Happiness Prediction

1. Introduction

The project was carried out as a part of Data Analysis course by:

Michał Święciło - 402247
Dawid Bogon - 402452

Problem formulation

The goal of our project was to develop a Bayesian model to predict the level of personal happiness of residents in the city of Somerville. The predicted level of happiness should be based on variables such as wellbeing, satisfaction with city services, and other relevant factors. There are many applications for developing such Bayesian models, for example:

- City planning

  Understanding which factors are crucial for residents' happiness can influence city planning decisions, from budget allocations to future urban development plans.

- Personalized Services

  If the model could incorporate demographic data, the city could even personalize its services or outreach to different resident groups based on what most significantly impacts their happiness.

- Benchmarking

  The city could use the findings from this model to compare its performance in delivering services and maintaining the happiness of its residents with other cities. This could help identify areas where the city is excelling or lagging behind.

About data

Every two years, the City of Somerville sends out a happiness survey to a random sample of Somerville residents. The survey asks residents to rate their personal happiness, wellbeing, and satisfaction with City services. The combined dataset includes the survey responses from 2011 to 2021. The dataset is intended for public access and can be found on many websites (official source). More information about the Happiness Survey is on the City of Somerville website.

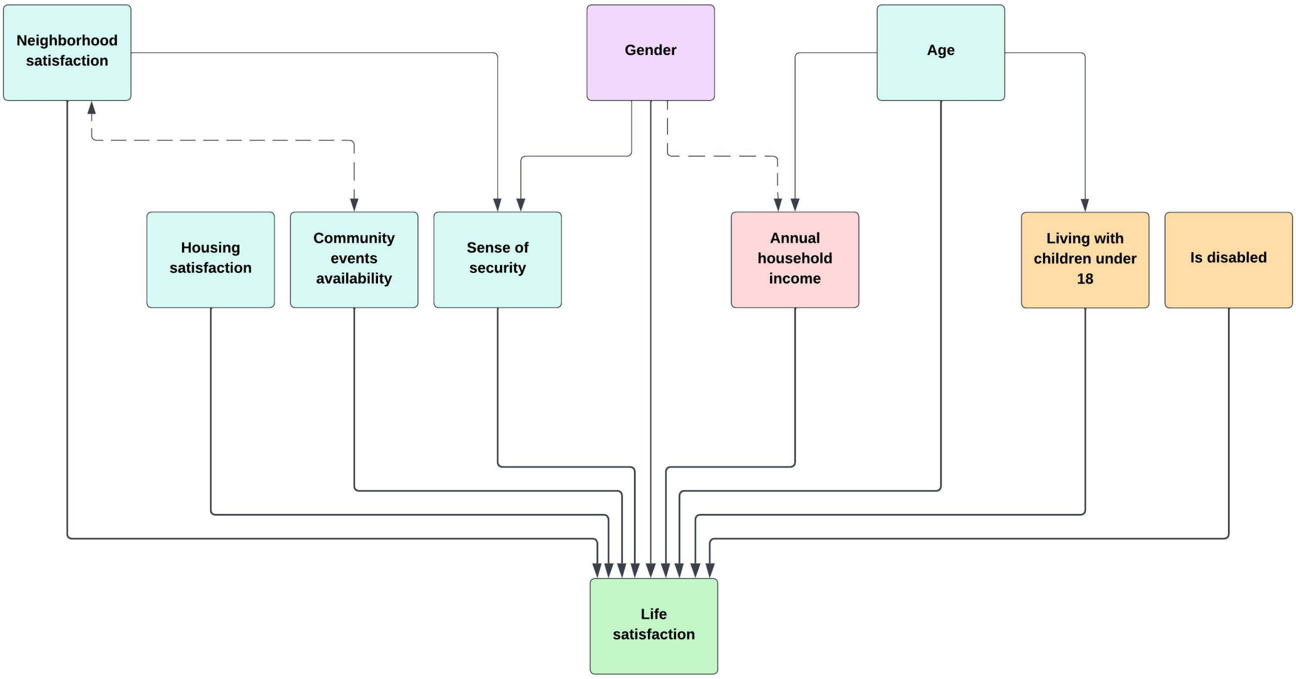
Obviously, the database required some preprocessing before it could be used with acual models. To make it happen, it was necessary to perform a number of operations like reducing the size, handling missing values and transforming the data. Here are the main steps we took during this process:

1. Restricting the scope of the data to needed columns and entries dating back to 2021.
2. Changing column names to easier and shorter ones.
3. Deleting all rows containing at least one null value.
4. Using Midpoint Coding to convert categorical "Annual household income" feature to real values.
5. Using One Hot Encoding to convert categorical "Living with children under 18" and "Disability" features to binary variables.
6. Using Ordinal Encoding to assigning integer values to given categories in "Age" and "Gender" features.
7. Changing the data type of individual columns.
8. Saving the processed database.

All these operations and the reasons for performing them are described in detail in data_preprocessing.ipynb.

Dependencies between variables

In order to demonstrate the relationships between selected features from the database and to better design future models, a DAG was created.



It's using the following designations:

- Blue block - Ordered categorical variable
- Purple block - Nominal categorical variable
- Red block - Real variable
- Orange block - Binary variable
- Green block - Target (ordered categorical variable)
- Continuous line - Association between block
- Dashed line - Weak influence between data

Possible confoundings

- Fork

Age is common cause for income and living with childern under 18.

Gender influences both sense of secuity and annual income (due to still existing income gap)

Neighbour satisfaction is common cause of life satisfaction, sens of secuirty and availiablity of community events.

- Collider

Sense of secuirty is infulenced by gender and neighbourhood satisfaction

Income is influenced by gender and age

Life satisafction is influenced by housing satisafction, neighbourhhod satisafction, availiability of comminuty events, sense of security, age, gender, income, disability and living with childern under 18. Age and Genter both influence it directly and are transmited throug other parameters therefore we are going to use them in second model.

- Pipe

Neighbour satisafction can influence community event availabilty and is transmited to life satisfaction.

In [1]:
```
# Import libraries
from cmdstanpy import CmdStanModel
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import arviz as az

# Set the default pandas displaing options
pd.reset_option("display.max_columns")
pd.reset_option("display.max_rows")
```

In [2]:
```
# Load processed data from .csv file
df_main = pd.read_csv('data/Somerville_Happiness_processed.csv', sep=',', header=0)

# Show basic informations about data
df_main.info()
df_main.head()
```

