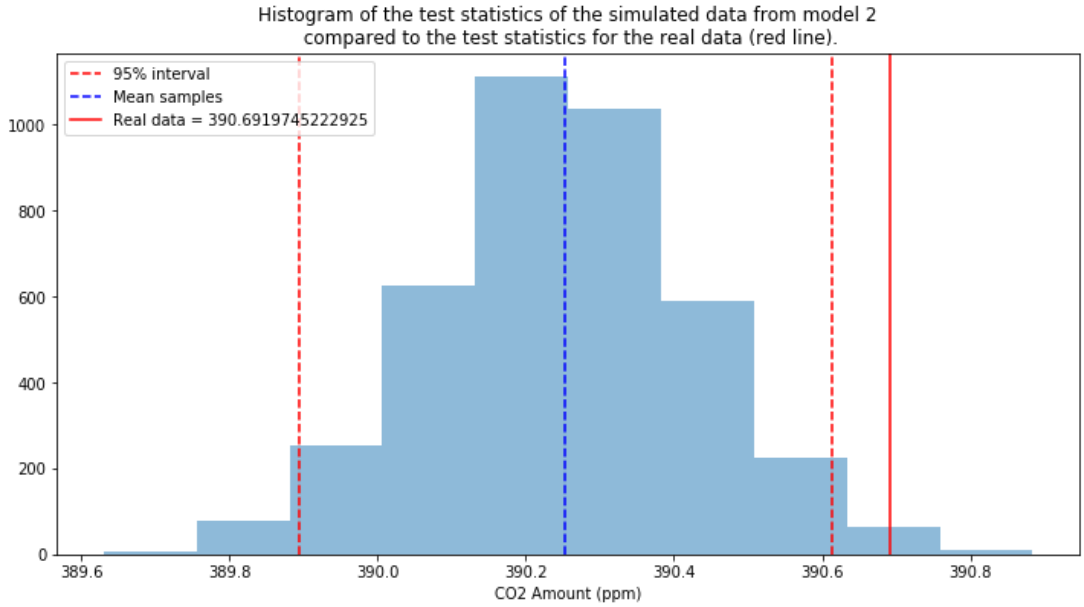


Fig 12: The histogram of the test statistics simulated from model 3.

To investigate further what might be the reason, we measure the errors overtime:
(we did not plot the 95% interval because it would look too clustery)

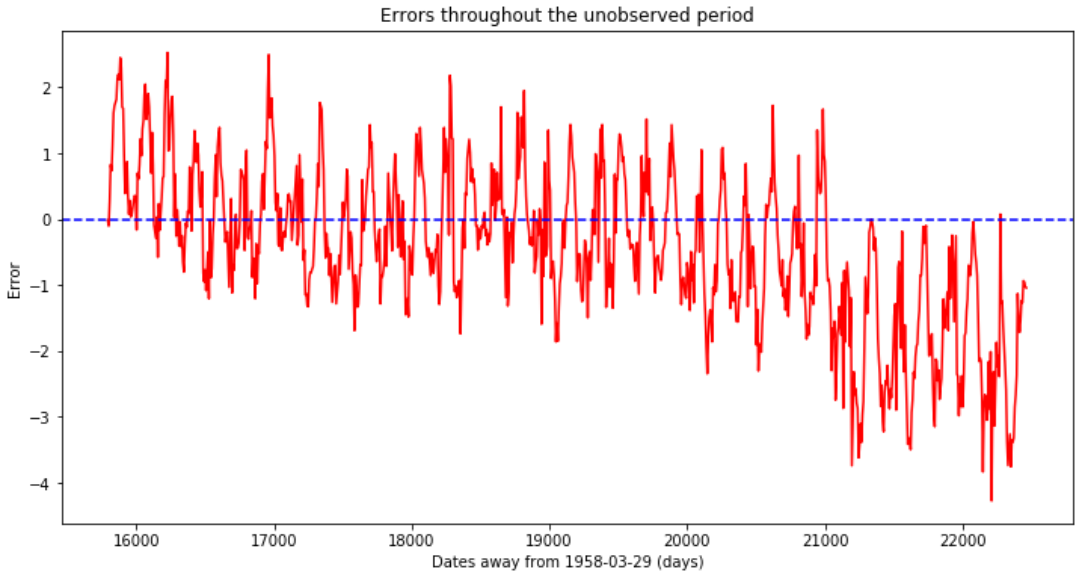


Fig 13: The average errors throughout the unobserved period.

As we can see, the error starts to underestimate the real values as time goes further. This could mean that our quadratic component did not increase fast enough to fit the data. We could try an exponential or a higher-order polynomial fit. However, for the scope of this project, and with the limited data, I would not add more parameters, which might overfit the data.

This graph would also raise concerns regarding our estimates for 2058 data and high-risk CO_2 levels, as those both are far in the future. Hence, we might underestimate even more for those values (2058 could have a higher CO_2 level, while the date with high probability for high-risk CO_2 could come faster). This will be further discussed in Part 9.

The set of the errors between all samples and the real data is very broad: with the median at -0.4 and the 95% interval at [-3.63686949, 2.46959212], and especially getting more negative while *t* gets larger.

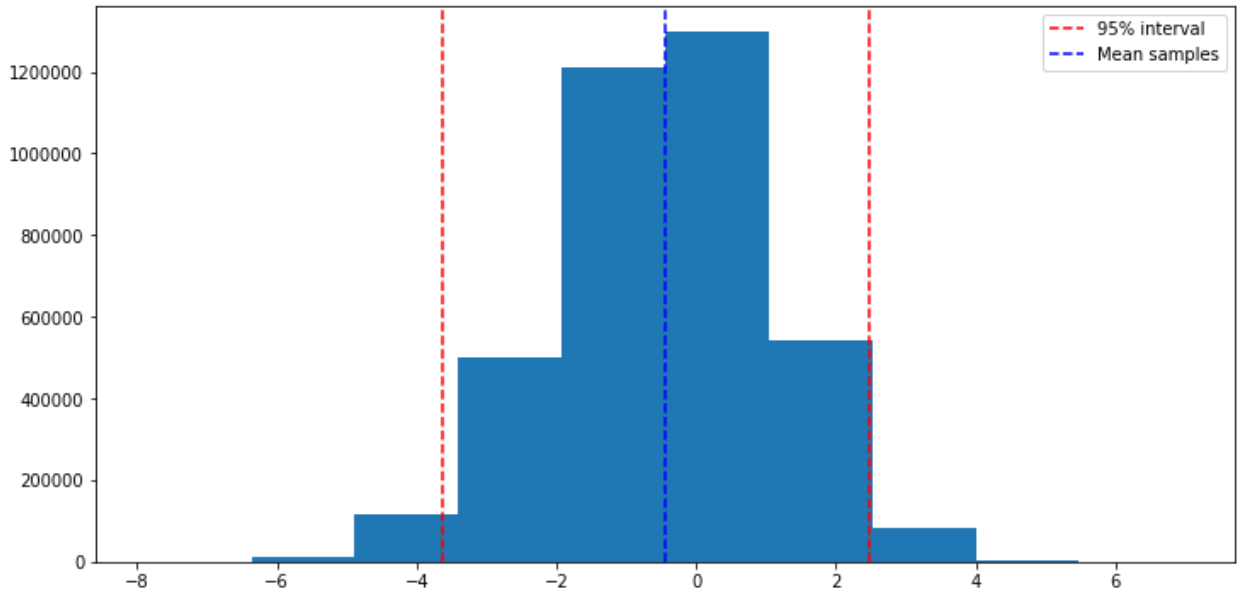


Fig 14: The histogram of all errors throughout the unobserved period.

Part 9: Discussion and Recommendations

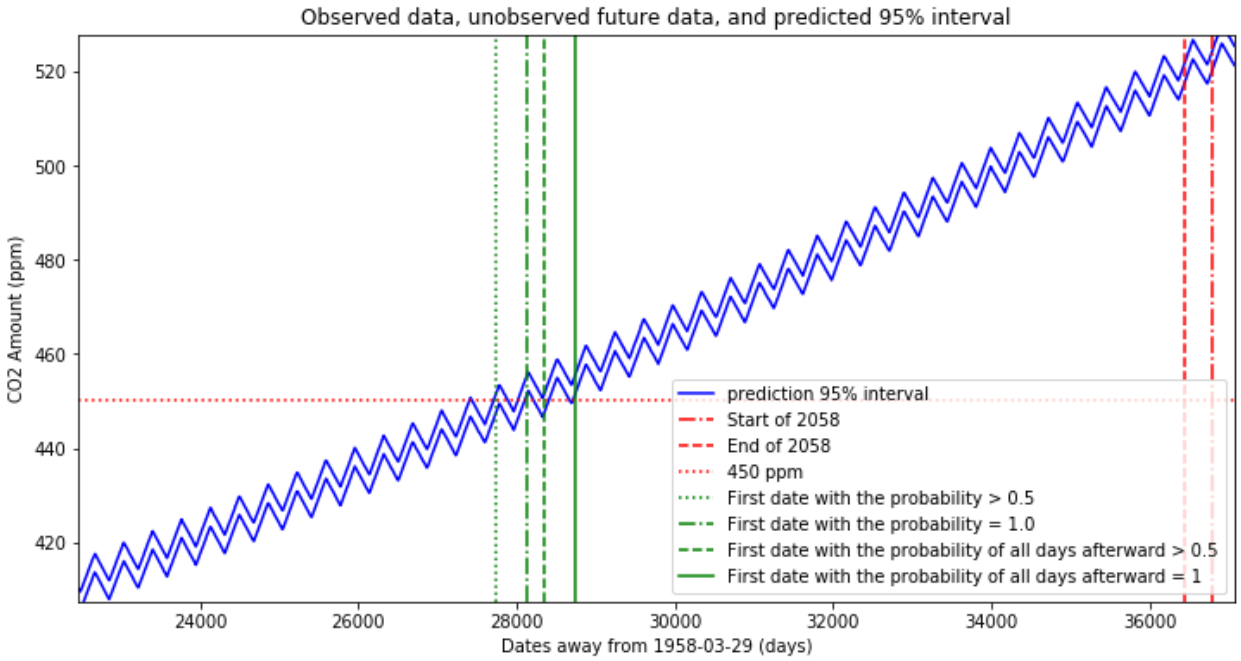


Fig 15: The plot of important components of this project

For the CO_2 modeling, we create a model with an increasing quadratic trend, a sawtooth component to represent seasonal factors and a Gaussian noise center at 0 with standard deviation σ.

The model fit the data quite well, with an average 1.6 ppm error when we subset the data into 70% train and 30% to validate. Overall, the model has a very closed estimation of the real data.

Notable drawback: the model tends to underestimate the real data as time gets further from the initial stage. Hence, the prediction for the year 2058 could be an underestimate (the values could be higher than what the model suggested). Also, the forecast for the high-risk CO_2 level could be earlier than what the model indicated.

Recommendations:

- For the high-risk CO_2 level (part 7), each probability is determined by 4000 samples from Stan sampling. If we want to have uncertainty for the probability estimate, I suggest running the Stan sampling multiple times to include uncertainties and a more accurate date range.
- Account for the possible underestimates while doing estimates: the current error is around -0.4 (the median of the errors set), with a confidence interval [-3.6, 2.5], with the errors getting more negative when t get larger.
- With the error sets data (972 time-values, each has 4000 error data points), try to figure a model to explain the systematic error (as it seems to be a systematic decrease).
- Or try fitting the model with either exponential or higher-order polynomials for the trend component to check if a more accurate model could be found.
- Save the environment!

HCs Applications

#communicationdesign: For this final project, I focus on demonstrating events and information through multiple data visualizations, as graphs are more intuitive to understand and interpret, both for statisticians and the general audience. This project is a good use of #communicationdesign as I take into consideration the audience and the representations of the deliverable. Also, I adhere to the cognitive principles of the communications design: reduce cognitive workload by using figures, each figure focuses on delivering one result, and all figures are clearly labeled and explained.

#testability: For all models, I focus on validating the advantage and disadvantages. I subset the data into train and validate data and conduct appropriate tests to analyze and compare with other models on different metrics (predictive error, the width of confidence intervals). Also, I design a test statistic that addresses the possible shortcomings of the models (the mean), which addresses the underestimation when t becomes large. This is a good use of #testability because I design testable models and test them thoroughly.

#casestudy: Using the data for this specific case: CO_2 level recorded at Mauna Loa Observatory, I try to model and extract as labeled insights as possible on the behaviors of the CO_2 level. I identify the suitable components to explain the data, describe the limitations, and provide recommendations to extend the study. This is a good use of #casestudy as I utilize a set of data on many aspects and infer trends when applicable

#selfawareness: While doing this project, I acknowledge that I might be subject to Dunning-Kruger effect (unable to recognize my incompetence). Hence, I try to start the project early and go through multiple stages of questions and feedback with Professor. Scheffler, which helps mitigate my weaknesses. This strategy works exceptionally well with me as I tend to procrastinate, which, through trying to discuss in Office Hour, helps me to motivate myself to work harder to have a more insightful conversation.

