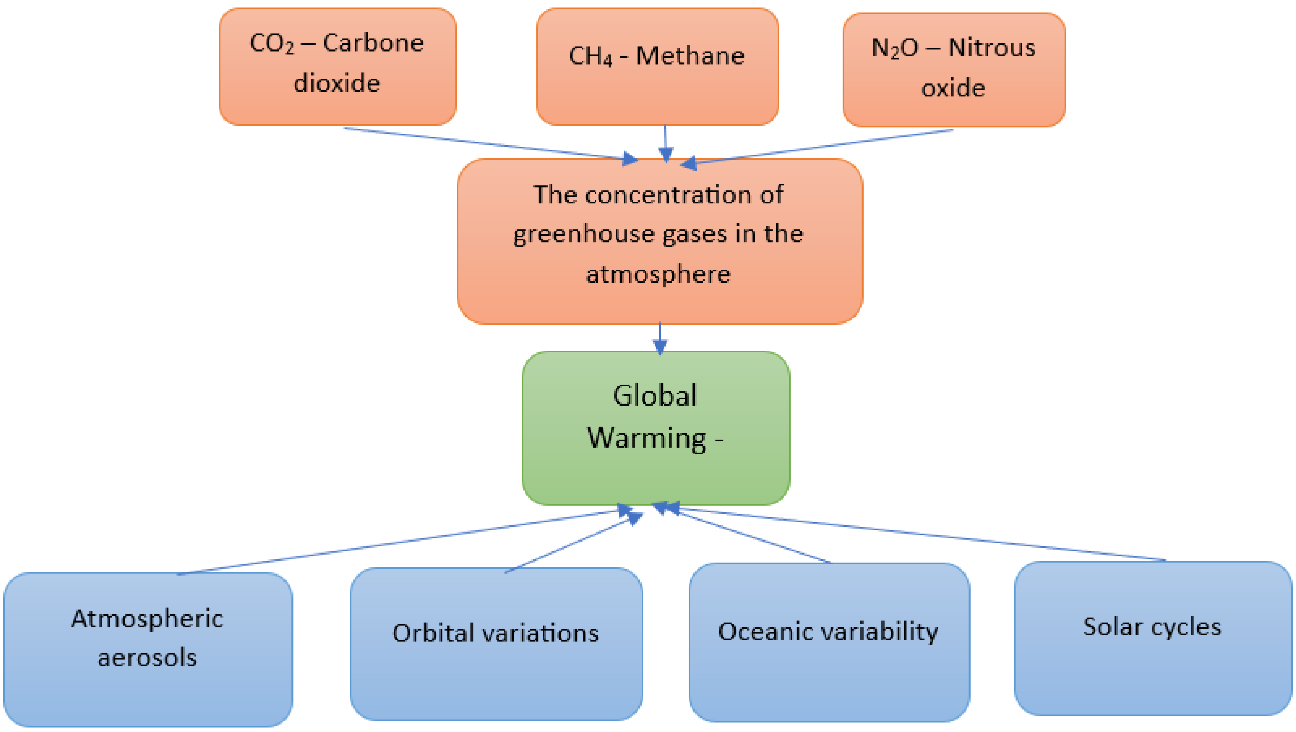
Problem formulation

The problem we focused on is predicting the global temperature anomaly based on concentration of gases in the atmosphere. Data is collected from year 2001 to 2022.

The result of our research can show directly how the global warming is going to change in the future years. This information can be useful for preparing society for the impact of temperature changes. It can also show how our population influence global weather changes because of gases emission.

Data used in our project come from two different sources. Data about global temperature anomaly comes from NASA Global Climate Change https://climate.nasa.gov/vital-signs/global-temperature/ and the gases concentration comes from Global Monitoring Laboratory https://gml.noaa.gov/. The first one contains yearly temperature anomaly around the globe. Original data was from years 1880-2023. The second data set has information about three gases concentration in the atmosphere. It not only has yearly data but also monthly.

Before starting the project we created the dag graph that shows what data impacts global warming.



.

The parameters we have taken into consideration are carbon dioxide(CO2), methan(CH4), nitrous oxide(N2O). Although there were more factors than only gases the ones we have chosen had the most significant impact on temperature changes.

During the given period of time there was a globla pandemic of COVID-19. Because of that the emission of gases were much lower due to for example smaller amount of flights. This may be potentially confounding for our results.

Data preprocessing

As the data were clear without any NaN or Null values we didn't change much while cleaning it. We had to compute the mean value for every year when it comes to gases because the data had values for every month which we didn't need to use. These actions were taken in different file "data_preprocessing.ipnyb".

Imports

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns
from cmdstanpy import CmdStanModel
from sklearn.preprocessing import StandardScaler
import arviz as az

df = pd.read_csv("/home/data.csv", index_col=0)
```

Data standarization

In [5]:

```
df.head()
```

Out[5]:

	year	CO2	CH4	N2O	Temperature
0	2001	371.319167	1771.269167	316.364167	0.54
1	2002	373.452500	1772.731667	316.942500	0.63
2	2003	375.983333	1777.334167	317.631667	0.62
3	2004	377.698333	1776.995833	318.262500	0.53
4	2005	379.983333	1774.180000	318.920000	0.68

In [6]:

```
df.describe()
```

Out[6]:

	year	CO2	CH4	N2O	Temperature
count	22.000000	22.000000	22.000000	22.000000	22.000000
mean	2011.500000	394.154091	1817.905000	325.018220	0.744091
std	6.493587	14.618055	43.282048	5.969065	0.158284
min	2001.000000	371.319167	1771.269167	316.364167	0.530000
25%	2006.250000	382.574375	1778.368750	319.980625	0.632500
50%	2011.500000	392.953333	1805.632917	324.638333	0.680000
75%	2016.750000	406.171875	1848.098542	329.547083	0.887500
max	2022.000000	418.564167	1911.968333	335.662500	1.020000

Data was standarized using StandardScaler class form sklearn. We wanted to have our parameters in the similar range and in not so big scale as they were before. The standard score of a sample x is calculated as: z = (x - u) / s where u is the mean of the training sample, and s is the standard deviation of the training samples.

In [9]:

```
scaler = StandardScaler()
df[['CO2', 'CH4', 'N2O']] = scaler.fit_transform(df[['CO2', 'CH4', 'N2O']])

print(df)
```

```
    year       CO2       CH4       N2O  Temperature
0   2001 -1.598865 -1.102843 -1.483935         0.54
1   2002 -1.449492 -1.068258 -1.384767         0.63
2   2003 -1.272287 -0.959418 -1.266593         0.62
```

```
2   2003 -1.272287 -0.959418 -1.266593        0.62
3   2004 -1.152206 -0.967419 -1.158423        0.53
4   2005 -0.992214 -1.034008 -1.045679        0.68
5   2006 -0.844650 -1.015326 -0.890782        0.64
6   2007 -0.709223 -0.861555 -0.782897        0.67
7   2008 -0.582723 -0.728771 -0.602136        0.54
8   2009 -0.455931 -0.575395 -0.469245        0.66
9   2010 -0.283744 -0.448149 -0.312204        0.73
10  2011 -0.161270 -0.348670 -0.137302        0.61
11  2012 -0.006880 -0.231750  0.007021        0.65
12  2013  0.180886 -0.105253  0.159489        0.68
13  2014  0.326174  0.114358  0.355969        0.75
14  2015  0.480156  0.389186  0.541731        0.90
15  2016  0.718277  0.598510  0.674336        1.02
16  2017  0.882529  0.752518  0.810657        0.92
17  2018  1.019531  0.934489  1.010995        0.85
18  2019  1.225327  1.153114  1.177752        0.98
19  2020  1.406266  1.447059  1.375375        1.02
20  2021  1.561182  1.833171  1.595432        0.85
21  2022  1.709154  2.224407  1.825205        0.90
```

Checking id data is suitable for normal model

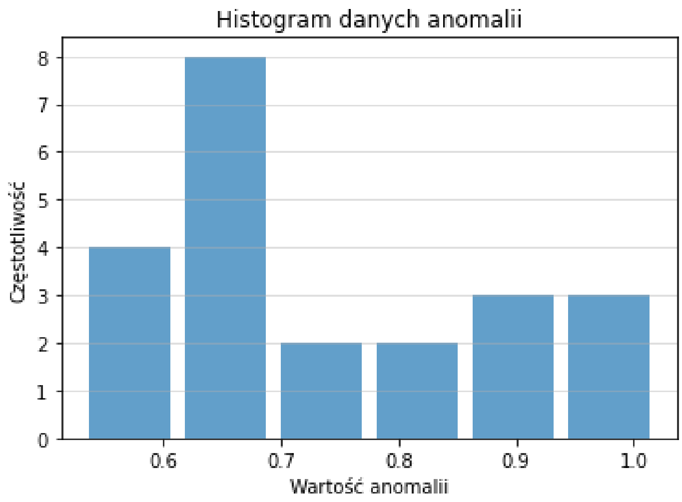
In [68]:

```
normality_test = stats.normaltest(np.array(df['Temperature']))

# Wyświetlanie wyników testu
print("Statystyka testu: ", normality_test.statistic)
print("Wartość p-wartości: ", normality_test.pvalue)

# Sprawdzenie interpretacji wyników testu
alpha = 0.05
if normality_test.pvalue < alpha:
    print("Dane nie pochodzą z rozkładu normalnego")
else:
    print("Dane są zgodne z rozkładem normalnym")
```

```
Statystyka testu:  3.3440751699526974
Wartość p-wartości:  0.18786388675829657
Dane są zgodne z rozkładem normalnym
```

In [10]:

```
plt.hist(np.array(df['Temperature']), bins='auto', alpha=0.7, rwidth=0.85)
plt.grid(axis='y', alpha=0.5)
plt.xlabel('Anomaly value')
plt.ylabel('Frequency')
plt.title('Histogram of anomaly data')
plt.show()
```



From histogram above we can see that the data doesn't have strong resemblance for any of distributions. At first we thought about Gamma distribution but after some actions the results weren't satisfying so we decided

to go with Normal Distribution

Model 1 - Normal Distribution

Our first approach was to create model with Normal Distribution. It is characterized by its symmetric bell shaped curve. It is defined by two parameters: mean and standard deviation. It is symmetric around mean value so it represents center of distribution while the standard deviation determines the spread of the data.

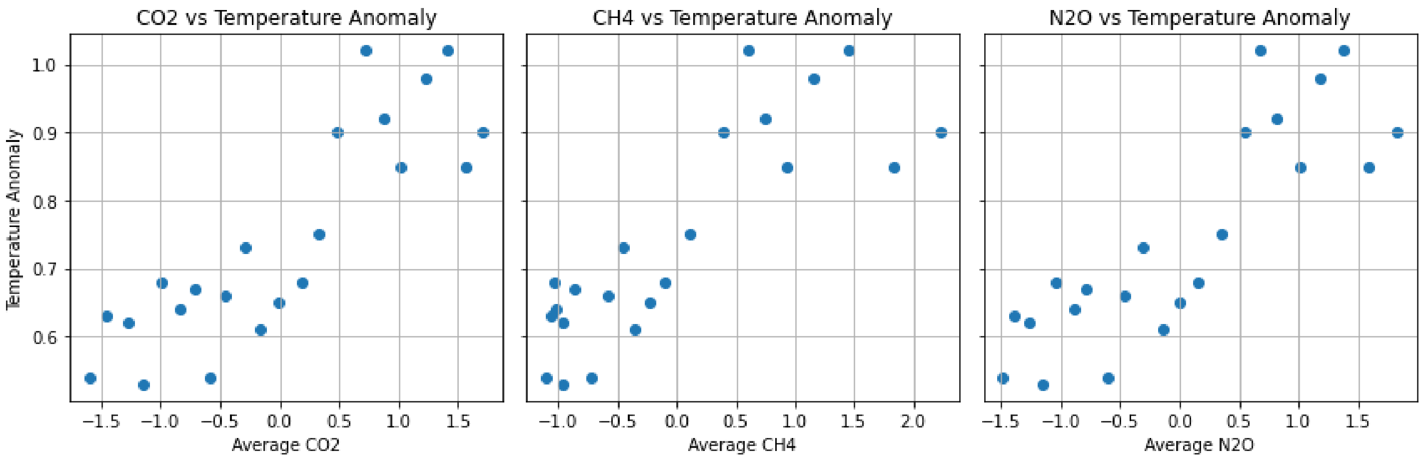
Below plots shows that correlation between our parameters and temperature data can be considered to be linear and that is why we have chosen this approach in the model.

Standard Bayesian model: $outcome_i$ **~ Normal(**μ_i**,** σ**)** μ_i **=** α**+** β ***** $predictor_i$ α **~ Normal(a,b)** β **~ Normal(c,d)** σ **~ Normal(f,g)**

The sum of multiplications beta and predictors is added two more times to the equation for every single predictor. α **is usually the mean value of the data and** σ **parameter is standard deviation.**

In [73]:

```
fig, axs = plt.subplots(1, 3, sharey=True, figsize=(12, 4))

axs[0].scatter(df['CO2'], df['Temperature'])
axs[0].set_xlabel('Average CO2')
axs[0].set_ylabel('Temperature Anomaly')
axs[0].set_title('CO2 vs Temperature Anomaly')
axs[0].grid()

axs[1].scatter(df['CH4'], df['Temperature'])
axs[1].set_xlabel('Average CH4')
axs[1].set_title('CH4 vs Temperature Anomaly')
axs[1].grid()

axs[2].scatter(df['N2O'], df['Temperature'])
axs[2].set_xlabel('Average N2O')
axs[2].set_title('N2O vs Temperature Anomaly')
axs[2].grid()

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



Prior

Alpha normal distribution is based on the mean value of 'Temperature Anomaly' in the data set and sigma is based on standard deviation of the same data. Beta for every predictor and sigma is also normally distributed. The parameters for distribution of beta where chosen considering the output of this prior model. It was the most

difficult challenge to fit beta parameters properly.

In [74]:

```
%%writefile root/stan_files/temp3_ppc.stan
data {
  int<lower=0> N;
  vector[N] CO2;
  vector[N] CH4;
  vector[N] N2O;
}

generated quantities {
  real alpha = normal_rng(0.7, 0.1);
  real beta_CO2 = normal_rng(0, 0.1);
  real beta_CH4 = normal_rng(0, 0.1);
  real beta_N2O = normal_rng(0, 0.1);
  real sigma = normal_rng(0.1, 0.02);
  vector[N] temperature;

  for (i in 1:N) {
    temperature[i] = normal_rng(alpha + beta_CO2 * CO2[i] + beta_CH4 * CH4[i] + beta_N2O
* N2O[i], sigma);
  }
}
```

```
Overwriting root/stan_files/temp3_ppc.stan
```

In [75]:

```
data_sim={'N':len(df), 'CO2':np.linspace(df.CO2.min(),df.CO2.max(),len(df)),'CH4':np.lins
pace(df.CH4.min(),df.CH4.max(),len(df)),'N2O':np.linspace(df.N2O.min(),df.N2O.max(),len(d
f))}
model_ppc1=CmdStanModel(stan_file='root/stan_files/temp3_ppc.stan')
R = 1000
sim=model_ppc1.sample(data=data_sim,
                      iter_sampling=R,
                      iter_warmup=0,
                      chains=1,
                      refresh=R,
                      fixed_param=True,
                      seed=29042020)
```

```
INFO:cmdstanpy:compiling stan file /root/stan_files/temp3_ppc.stan to exe file /root/stan
_files/temp3_ppc
INFO:cmdstanpy:compiled model executable: /root/stan_files/temp3_ppc
INFO:cmdstanpy:CmdStan start processing
chain 1 |██████████| 00:00 Sampling completed
```

```
INFO:cmdstanpy:CmdStan done processing.
```

In [76]:

```
ppc_df = sim.draws_pd()
ppc_df.head()
```

Out[76]:

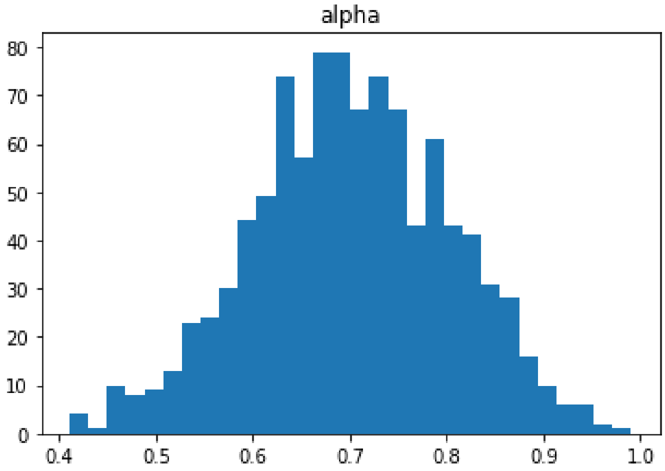
	lp__	accept_stat__	alpha	beta_CO2	beta_CH4	beta_N2O	sigma	temperature[1]	temperature[2]	temperature[3]	...
0	0.0	0.0	0.970817	0.077718	-0.127227	0.012759	0.072212	0.985710	1.048590	1.119710	...
1	0.0	0.0	0.635055	0.118083	-0.014296	-0.015998	0.112775	0.534230	0.179542	0.630123	...
2	0.0	0.0	0.699322	-0.000371	0.121486	0.075957	0.114794	0.588463	0.623650	0.606261	...
3	0.0	0.0	0.606081	0.106421	-0.002303	0.037027	0.088234	0.462742	0.446727	0.355780	...
4	0.0	0.0	0.577813	0.141649	-0.018822	-0.016031	0.112103	0.440043	0.511638	0.191112	...

5 rows × 29 columns

5 rows × 29 columns

In [84]:

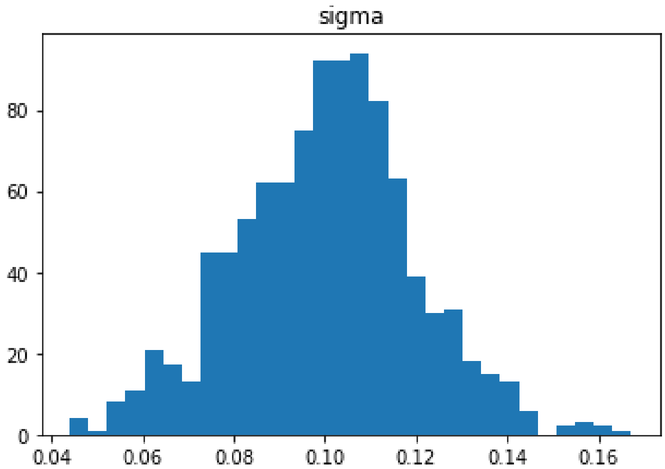
```
plt.hist(ppc_df['alpha'], bins=30)
plt.title('alpha')
plt.show()
```



Parameter alpha is in the right range and is wider on the center which is good.

In [85]:

```
plt.hist(ppc_df['sigma'], bins=30)
plt.title('sigma')
plt.show()
```