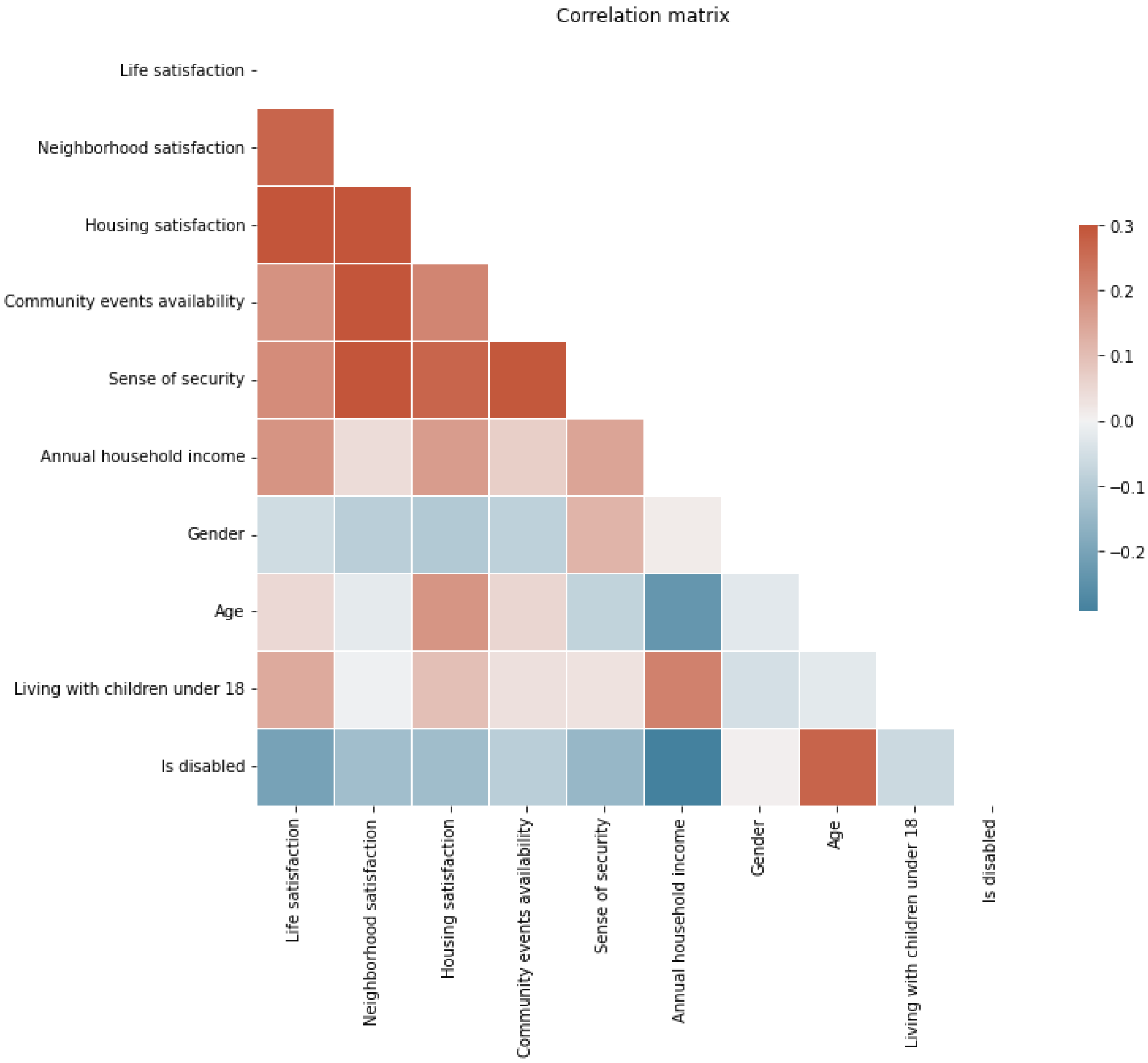
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1049 entries, 0 to 1048
Data columns (total 10 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Life satisfaction             1049 non-null   int64  
 1   Neighborhood satisfaction     1049 non-null   int64  
 2   Housing satisfaction          1049 non-null   int64  
 3   Community events availability 1049 non-null   int64  
 4   Sense of security             1049 non-null   int64  
 5   Annual household income       1049 non-null   float64
 6   Gender                        1049 non-null   int64  
 7   Age                           1049 non-null   int64  
 8   Living with children under 18 1049 non-null   int64  
 9   Is disabled                   1049 non-null   int64  
dtypes: float64(1), int64(9)
memory usage: 82.1 KB
```

Out[2]:

	Life satisfaction	Neighborhood satisfaction	Housing satisfaction	Community events availability	Sense of security	Annual household income	Gender	Age	Living with children under 18	Is disabled
0	9	9	9	5	9	87499.5	1	6	0	0
1	8	7	6	4	6	87499.5	2	7	0	1
2	7	8	3	4	5	62499.5	1	3	0	0
3	3	8	8	1	4	62499.5	1	4	0	0
4	5	7	7	4	7	124999.5	2	6	1	0

In [3]:
```
N = 1049
df_trimed = df_main.head(N)

# All database columns
life_satisfaction = df_trimed['Life satisfaction'].to_numpy()
neighborhood_satisfaction = df_trimed['Neighborhood satisfaction'].to_numpy()
housing_satisfaction = df_trimed['Housing satisfaction'].to_numpy()
community_events_availability = df_trimed['Community events availability'].to_numpy()
sense_of_security = df_trimed['Sense of security'].to_numpy()
annual_household_income = df_trimed['Annual household income'].to_numpy()
gender = df_trimed['Gender'].to_numpy()
age = df_trimed['Age'].to_numpy()
living_with_children = df_trimed['Living with children under 18'].to_numpy()
is_disabled = df_trimed['Is disabled'].to_numpy()

# Normalization
annual_household_income = (annual_household_income/annual_household_income.std(axis=0))
annual_household_income = annual_household_income-annual_household_income.mean(axis=0)
```

In [4]:
```
# Correlation between all variables
corr = df_trimed.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.title("Correlation matrix")
plt.show()
```



From corelation matrix we can see that on direct level correlation between chosen variables is not very strong. Such behaviour is desired because ordered logistic regression in our models requires low association between features.

2. Priors and validation

Ordered logistics regression

We have decied to use ordered logistics regression analysis in order to determine factors affecting happines levels. It has been used before for such task.

Ordered logistic reggression model is used when varabile has more than two categories and is sortable. In our case we have 10 diffrent levels of happines, and we can sort them from least happy to most happy. Other use cases for such models would be predicting education level or descriptions of parameters with outcomes as: "poor", "fair", "good", "very good", and "excellent".

We are loooking for values of cutpoints which would split values of linear combination for given person into 10 categories.

Chosing Priors

The chosen priors distribution (for predictors coefficients) is the normal distribution with mean 0 and standard deviation 1, except for the cutpoints which have standard deviation 3. We have decided to use such uninformative ("weak") priors because we couldn't find any specific values for parameters. This was not surprising because in the social sciences it is hard to come up with scientific constants or specific formulas to describe feelings. We therefore assume the true value of the coefficient can plausibly be anywhere on a relatively wide range on the real number line.

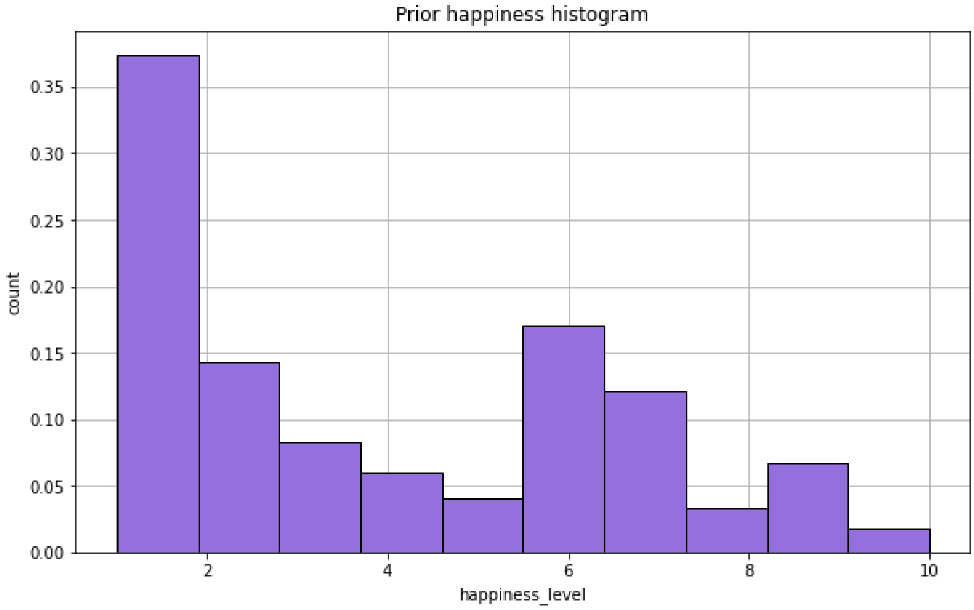
Values for c - cutpoints in prior check are defined in model because ordered vector can`t be used in generated quantities block.

Prior predictive checks

In [5]:
```
m1_ppc = CmdStanModel(stan_file='stan_files/model1_ppc.stan')

rng_num =  np.random.randint(low=0, high=100)

d = {'K' : 10,
```

```
    'y' : life_satisfaction[rng_num],
    'neigh_sat' : neighborhood_satisfaction[rng_num],
    'hous_sat' : housing_satisfaction[rng_num],
    'com_even_avail' : community_events_availability[rng_num],
    'sen_of_sec' : sense_of_security[rng_num],
    'ann_hous_inc' : annual_household_income[rng_num],
    'liv_with_child' : living_with_children[rng_num],
    'is_disabled' : is_disabled[rng_num]}