References:

Sampling Station Records - Scripps CO2 Program - Mauna Loa Observatory, Hawaii. (n.d.). Retrieved December 2019, from https://scrippsco2.ucsd.edu/data/atmospheric_co2/mlo.html.

```
# import packages

import pystan
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import pandas as pd
import seaborn
import datetime
from datetime import timedelta
%matplotlib inline
np.random.seed(2019)
```

Data Processing

```
data = pd.read_csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vTX5nZNL-okv1cEdQ00ue57BEI6ARhDZwMJWiUyD4xupTcIx0aEI75UTxp8P9zx
```

```
data.columns = ["Date", "Amount"]
```

```
max(data["Date"])
```

```
'2019-09-28'
```

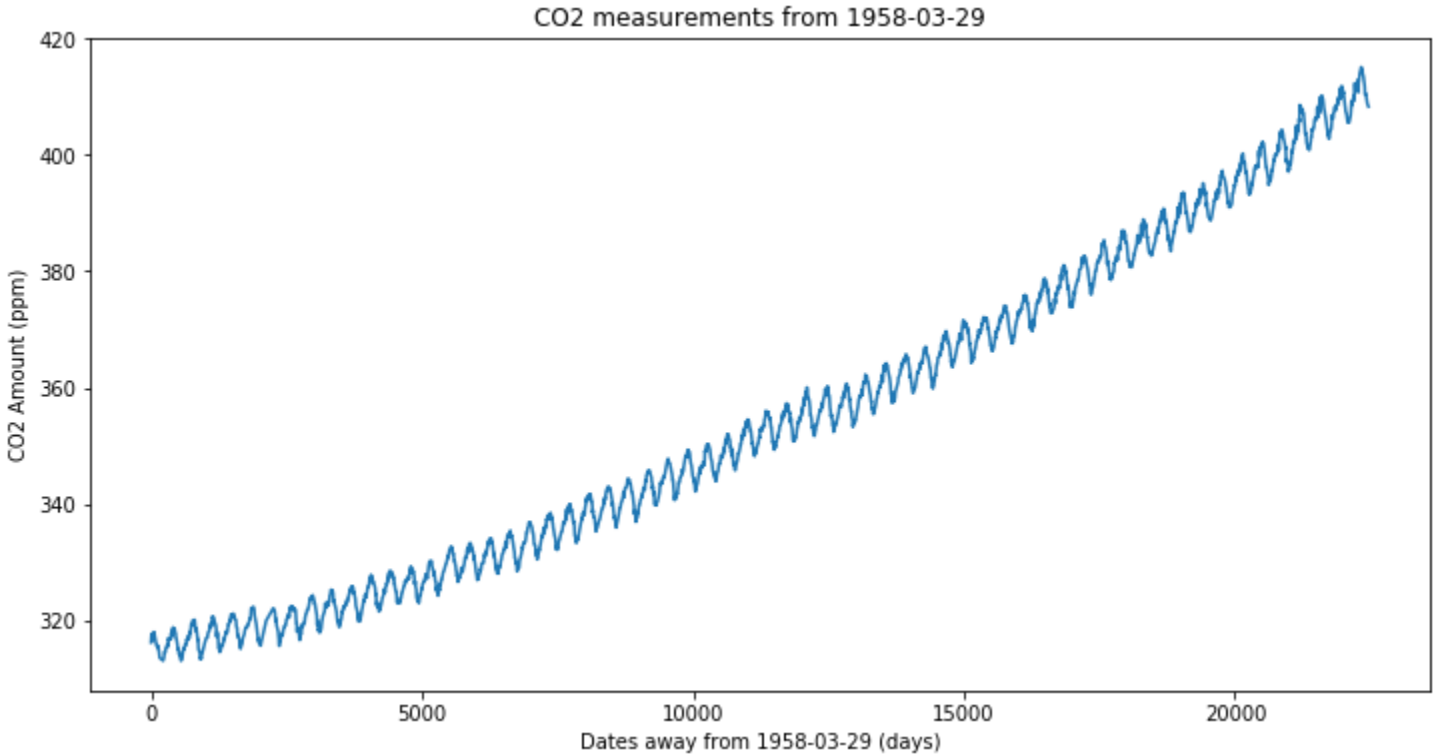
```
# subtract the date values from the first date
time = []
for i in range(len(data)):
  time.append((datetime.datetime.strptime(data["Date"][i], "%Y-%m-%d") - datetime.datetime.strptime(data["Date"][0], "%Y-%m-%d")).d
```

```
data["Time"] = time
```

```
data.head()
```

	Date	Amount	Time
0	1958-03-29	316.19	0
1	1958-04-05	317.31	7
2	1958-04-12	317.69	14
3	1958-04-19	317.58	21
4	1958-04-26	316.48	28

```
plt.figure(figsize = (12,6))
plt.title('CO2 measurements from 1958-03-29')
plt.plot(data["Time"], data["Amount"], label = "Real data")
plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("CO2 Amount (ppm)")
plt.show()
```



Simple model: Linear + cos periodic

```
# The Stan model. Running this cell compiles the Stan model, which takes
# some time.
  # Stan gets to see the first n data values only

stan_code = """
data {
    int<lower=0> n;        // The number of data
    int<lower=0> n_future; // the number of predictions
    real x[n];             // The data
    real t[n];             // time data
    real t_future[n_future]; // future time data
}
```

```
parameters {
    real<lower = 0> c0;
    real<lower = 0> c1;
    real<lower = 0> c2;
    real<lower = 0> c3;
    real<lower = 0> c4;
}

model {
    for(i in 1:n) {
        x[i] ~ normal(
            c0 + c1*t[i] + c2*cos(2*pi()*t[i]/365.25 + c3),
            c4);
    }
}

// Generate the predicted function values for the next n_future steps.
generated quantities {
    real x_future[n_future];
    for(i in 1:n_future) {
        x_future[i] = normal_rng(
            c0 + c1*t_future[i] + c2*cos(2*pi()*t_future[i]/365.25 + c3),
            c4);
    }
}
"""

stan_model = pystan.StanModel(model_code=stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_ebe2c6f5518b031d8f01ff4ffba65ddc NOW.
```

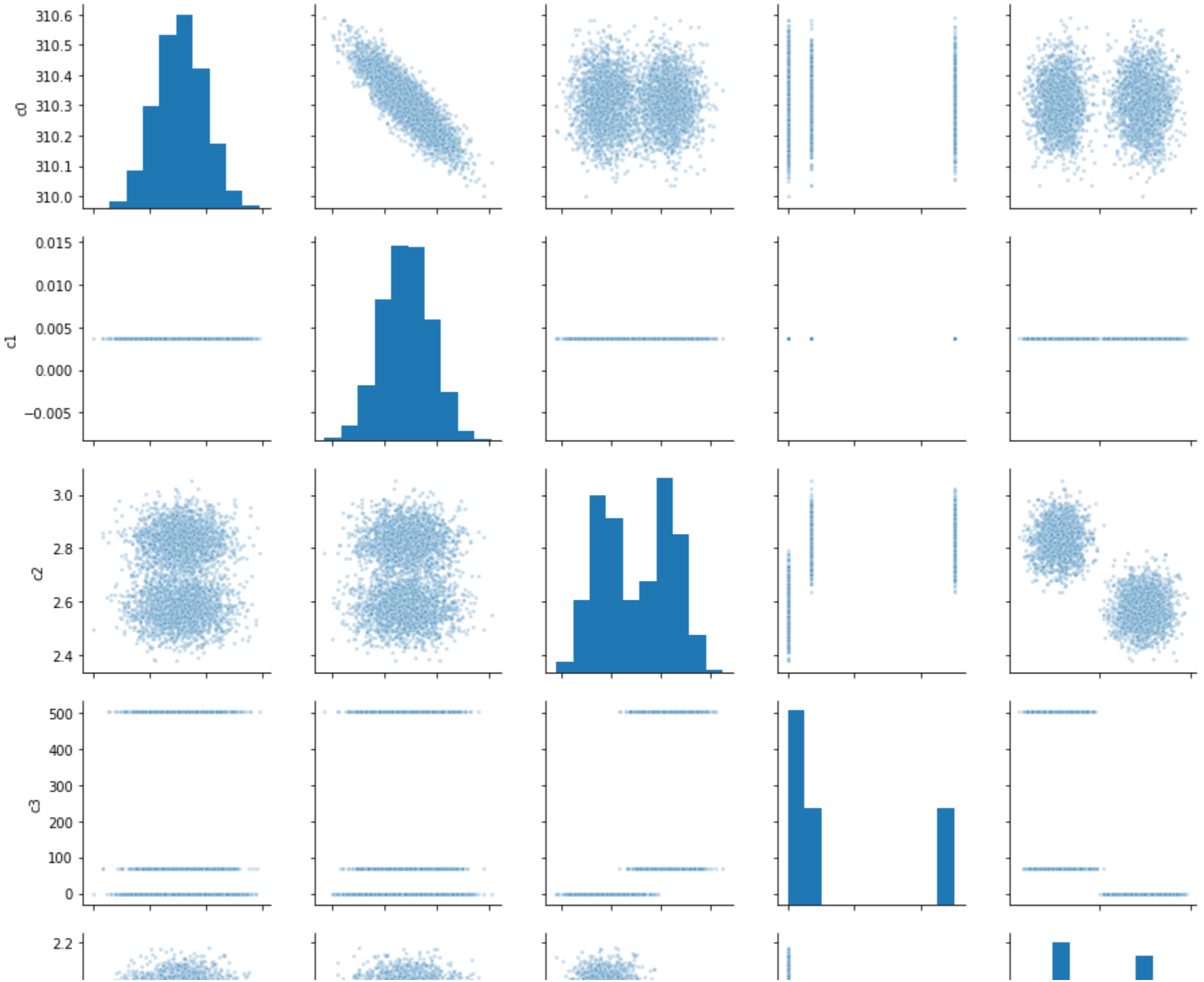
```
# subsetting the data
n = len(data)*7//10
n_future = len(data) - n

# stan data
stan_data = {
    'n': n,
```

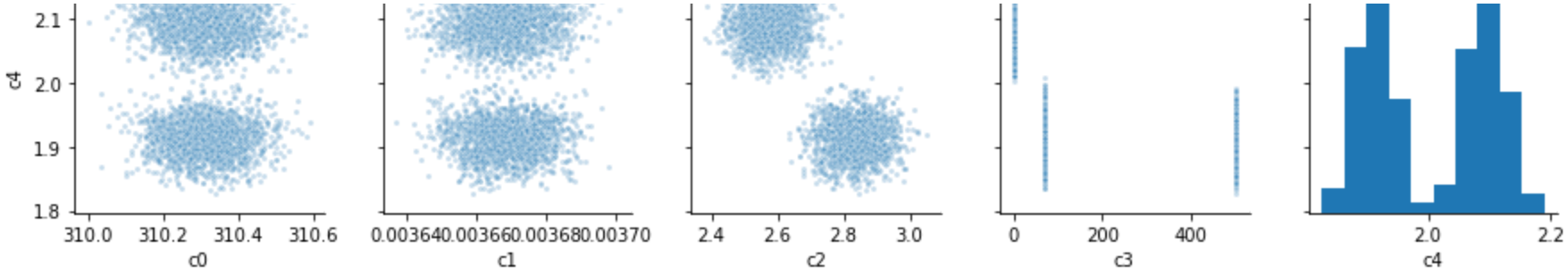
```
    'n_future': n_future,
    'x': data["Amount"][:n],
    't': data["Time"][:n],
    "t_future" : data["Time"][n:]}

# Run Hamiltonian Monte Carlo using Stan with 4 Markov chains, a 1000 step
# warm-up phase and 1000 step sampling phase for each chain. The warm-up samples
# are discarded, so we are left with 4 x 1000 = 4000 samples.

# This cell will take a minute or two to run.

parameters = ['c0', 'c1', 'c2', 'c3', 'c4']

results = stan_model.sampling(data=stan_data)
print(results.stansummary(pars=parameters))
samples = results.extract()
```

```
WARNING:pystan:n_eff / iter below 0.001 indicates that the effective sample size has likely been overestimated
WARNING:pystan:Rhat above 1.1 or below 0.9 indicates that the chains very likely have not mixed
Inference for Stan model: anon_model_ebe2c6f5518b031d8f01ff4ffba65ddc.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

     mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
c0 310.31  1.4e-3   0.09 310.15 310.25 310.31 310.37 310.48   3822    1.0
c1 3.7e-3  1.6e-7 9.3e-6 3.7e-3 3.7e-3 3.7e-3 3.7e-3 3.7e-3   3546    1.0
c2    2.7    0.09   0.14   2.47   2.58    2.7   2.83   2.94      2   2.39
c3 142.72  148.08 209.46 4.9e-5 7.9e-4  34.31 307.12 502.24      2  1.6e4
c4    2.0    0.07    0.1   1.87   1.91   2.01   2.09   2.15      2   3.39

Samples were drawn using NUTS at Wed Dec 18 13:51:30 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
# plot the pair plot

df = pd.DataFrame(
    data=np.transpose([samples[param] for param in parameters]),
    columns=parameters)
```

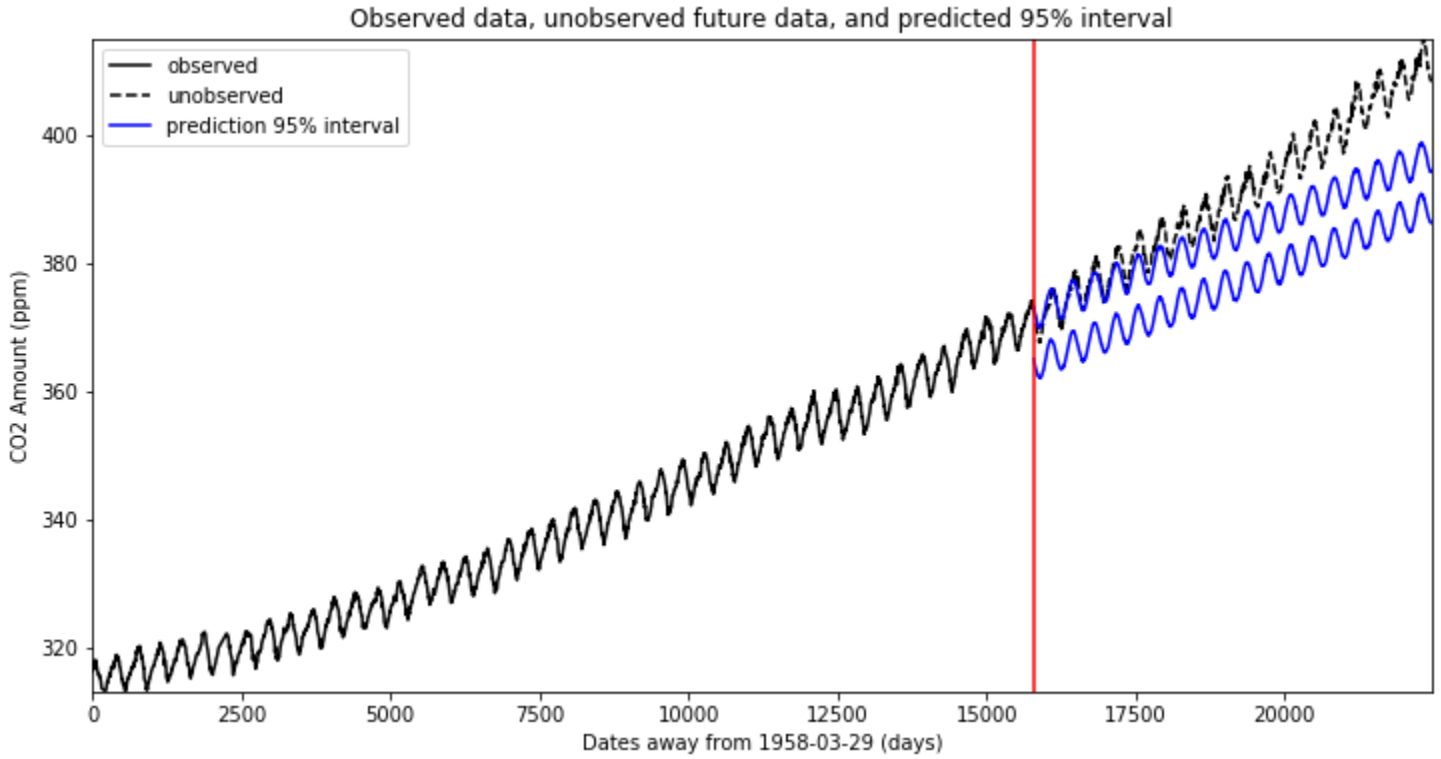
```
seaborn.pairplot(df, height=2.5, plot_kws={'marker': '.', 'alpha': 0.25})
plt.show()
```