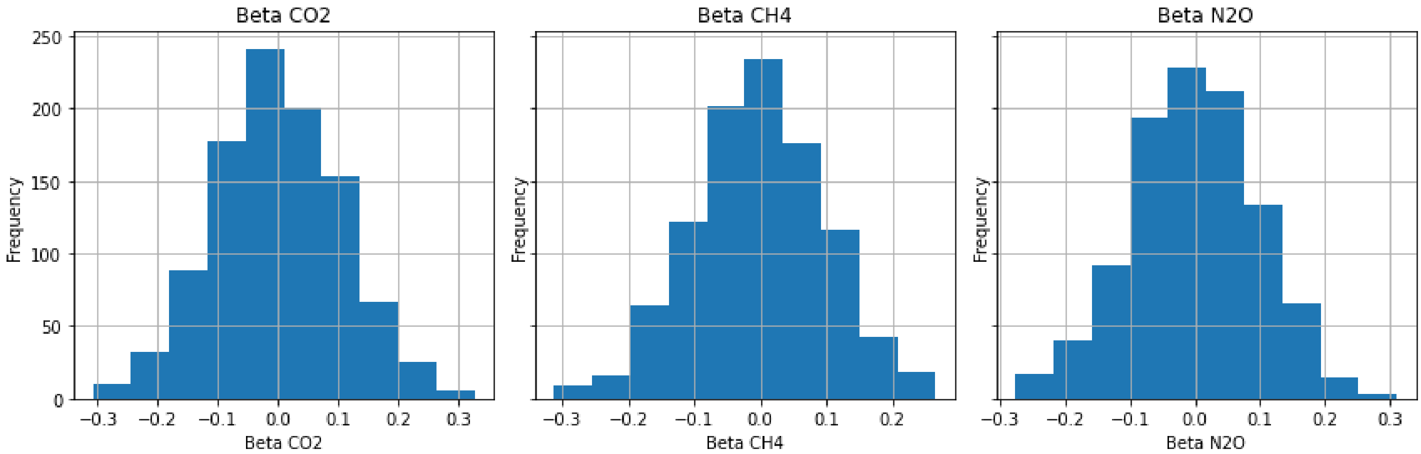


Parameter sigma looks also okay with tha values at the meant range

In [87]:

```
fig, axs = plt.subplots(1, 3, sharey=True, figsize=(12, 4))

axs[0].hist(ppc_df['beta_CO2'])
axs[0].set_xlabel('Beta CO2')
axs[0].set_ylabel('Frequency')
axs[0].set_title('Beta CO2')
axs[0].grid()

axs[1].hist(ppc_df['beta_CH4'])
axs[1].set_xlabel('Beta CH4')
axs[1].set_ylabel('Frequency')
axs[1].set_title('Beta CH4')
axs[1].grid()

axs[2].hist(ppc_df['beta_N2O'])
axs[2].set_xlabel('Beta N2O')
```

```
axs[2].set_ylabel('Frequency')
axs[2].set_title('Beta N2O')
axs[2].grid()

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



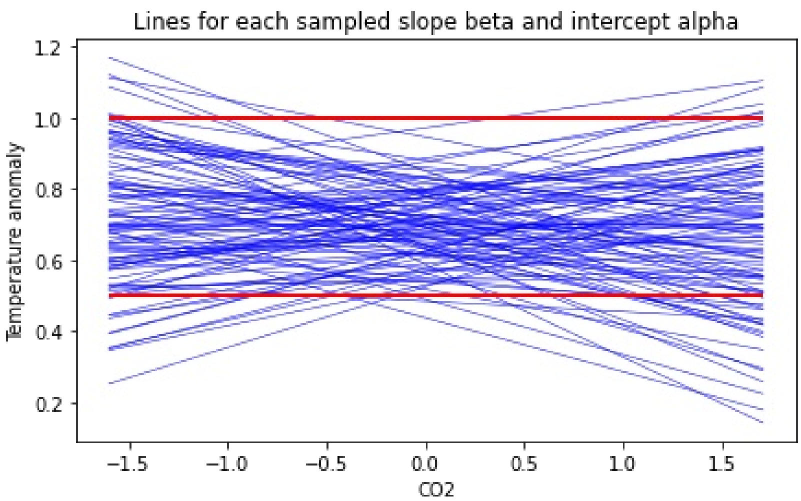
As we put the same values for beta distribution parameters for predictors their histograms look almost the same. We wanted for all the predictors to have the same impact on our model.

In [78]:

```
fig, axes = plt.subplots(1,1,figsize=(7,4))

beta_humid = sim.stan_variable('beta_CO2')
alpha_humid = sim.stan_variable('alpha')
for i in range(100):
    axes.plot(df['CO2'], alpha_humid[i]+beta_humid[i]*np.array(df['CO2']), linewidth = 0
.5, color='b')
plt.title("Lines for each sampled slope beta and intercept alpha")
axes.set_xlabel('CO2')
axes.set_ylabel('Temperature anomaly')
axes.hlines([0.5, 1],xmin = df['CO2'].min(), xmax = df['CO2'].max(), linestyles = '-',li
newidth = 2, color = 'r')
axes.annotate(text='max',xy=(80,320), weight = 'bold', color = 'r', fontsize = 15)
axes.annotate(text='min',xy=(80,20), weight = 'bold', color = 'r', fontsize = 15)
```

Out[78]:

```
Text(80, 20, 'min')
```



This model fits just fine. Those lines that are under or above min and max values on the plot are acceptable because temperature anomaly can go below 0.5 (even below 0) and above 1. After fitting the data to the model everything should be between those lines perfectly.

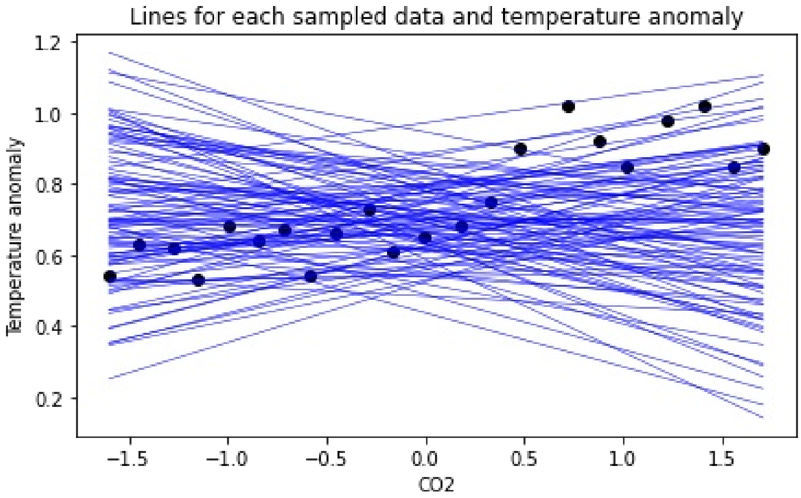
Lets see how the actual temperature data corresponds with the data from prior.

In [81]:

```
fig, axes = plt.subplots(1,1,figsize=(7,4))

beta_humid = sim.stan_variable('beta_CO2')
alpha_humid = sim.stan_variable('alpha')
for i in range(100):
    axes.plot(df['CO2'], alpha_humid[i]+beta_humid[i]*np.array(df['CO2']), linewidth = 0
.5, color='b')
plt.title("Lines for each sampled data and temperature anomaly")
axes.scatter(df['CO2'], df['Temperature'], color= 'black')
axes.set_xlabel('CO2')
axes.set_ylabel('Temperature anomaly')
axes.annotate(text='max',xy=(80,320), weight = 'bold', color = 'r', fontsize = 15)
axes.annotate(text='min',xy=(80,20), weight = 'bold', color = 'r', fontsize = 15)
```

Out[81]:

```
Text(80, 20, 'min')
```



Posterior predictive check

Fitting model to data

In [48]:

```
%%writefile root/stan_files/temp4_ppc.stan

data {
    int<lower=0> N;
    vector[N] temp;
    vector[N] CO2;
    vector[N] CH4;
    vector[N] N2O;
}

parameters {
    real<lower=0> alpha;
    real<lower=0> sigma;
    real<lower=0> beta_CO2;
    real<lower=0> beta_CH4;
    real<lower=0> beta_N2O;
}

transformed parameters {
    vector[N] mean;
    for (i in 1:N) {
        mean[i] = alpha + beta_CO2 * CO2[i] + beta_CH4 * CH4[i] + beta_N2O * N2O[i];
    }
}
```

```
model {
    alpha ~ normal(0.7, 0.1);
    sigma ~ normal(0.1, 0.02);
    beta_CO2 ~ normal(0, 0.1);
    beta_CH4 ~ normal(0, 0.1);
    beta_N2O ~ normal(0, 0.1);
    for (i in 1:N) {
        temp[i] ~ normal(mean[i], sigma);
    }
}

generated quantities {
    vector[N] temp_;
    vector[N] log_lik;
    for (i in 1:N) {
        temp_[i] = normal_rng(mean[i], sigma);
        log_lik[i] = normal_lpdf(temp_[i]|mean[i], sigma);
    }
}
```

```
Overwriting root/stan_files/temp4_ppc.stan
```

In [52]:

```
model_1_fit=CmdStanModel(stan_file='root/stan_files/temp4_ppc.stan')
N = len(df)
data_fit = {'N': N, 'CO2': df.CO2.values[:N], 'temp': df.Temperature.values[:N], 'CH4':
df.CH4.values[:N], 'N2O': df.N2O.values[:N]}
fit=model_1_fit.sample(data=data_fit,seed=28052020)
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
INFO:cmdstanpy:CmdStan start processing
chain 1 |          | 00:00 Status


chain 1 |█         | 00:00 Iteration:  100 / 2000 [  5%]  (Warmup)


chain 1 |██████    | 00:00 Iteration: 1200 / 2000 [ 60%]  (Sampling)


chain 1 |██████████| 00:00 Sampling completed
chain 2 |██████████| 00:00 Sampling completed
chain 3 |██████████| 00:00 Sampling completed
chain 4 |██████████| 00:00 Sampling completed
```

```
INFO:cmdstanpy:CmdStan done processing.
```

There were no issues with the sampling

In [50]:

```
df_ = fit.draws_pd()
df_.head()
```

Out[50]:

	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__	alpha	sigma	beta_CO2	...	log
0	27.9625	0.985636	0.167017	5.0	31.0	0.0	-27.1543	0.742672	0.098222	0.037101	...	0.
1	30.4195	0.999059	0.167017	5.0	31.0	0.0	-27.0724	0.747539	0.082064	0.046266	...	1.
2	31.0292	0.994093	0.167017	5.0	31.0	0.0	-29.3316	0.736152	0.099784	0.070291	...	0.
3	29.5738	0.978468	0.167017	4.0	15.0	0.0	-28.5692	0.707534	0.084646	0.040462	...	1.
4	28.9220	0.907824	0.167017	4.0	15.0	0.0	-26.5229	0.718149	0.100418	0.032382	...	0.

	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__	alpha	sigma	beta_CO2	...	log

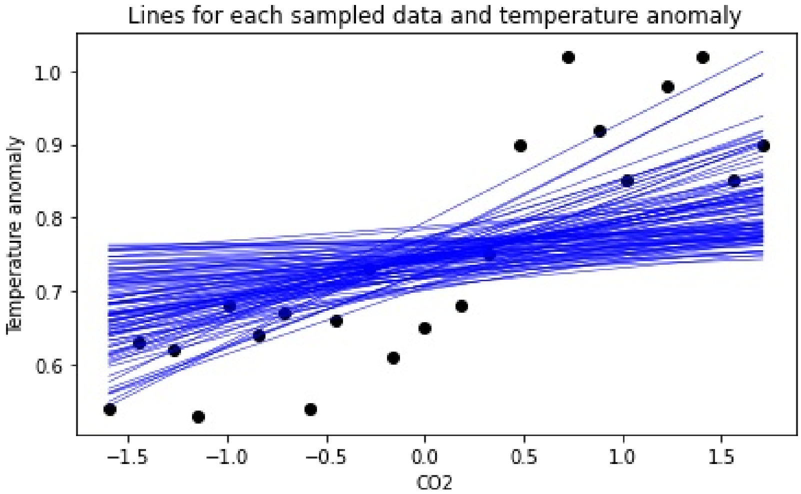
5 rows × 78 columns

In [51]:

```
fig, axes = plt.subplots(1,1,figsize=(7,4))

beta_humid = fit.stan_variable('beta_CO2')
alpha_humid = fit.stan_variable('alpha')
for i in range(100):
    axes.plot(df['CO2'], alpha_humid[i]+beta_humid[i]*np.array(df['CO2']), linewidth = 0
.5, color='b')
plt.title("Lines for each sampled data and temperature anomaly")
axes.scatter(df['CO2'], df['Temperature'], color= 'black')
axes.set_xlabel('CO2')
axes.set_ylabel('Temperature anomaly')
axes.annotate(text='max',xy=(80,320), weight = 'bold', color = 'r', fontsize = 15)
axes.annotate(text='min',xy=(80,20), weight = 'bold', color = 'r', fontsize = 15)
```

Out[51]:

```
Text(80, 20, 'min')
```

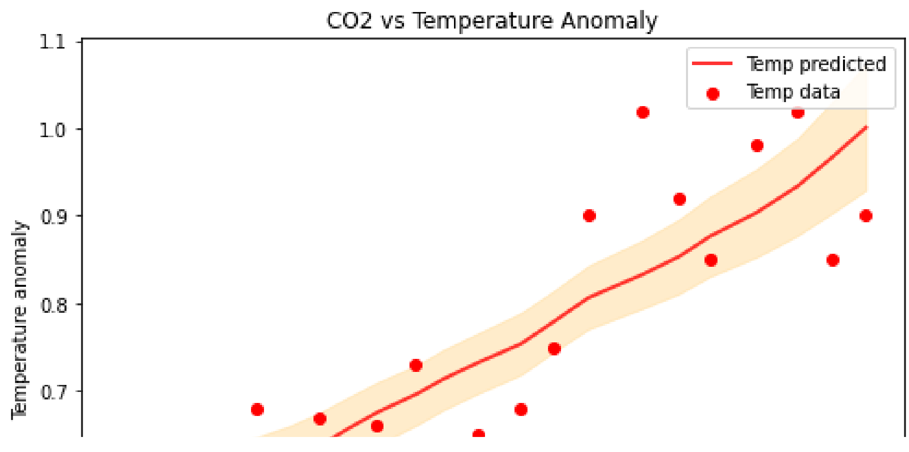


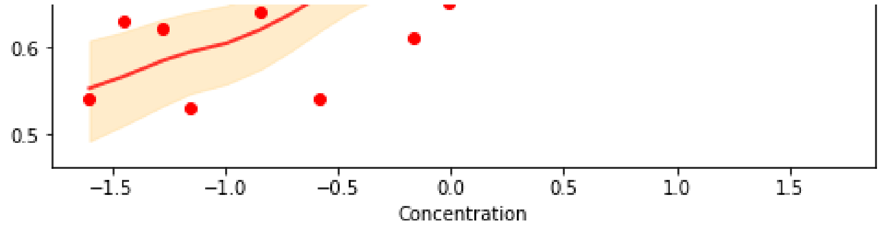
Now the lines are more adjusted to actual data. There are less data points on the center and so there are less wide lines there. As the data is spread at the ends so are the lines

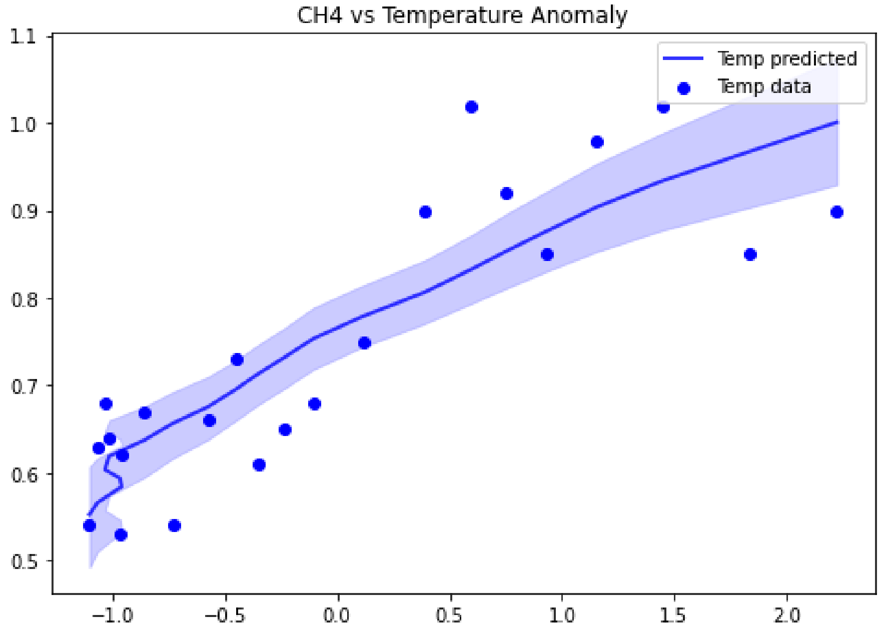
In [103]:

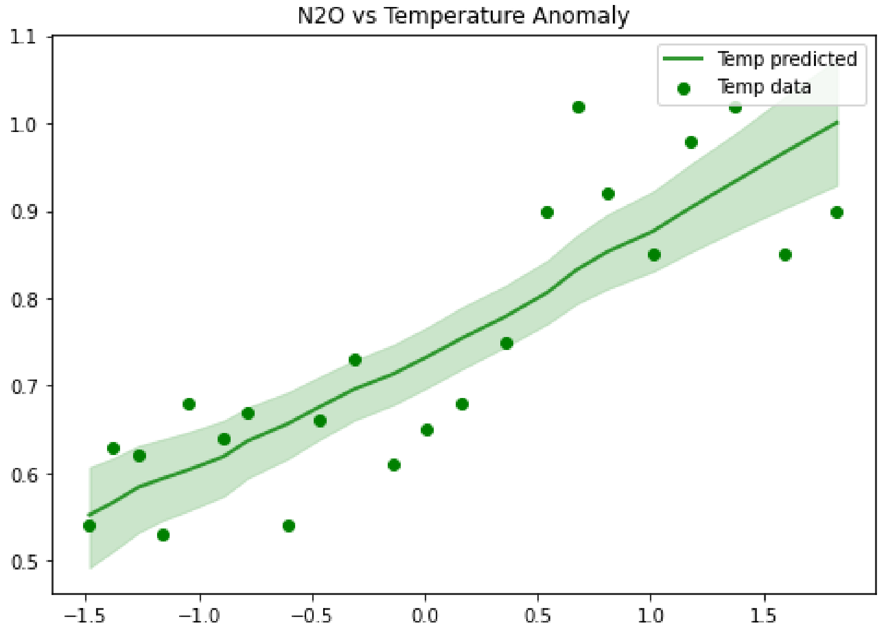
```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

CO2 = np.array(df['CO2'])
CH4 = np.array(df['CH4'])
N2O = np.array(df['N2O'])
Temperature = np.array(df['Temperature'])
mu_CO2 = fit.stan_variable('mean')
mu_CH4 = fit.stan_variable('mean')
mu_N2O = fit.stan_variable('mean')


fig, ax = plt.subplots(3, 1, figsize=(7 ,15))

ax[0].fill_between(
    CO2,
    np.percentile(mu_CO2, 5, axis=0),
    np.percentile(mu_CO2, 95, axis=0),
    color=1 - 0.4 * (1 - np.array(mpl.colors.to_rgb('orange'))),
    alpha=0.5
)

ax[1].fill_between(
```

```
    CH4,
    np.percentile(mu_CH4, 5, axis=0),
    np.percentile(mu_CH4, 95, axis=0),
    color=1 - 0.4 * (1 - np.array(mpl.colors.to_rgb('blue'))),
    alpha=0.5
)

ax[2].fill_between(
    N2O,
    np.percentile(mu_N2O, 5, axis=0),
    np.percentile(mu_N2O, 95, axis=0),
    color=1 - 0.4 * (1 - np.array(mpl.colors.to_rgb('green'))),
    alpha=0.5
)

ax[0].plot(
    CO2,
    np.percentile(mu_CO2, 50, axis=0),
    color='red',
    linewidth=2,
    alpha=0.8,
    label='Temp predicted'
)
ax[1].plot(
    CH4,
    np.percentile(mu_CH4, 50, axis=0),
    color='blue',
    linewidth=2,
    alpha=0.8,
    label='Temp predicted'
)
ax[2].plot(
    N2O,
    np.percentile(mu_N2O, 50, axis=0),
    color='green',
    linewidth=2,
    alpha=0.8,
    label='Temp predicted'
)

ax[0].scatter(CO2, Temperature, color='red', label='Temp data')
ax[1].scatter(CH4, Temperature, color='blue', label='Temp data')
ax[2].scatter(N2O, Temperature, color='green', label='Temp data')

ax[0].set_xlabel('Concentration')
ax[0].set_ylabel('Temperature anomaly')


ax[0].legend()
ax[1].legend()
ax[2].legend()

ax[0].set_title('CO2 vs Temperature Anomaly')
ax[1].set_title('CH4 vs Temperature Anomaly')
ax[2].set_title('N2O vs Temperature Anomaly')
plt.tight_layout()
plt.show()
```









Shown output data is consistent with the provided data of temperatures anomaly. For each gases the output mean fits great.

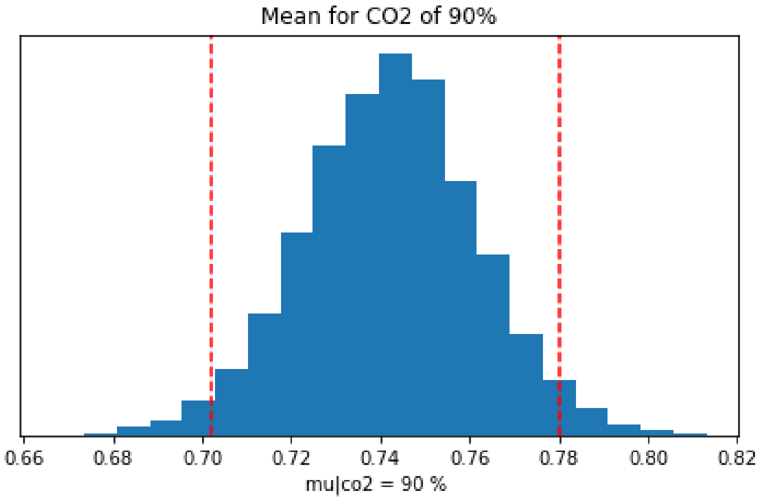
Marginal Distribution

In [111]:

```
alpha_post =  fit.stan_variable('alpha')
beta_post = fit.stan_variable('beta_CO2')
mu_post = fit.stan_variable('mean')

mu90 = alpha_post+beta_post*(np.mean(df['CO2']))
mu_95p = az.hdi(mu90,.95)
fig, ax = plt.subplots(1, 1, figsize=(7, 4))
ax.hist(mu90,bins=20,density=True)
plt.axvline(mu_95p[0], linestyle = '--', color = 'r')
plt.axvline(mu_95p[1], linestyle = '--', color = 'r')
ax.set_title('Mean for CO2 of 90%')
```

```
ax.set_yticks(())
ax.set_xlabel('mu|co2 = 90 % ')
plt.show()
print('Mean: {:4.2f}'.format(np.mean(mu90)))
print('95% confidence interval: ',['{:4.2f}'.format(k) for k in az.hdi(mu90,.95)])
```

```
0.7803419999999998
```




```
Mean: 0.74
95% confidence interval:  ['0.70', '0.78']
```

From above histogram we can see that on 90% propability the temperature anomaly will be in range 0.7 and 0.78. The other gases are not gonna be checked due to the similar data with CO2

Model 2 - Student_t distribution

Our second model approach will be to apply a Student-t distribution

The Student-t distribution is similar in shape to the Gaussian but has heavier tails. It is symmetric and bell-shaped, with single peak. This model have more probability mass and allows to better represnt data that may have outliers.

The t-distribution is characterized by called degrees of freedom ν (nu) which determines the shape of the distribution. The more degrees of freedom the closer it is to the Gaussian distribution. The model have two other parameters: μ (mu) – location, σ (sigma)- scale witch are similar to the Gaussian model.

For this model we used the linear relation too.

Prior

Alpha, beta and sigma data were selected similarly as in the first model. The main difference is new nu parameter which we choose based on internet: https://statmodeling.stat.columbia.edu/2015/05/17/do-we-have-any-recommendations-for-priors-for-student_ts-degrees-of-freedom-parameter/

In [12]:

```
%%writefile /home/temp7_ppc_prior.stan
data {
  int<lower=0> N;
  vector[N] CO2;
  vector[N] CH4;
  vector[N] N2O;
}

generated quantities {
  real sigma = normal_rng(0.1,0.05);
```

```
  real nu = gamma_rng(2, 0.1);
  real alpha = normal_rng(0.7, 0.1);
  real beta_CO2 = normal_rng(0, 0.1);
  real beta_CH4 = normal_rng(0, 0.1);
  real beta_N2O = normal_rng(0, 0.1);
  vector[N] temperature;

  for (i in 1:N) {
    temperature[i] = student_t_rng(nu, alpha + beta_CO2 * CO2[i] + beta_CH4 * CH4[i] + b
eta_N2O * N2O[i], sigma);
  }
}
```

```
Overwriting /home/temp7_ppc_prior.stan
```

In [21]:

```
data_sim={'N':len(df), 'CO2':np.linspace(df.CO2.min(),df.CO2.max(),len(df)),'CH4':np.lins
pace(df.CH4.min(),df.CH4.max(),len(df)),'N2O':np.linspace(df.N2O.min(),df.N2O.max(),len(d
f))}
```

In [15]:

```
model_ppc7_p=CmdStanModel(stan_file='/home/temp7_ppc_prior.stan')
R = 1000
sim7=model_ppc7_p.sample(data=data_sim,
                         iter_sampling=R,
                         iter_warmup=0,
                         chains=1,
                         refresh=R,
                         fixed_param=True,
                         seed=29042020)
df_7_p = sim7.draws_pd()
df_7_p.head()
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
INFO:cmdstanpy:CmdStan start processing
chain 1 |██████████| 00:00 Sampling completed
```

```
INFO:cmdstanpy:CmdStan done processing.
```

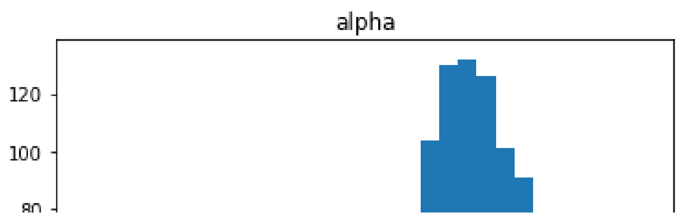
Out[15]:

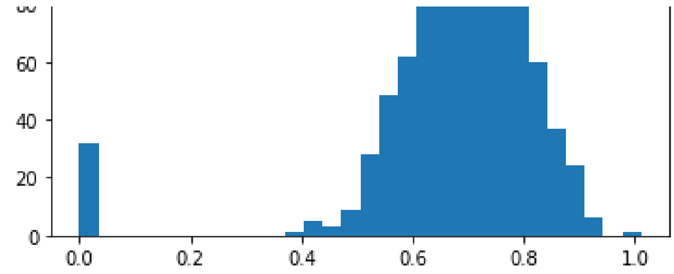
	lp__	accept_stat__	sigma	nu	alpha	beta_CO2	beta_CH4	beta_N2O	temperature[1]	temperature[2]	...	tempe
0	0.0	0.0	0.235409	10.8336	0.693634	-0.127227	0.012759	-0.138942	1.146710	1.216910	...	
1	0.0	0.0	0.122755	20.6335	0.643012	0.104238	0.116276	0.188285	0.299489	0.171137	...	
2	0.0	0.0	0.048386	26.1987	0.650050	0.024688	-0.058932	-0.001650	0.727714	0.671177	...	
3	0.0	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
4	0.0	0.0	0.101699	9.8558	0.743006	0.110379	-0.049400	-0.065600	0.685745	0.779902	...	

5 rows × 30 columns

In [16]:

```
plt.hist(df_7_p['alpha'], bins=30)
plt.title('alpha')
plt.show()
```

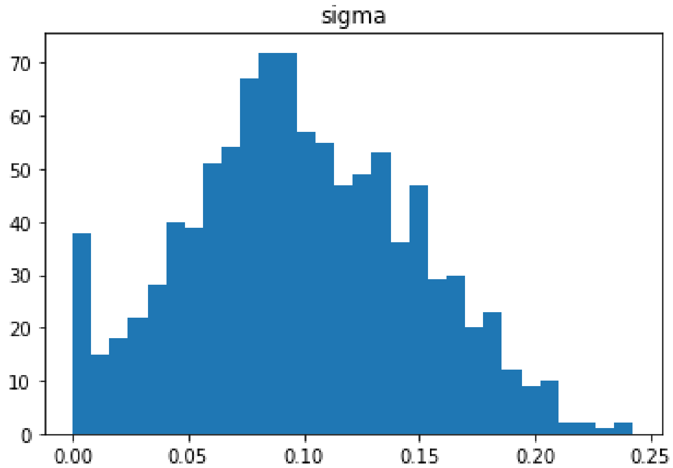




Parameter alpha is in right range and most of the values are in the middle but some stick out.

In [17]:

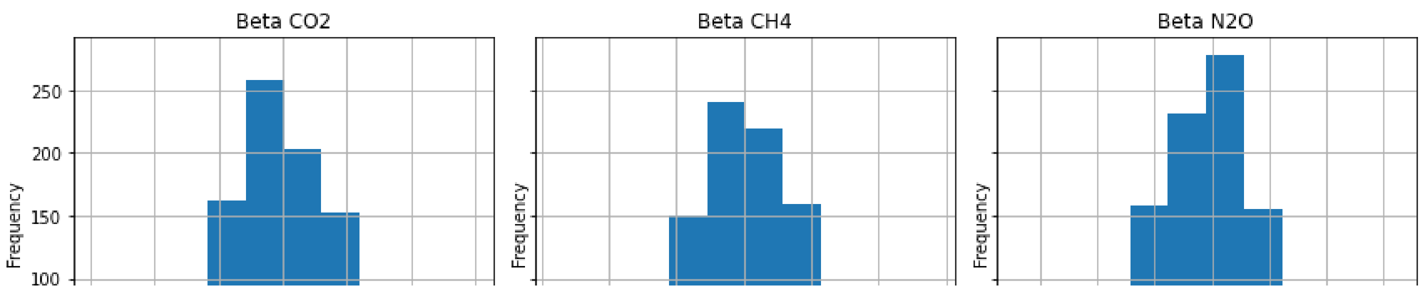
```
plt.hist(df_7_p['sigma'], bins=30)
plt.title('sigma')
plt.show()
```

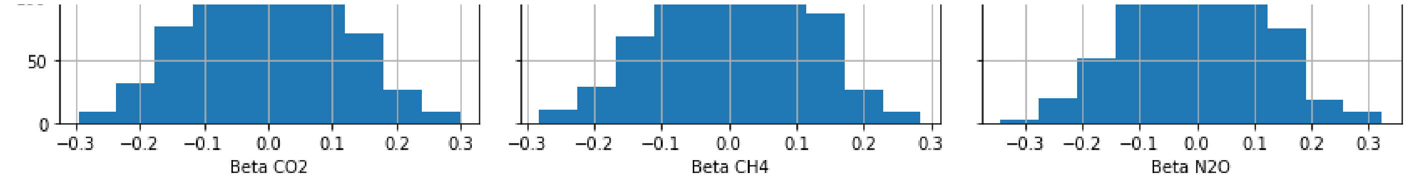


Parameter sigma have values in the middle and this is okay

In [18]:

```
fig, axs = plt.subplots(1, 3, sharey=True, figsize=(12, 4))

axs[0].hist(df_7_p['beta_CO2'])
axs[0].set_xlabel('Beta CO2')
axs[0].set_ylabel('Frequency')
axs[0].set_title('Beta CO2')
axs[0].grid()

axs[1].hist(df_7_p['beta_CH4'])
axs[1].set_xlabel('Beta CH4')
axs[1].set_ylabel('Frequency')
axs[1].set_title('Beta CH4')
axs[1].grid()

axs[2].hist(df_7_p['beta_N2O'])
axs[2].set_xlabel('Beta N2O')
axs[2].set_ylabel('Frequency')
axs[2].set_title('Beta N2O')
axs[2].grid()