# Compilation of model1_ppc.stan and get 1000 samples for 4 chains
samples = m1_ppc.sample(data=d, fixed_param=True, iter_sampling=1000, chains=4)

ppc_happy = samples.stan_variable('happy').flatten()
plt.figure(figsize=(10, 6))
x, y = np.unique(life_satisfaction, return_counts=True)
plt.hist(ppc_happy, align='mid', density=True, label='predicted', color='mediumpurple', edgecolor="black", zorder=2)
plt.title("Prior happiness histogram")
plt.xlabel("happiness_level")
plt.ylabel("count")
plt.grid(zorder=0)
plt.show()
```

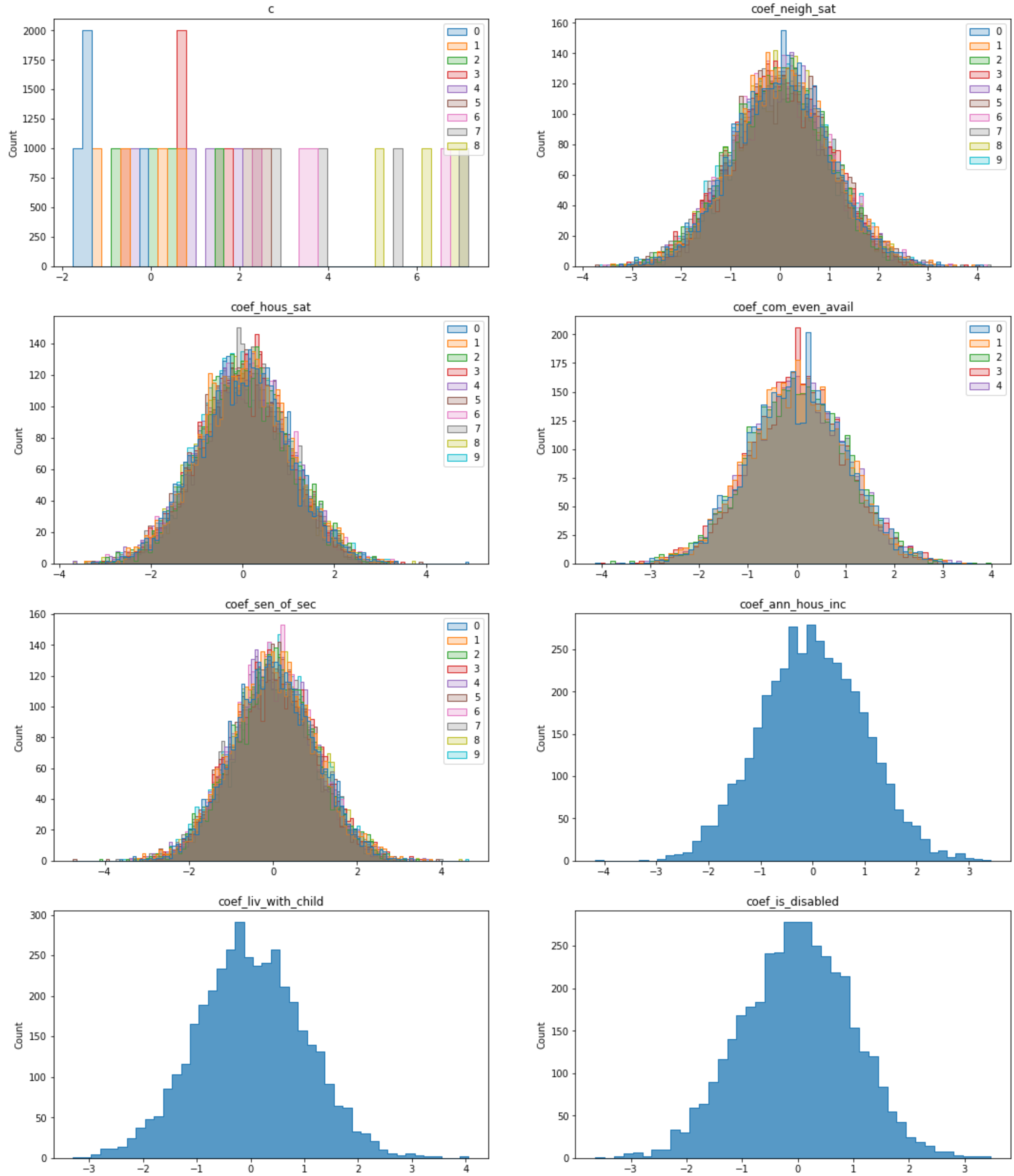
```
INFO:cmdstanpy:found newer exe file, not recompiling
INFO:cmdstanpy:CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

INFO:cmdstanpy:CmdStan done processing.
```



For random choosen sample prior predictive check does not fit data very well. The main reason behind this behaviour are choosen priors. Model does not yet know that beta paramteres in vector for each categorical varaible could be correlated.

Prior parameteres check

In [6]:
```
lst = ['c','coef_neigh_sat','coef_hous_sat','coef_com_even_avail','coef_sen_of_sec','coef_ann_hous_inc','coef_liv_with_ch

fig, axs = plt.subplots(4, 2, figsize=(18, 22))
for ax, param in zip(axs.flat, lst):
    coeffs = samples.stan_variable(param)
    sns.histplot(ax=ax, data=coeffs, element="step")
    ax.set_title(param)

plt.show()
```



Distribution of coefficients in prior model are consistent with those specfied in model.

C - cutoff does not change because it can only be specified in model (it is an oredered vector), and we do not want to update it in prior predictive check.

3. First model

Model description

This is a Bayesian ordinal logistic regression model implemented in Stan, which aims to predict an ordinal outcome based on seven predictors, including neighborhood satisfaction, housing satisfaction, community events availability, sense of security, annual household income, living with children, and disability status. The model, defined by specific cutpoints and coefficients for each predictor, assumes priors to follow a normal distribution. The estimation process uses a loop to run through all observations, calculating the sum of each predictor times its respective coefficient for the ordinal logistic regression model.

Inputs:

- N - Number of samples
- K - Number of ordinal categories
- y[N] - Ordinal outcome
- neigh_sat[N] - Predictor 1 (neighborhood_satisfaction)
- hous_sat[N] - Predictor 2 (housing_satisfaction)
- com_even_avail[N] - Predictor 3 (community_events_availability)
- sen_of_sec[N] - Predictor 4 (sense_of_security)
- ann_hous_inc[N] - Predictor 5 (annual_household_income)
- liv_with_child[N] - Predictor 6 (living_with_children)
- is_disabled[N] - Predictor 7 (is_disabled)

Parameters:

- c[K-1] - Cutpoints
- coef_neigh_sat[10] - Coefficient 1 (neighborhood_satisfaction)
- coef_hous_sat[10] - Coefficient 2 (housing_satisfaction)
- coef_com_even_avail[5] - Coefficient 3 (community_events_availability)
- coef_sen_of_sec[10] - Coefficient 4 (sense_of_security)
- coef_ann_hous_inc - Coefficient 5 (annual_household_income)
- coef_liv_with_child - Coefficient 6 (living_with_children)
- coef_is_disabled - Coefficient 7 (is_disabled)

Formulas:

```
happy = ordered_logistic(coef_neigh_sat[neigh_sat] +
                         coef_hous_sat[hous_sat] +
                         coef_com_even_avail[com_even_avail] +
                         coef_sen_of_sec[sen_of_sec] +
                         coef_ann_hous_inc * ann_hous_inc +
                         coef_liv_with_child * liv_with_child +
                         coef_is_disabled * is_disabled, c)
```

$$c \sim Normal(0, 3)$$
$$coef_{neighsat} \sim Normal(0, 1)$$
$$coef_{houssat} \sim Normal(0, 1)$$
$$coef_{comevenavail} \sim Normal(0, 1)$$
$$coef_{senofsec} \sim Normal(0, 1)$$
$$coef_{annhousinc} \sim Normal(0, 1)$$
$$coef_{livwithchild} \sim Normal(0, 1)$$
$$coef_{isdisabled} \sim Normal(0, 1)$$

Model fit and evaluation

In first model we have decied to use neighbourhhood satisfaction, housing satisafcion, rating of availability of community events, sens of security, disability and income and whether or not are you living with childern under 18.

Happines is an ordered categorical variable, therefore we have used ordered logistic model. This model is suited for categorical varaible, we are searching beta parameters for each coefficient and cutoffpoints which define values in which the level of happiness changes.

In [7]:
```
model1_fit = CmdStanModel(stan_file='stan_files/model1_fit.stan')

d = {'N' : N,
     'K' : 10,
     'y' : life_satisfaction,
     'neigh_sat' : neighborhood_satisfaction,
     'hous_sat' : housing_satisfaction,
     'com_even_avail' : community_events_availability,
     'sen_of_sec' : sense_of_security,
     'ann_hous_inc' : annual_household_income,
     'liv_with_child' : living_with_children,
     'is_disabled' : is_disabled}