c0
310.6
310.5
310.4
310.3
310.2
310.1
310.0
c1
0.015
0.010
0.005
0.000
−0.005
c2
3.0
2.8
2.6
2.4
c3
500
400
300
200
100
0
2.2




```
# draw predictions
prediction = samples['x_future']

# Compute 95% interval of the predicted values
prediction_interval = np.percentile(prediction, axis=0, q=[2.5, 97.5])

# Plot mean and 95% interval of predictions
plt.figure(figsize=(12, 6))
# observed data
plt.plot(data["Time"][:n], data["Amount"][:n], 'k-', label='observed')
#unobserved data
plt.plot(data["Time"][n:n+n_future], data["Amount"][n:n+n_future], 'k--', label='unobserved')

# set the grid
plt.xlim(0, data["Time"][n+n_future-1])
plt.ylim(
    min(min(prediction[0,:]), min(data["Amount"])),
    max(max(prediction[1,:]), max(data["Amount"])))

# confidence interval
plt.plot(
    data["Time"][n:],
    prediction_interval[0,:],
    'b-', label='prediction 95% interval')
plt.plot(
    data["Time"][n:],
    prediction_interval[1,:],
    'b-')
```

```
29
30 plt.axvline(data["Time"][n - 1], color='red')
31 plt.xlabel("Dates away from 1958-03-29 (days)")
32 plt.ylabel("CO2 Amount (ppm)")
33 plt.legend()
34 plt.title('Observed data, unobserved future data, and predicted 95% interval')
35 plt.show()
```




```
1
```

Revised model: quadratic + cos periodic

```
1 # The Stan model. Running this cell compiles the Stan model, which takes
2 # some time.
3   # Stan gets to see the first n data values only
4
```

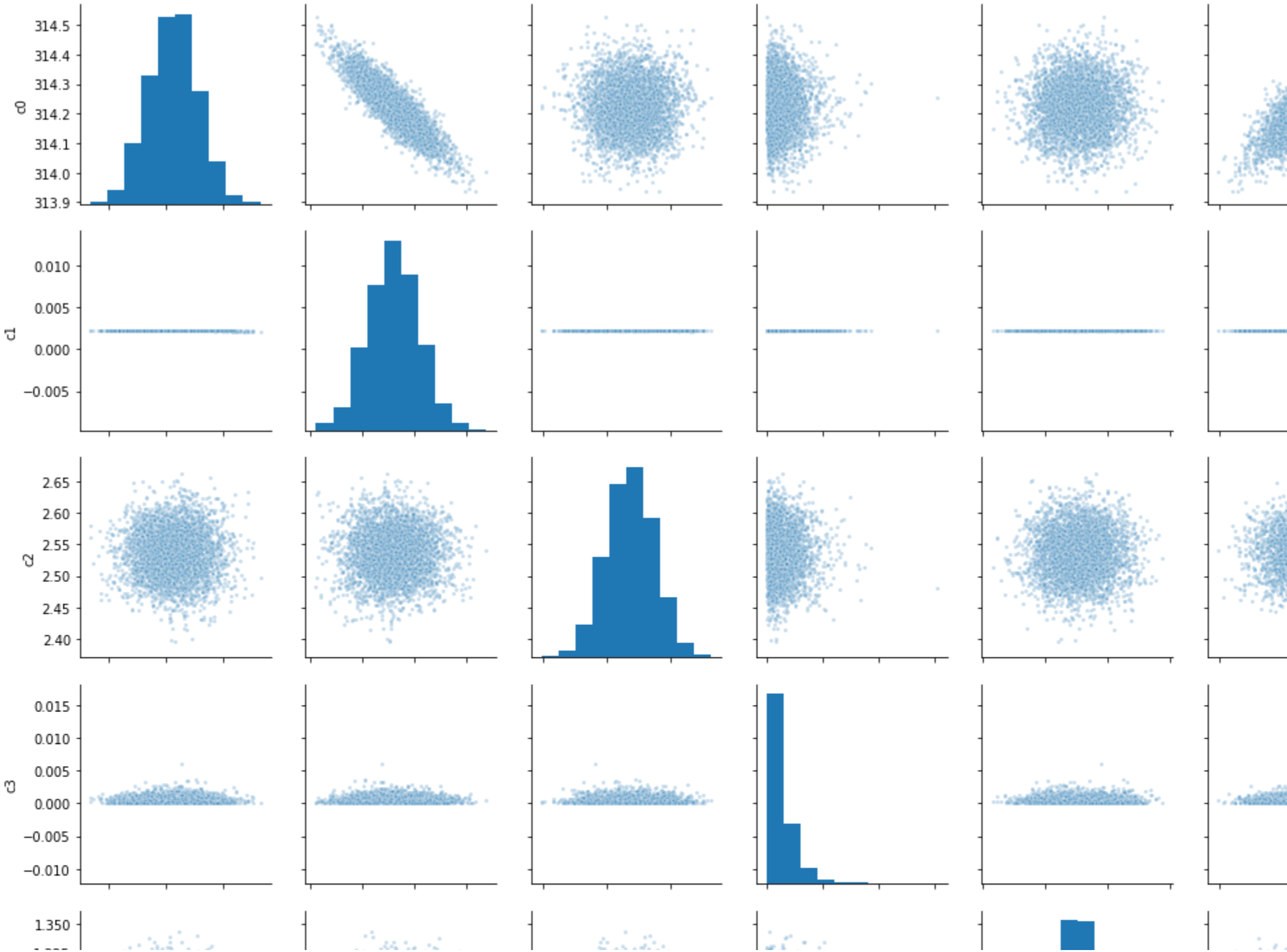
```
stan_code = """
data {
    int<lower=0> n;        // The number of data
    int<lower=0> n_future; // the number of predictions
    real x[n];             // The data
    real t[n];             // time data
    real t_future[n_future]; // future time data
}

parameters {
    real<lower = 0> c0;
    real<lower = 0> c1;
    real<lower = 0> c2;
    real<lower = 0, upper = pi()> c3;
    real<lower = 0> c4;
    real<lower = 0> c5;
}

model {
    for(i in 1:n) {
        x[i] ~ normal(
            c0 + c1*t[i] + c5*t[i]*t[i] + c2*cos(2*pi()*t[i]/365.25 + c3),
            c4);
    }
}

// Generate the predicted function values for the next n_future steps.
generated quantities {
    real x_future[n_future];
    for(i in 1:n_future) {
        x_future[i] = normal_rng(
            c0 + c1*t_future[i] + c5*t_future[i]*t_future[i] + c2*cos(2*pi()*t_future[i]/365.25 + c3),
            c4);
    }
}
"""

stan_model = pystan.StanModel(model_code=stan_code)
```

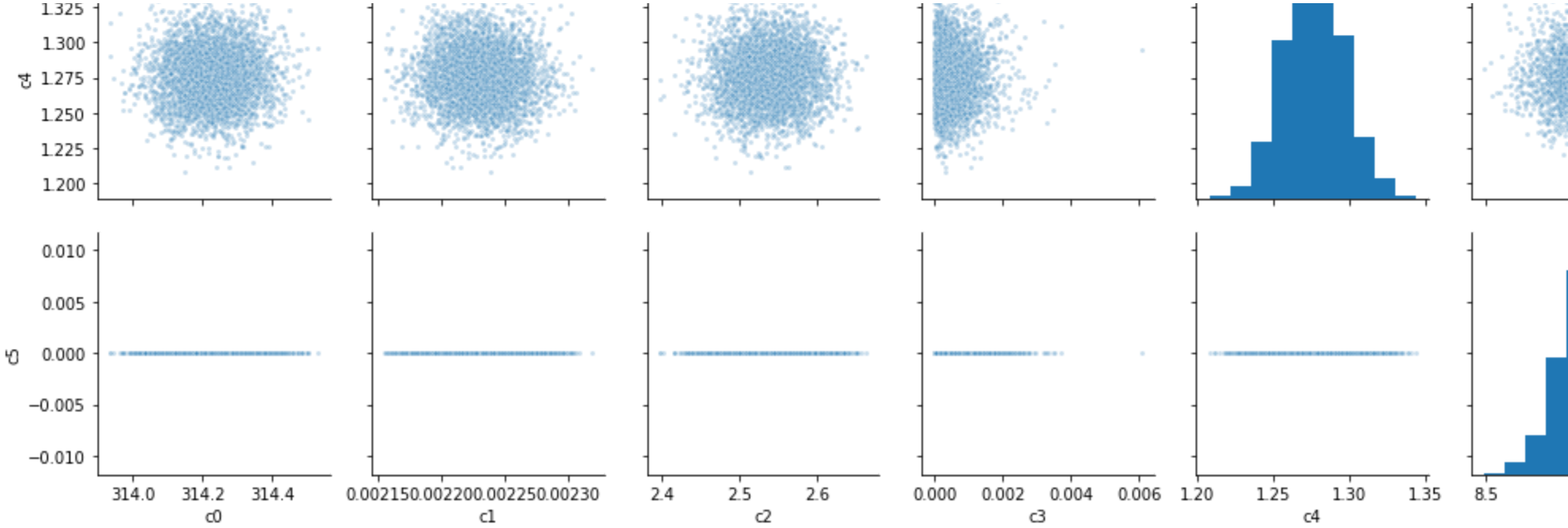
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_84f7f23059da5ade2b15b1965e6fdfad NOW.
```

```
# subsetting the data
n = len(data)*7//10
n_future = len(data) - n

stan_data = {
    'n': n,
    'n_future': n_future,
    'x': data["Amount"][:n],
    't': data["Time"][:n],
    "t_future" : data["Time"][n:]}

# Run Hamiltonian Monte Carlo using Stan with 4 Markov chains, a 1000 step
# warm-up phase and 1000 step sampling phase for each chain. The warm-up samples
# are discarded, so we are left with 4 x 1000 = 4000 samples.

# This cell will take a minute or two to run.

parameters = ['c0', 'c1', 'c2', 'c3', 'c4', 'c5']

results = stan_model.sampling(data=stan_data)
print(results.stansummary(pars=parameters))
samples = results.extract()
```

```
Inference for Stan model: anon_model_84f7f23059da5ade2b15b1965e6fdfad.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

     mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
c0 314.23  1.8e-3   0.09 314.06 314.17 314.23 314.28 314.39   2307    1.0
c1 2.2e-3  5.8e-7 2.4e-5 2.2e-3 2.2e-3 2.2e-3 2.2e-3 2.3e-3   1763    1.0
c2   2.54  6.7e-4   0.04   2.46   2.51   2.54   2.56   2.61   3226    1.0
c3 5.0e-4  8.4e-6 4.7e-4 1.3e-5 1.6e-4 3.6e-4 6.8e-4 1.7e-3   3097    1.0
c4   1.28  3.7e-4   0.02   1.24   1.26   1.28   1.29   1.32   2875    1.0
c5 9.0e-8 3.5e-11 1.5e-9 8.7e-8 8.9e-8 9.0e-8 9.1e-8 9.3e-8   1715    1.0

Samples were drawn using NUTS at Wed Dec 18 13:59:27 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

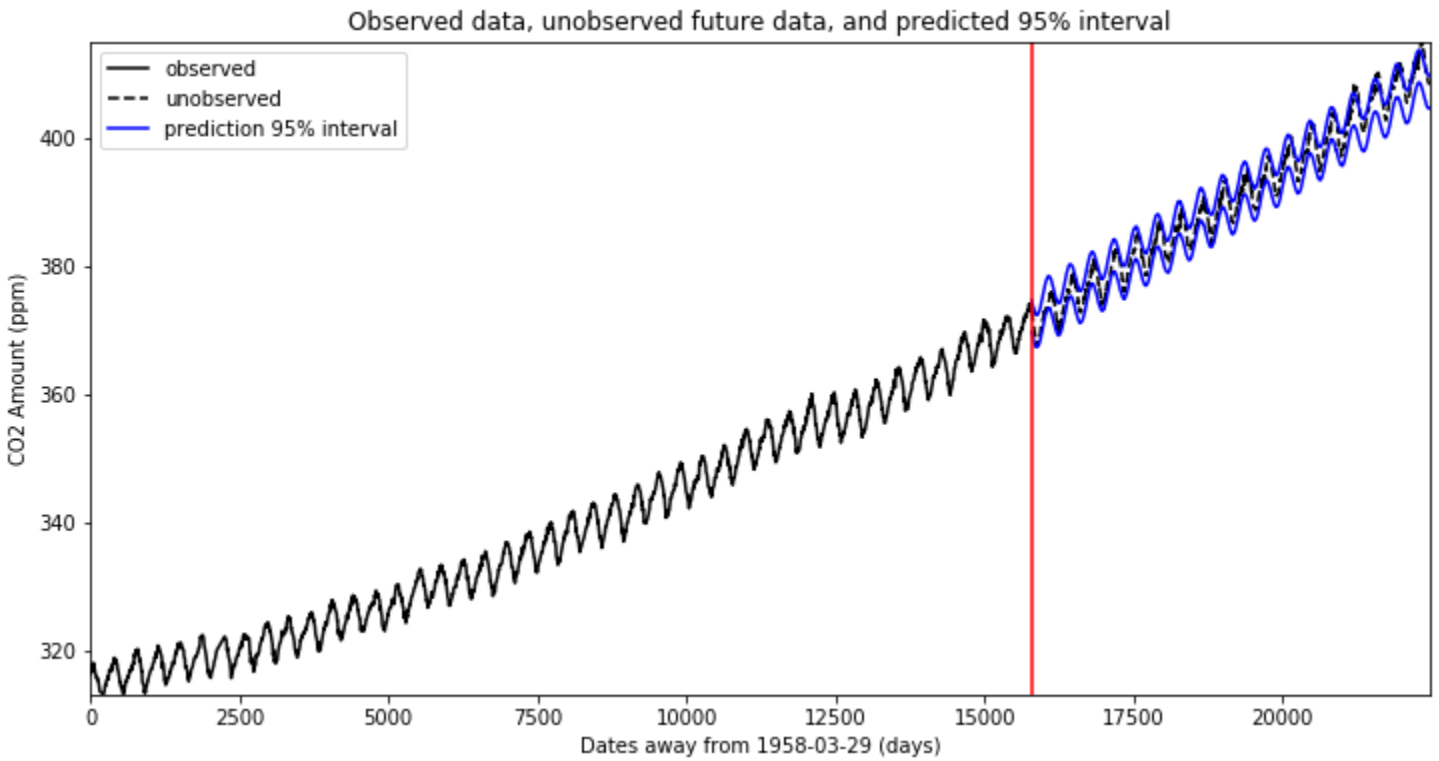
```
# pair plots

df = pd.DataFrame(
    data=np.transpose([samples[param] for param in parameters]),
    columns=parameters)
seaborn.pairplot(df, height=2.5, plot_kws={'marker': '.', 'alpha': 0.25})
plt.show()
```