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





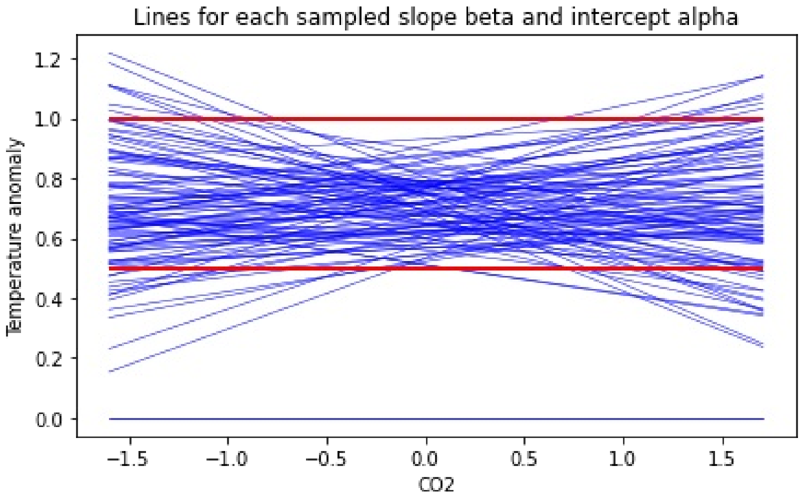
In this model we put the same values for beta distribution like in the first model and the historams look almost the same

In [19]:

```
fig, axes = plt.subplots(1,1,figsize=(7,4))

beta_humid = sim7.stan_variable('beta_CO2')
alpha_humid = sim7.stan_variable('alpha')
for i in range(100):
    axes.plot(df['CO2'], alpha_humid[i]+beta_humid[i]*np.array(df['CO2']), linewidth = 0
.5, color='b')
plt.title("Lines for each sampled slope beta and intercept alpha")
axes.set_xlabel('CO2')
axes.set_ylabel('Temperature anomaly')
axes.hlines([0.5, 1],xmin = df['CO2'].min(), xmax = df['CO2'].max(), linestyles = '-',li
newidth = 2, color = 'r')
axes.annotate(text='max',xy=(80,320), weight = 'bold', color = 'r', fontsize = 15)
axes.annotate(text='min',xy=(80,20), weight = 'bold', color = 'r', fontsize = 15)
```

Out[19]:

```
Text(80, 20, 'min')
```

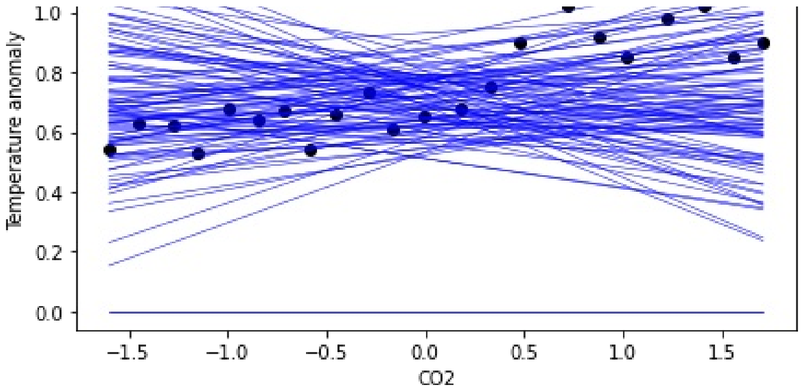


In [20]:

```
fig, axes = plt.subplots(1,1,figsize=(7,4))

beta_humid = sim7.stan_variable('beta_CO2')
alpha_humid = sim7.stan_variable('alpha')
for i in range(100):
    axes.plot(df['CO2'], alpha_humid[i]+beta_humid[i]*np.array(df['CO2']), linewidth = 0
.5, color='b')
plt.title("Lines for each sampled data and temperature anomaly")
axes.scatter(df['CO2'], df['Temperature'], color= 'black')
axes.set_xlabel('CO2')
axes.set_ylabel('Temperature anomaly')
axes.annotate(text='max',xy=(80,320), weight = 'bold', color = 'r', fontsize = 15)
axes.annotate(text='min',xy=(80,20), weight = 'bold', color = 'r', fontsize = 15)
```

Out[20]:

```
Text(80, 20, 'min')
```



In this two plots we can see that the actual temperature data correspods with data from priors and the most of samples fits.

Prosterior predictive check

Fitting model to data

In [23]:

```
%%writefile /home/temp7_ppc.stan
data {
  int<lower=0> N; // number of data points
  vector[N] CO2;
  vector[N] CH4;
  vector[N] N2O;
  vector[N] temp;
}
parameters {
  real<lower=0> alpha;
  real<lower=0> beta_CO2;
  real<lower=0> beta_CH4;
  real<lower=0> beta_N2O;
  real<lower=0> sigma;
  real<lower=1, upper=80> nu;
}
transformed parameters {
  vector[N] mu;
  mu = alpha + beta_CO2 * CO2 + beta_CH4 * CH4 + beta_N2O * N2O;
}
model {
  nu ~ gamma(2, 0.1); // found this online: Juarez and Steel(2010)
  temp ~ student_t(nu, mu, sigma);
  alpha ~ normal(0.7, 0.1);
  beta_CO2 ~ normal(0, 0.1);
  beta_CH4 ~ normal(0, 0.1);
  beta_N2O ~ normal(0, 0.1);
  sigma ~ normal(0.1,0.05);

}
generated quantities {
    vector[N] temp_i;
    vector[N] log_lik;
    for (i in 1:N) {
        temp_i[i] = student_t_rng(nu,mu[i],sigma);
        log_lik[i] = student_t_lpdf(temp[i] | nu, mu[i], sigma);
    }
}
```

```
Overwriting /home/temp7_ppc.stan
```

In [27]:

```
N = len(df)
data_fit = {'N':N, 'CO2': df.CO2.values[:N], 'temp': df.Temperature.values[:N], 'CH4': d
```

```
f.CH4.values[:N], 'N2O': df.N2O.values[:N]}
```

In [28]:

```
model_ppc7=CmdStanModel(stan_file='/home/temp7_ppc.stan')
fit7=model_ppc7.sample(data=data_fit,seed=28052020)
```

```
INFO:cmdstanpy:compiling stan file /home/temp7_ppc.stan to exe file /home/temp7_ppc
INFO:cmdstanpy:compiled model executable: /home/temp7_ppc
INFO:cmdstanpy:CmdStan start processing
chain 1 |          | 00:00 Status


chain 1 |          | 00:00 Status






chain 1 |          | 00:01 Iteration:    1 / 2000 [  0%]  (Warmup)


chain 1 |          | 00:01 Iteration:  200 / 2000 [ 10%]  (Warmup)

chain 1 |          | 00:01 Iteration:  400 / 2000 [ 20%]  (Warmup)


chain 1 |          | 00:01 Iteration:  600 / 2000 [ 30%]  (Warmup)

chain 1 |          | 00:01 Iteration:  800 / 2000 [ 40%]  (Warmup)


chain 1 |          | 00:01 Iteration: 1001 / 2000 [ 50%]  (Sampling)


chain 1 |          | 00:02 Iteration: 1200 / 2000 [ 60%]  (Sampling)


chain 1 |          | 00:02 Iteration: 1400 / 2000 [ 70%]  (Sampling)

chain 1 |          | 00:02 Iteration: 1500 / 2000 [ 75%]  (Sampling)

chain 1 |          | 00:02 Iteration: 1600 / 2000 [ 80%]  (Sampling)





chain 1 |          | 00:03 Iteration: 1800 / 2000 [ 90%]  (Sampling)
chain 1 |          | 00:03 Sampling completed
chain 2 |          | 00:03 Sampling completed

chain 3 |          | 00:03 Sampling completed
chain 4 |          | 00:03 Sampling completed
```

```
INFO:cmdstanpy:CmdStan done processing.
```

In [29]:

```
df_7 = fit7.draws_pd()
df_7.head()
```

Out[29]:

	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__	alpha	beta_CO2	beta_CH4	...	log
0	25.7689	0.998325	0.161263	5.0	31.0	0.0	-24.0963	0.728749	0.093085	0.020597	...	
1	25.7824	0.915042	0.161263	4.0	31.0	0.0	-24.4012	0.739005	0.076416	0.029738	...	
2	26.5245	0.846326	0.161263	4.0	15.0	0.0	-23.1515	0.740603	0.012135	0.044226	...	
3	25.5080	0.970196	0.161263	4.0	31.0	0.0	-24.3903	0.720865	0.011273	0.081586	...	
4	26.0037	0.991388	0.161263	5.0	31.0	0.0	-23.2490	0.713015	0.029480	0.059188	...	

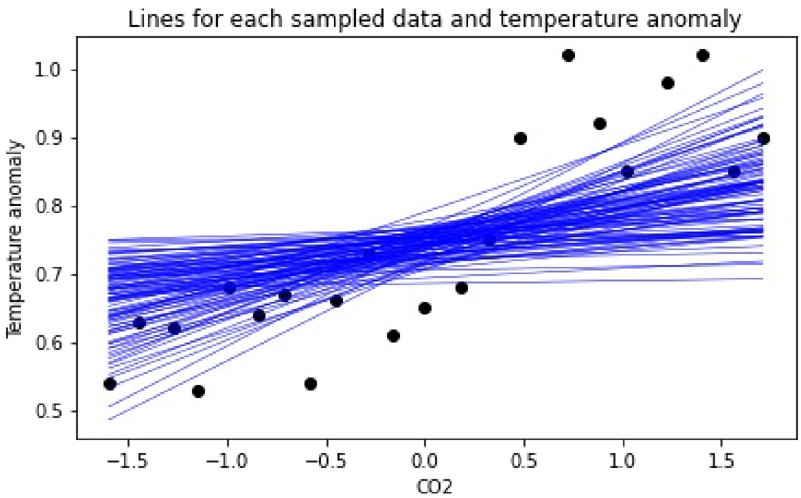
5 rows × 79 columns

In [30]:

```
fig, axes = plt.subplots(1,1,figsize=(7,4))

beta_humid = fit7.stan_variable('beta_CO2')
alpha_humid = fit7.stan_variable('alpha')
for i in range(100):
    axes.plot(df['CO2'], alpha_humid[i]+beta_humid[i]*np.array(df['CO2']), linewidth = 0
.5, color='b')
plt.title("Lines for each sampled data and temperature anomaly")
axes.scatter(df['CO2'], df['Temperature'], color= 'black')
axes.set_xlabel('CO2')
axes.set_ylabel('Temperature anomaly')
axes.annotate(text='max',xy=(80,320), weight = 'bold', color = 'r', fontsize = 15)
axes.annotate(text='min',xy=(80,20), weight = 'bold', color = 'r', fontsize = 15)
```