# Compilation of test2.stan and get 1000 samples
```

```python
samples1 = model1_fit.sample(data=d, iter_sampling=1000, iter_warmup=1000, chains=4)
print(samples1.diagnose())
```

```
INFO:cmdstanpy:compiling stan file /root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan to exe file /root/ISZ_DA/DA_final_project/stan_files/model1_fit
INFO:cmdstanpy:compiled model executable: /root/ISZ_DA/DA_final_project/stan_files/model1_fit
WARNING:cmdstanpy:Stan compiler has produced 10 warnings:
WARNING:cmdstanpy:
--- Translating Stan model to C++ code ---
bin/stanc  --o=/root/ISZ_DA/DA_final_project/stan_files/model1_fit.hpp /root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 7, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 10, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 11, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 12, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 13, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 14, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 15, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 16, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 55, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc
Warning in '/root/ISZ_DA/DA_final_project/stan_files/model1_fit.stan', line 56, column 2: Declaration
    of arrays by placing brackets after a variable name is deprecated and
    will be removed in Stan 2.32.0. Instead use the array keyword before the
    type. This can be changed automatically using the auto-format flag to
    stanc

--- Compiling, linking C++ code ---
g++ -std=c++1y -pthread -D_REENTRANT -Wno-sign-compare -Wno-ignored-attributes      -I stan/lib/stan_math/lib/tbb_2020.3/include    -O3 -I src -I stan/src -I lib/rapidjson_1.1.0/ -I lib/CLI11-1.9.1/ -I stan/lib/stan_math/ -I stan/lib/stan_math/lib/eigen_3.3.9 -I stan/lib/stan_math/lib/boost_1.75.0 -I stan/lib/stan_math/lib/sundials_6.0.0/include -I stan/lib/stan_math/lib/sundials_6.0.0/src/sundials    -DBOOST_DISABLE_ASSERTS          -c -Wno-ignored-attributes   -x c++ -o /root/ISZ_DA/DA_final_project/stan_files/model1_fit.o /root/ISZ_DA/DA_final_project/stan_files/model1_fit.hpp
g++ -std=c++1y -pthread -D_REENTRANT -Wno-sign-compare -Wno-ignored-attributes      -I stan/lib/stan_math/lib/tbb_2020.3/include    -O3 -I src -I stan/src -I lib/rapidjson_1.1.0/ -I lib/CLI11-1.9.1/ -I stan/lib/stan_math/ -I stan/lib/stan_math/lib/eigen_3.3.9 -I stan/lib/stan_math/lib/boost_1.75.0 -I stan/lib/stan_math/lib/sundials_6.0.0/include -I stan/lib/stan_math/lib/sundials_6.0.0/src/sundials    -DBOOST_DISABLE_ASSERTS               -Wl,-L,"/opt/cmdstan-2.29.0/stan/lib/stan_math/lib/tbb" -Wl,-rpath,"/opt/cmdstan-2.29.0/stan/lib/stan_math/lib/tbb"      /root/ISZ_DA/DA_final_project/stan_files/model1_fit.o src/cmdstan/main.o        -Wl,-L,"/opt/cmdstan-2.29.0/stan/lib/stan_math/lib/tbb" -Wl,-rpath,"/opt/cmdstan-2.29.0/stan/lib/stan_math/lib/tbb"   stan/lib/stan_math/lib/sundials_6.0.0/lib/libsundials_nvecserial.a stan/lib/stan_math/lib/sundials_6.0.0/lib/libsundials_cvodes.a stan/lib/stan_math/lib/sundials_6.0.0/lib/libsundials_idas.a stan/lib/stan_math/lib/sundials_6.0.0/lib/libsundials_kinsol.a  stan/lib/stan_math/lib/tbb/libtbb.so.2 -o /root/ISZ_DA/DA_final_project/stan_files/model1_fit
rm -f /root/ISZ_DA/DA_final_project/stan_files/model1_fit.o

INFO:cmdstanpy:CmdStan start processing
chain 1 |          | 00:00 Status
```

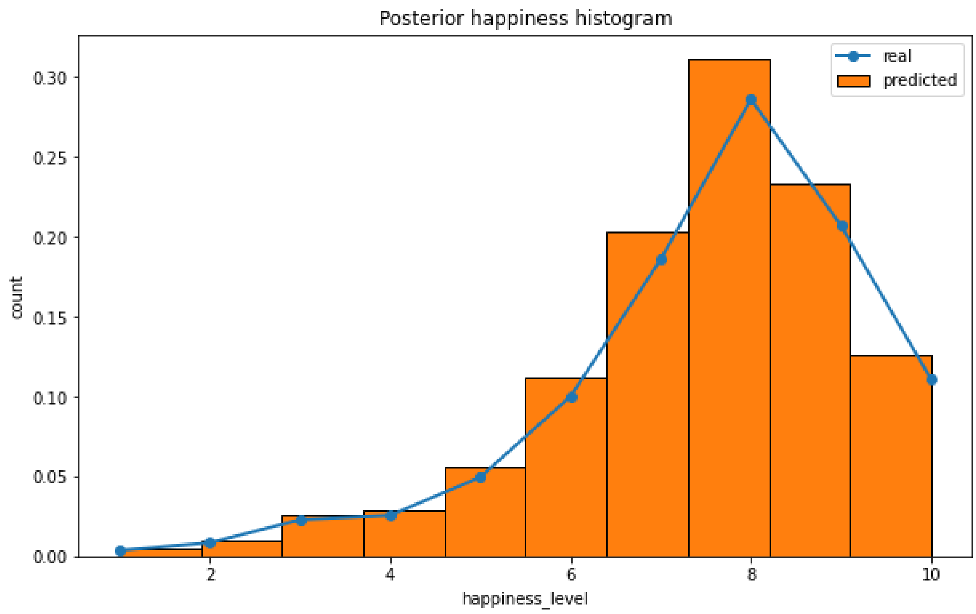
```
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status
```

```
INFO:cmdstanpy:CmdStan done processing.
Processing csv files: /tmp/tmpxqn1hebi/model1_fit-20230710202353_1.csv, /tmp/tmpxqn1hebi/model1_fit-20230710202353_2.csv, /tmp/tmpxqn1hebi/model1_fit-20230710202353_3.csv, /tmp/tmpxqn1hebi/model1_fit-20230710202353_4.csv

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Effective sample size satisfactory.

Split R-hat values satisfactory all parameters.

Processing complete, no problems detected.
```

Posterior anlysis

Below we can see the distribution of happines, cutoffpoints and beta parameters of the model. Fitting proces caused changes in parameteres. The cooeficients make sens from logical point of view, but it would be hard to get this knowledge apriori. We would expected to see more ordered structure for categorical parrameters like housing satisafction or sense of security. It is still visible that in most of categorical parameteres higher values correspond to higher sense of security.

In [8]:
```
model1_happy = samples1.stan_variable('happy').flatten()

plt.figure(figsize=(10, 6))
x, y = np.unique(life_satisfaction, return_counts=True)
plt.plot(x, y/sum(y), marker='o', linewidth=2, label='real')
plt.hist(model1_happy, align='mid', density=True, label='predicted', edgecolor="black")
plt.title("Posterior happiness histogram")
plt.xlabel("happiness_level")
plt.ylabel("count")
plt.legend()
plt.show()