c0
314.5
314.4
314.3
314.2
314.1
314.0
313.9
c1
0.010
0.005
0.000
−0.005
c2
2.65
2.60
2.55
2.50
2.45
2.40
c3
0.015
0.010
0.005
0.000
−0.005
−0.010
1.350




```
# draw predictions
prediction = samples['x_future']

# Compute 95% interval of the predicted values
prediction_interval = np.percentile(prediction, axis=0, q=[2.5, 97.5])

# Plot mean and 95% interval of predictions
plt.figure(figsize=(12, 6))
plt.plot(data["Time"][:n], data["Amount"][:n], 'k-', label='observed')
plt.plot(data["Time"][n:n+n_future], data["Amount"][n:n+n_future], 'k--', label='unobserved')
plt.xlim(0, data["Time"][n+n_future-1])
plt.ylim(
    min(min(prediction[0,:]), min(data["Amount"])),
    max(max(prediction[1,:]), max(data["Amount"])))
plt.plot(
    data["Time"][n:],
    prediction_interval[0,:],
    'b-', label='prediction 95% interval')
plt.plot(
```

```
    data["Time"][n:],
    prediction_interval[1,:],
    'b-')
plt.axvline(data["Time"][n - 1], color='red')
plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("CO2 Amount (ppm)")
plt.legend()
plt.title('Observed data, unobserved future data, and predicted 95% interval')
plt.show()
```




```
# we store this sampling for later comparison
cos_prediction_interval = prediction_interval
cos_prediction = prediction
```

New model: quadratic + sawtooth periodic

```
# The Stan model. Running this cell compiles the Stan model, which takes
# some time.
  # Stan gets to see the first n data values only

stan_code = """
data {
    int<lower=0> n;        // The number of data
    int<lower=0> n_future; // the number of predictions
    real x[n];             // The data
    real t[n];             // time data
    real t_future[n_future]; // future time data
}

parameters {
    real<lower=0> A;
    real<lower=0> sigma;
    real<lower = 0> a;
    real<lower = 0> b;
    real<lower = 0> c;
    real phi_x;
    real phi_y;
}

transformed parameters {
    real<lower=0,upper=1> phi;
    real<upper = 0> log_a;
    // The atan2 function returns a value in the range [-pi, pi], which we
    // then transform linearly into the range [0, 1].
    phi = atan2(phi_x, phi_y) / (2 * pi()) + 0.5;

    log_a = log(a);
}

model {
    A ~ normal(0, 5);
    sigma ~ gamma(1, 0.3);
    log_a ~ normal(-16, 6);
    b ~ normal(0, 0.1);
```

```
    c ~ normal(300, 100);
    phi_x ~ normal(0, 1);
    phi_y ~ normal(0, 1);
    for(i in 1:n) {
        x[i] ~ normal(
            A * (4 * fabs(fmod(1/365.25 * t[i] + phi, 1) - 0.5) - 1) +  a*t[i]*t[i] + b*t[i] + c,
            sigma);
    }
}

// Generate the predicted function values for the next n_future steps.
generated quantities {
    real x_future[n_future];
    for(i in 1:n_future) {
        x_future[i] = normal_rng(
            A * (4 * fabs(fmod(1/365.25 * t_future[i] + phi, 1) - 0.5) - 1) +  a*t_future[i]*t_future[i] + b*t_future[i] + c,
            sigma);
    }
}
"""

stan_model = pystan.StanModel(model_code=stan_code)
```

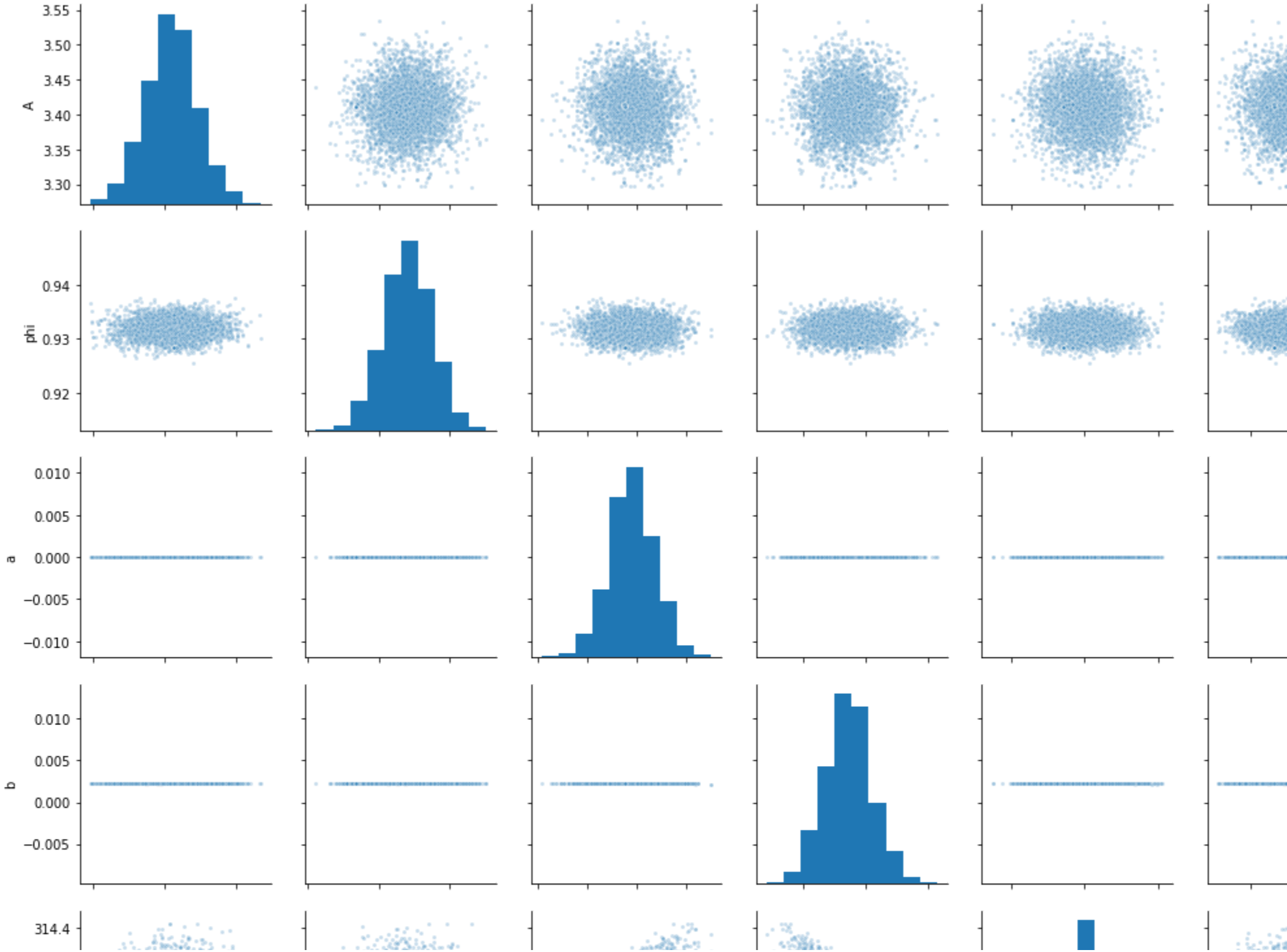
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_e495cf1d5c27bdec1a1bdddf6401e82a NOW.
```

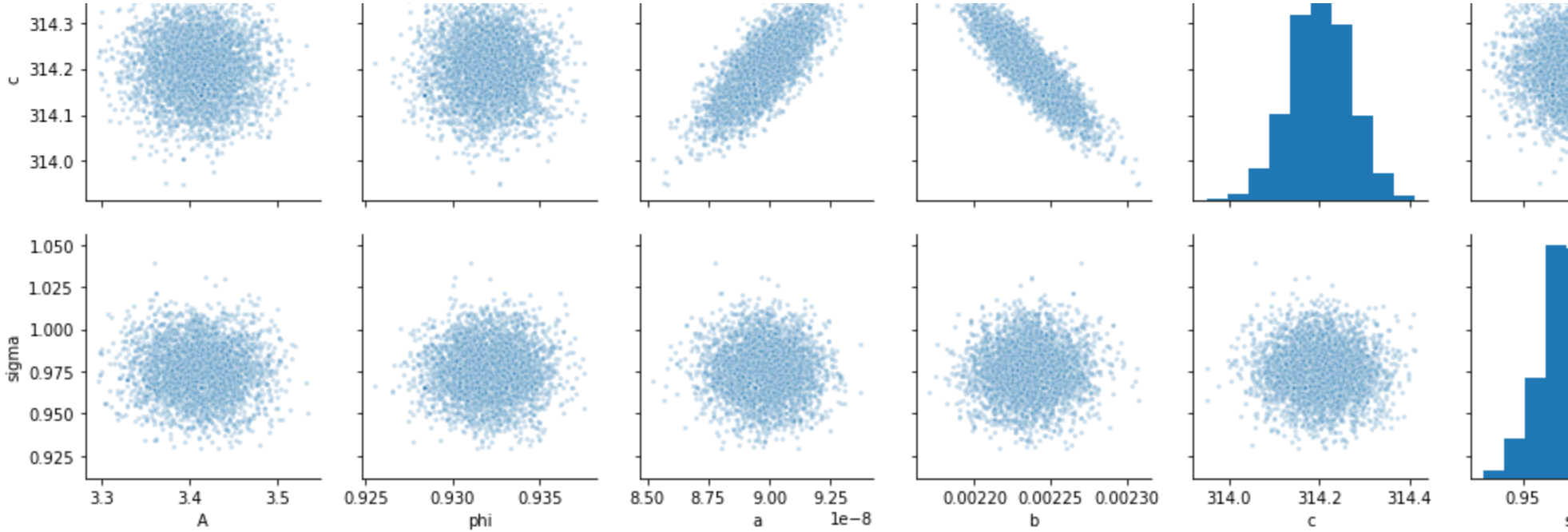
```
# subsetting the data
n = len(data)*7//10
n_future = len(data) - n

stan_data = {
    'n': n,
    'n_future': n_future,
    'x': data["Amount"][:n],
    't': data["Time"][:n],
    "t_future" : data["Time"][n:]}

# Run Hamiltonian Monte Carlo using Stan with 4 Markov chains, a 1000 step
# warm-up phase and 1000 step sampling phase for each chain. The warm-up samples
# are discarded, so we are left with 4 x 1000 = 4000 samples.
```

```
15
16 # This cell will take a minute or two to run.
17
18 parameters = ['A', 'phi', 'a', "b", "c", "sigma"]
19
20 results = stan_model.sampling(data=stan_data)
21 print(results.stansummary(pars=parameters))
22 samples = results.extract()
```

```
WARNING:pystan:73 of 4000 iterations ended with a divergence (1.82 %).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the divergences.
Inference for Stan model: anon_model_e495cf1d5c27bdec1a1bdddf6401e82a.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
A       3.41  8.1e-4   0.04   3.34   3.39   3.41   3.43   3.48   1997    1.0
phi     0.93  2.8e-5 1.7e-3   0.93   0.93   0.93   0.93   0.94   3661    1.0
a     9.0e-8 3.1e-11 1.1e-9 8.7e-8 8.9e-8 9.0e-8 9.1e-8 9.2e-8   1355    1.0
b     2.2e-3  5.1e-7 1.9e-5 2.2e-3 2.2e-3 2.2e-3 2.2e-3 2.3e-3   1375    1.0
c      314.2  1.6e-3   0.07 314.07 314.16  314.2 314.25 314.33   1694    1.0
sigma   0.98  3.3e-4   0.01   0.95   0.97   0.98   0.99   1.01   2011    1.0

Samples were drawn using NUTS at Wed Dec 18 14:12:00 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