Out[30]:

```
Text(80, 20, 'min')
```

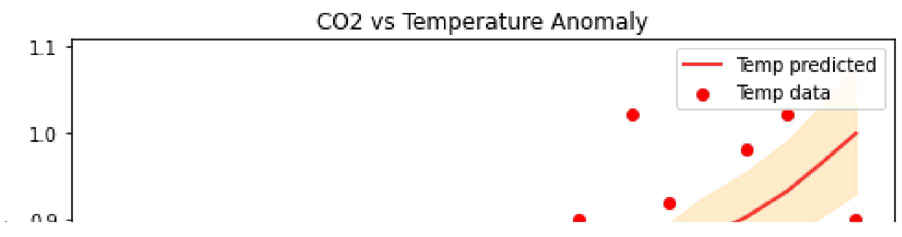


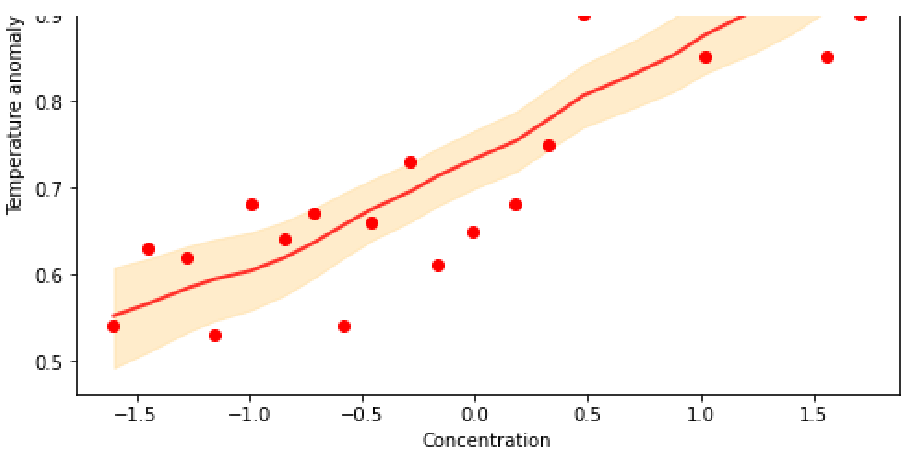
In this plot we can see lines adjusted to actual data. In the model we can see more spread at the end of the x-axis than in Gaussian model and model fits better.

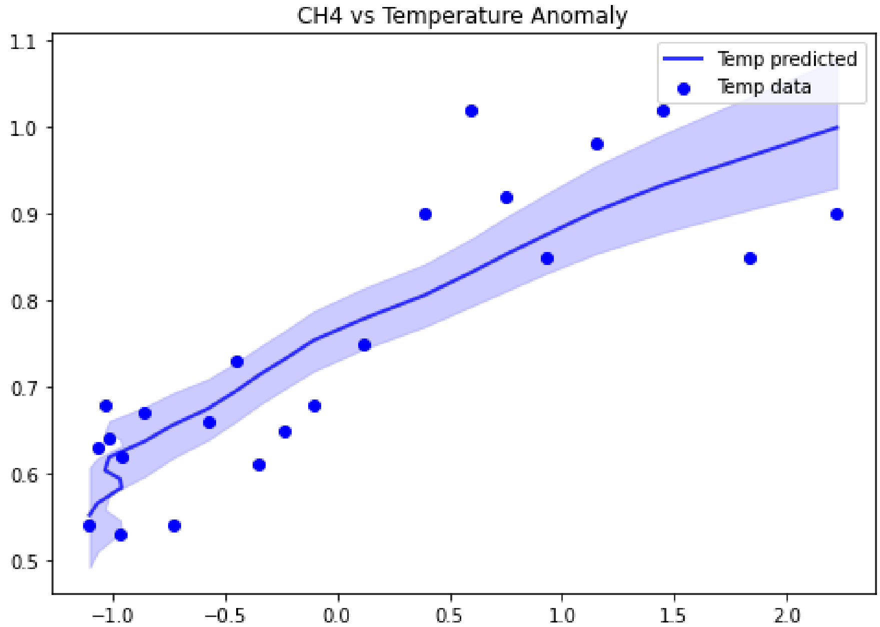
In [33]:

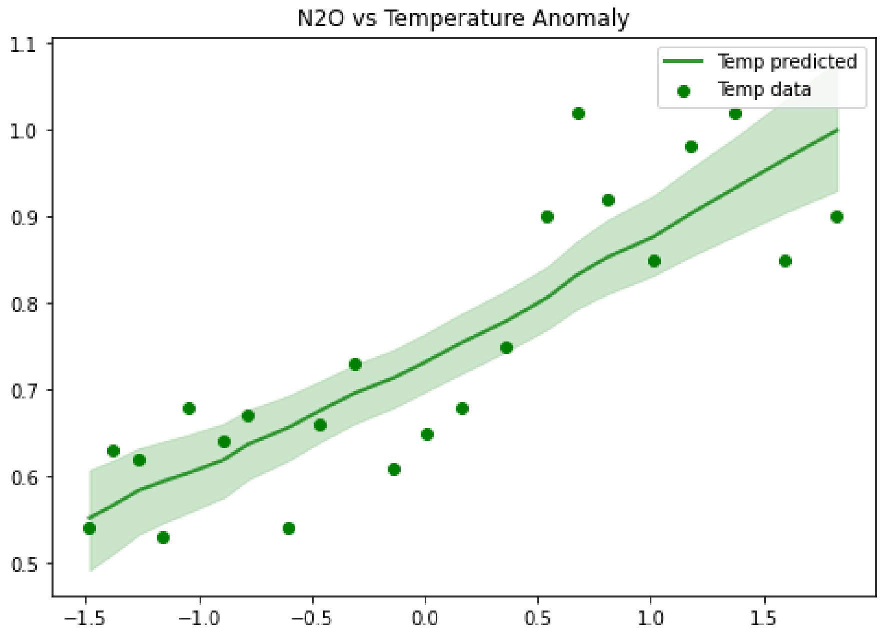
```
import matplotlib as mpl
CO2 = np.array(df['CO2'])
CH4 = np.array(df['CH4'])
N2O = np.array(df['N2O'])
Temperature = np.array(df['Temperature'])
mu_CO2 = fit7.stan_variable('mu')
mu_CH4 = fit7.stan_variable('mu')
mu_N2O = fit7.stan_variable('mu')


fig, ax = plt.subplots(3, 1, figsize=(7 ,15))

ax[0].fill_between(
    CO2,
```

```
    np.percentile(mu_CO2, 5, axis=0),
    np.percentile(mu_CO2, 95, axis=0),
    color=1 - 0.4 * (1 - np.array(mpl.colors.to_rgb('orange'))),
    alpha=0.5
)

ax[1].fill_between(
    CH4,
    np.percentile(mu_CH4, 5, axis=0),
    np.percentile(mu_CH4, 95, axis=0),
    color=1 - 0.4 * (1 - np.array(mpl.colors.to_rgb('blue'))),
    alpha=0.5
)

ax[2].fill_between(
    N2O,
    np.percentile(mu_N2O, 5, axis=0),
    np.percentile(mu_N2O, 95, axis=0),
    color=1 - 0.4 * (1 - np.array(mpl.colors.to_rgb('green'))),
    alpha=0.5
)

ax[0].plot(
    CO2,
    np.percentile(mu_CO2, 50, axis=0),
    color='red',
    linewidth=2,
    alpha=0.8,
    label='Temp predicted'
)
ax[1].plot(
    CH4,
    np.percentile(mu_CH4, 50, axis=0),
    color='blue',
    linewidth=2,
    alpha=0.8,
    label='Temp predicted'
)
ax[2].plot(
    N2O,
    np.percentile(mu_N2O, 50, axis=0),
    color='green',
    linewidth=2,
    alpha=0.8,
    label='Temp predicted'
)

ax[0].scatter(CO2, Temperature, color='red', label='Temp data')
ax[1].scatter(CH4, Temperature, color='blue', label='Temp data')
ax[2].scatter(N2O, Temperature, color='green', label='Temp data')

ax[0].set_xlabel('Concentration')
ax[0].set_ylabel('Temperature anomaly')


ax[0].legend()
ax[1].legend()
ax[2].legend()

ax[0].set_title('CO2 vs Temperature Anomaly')
ax[1].set_title('CH4 vs Temperature Anomaly')
ax[2].set_title('N2O vs Temperature Anomaly')
plt.tight_layout()
plt.show()
```







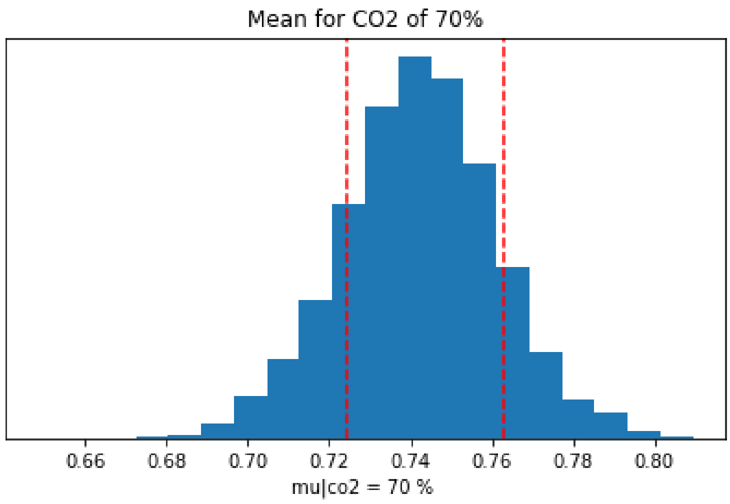


In the output data we can see that provided data of temperatures anomaly fits great.

In [39]:

```
alpha_post =  fit7.stan_variable('alpha')
beta_post = fit7.stan_variable('beta_CO2')
mu_post = fit7.stan_variable('mu')

mu70 = alpha_post+beta_post*(np.mean(df['CO2']))
mu_70p = az.hdi(mu70,.70)
fig, ax = plt.subplots(1, 1, figsize=(7, 4))
ax.hist(mu70,bins=20,density=True)
plt.axvline(mu_70p[0], linestyle = '--', color = 'r')
```

```
plt.axvline(mu_70p[1], linestyle = '--', color = 'r')
ax.set_title('Mean for CO2 of 70%')
ax.set_yticks(())
ax.set_xlabel('mu|co2 = 70 % ')
plt.show()
print('Mean: {:4.2f}'.format(np.mean(mu70)))
print('70% confidence interval: ',['{:4.2f}'.format(k) for k in az.hdi(mu70,.70)])
```




```
Mean: 0.74
70% confidence interval:  ['0.72', '0.76']
```

Histogram shows that on 70% probability will be in range 0.72 and 0.76.