df_res = samples1.draws_pd()
display(df_res)
```

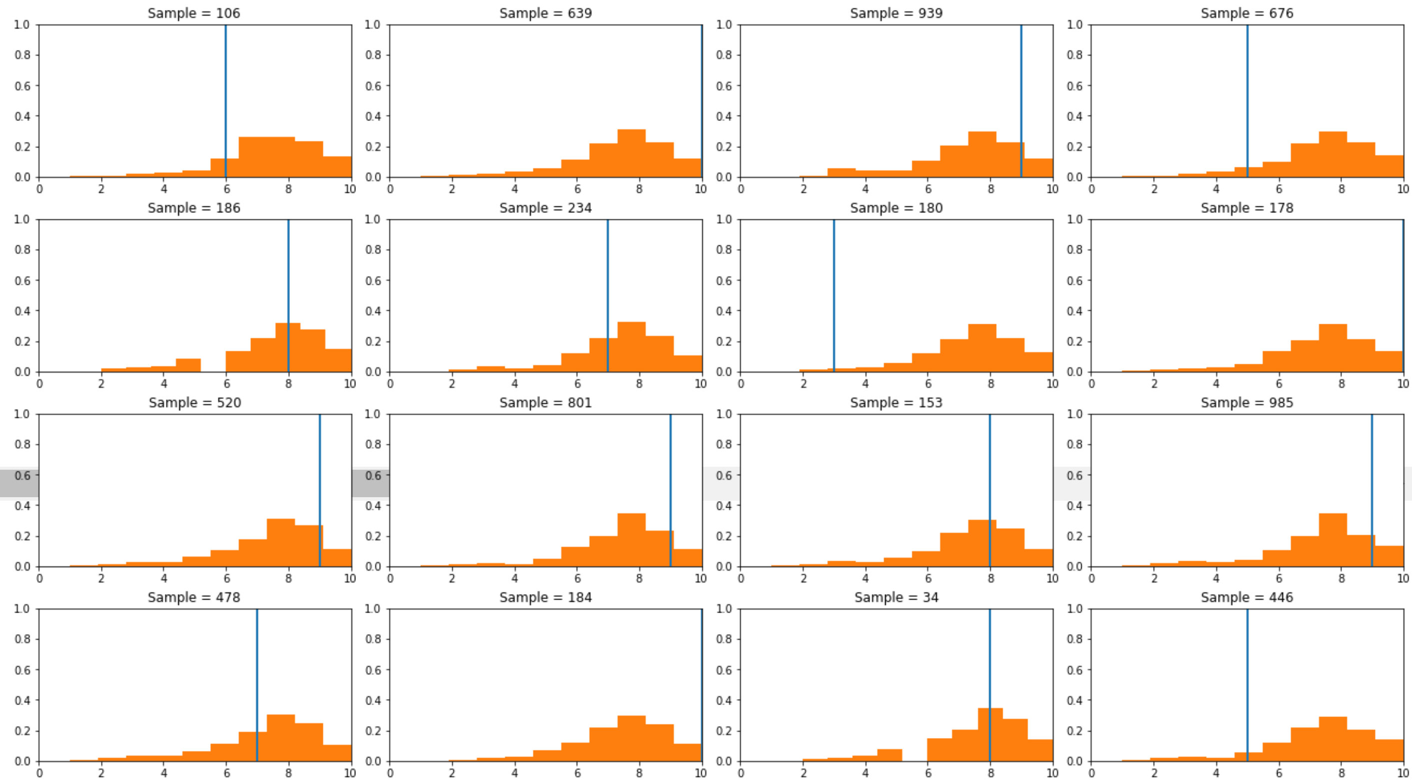


	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__	c[1]	c[2]	c[3]	...	log_lik[1040
0	-1847.47	0.984291	0.055224	6.0	63.0	0.0	1872.70	-5.07697	-3.89301	-2.92909	...	-1.7673
1	-1841.84	0.967563	0.055224	6.0	63.0	0.0	1866.52	-5.19399	-4.00771	-2.85674	...	-1.7525
2	-1840.34	0.894710	0.055224	6.0	63.0	0.0	1865.51	-5.18862	-3.78780	-2.79982	...	-1.8811
3	-1846.16	0.912316	0.055224	6.0	63.0	0.0	1862.43	-5.24995	-4.18277	-3.20249	...	-2.0165
4	-1846.59	0.951071	0.055224	6.0	63.0	0.0	1868.31	-4.60696	-3.69611	-2.75500	...	-1.6335
...	...	...	...	...	...	...	...	...	...	...	...	
3995	-1852.62	0.833181	0.059557	6.0	63.0	0.0	1879.92	-6.00542	-4.16528	-3.09661	...	-1.8234
3996	-1845.69	0.996084	0.059557	6.0	127.0	0.0	1867.68	-6.23408	-4.51980	-3.42029	...	-1.7188
3997	-1846.25	0.921971	0.059557	6.0	63.0	0.0	1869.25	-4.88479	-3.74253	-2.68829	...	-1.5580
3998	-1858.33	0.604213	0.059557	6.0	63.0	0.0	1878.34	-5.79412	-4.12216	-3.33326	...	-1.8644
3999	-1852.56	0.995383	0.059557	6.0	127.0	0.0	1879.18	-4.75651	-3.53009	-2.72757	...	-1.3206

4000 rows × 2152 columns

As we can see on the plot, distribution of target samples in posterior model fits distribution in our dataset.

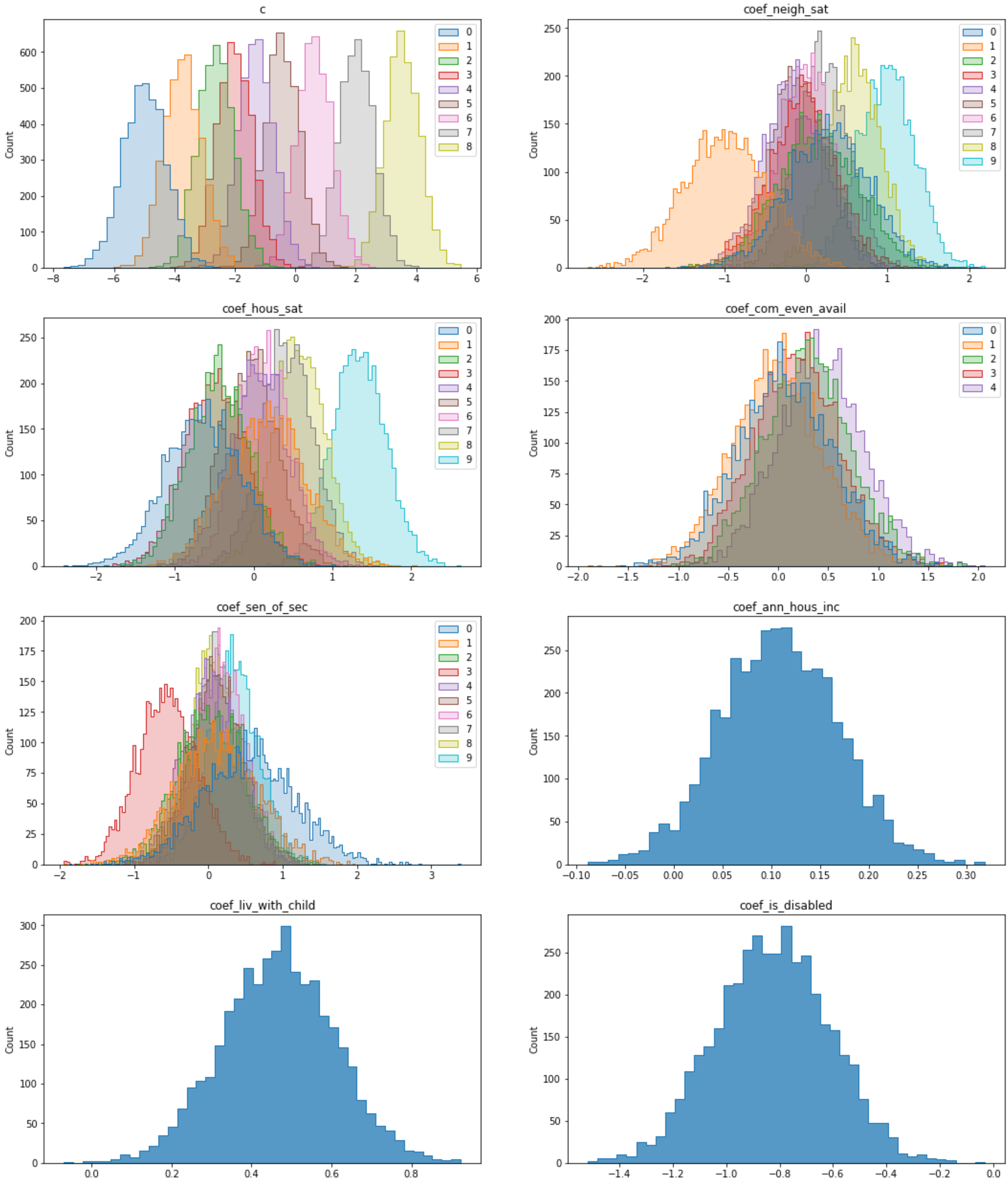
In [9]:
```
model_happy = samples1.stan_variable('happy')
arr = np.random.randint(low=1, high=N, size = 16)