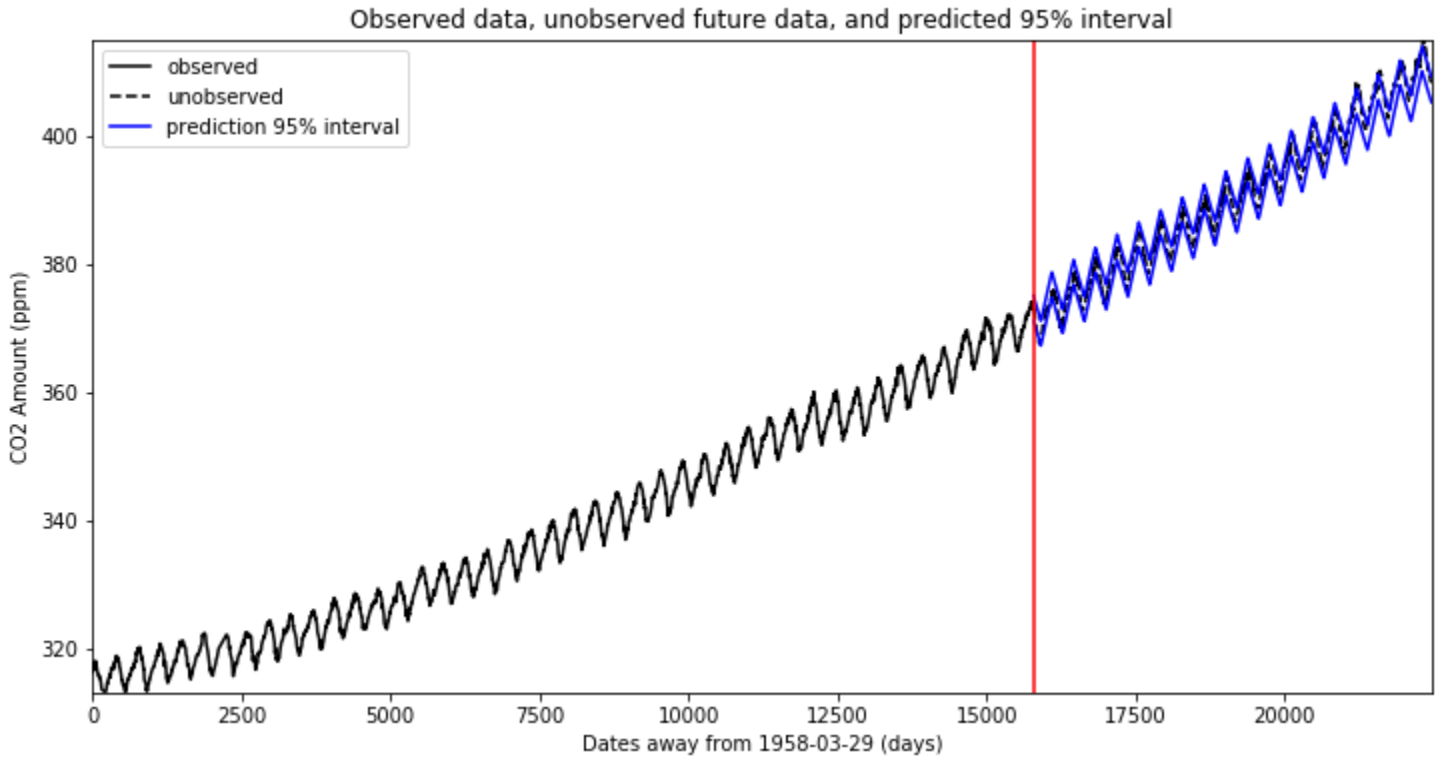
```
1 df = pd.DataFrame(
2     data=np.transpose([samples[param] for param in parameters]),
3     columns=parameters)
4 seaborn.pairplot(df, height=2.5, plot_kws={'marker': '.', 'alpha': 0.25})
5 plt.show()
```


A
phi
a
b
3.55
3.50
3.45
3.40
3.35
3.30
0.94
0.93
0.92
0.010
0.005
0.000
−0.005
−0.010
0.010
0.005
0.000
−0.005
314.4




```
prediction = samples['x_future']

# Compute 95% interval of the predicted values
prediction_interval = np.percentile(prediction, axis=0, q=[2.5, 97.5])

# Plot mean and 95% interval of predictions
plt.figure(figsize=(12, 6))
plt.plot(data["Time"][:n], data["Amount"][:n], 'k-', label='observed')
plt.plot(data["Time"][n:n+n_future], data["Amount"][n:n+n_future], 'k--', label='unobserved')
plt.xlim(0, data["Time"][n+n_future-1])

# set scaling
plt.ylim(
    min(min(prediction[0,:]), min(data["Amount"])),
    max(max(prediction[1,:]), max(data["Amount"])))

# plot confidence interval
plt.plot(
    data["Time"][n:],
```

```
    prediction_interval[0,:],
    'b-', label='prediction 95% interval')
plt.plot(
    data["Time"][n:],
    prediction_interval[1,:],
    'b-')

plt.axvline(data["Time"][n - 1], color='red')
plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("CO2 Amount (ppm)")
plt.legend()
plt.title('Observed data, unobserved future data, and predicted 95% interval')
plt.show()
```




```
# we store this sampling for later comparison
saw_prediction_interval = prediction_interval
saw_prediction = prediction
```

Model Comparison

```
# confidence interval width comparison
saw_interval_length = saw_prediction_interval[1:,][0] - saw_prediction_interval[0:,][0]
cos_interval_length = cos_prediction_interval[1:,][0] - cos_prediction_interval[0:,][0]
rate = np.mean(saw_interval_length < cos_interval_length)*100
print("The percentage of the width of model 3 interval is smaller than of model 2 interval: {}%".format(rate))
```

```
The percentage of the width of model 3 interval is smaller than of model 2 interval: 100.0%
```

Accuracy testing

```
# the real data to validate
validate = np.array(data["Amount"][n:])

# the mean of prediction
saw_mean = np.mean(saw_prediction, axis = 0)
cos_mean = np.mean(cos_prediction, axis = 0)

# compute the SME errors
saw_SME = np.mean((saw_mean - validate)**2)
cos_SME = np.mean((cos_mean - validate)**2)
saw_all_SME = np.mean((saw_prediction - validate)**2)
cos_all_SME = np.mean((cos_prediction - validate)**2)

print("Sawtooth SME wrt to samples mean:", saw_SME)
print("Cos SME wrt to samples mean:", cos_SME)
print("Sawtooth SME wrt to all samples values:", saw_all_SME)
print("Cos SME wrt to all samples values:", cos_all_SME)
```

```
Sawtooth SME wrt to samples mean: 1.612156603044492
Cos SME wrt to samples mean: 2.1612224291754356
Sawtooth SME wrt to all samples values: 2.5857090803350777
Cos SME wrt to all samples values: 3.824635218251834
```

```
1
```

Test statistics

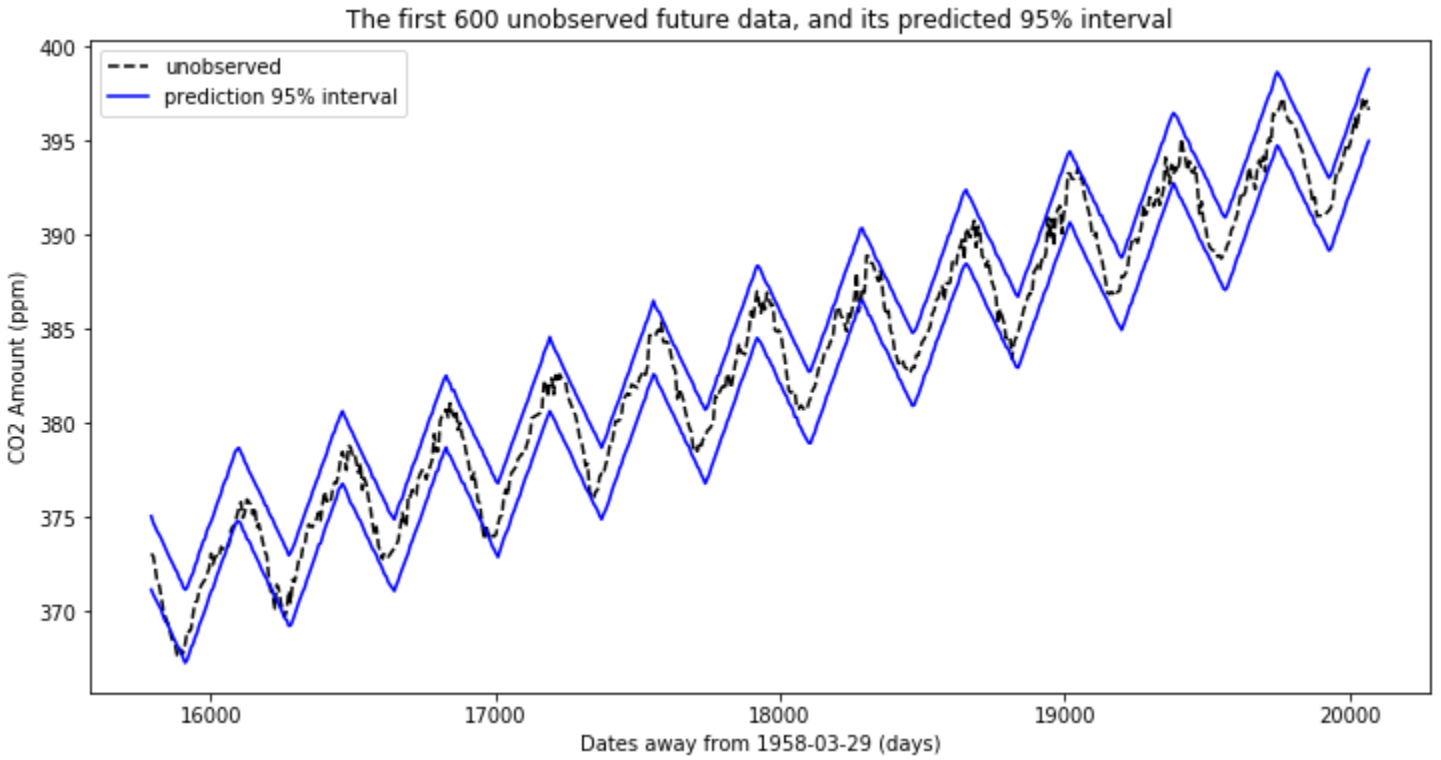
As discussed above, we use the first 70% of the data (2197 values) as data for computing the posterior distributions over the parameters above using Stan. We use the other 30% of the data (the next 942 values) to test our prediction of future function values.

Our test statistic is the mean of future data: 942 last data points in the dataset.

The reason is that: we suspect the model, even though overall fit the future data very well (based on the plot), fails to predict the few anomalies in the original data fully. For example, when we zoom in the first 600 data points in the test future data, we can see at those peaks, our real data is slightly off compared to the predicted interval.

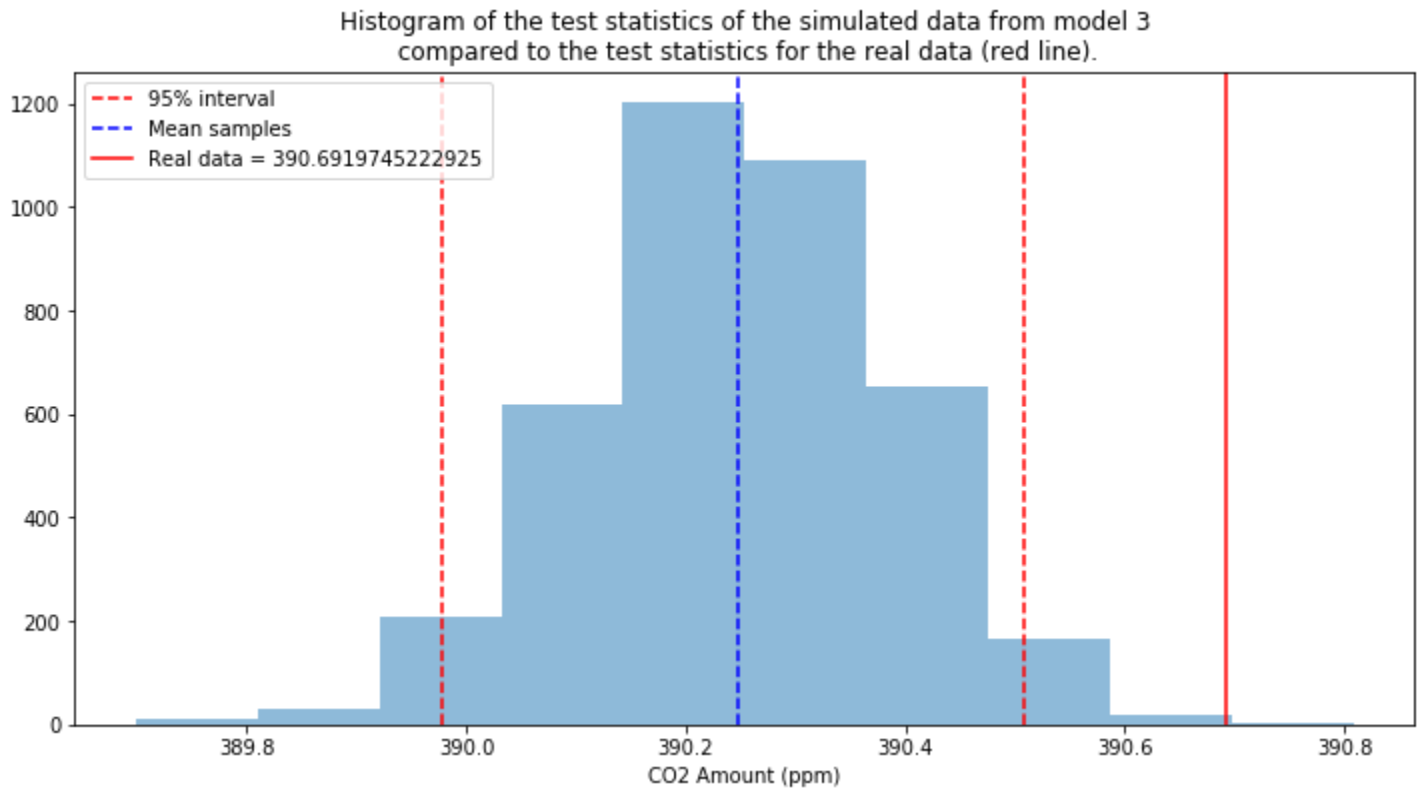
As the mean is sensitive to the outliers, we would expect the mean for the real future data to be in the upper or bottom percentile of the test statistics results of the sampled data.

```
# zoom in the data for the first 600 points
plt.figure(figsize = (12,6))
plt.plot(data["Time"][n:n+600], data["Amount"][n:n+600], 'k--', label='unobserved')
plt.plot(
    data["Time"][n:n+600],
    prediction_interval[0,:][:600],
    'b-', label='prediction 95% interval')
plt.plot(
    data["Time"][n:n+600],
    prediction_interval[1,:][:600],
    'b-')
plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("CO2 Amount (ppm)")
plt.legend()
plt.title('The first 600 unobserved future data, and its predicted 95% interval')
plt.show()
```




```
# test statistics: mean of samples
mean_samples = np.mean(saw_prediction, axis = 1)
benchmark = np.mean(data["Amount"][n:])

# plot the histogram of the test statistics
plt.figure(figsize = (12,6))
plt.hist(mean_samples, alpha = 0.5)
plt.axvline(np.quantile(mean_samples, 0.025), color='red', label ="95% interval", linestyle = "--")
plt.axvline(np.quantile(mean_samples, 0.975), color='red', linestyle = "--")
plt.axvline(np.mean(mean_samples), color='blue', linestyle = "--", label = "Mean samples")
plt.axvline(benchmark, color='red', label ="Real data = " + str(benchmark))
plt.xlabel("CO2 Amount (ppm)")
plt.title("Histogram of the test statistics of the simulated data from model 3\n compared to\
 the test statistics for the real data (red line).")
plt.legend()
plt.show()
```