Model comparison

In [55]:

```
print("Summary - Normal model:")
fit.summary()
```

```
Summary - Normal model:
```

Out[55]:

name	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
lp__	29.000	0.05200	1.800	25.0000	29.000	31.000	1200.0	2600.0	1.0
alpha	0.740	0.00041	0.020	0.7100	0.740	0.770	2400.0	5200.0	1.0
sigma	0.092	0.00024	0.012	0.0730	0.091	0.110	2600.0	5700.0	1.0
beta_CO2	0.049	0.00064	0.032	0.0046	0.045	0.110	2500.0	5500.0	1.0
beta_CH4	0.042	0.00054	0.030	0.0035	0.037	0.097	3000.0	6600.0	1.0
...	...	...	...	...	...	...	...	...	...
log_lik[18]	0.980	0.01100	0.710	-0.4700	1.200	1.600	3910.0	8613.0	1.0
log_lik[19]	0.970	0.01100	0.690	-0.4500	1.200	1.600	3980.0	8766.0	1.0
log_lik[20]	0.970	0.01200	0.740	-0.4800	1.200	1.600	3759.0	8279.0	1.0
log_lik[21]	0.990	0.01100	0.690	-0.3700	1.200	1.600	3864.0	8512.0	1.0
log_lik[22]	0.980	0.01200	0.760	-0.4600	1.200	1.600	4136.0	9109.0	1.0

72 rows × 9 columns

In [56]:

```
print("Summary - Student model:")
fit7.summary()
```

```
Summary - Student model:
```

Out[56]:

name	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
lp__	24.000	0.05500	2.000	21.0000	25.000	27.000	1300.0	1800.0	1.0
alpha	0.740	0.00038	0.019	0.7100	0.740	0.770	2600.0	3700.0	1.0
beta_CO2	0.049	0.00060	0.032	0.0053	0.046	0.110	2800.0	3900.0	1.0
beta_CH4	0.043	0.00053	0.030	0.0040	0.038	0.099	3200.0	4500.0	1.0
beta_N2O	0.043	0.00059	0.031	0.0040	0.037	0.100	2800.0	3900.0	1.0
...	...	...	...	...	...	...	...	...	...
log_lik[18]	1.400	0.00380	0.200	1.1000	1.400	1.700	2688.0	3765.0	1.0
log_lik[19]	1.000	0.00580	0.340	0.3900	1.100	1.500	3362.0	4708.0	1.0
log_lik[20]	0.900	0.00700	0.410	0.1000	0.970	1.400	3511.0	4918.0	1.0
log_lik[21]	0.480	0.00950	0.590	-0.6300	0.570	1.300	3858.0	5403.0	1.0
log_lik[22]	0.690	0.00960	0.600	-0.4800	0.800	1.500	3945.0	5525.0	1.0

73 rows × 9 columns

Values for returned parameters are quite similar. We assumed that in both models that may look alike

In [65]:

```
fitStudent_ = az.from_cmdstanpy(posterior=fit7,
                                log_likelihood='log_lik',
                                posterior_predictive='temp_i',
                                observed_data={'temperature':df['Temperature']})
fitStudent_
```

Out[65]:

arviz.InferenceData

► **posterior**

► **posterior_predictive**

► **log_likelihood**

► **sample_stats**

► **observed_data**

In [61]:

```
fitNormal_ = az.from_cmdstanpy(posterior=fit,
                               log_likelihood='log_lik',
                               posterior_predictive='temp_',
                               observed_data={'temperature':df['Temperature']})
```

In [66]:

```
az.loo(fitStudent_)
```

Out[66]:

```
Computed from 4000 by 22 log-likelihood matrix

        Estimate       SE
```

```
          Estimate       SE
elpd_loo    20.87     2.62
p_loo        2.55        -
------

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)       22  100.0%
 (0.5, 0.7]   (ok)          0    0.0%
   (0.7, 1]   (bad)         0    0.0%
   (1, Inf)   (very bad)    0    0.0%
```

In [67]:

```
az.waic(fitStudent_)
```

```
/usr/local/lib/python3.9/site-packages/arviz/stats/stats.py:1635: UserWarning: For one or
more samples the posterior variance of the log predictive densities exceeds 0.4. This cou
ld be indication of WAIC starting to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
```

Out[67]:

```
Computed from 4000 by 22 log-likelihood matrix

          Estimate       SE
elpd_waic    20.92     2.60
p_waic        2.51        -

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

The model with students distribution gives very similar result for WAIC and LOO

In [68]:

```
az.loo(fitNormal_)
```

```
/usr/local/lib/python3.9/site-packages/arviz/stats/stats.py:811: UserWarning: Estimated s
hape parameter of Pareto distribution is greater than 0.7 for one or more samples. You sh
ould consider using a more robust model, this is because importance sampling is less like
ly to work well if the marginal posterior and LOO posterior are very different. This is m
ore likely to happen with a non-robust model and highly influential observations.
  warnings.warn(
```

Out[68]:

```
Computed from 4000 by 22 log-likelihood matrix

          Estimate       SE
elpd_loo     6.01     0.73
p_loo       18.99        -

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

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)        0    0.0%
 (0.5, 0.7]   (ok)          1    4.5%
   (0.7, 1]   (bad)        16   72.7%
   (1, Inf)   (very bad)    5   22.7%
```

In [69]:

```
az.waic(fitStudent_)
```

```
/usr/local/lib/python3.9/site-packages/arviz/stats/stats.py:1635: UserWarning: For one or
more samples the posterior variance of the log predictive densities exceeds 0.4. This cou
ld be indication of WAIC starting to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
```

Out[69]:

```
Computed from 4000 by 22 log-likelihood matrix

          Estimate       SE
elpd_waic    20.92     2.60
p_waic        2.51        -

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

In the model with Normal distribution only WAIC is the same as WAIC nad LOO for Student-t model. The reason why LOO and WAIC varies here it is because they have different evaluation strategies. WAIC focus on entire dataset while LOO on every point of data. When it comes to LOO it shows that model is not so good but focusing on whole dataset the evaluation is much better.

LOO

In [71]:

```
LOO_compare = az.compare({'Student model':fitStudent_, 'Normal model':fitNormal_}, ic='l
oo')
LOO_compare
```

```
/usr/local/lib/python3.9/site-packages/arviz/stats/stats.py:811: UserWarning: Estimated s
hape parameter of Pareto distribution is greater than 0.7 for one or more samples. You sh
ould consider using a more robust model, this is because importance sampling is less like
ly to work well if the marginal posterior and LOO posterior are very different. This is m
ore likely to happen with a non-robust model and highly influential observations.
  warnings.warn(
```

Out[71]:

	rank	loo	p_loo	d_loo	weight	se	dse	warning	loo_scale
Student model	0	20.873971	2.553913	0.000000	1.0	2.616999	0.000000	False	log
Normal model	1	6.009034	18.990402	14.864938	0.0	0.731370	2.418414	True	log

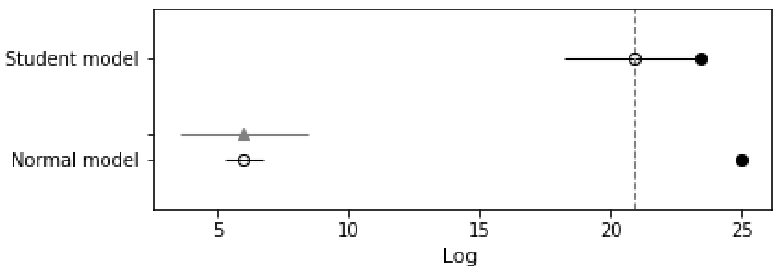
Smaller rank indicates which model is better. Here we can see that Student model is much better from the other one. But lets see the output when it comes to whole dataset -WAIC.

In [72]:

```
az.plot_compare(LOO_compare)
```

Out[72]:

```
&lt;AxesSubplot:xlabel='Log'&gt;
```



WAIC

In [74]:

```
WAIC_compare = az.compare({'Student model':fitStudent_, 'Gaussian model':fitNormal_}, ic
='waic')
WAIC_compare
```

```
/usr/local/lib/python3.9/site-packages/arviz/stats/stats.py:1635: UserWarning: For one or
```

```
/usr/local/lib/python3.9/site-packages/arviz/stats/stats.py:1635: UserWarning: For one or
more samples the posterior variance of the log predictive densities exceeds 0.4. This cou
ld be indication of WAIC starting to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
/usr/local/lib/python3.9/site-packages/arviz/stats/stats.py:1635: UserWarning: For one or
more samples the posterior variance of the log predictive densities exceeds 0.4. This cou
ld be indication of WAIC starting to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
```

Out[74]:

	rank	waic	p_waic	d_waic	weight	se	dse	warning	waic_scale
Student model	0	20.916747	2.511138	0.000000	1.0	2.602951	0.000000	True	log
Gaussian model	1	13.469474	11.529962	7.447273	0.0	0.163882	2.600035	True	log

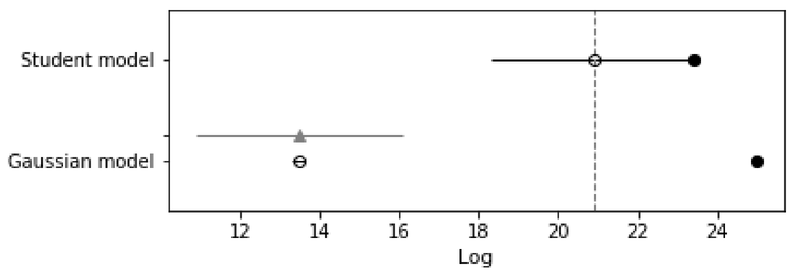
Here also the output says that the Student model is better than the Gaussian Model.

In [75]:

```
az.plot_compare(WAIC_compare)
```

Out[75]:

<AxesSubplot:xlabel='Log'>



With WAIC evaluation the difference is much smaller than with LOO but the better model is still the same.

To sum up both models adjusted to the data pretty well. The secound approach turned out to be better than the first one - Gaussian. In the future we could add more parameters to the model or change the impact of the parameters to be different from each other. Also we could consider wider time range than just from 2001 to 2022.