fig, axs = plt.subplots(4, 4, figsize=(18, 10), layout='constrained')
for ax, sample in zip(axs.flat, arr):
    x = life_satisfaction[sample]
    ax.plot([x,x], [-0.2,1.2], linewidth=2, label='real')
    ax.hist(model_happy[sample], align='mid', density=True, label='predicted')
    ax.set_xlim([0, 10])
    ax.set_ylim([0, 1])
    ax.set_title(f"Sample = {sample}")
plt.show()
```



When looking for random chossen specfic indivdulas we can see that the distrbution does not follow the real value, this means that model is overfitted. Possible reason for this phenomena could be using to many categorical variables in ordered logistic model.

In [10]:
```
lst = ['c','coef_neigh_sat','coef_hous_sat','coef_com_even_avail','coef_sen_of_sec',
       'coef_ann_hous_inc','coef_liv_with_child','coef_is_disabled']

fig, axs = plt.subplots(4, 2, figsize=(18, 22))
for ax, param in zip(axs.flat, lst):
    coeffs = samples1.stan_variable(param)
    sns.histplot(ax=ax, data=coeffs, element="step")
    ax.set_title(param)
```

```
plt.show()
```



For most paramteters distributions are narrower. Mean value also has changed, but we expected it to change because it is hard to assume impact of parameters on life satisafction.

On our the first approach we encountered issue during sampling which was propably triggered by using ordered vectors for all coefficients. To fix this problem we decided to use normal vectors witch resulted in better coefficients spread. Another minor problem was an fixed_parameter flag that was set to True. This initially blocked the change of coefficients.

4. Second model

Model description

For second model we have added age and gender of person. Age and gender can have impact on human happines. As described in [A Research Note: Happiness by Age is More Complex than U-Shaped].

Inputs:

- N - Number of samples
- K - Number of ordinal categories
- y[N] - Ordinal outcome
- neigh_sat[N] - Predictor 1 (neighborhood_satisfaction)
- hous_sat[N] - Predictor 2 (housing_satisfaction)
- com_even_avail[N] - Predictor 3 (community_events_availability)
- sen_of_sec[N] - Predictor 4 (sense_of_security)
- ann_hous_inc[N] - Predictor 5 (annual_household_income)
- gender[N] - Predictor 6 (gender)
- age[N] - Predictor 7 (age)
- liv_with_child[N] - Predictor 8 (living_with_children)
- is_disabled[N] - Predictor 9 (is_disabled)

Parameters:

- c[K-1] - Cutpoints
- coef_neigh_sat[10] - Coefficient 1 (neighborhood_satisfaction)
- coef_hous_sat[10] - Coefficient 2 (housing_satisfaction)
- coef_com_even_avail[5] - Coefficient 3 (community_events_availability)
- coef_sen_of_sec[10] - Coefficient 4 (sense_of_security)
- coef_ann_hous_inc - Coefficient 5 (annual_household_income)
- coef_gender[N] - Coefficient 6 (gender)
- coef_age[N] - Coefficient 7 (age)
- coef_liv_with_child - Coefficient 8 (living_with_children)
- coef_is_disabled - Coefficient 9 (is_disabled)

Formulas:

```
happy = ordered_logistic(coef_neigh_sat[neigh_sat] +
                         coef_hous_sat[hous_sat] +
                         coef_com_even_avail[com_even_avail] +
                         coef_sen_of_sec[sen_of_sec] +
                         coef_ann_hous_inc * ann_hous_inc +
                         coef_gender[gender[n]] +
                         coef_age[age[n]] +
                         coef_liv_with_child * liv_with_child +
                         coef_is_disabled * is_disabled, c)
```

$$c \sim Normal(0, 3)$$

$$coef_{neighsat} \sim Normal(0, 1)$$

$$coef_{houssat} \sim Normal(0, 1)$$

$$coef_{comevenavail} \sim Normal(0, 1)$$

$$coef_{senofsec} \sim Normal(0, 1)$$

$$coef_{annhousinc} \sim Normal(0, 1)$$

$$coef_{livwithchild} \sim Normal(0, 1)$$

$$coef_{isdisabled} \sim Normal(0, 1)$$

$$coef_{gender} \sim Normal(0, 1)$$

$$coef_{age} \sim Normal(0, 1)$$

Model fit and evaluation

In [11]:

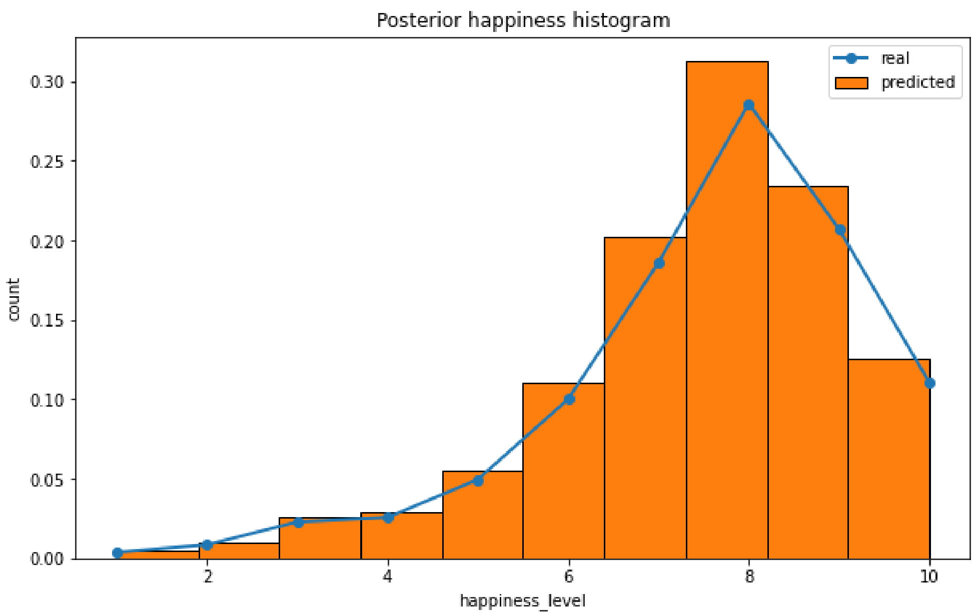
```
model2_fit = CmdStanModel(stan_file='stan_files/model2_fit.stan')

d = {'N' : N,
     'K' : 10,
     'y' : life_satisfaction,
     'neigh_sat' : neighborhood_satisfaction,
     'hous_sat' : housing_satisfaction,
     'com_even_avail' : community_events_availability,
     'sen_of_sec' : sense_of_security,
     'ann_hous_inc' : annual_household_income,
```

```
    'gender' : gender,
    'age' : age,
    'liv_with_child' : living_with_children,
    'is_disabled' : is_disabled}

# Compilation of test2.stan and get 1000 samples
samples2 = model2_fit.sample(data=d, iter_sampling=1000, iter_warmup=1000, chains=4)
print(samples2.diagnose())
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
INFO:cmdstanpy:CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

INFO:cmdstanpy:CmdStan done processing.
Processing csv files: /tmp/tmpxqn1hebi/model2_fit-20230710202823_1.csv, /tmp/tmpxqn1hebi/model2_fit-20230710202823_2.csv, /tmp/tmpxqn1hebi/model2_fit-20230710202823_3.csv, /tmp/tmpxqn1hebi/model2_fit-20230710202823_4.csv

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Effective sample size satisfactory.

Split R-hat values satisfactory all parameters.