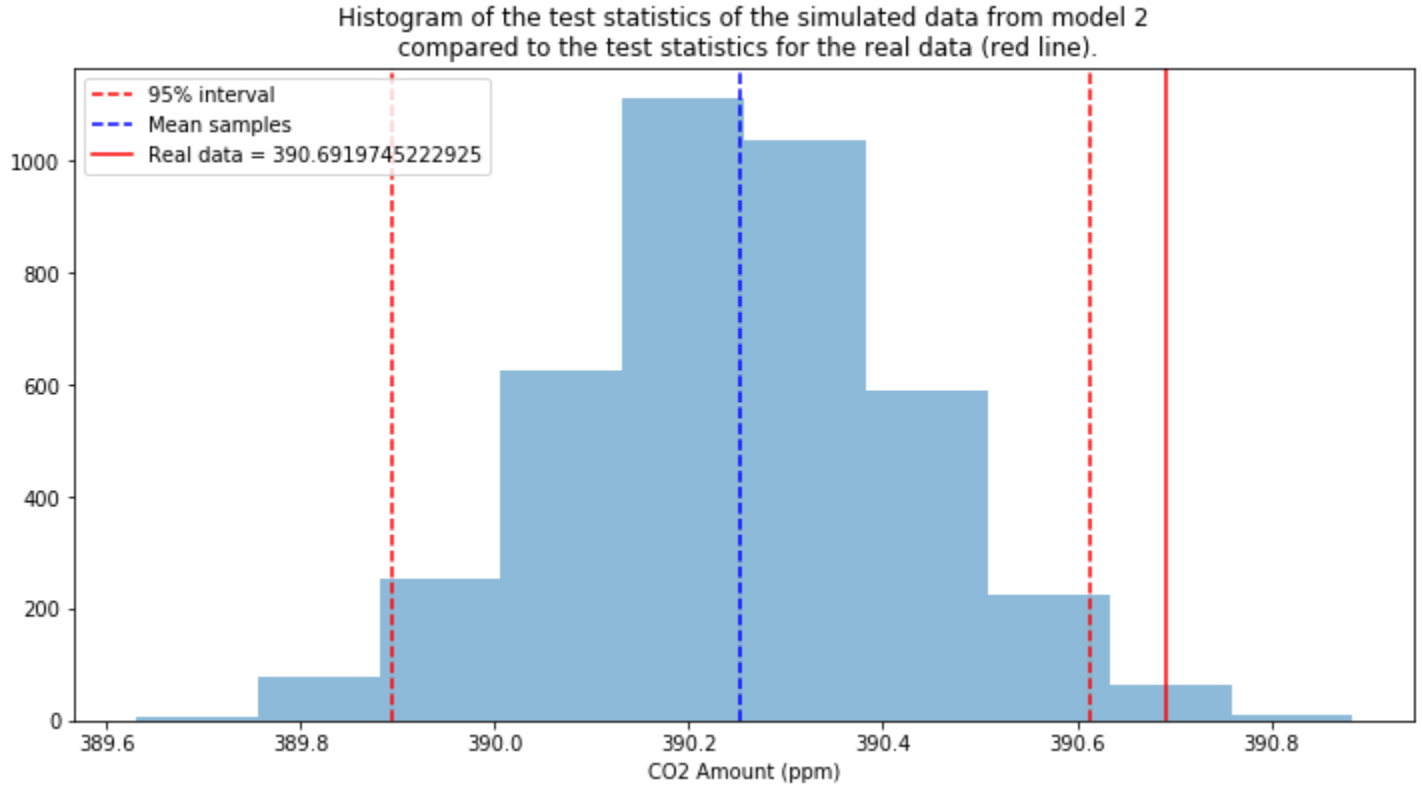
```
17
18 final = np.mean(mean_samples > benchmark)
19 print("percentage result > real statistic value: ", final*100, "%")
20 print("p value: ", 1-final)
```




```
percentage result > real statistic value:  0.1 %
p value:  0.999
```

```
1 # test statistics: mean of samples
2 mean_samples = np.mean(cos_prediction, axis = 1)
3 benchmark = np.mean(data["Amount"][n:])
4
5 # plot the histogram of the test statistics
6 plt.figure(figsize = (12,6))
7 plt.hist(mean_samples, alpha = 0.5)
8 plt.axvline(np.quantile(mean_samples, 0.025), color='red', label ="95% interval", linestyle = "--")
9 plt.axvline(np.quantile(mean_samples, 0.975), color='red', linestyle = "--")
```

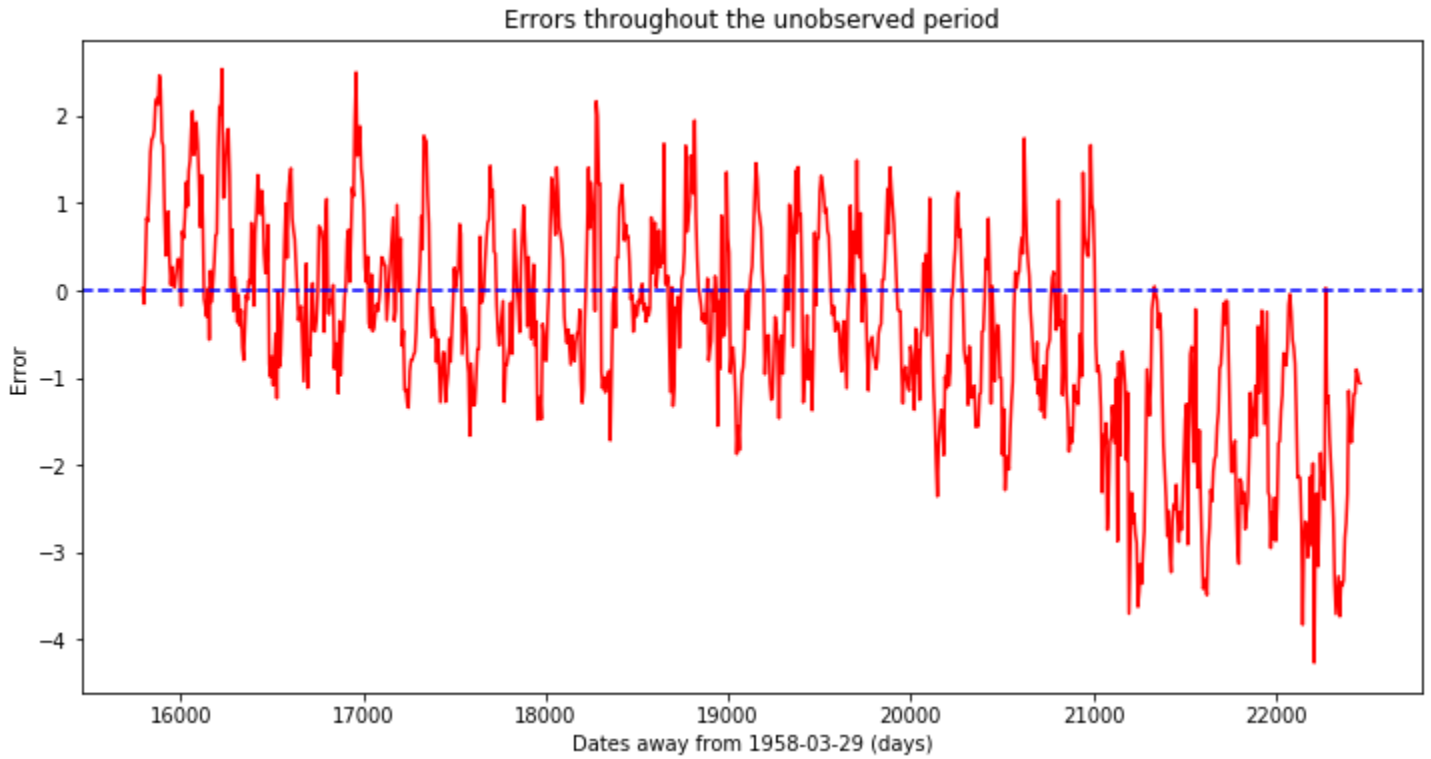
```
plt.axvline(np.mean(mean_samples), color='blue', linestyle = "--", label = "Mean samples")
plt.axvline(benchmark, color='red', label ="Real data = " + str(benchmark))
plt.xlabel("CO2 Amount (ppm)")
plt.title("Histogram of the test statistics of the simulated data from model 2\n compared to\
 the test statistics for the real data (red line).")
plt.legend()
plt.show()

final = np.mean(mean_samples > benchmark)
print("percentage result > real statistic value: ", final*100, "%")
print("p value: ", 1-final)
```




```
percentage result > real statistic value:  0.7250000000000001 %
p value:  0.99275
```

```
# the mean of errors by time
error_by_time = np.mean(saw_prediction - validate, axis = 0)

```

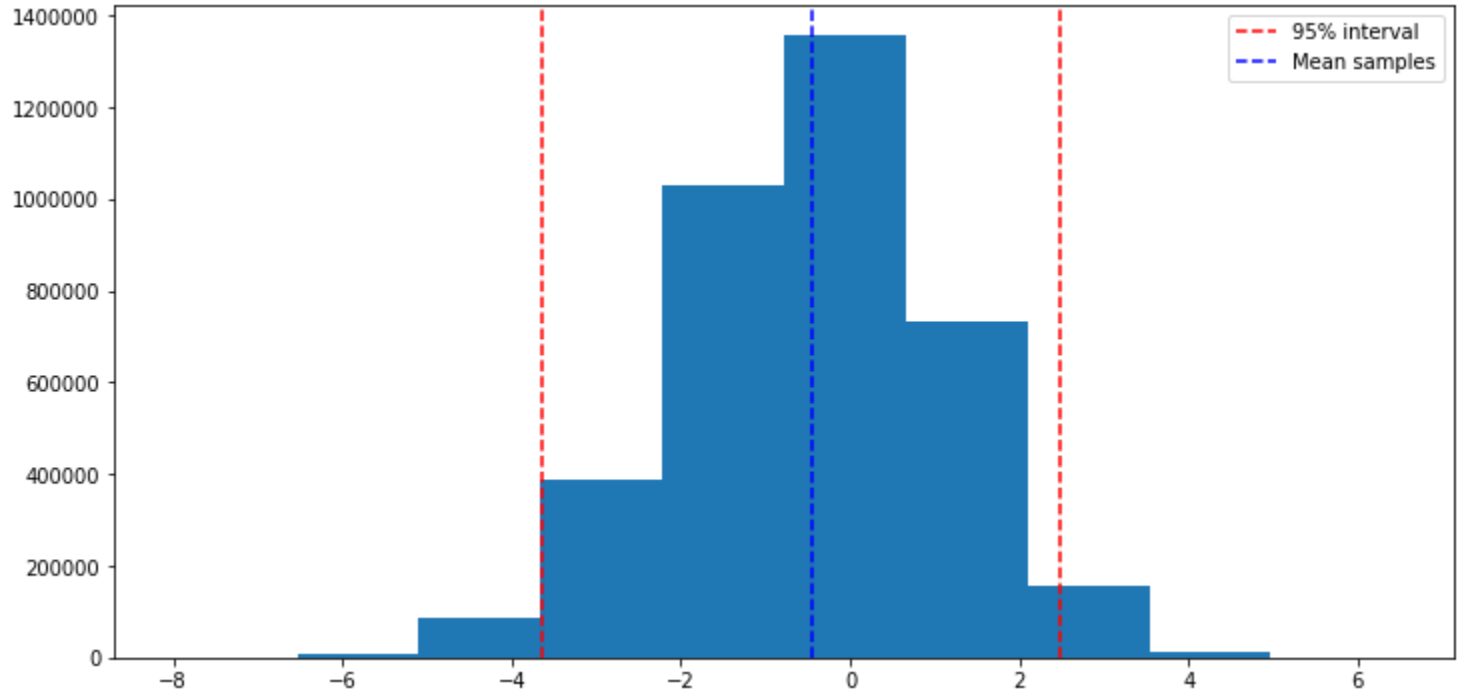
```
# plot
plt.figure(figsize = (12,6))
plt.plot(data["Time"][n:], error_by_time, 'r-', label = "error")
plt.axhline(0, color = "blue", linestyle = "--")
plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("Error")
plt.title('Errors throughout the unobserved period')
plt.show()
```




```
# combine of every errors
error_sets = (saw_prediction - validate).flatten()
print("Median:", np.median(error_sets))
print("95% Interval:", np.quantile(error_sets, [0.025, 0.975]))

# plot
plt.figure(figsize = (12,6))
plt.hist(error_sets)
```

```
plt.axvline(np.quantile(error_sets, 0.025), color='red', label ="95% interval", linestyle = "--")
plt.axvline(np.quantile(error_sets, 0.975), color='red', linestyle = "--")
plt.axvline(np.mean(error_sets), color='blue', linestyle = "--", label = "Mean samples")
plt.legend()
plt.show()
```

```
Median: -0.39648976903481525
95% Interval: [-3.63786624  2.4763771 ]
```



Predictions

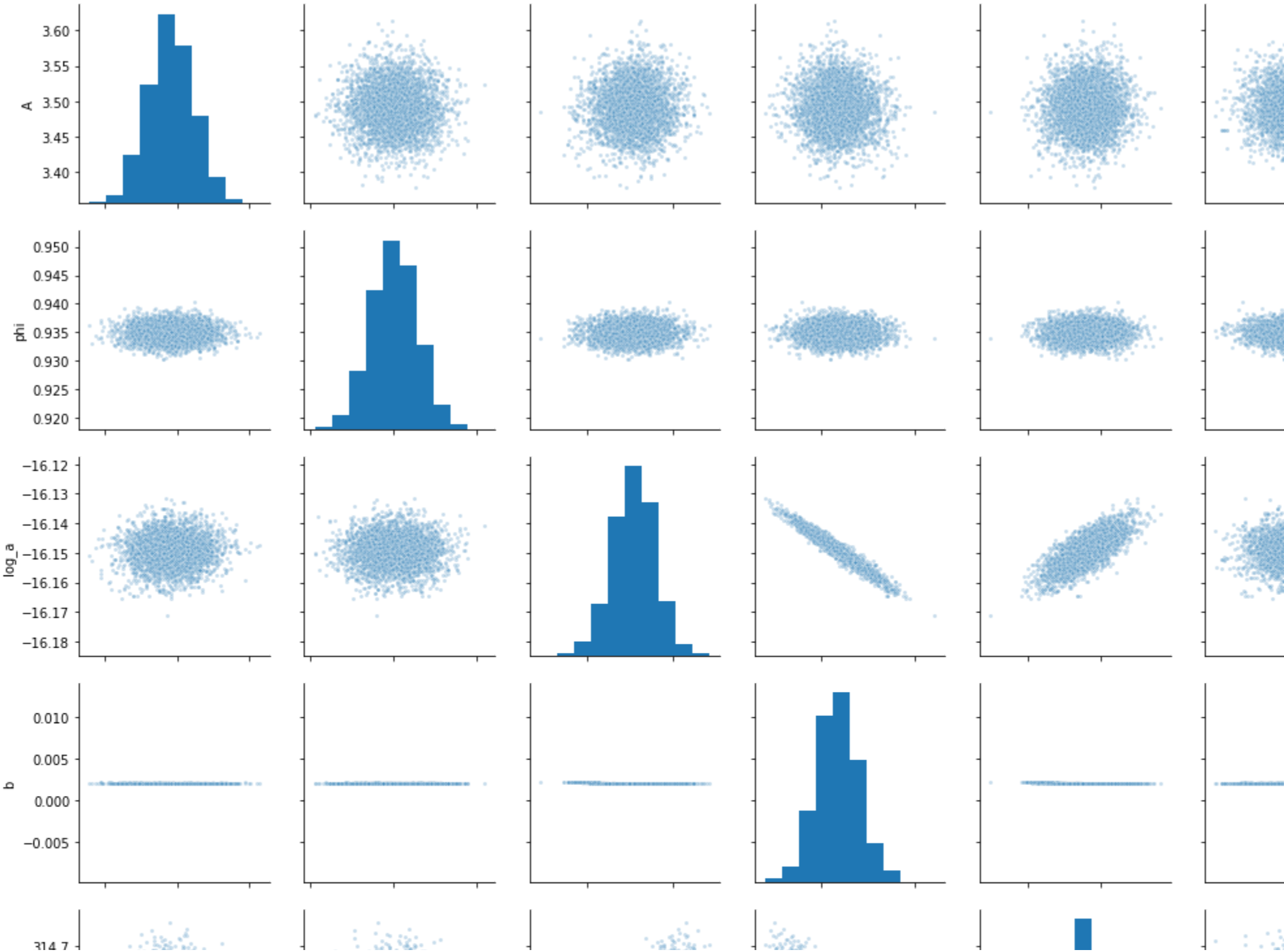
```
# the dates need for calculation: 1st date of data, end date of data, 2058 dates (start and end)
first_date = datetime.datetime.strptime(data["Date"][0], "%Y-%m-%d")
last_date = datetime.datetime.strptime(data["Date"][len(data)-1], "%Y-%m-%d")
desire_first = datetime.datetime.strptime("2058-01-01", "%Y-%m-%d")
desire_last = datetime.datetime.strptime("2058-12-31", "%Y-%m-%d")

# convert into readable dates by pystan
stop1 = (desire_first - first_date).days//7+1
```

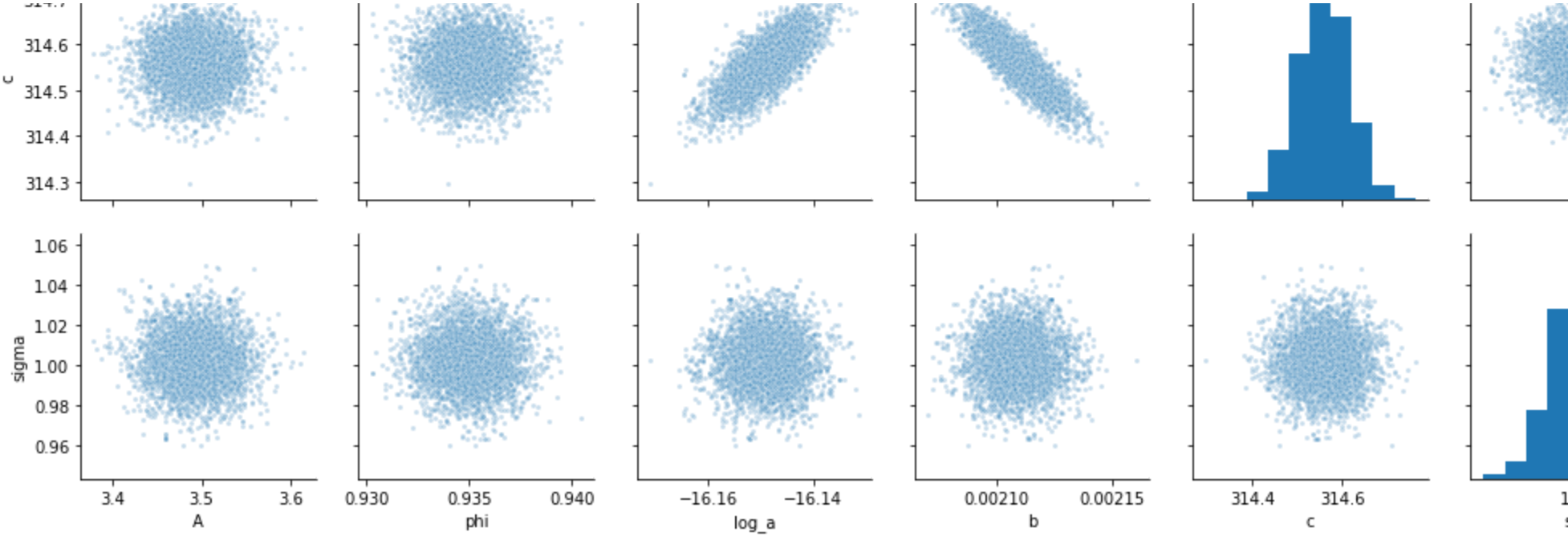
```
stop1 = (desire_first - first_date).days//7+1
stop2 = (desire_last - first_date).days//7
start = (last_date - first_date).days/7

# generate the future time parameters
new_time = np.array(list(range(int(start + 1), int(start + 1) + int(40*365.25//7))))*7
new_time = list(new_time)
```

```
# rerun the model with all the data to predict the future
n = len(data)
n_future = len(new_time)

stan_data = {
    'n': n,
    'n_future': n_future,
    'x': data["Amount"],
    't': data["Time"],
    "t_future" : new_time}

# Run Hamiltonian Monte Carlo using Stan with 4 Markov chains, a 1000 step
# warm-up phase and 1000 step sampling phase for each chain. The warm-up samples
# are discarded, so we are left with 4 x 1000 = 4000 samples.

# This cell will take a minute or two to run.

parameters = ['A', 'phi', 'log_a', 'b', 'c', "sigma"]

results = stan_model.sampling(data=stan_data)
print(results.stansummary(pars=parameters))
samples = results.extract()
```

```
WARNING:pystan:Maximum (flat) parameter count (1000) exceeded: skipping diagnostic tests for n_eff and Rhat.
To run all diagnostics call pystan.check_hmc_diagnostics(fit)
WARNING:pystan:54 of 4000 iterations ended with a divergence (1.35 %).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the divergences.
WARNING:pystan:100 of 4000 iterations saturated the maximum tree depth of 10 (2.5 %)
WARNING:pystan:Run again with max_treedepth larger than 10 to avoid saturation
Inference for Stan model: anon_model_e495cf1d5c27bdec1a1bdddf6401e82a.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
A       3.49  7.2e-4   0.03   3.43   3.47   3.49   3.51   3.55   1939    1.0
phi     0.94  2.2e-5 1.4e-3   0.93   0.93   0.94   0.94   0.94   4093    1.0
log_a -16.15  1.2e-4 5.1e-3 -16.16 -16.15 -16.15 -16.15 -16.14   1665    1.0
b     2.1e-3  2.8e-7 1.1e-5 2.1e-3 2.1e-3 2.1e-3 2.1e-3 2.1e-3   1696    1.0
c     314.56  1.2e-3   0.06 314.45 314.52 314.56  314.6 314.67   2152    1.0
sigma    1.0  3.0e-4   0.01   0.98   0.99    1.0   1.01   1.03   1881    1.0

Samples were drawn using NUTS at Wed Dec 18 14:32:04 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

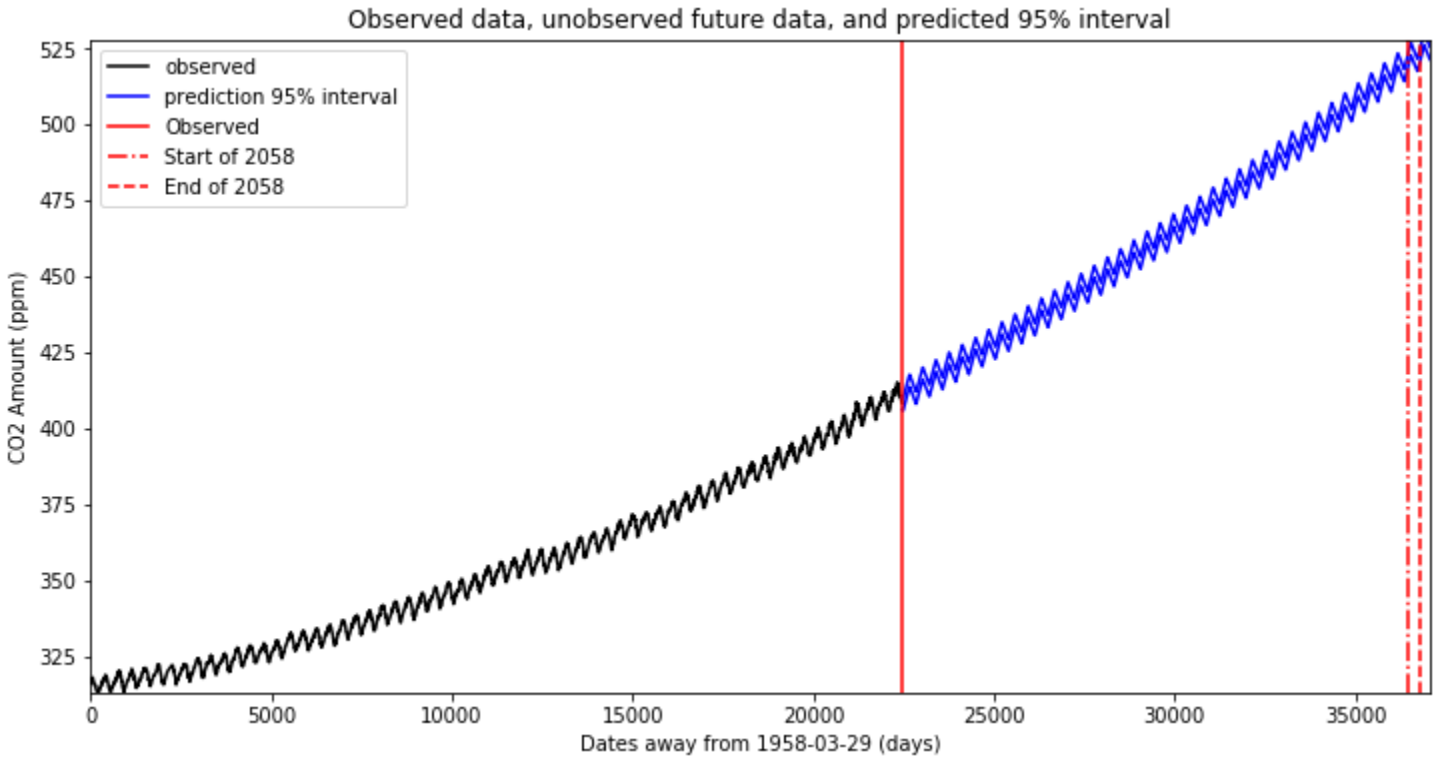
```
# pair plots
df = pd.DataFrame(
    data=np.transpose([samples[param] for param in parameters]),
    columns=parameters)
seaborn.pairplot(df, height=2.5, plot_kws={'marker': '.', 'alpha': 0.25})
plt.show()
```


A
3.60
3.55
3.50
3.45
3.40
phi
0.950
0.945
0.940
0.935
0.930
0.925
0.920
log_a
−16.12
−16.13
−16.14
−16.15
−16.16
−16.17
−16.18
b
0.010
0.005
0.000
−0.005
314.7




```
prediction = samples['x_future']

# Compute 95% interval of the predicted values
prediction_interval = np.percentile(prediction, axis=0, q=[2.5, 97.5])

# Plot mean and 95% interval of predictions
plt.figure(figsize=(12, 6))
plt.plot(data["Time"], data["Amount"], 'k-', label='observed')
plt.xlim(0, max(new_time))
plt.ylim(
    min(min(prediction[0,:]), min(data["Amount"])),
    max(max(prediction[1,:]), max(data["Amount"])))

plt.plot(
    new_time,
    prediction_interval[0,:],
    'b-', label='prediction 95% interval')

plt.plot(
```

```
    new_time,
    prediction_interval[1,:],
    'b-')

plt.axvline(data["Time"][n - 1], color='red', label = "Observed")
plt.axvline(stop1 * 7, color = "red", linestyle = "-.", label ="Start of 2058")
plt.axvline(stop2 * 7, color = "red", linestyle = "--", label ="End of 2058")
plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("CO2 Amount (ppm)")
plt.legend()
plt.title('Observed data, unobserved future data, and predicted 95% interval')
plt.show()
```

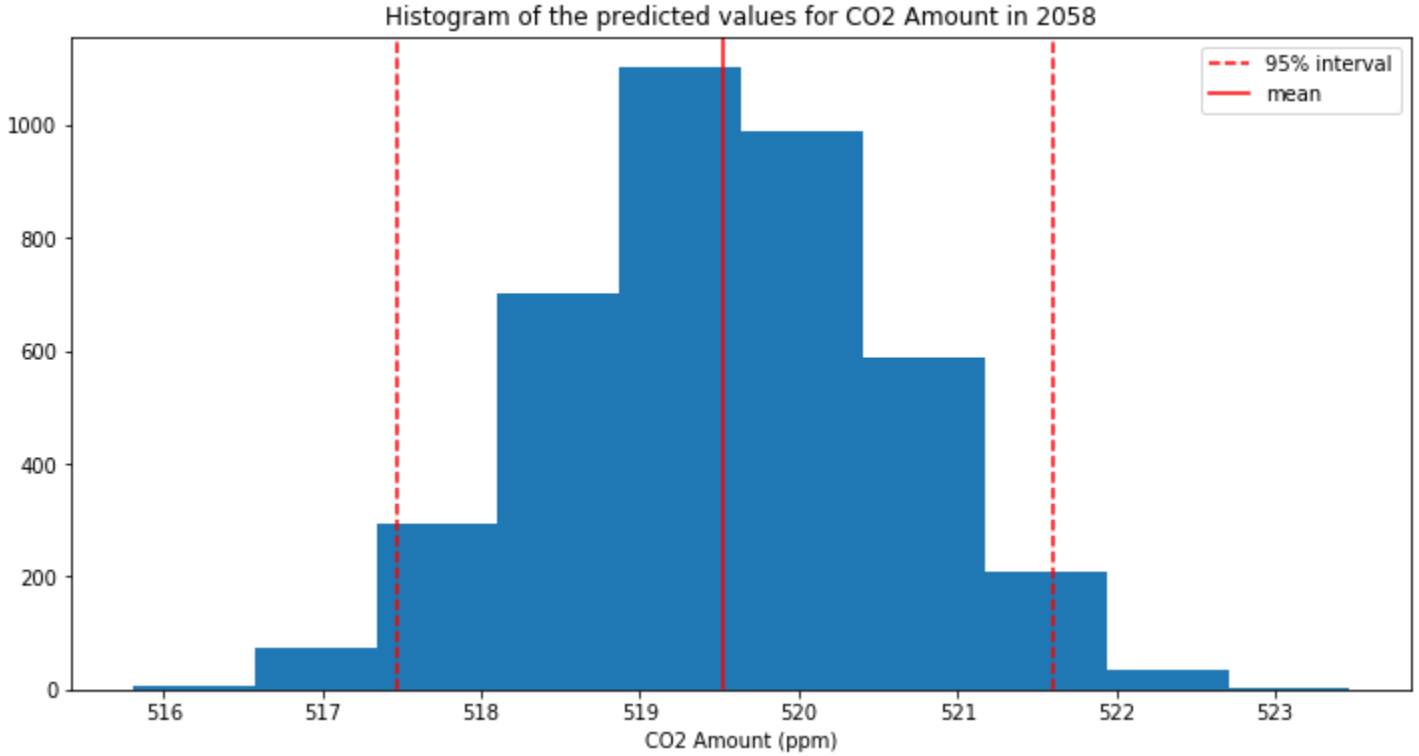


year 2058