Processing complete, no problems detected.
```

Posterior anlysis

In [12]:
```
model2_happy = samples2.stan_variable('happy').flatten()

plt.figure(figsize=(10, 6))
x, y = np.unique(life_satisfaction, return_counts=True)
plt.plot(x, y/sum(y), marker='o', linewidth=2, label='real')
plt.hist(model2_happy, align='mid', density=True, label='predicted', edgecolor="black")
plt.title("Posterior happiness histogram")
plt.xlabel("happiness_level")
plt.ylabel("count")
plt.legend()
plt.show()

df_res = samples2.draws_pd()
display(df_res)
```

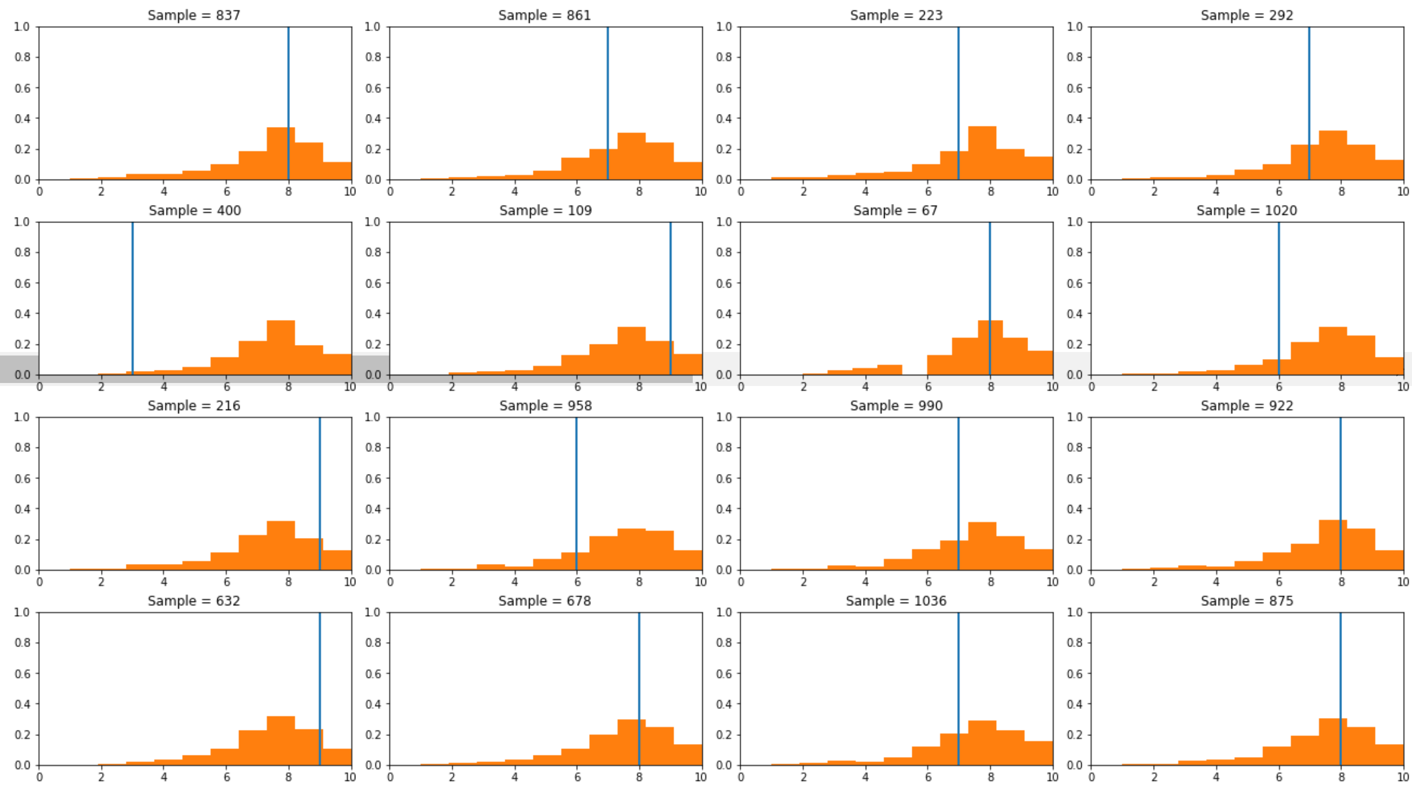


	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__	c[1]	c[2]	c[3]	...	log_lik[104
0	-1841.76	0.992868	0.047128	6.0	127.0	0.0	1870.85	-5.01493	-4.07503	-2.86812	...	-1.8606
1	-1849.40	0.950680	0.047128	7.0	127.0	0.0	1871.16	-4.88874	-3.59861	-2.46249	...	-2.0720
2	-1852.12	0.918759	0.047128	7.0	127.0	0.0	1892.01	-5.54296	-4.08331	-3.24530	...	-1.2631
3	-1853.14	0.782263	0.047128	6.0	63.0	0.0	1880.59	-5.42495	-4.12209	-3.13397	...	-1.5558
4	-1854.79	0.999079	0.047128	7.0	191.0	0.0	1889.59	-4.56539	-3.28938	-2.19560	...	-1.4241
...	...	...	...	...	...	...	...	...	...	...	...	
3995	-1859.94	0.994283	0.043507	6.0	63.0	0.0	1885.13	-4.34495	-2.84745	-1.41942	...	-1.8741
3996	-1856.92	0.975970	0.043507	6.0	63.0	0.0	1895.16	-3.62266	-3.03983	-1.95594	...	-2.0665
3997	-1856.55	0.859484	0.043507	6.0	127.0	0.0	1894.97	-3.16488	-2.55579	-1.40362	...	-1.6581
3998	-1853.51	0.840398	0.043507	6.0	63.0	0.0	1892.78	-3.48895	-2.67679	-1.35653	...	-2.0337
3999	-1854.00	0.925420	0.043507	6.0	127.0	0.0	1895.47	-4.34023	-3.39489	-1.92758	...	-1.8265

4000 rows × 2164 columns

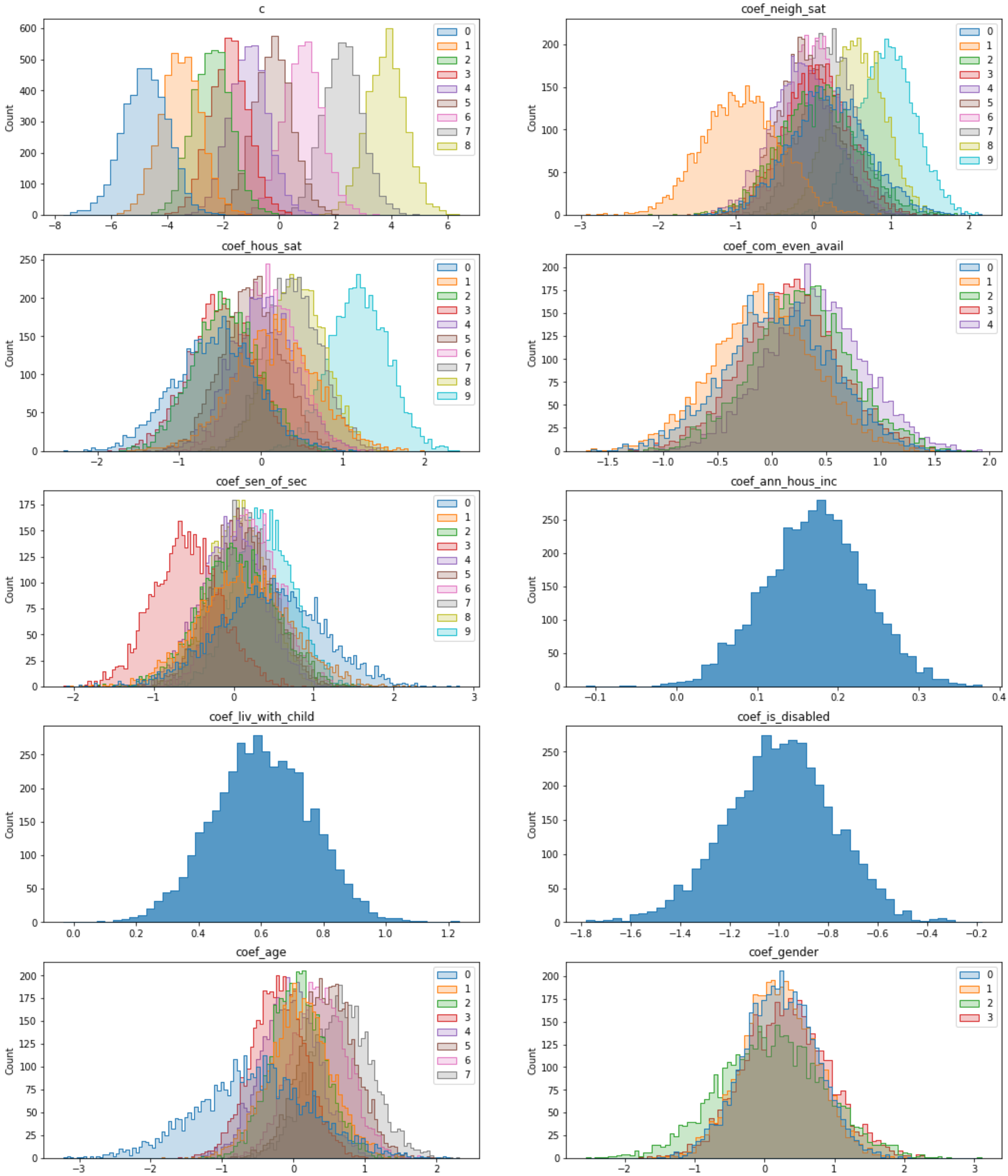
In [13]:

```
model2_happy = samples2.stan_variable('happy')
arr = np.random.randint(low=1, high=N, size = 16)


fig, axs = plt.subplots(4, 4, figsize=(18, 10), layout='constrained')
for ax, sample in zip(axs.flat, arr):
    x = life_satisfaction[sample]
    ax.plot([x,x], [-0.2,1.2], linewidth=2, label='real')
    ax.hist(model2_happy[sample], align='mid', density=True, label='predicted')
    ax.set_xlim([0, 10])
    ax.set_ylim([0, 1])
    ax.set_title(f"Sample = {sample}")
plt.show()
```



In [14]:

```
lst = ['c','coef_neigh_sat','coef_hous_sat','coef_com_even_avail','coef_sen_of_sec',
       'coef_ann_hous_inc','coef_liv_with_child','coef_is_disabled','coef_age','coef_gender']

fig, axs = plt.subplots(5, 2, figsize=(18, 22))
for ax, param in zip(axs.flat, lst):
    coeffs = samples2.stan_variable(param)
    sns.histplot(ax=ax, data=coeffs, element="step")
    ax.set_title(param)

plt.show()
```



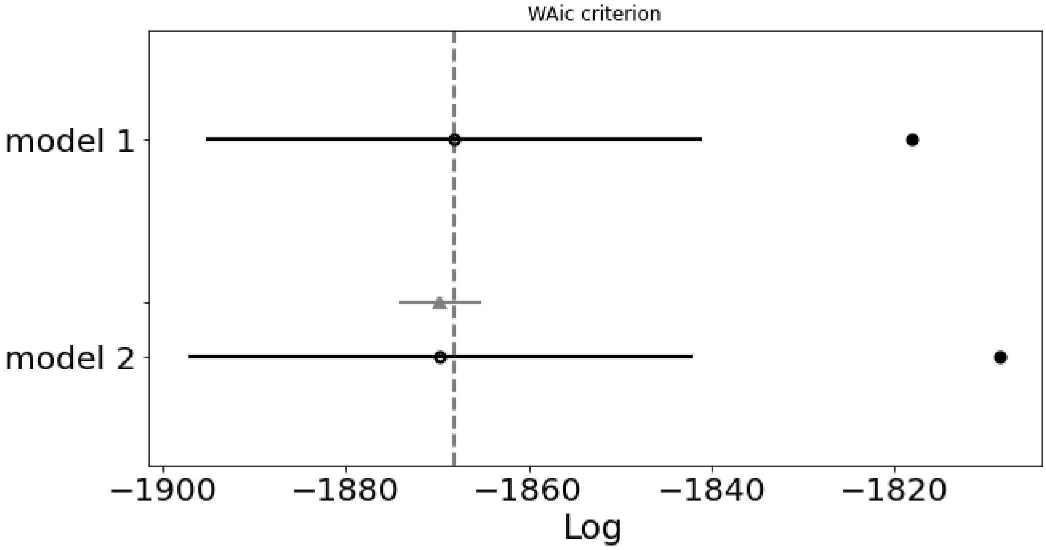
Coefficents for second model are similar to first one. We can see mean of cooeficents for age and sex is aroud 0. But we cas see that age does have impact on our model accoring to paper mentioned above.

5. Model Comparison

In [15]:

```
az_pred1 = az.from_cmdstanpy(samples1)
az_pred2 = az.from_cmdstanpy(samples2)
waic_compare = az.compare({"model 1" : az_pred1, "model 2" : az_pred2}, ic="waic")
az.plot_compare(waic_compare, figsize=(10, 5))
plt.title("WAic criterion")
plt.show()
```

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



In [16]:
```
waic_compare
```

Out[16]:

	rank	waic	p_waic	d_waic	weight	se	dse	warning	waic_scale
model 1	0	-1868.093918	50.038911	0.000000	0.574616	27.090107	0.000000	True	log
model 2	1	-1869.629942	61.171985	1.536024	0.425384	27.519176	4.417561	True	log

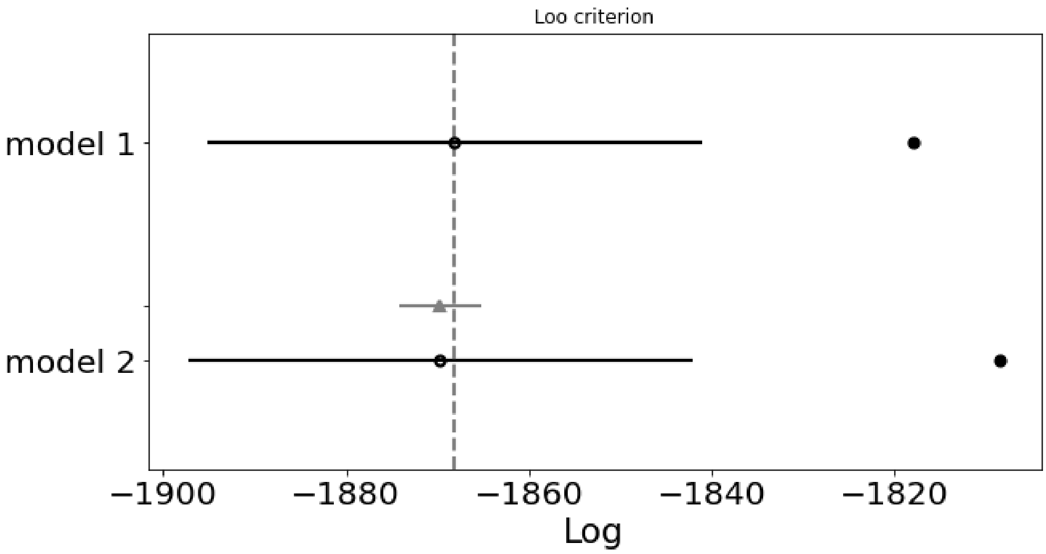
According to WAIC models are overlapping. Which is not expected as we have thought that adding parameters like age and gender would make the fit better.

There is one warning while calulcating the WAIC score:

UserWarning: For one or more samples the posterior variance of the log predictive densities exceeds 0.4. This could be indication of WAIC starting to fail.

This warning can indicate model misspecification, overfitting, or other issues related to the data or modeling assumptions.

In [17]:
```
loo_compare = az.compare({"model 1" : az_pred1, "model 2" : az_pred2}, ic="loo")
az.plot_compare(loo_compare, figsize=(10, 5))
plt.title("Loo criterion")
plt.show()
```



In [18]:
```
loo_compare
```

Out[18]:

	rank	loo	p_loo	d_loo	weight	se	dse	warning	loo_scale
model 1	0	-1868.131389	50.076382	0.000000	0.577287	27.090797	0.000000	False	log
model 2	1	-1869.720103	61.262147	1.588714	0.422713	27.521378	4.418562	False	log

Similar to WAIC models according to LOO criterion models are also overlapping. There is no winner, which we did not expect.

Loo creterion does not raise any warnings or erros.

The difference between both models is insignificant. It was surprising to learn because based on our assumptions age and gender should have a significant impact on happiness level. This could indicate more complex hidden confounds. We are content with the results, but predicting human happiness is a complex problem and we have only scratched the surface. Models require more analysis.

It is worth mentioning that finding an appropriate dataset was a difficult task. Most available datasets contained only data summaries, we were not able to get individual responses.