```
# Get data for this specific measurement and compute the statistics
```

```
sample_2058 = prediction.T[int(stop1-start)-1]
print("Mean of the estimate: {}".format(np.mean(sample_2058)))
print("Median of the estimate: {}".format(np.median(sample_2058)))
print("STD of the estimate:", np.std(sample_2058))
lower_2058 = prediction_interval[0,:][int(stop1-start)-1]
upper_2058 = prediction_interval[1,:][int(stop1-start)-1]
print("Confidence interval in 2058: [{}, {}]".format(lower_2058, upper_2058))

# plot the results
plt.figure(figsize = (12,6))
plt.hist(sample_2058)
plt.xlabel("CO2 Amount (ppm)")
plt.title("Histogram of the predicted values for CO2 Amount in 2058")
plt.axvline(lower_2058, color = "red", linestyle = "--")
plt.axvline(upper_2058, color = "red", linestyle = "--", label = "95% interval")
plt.axvline(np.mean(sample_2058), color = "red", linestyle = "-", label = "mean")
plt.legend()
plt.show()
```

```
Mean of the estimate: 519.5246199645143
Median of the estimate: 519.5167440251244
STD of the estimate: 1.05966781226145
Confidence interval in 2058: [517.474873011385, 521.5981563346509]
```

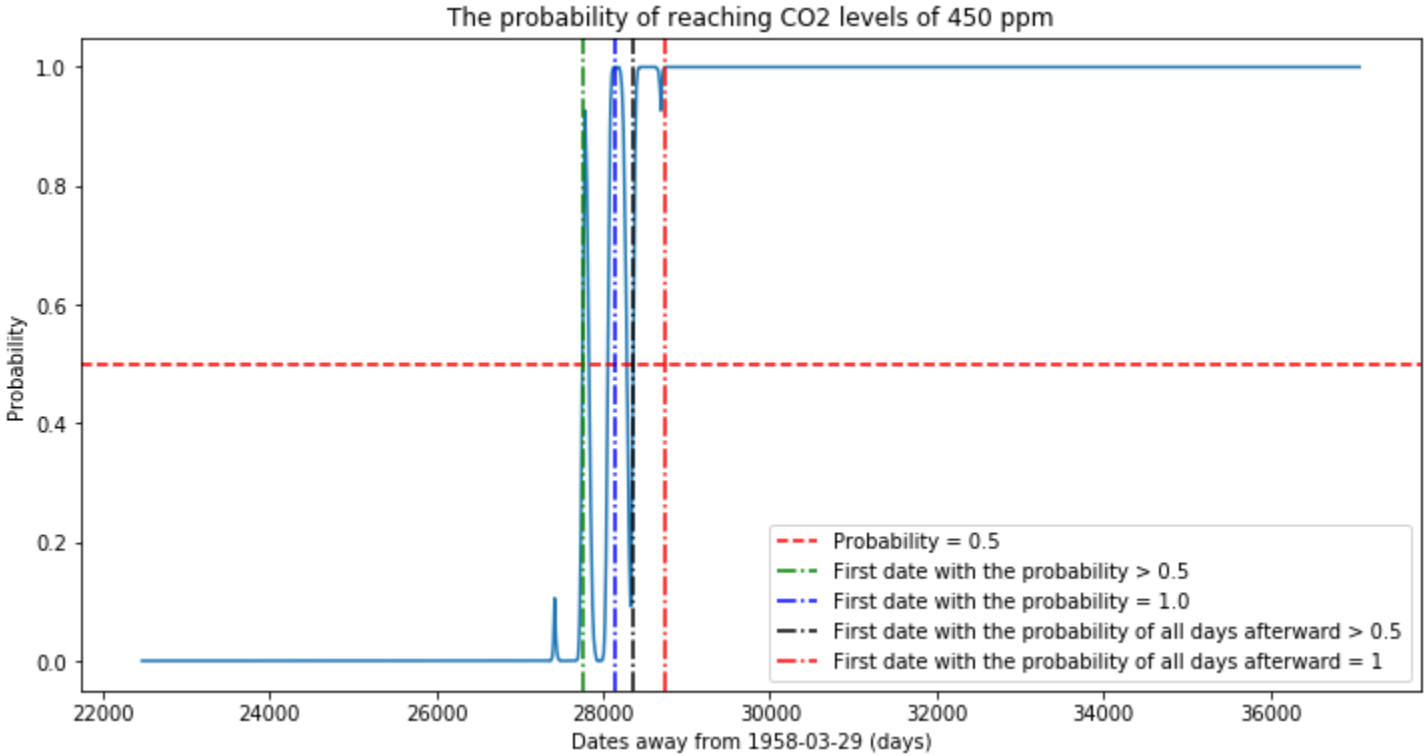



```
# the probabilities of certain dates > 450 ppm
probability = np.mean(prediction.T >= 450, axis = 1)

# first value to have 1.0 probability
idx_100 = next(x[0] for x in enumerate(list(probability)) if x[1] >= 1)

# first value to have 0.5 probability
idx_50 = next(x[0] for x in enumerate(list(probability)) if x[1] >= 0.5)

# first value to have all values onward = 1.0
for i in range(len(probability)):
  if np.mean(probability[i:]) == 1:
```

```
    idx_full_100 = i
    break

# first value to have all values onward >= 0.5
for i in range(len(probability)):
  if np.mean(probability[i:] >= 0.5) == 1:
    idx_full_50 = i
    break

print("1st date to have 0.5 probability:", timedelta(idx_50*7) + last_date)
print("1st date to have 1.0 probability:", timedelta(idx_100*7) + last_date)
print("1st date to have all 0.5 probability onward:", timedelta(idx_full_50*7) + last_date)
print("1st date to have all 1.0 probability onward:", timedelta(idx_full_100*7) + last_date)
```

```
1st date to have 0.5 probability: 2034-03-18 00:00:00
1st date to have 1.0 probability: 2035-04-07 00:00:00
1st date to have all 0.5 probability onward: 2035-11-17 00:00:00
1st date to have all 1.0 probability onward: 2036-11-29 00:00:00
```

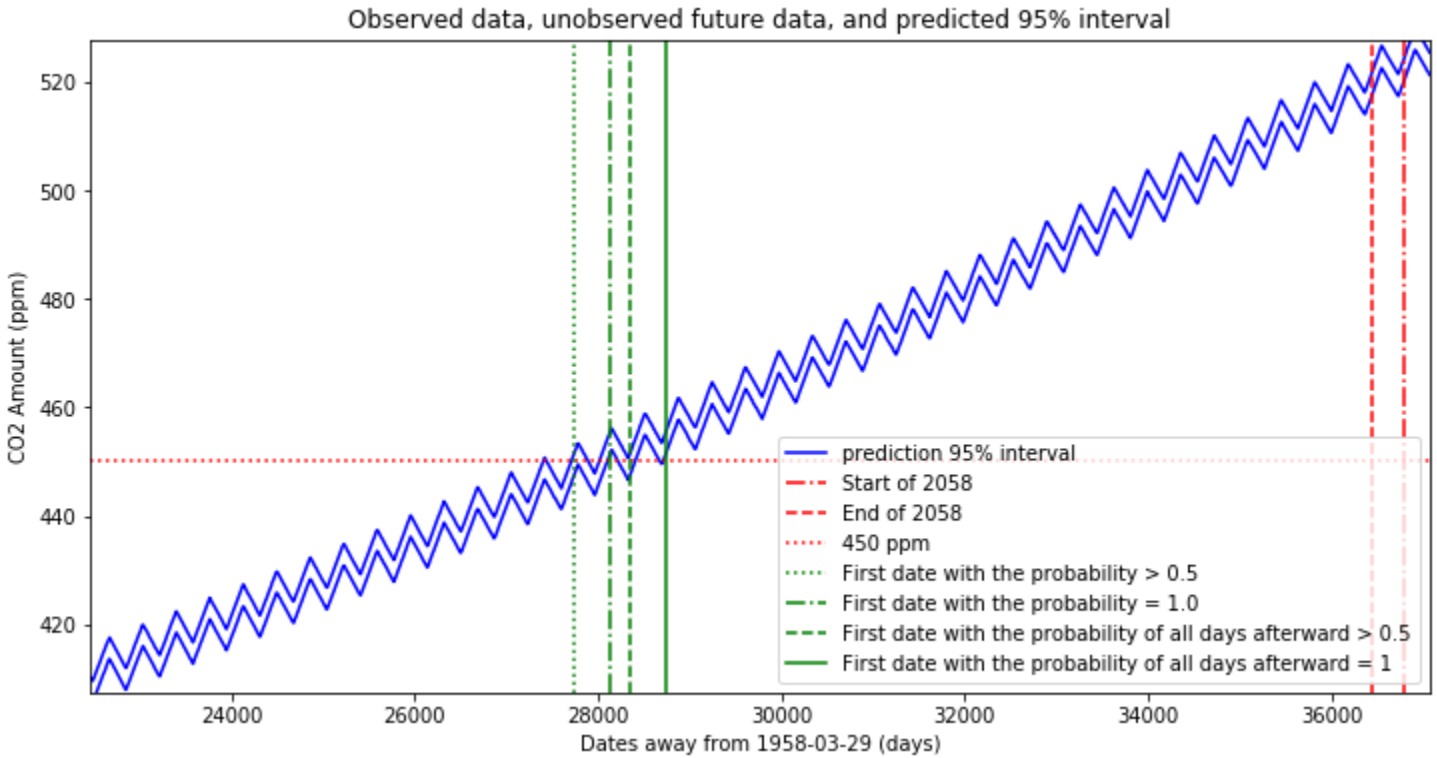
```
# plot the probability
plt.figure(figsize = (12,6))
plt.plot(new_time,probability)
plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("Probability")
plt.axhline(0.5, color = "red", linestyle = "--", label = "Probability = 0.5")
plt.axvline(idx_50 * 7 + data["Time"][n - 1], color = "green", linestyle = "-.", label ="First date with the probability > 0.5")
plt.axvline(idx_100 * 7 + data["Time"][n - 1], color = "blue", linestyle = "-.", label ="First date with the probability = 1.0")
plt.axvline(idx_full_50 * 7 + data["Time"][n - 1], color = "black", linestyle = "-.", label ="First date with the probability of al
plt.axvline(idx_full_100 * 7 + data["Time"][n - 1], color = "red", linestyle = "-.", label ="First date with the probability of all
plt.title("The probability of reaching CO2 levels of 450 ppm")
plt.legend()
plt.show()
```



All plot at once

```
prediction = samples['x_future']

# Compute 95% interval of the predicted values
prediction_interval = np.percentile(prediction, axis=0, q=[2.5, 97.5])

# Plot mean and 95% interval of predictions
plt.figure(figsize=(12, 6))
# plt.plot(data["Time"], data["Amount"], 'k-', label='observed')
plt.xlim(data["Time"][n - 1], max(new_time))
plt.ylim(
    min(prediction[0,:]),
    max(prediction[1,:]))

```

```
plt.plot(
    new_time,
    prediction_interval[0,:],
    'b-', label='prediction 95% interval')

plt.plot(
    new_time,
    prediction_interval[1,:],
    'b-')

plt.axvline(stop2 * 7, color = "red", linestyle = "-.", label ="Start of 2058")
plt.axvline(stop1 * 7, color = "red", linestyle = "--", label ="End of 2058")
plt.axhline(450, color = "red", linestyle = ":", label = "450 ppm")
plt.axvline(idx_50 * 7 + data["Time"][n - 1], color = "green", linestyle = ":", label ="First date with the probability > 0.5")
plt.axvline(idx_100 * 7 + data["Time"][n - 1], color = "green", linestyle = "-.", label ="First date with the probability = 1.0")
plt.axvline(idx_full_50 * 7 + data["Time"][n - 1], color = "green", linestyle = "--", label ="First date with the probability of al
plt.axvline(idx_full_100 * 7 + data["Time"][n - 1], color = "green", linestyle = "-", label ="First date with the probability of al

plt.xlabel("Dates away from 1958-03-29 (days)")
plt.ylabel("CO2 Amount (ppm)")
plt.legend()
plt.title('Observed data, unobserved future data, and predicted 95% interval')
plt.show()
```


Observed data, unobserved future data, and predicted 95% interval
CO2 Amount (ppm)
Dates away from 1958-03-29 (days)
520
500
480
460
440
420
24000
26000
28000
30000
32000
34000
36000
prediction 95% interval
Start of 2058
End of 2058
450 ppm
First date with the probability > 0.5
First date with the probability = 1.0
First date with the probability of all days afterward > 0.5
First date with the probability of all days afterward = 1

```
1
```