Diamond Price Analysis

Authors:

- Jan Kwapień
- Jakub Pawłowski

The problem:

The purpose of the project was to analyze car's fuel consumption based on it's engine size and taking into account its other characteristics such as drag coefficient and number of cylinders in engine.

The goal:

We hope, that after creating sufficient model it will be possible to predict a fuel consumption for a car given information about it's engine size also taking into account it's drag coefficient and number of cylinders in engine.

It may be possible to estimate a fuel consumption for a car without any domain specific knowledge, which could help to recognise problems in working of mentioned car.

Table of contents

Dataset

Return to table of contents

The data was sourced from Kaggle.com which is an online community of data scientists. The dataset can be downloaded **here**.

Dataset contains 946 records containing description of 11 car properties.

The colums are as follows:

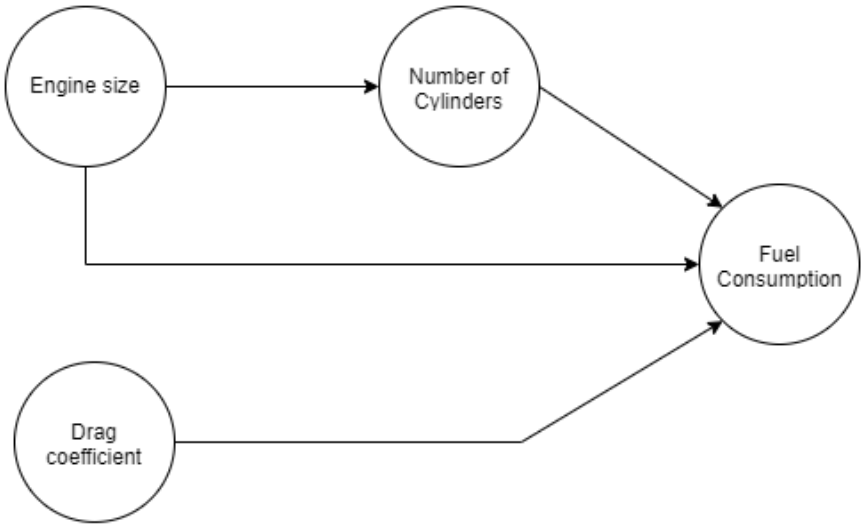
- **Model Year** - year of production
- **Company** - the company that produced car
- **Model** - name of the car model
- **Vehicle Class** - class of the car
- **Engine Size** - engine size in liters
- **Cylinders** - number of cylinders
- **Transmission** - type of transmission
- **Fuel Type** - type of used fuel
- **City Fuel Consumption** - city fuel consumption ratings shown in liters per 100 kilometers
- **Highway Fuel Consumption** - highway fuel consumption ratings shown in liters per 100 kilometers

---

Directed Acyclic Graph

Return to table of contents

Below is presented proposition of Directed Acyclic Graph connected to described problem. Chosen predictor parameters are Engine size, Number of cylinders and Drag coefficient.



There are visible structures such as:

- pipe - connection from Drag coefficient to Fuel consumption
- fork - Engine size as the parent node, create fork leading to Number of cylinders and Fuel consumption
- collider - Fuel Consumption is a collider node for nodes Engine size, Number of Cylinders, and Drag coefficient

---

Imports

Return to table of contents

Necessary python modules for data analysis.

In [ ]:
```
from cmdstanpy import CmdStanModel

import arviz as az
import numpy as np
import scipy.stats as stats

import matplotlib.pyplot as plt
import pandas as pd
import random as rd
import seaborn as sns
```

Data preprocessing

Return to table of contents

Before starting analysis it may be necessary to clean up and make adjustment to the dataset.

Drag coefficient

Chosen dataset does not contain column describing drag coefficient parameter. Because it is crucial information it was necessary to add this from outside source. Because of insufficient data it was decided to assign mean value of drag coefficient to record based on its class (the difference in parameter values is small for cars belonging to the same vehicle class).

Read the data set

In [ ]:
```
dataset = pd.read_csv("Data/MY2022 Fuel Consumption Ratings.csv")
```

Make a list of all the unique car types and assign them to a variable

In [ ]:
```
carTypes = dataset['Vehicle Class']
uniqueTypes = pd.DataFrame(carTypes.unique())

compacts = dataset[dataset['Vehicle Class']== 'Compact']
suvSmall =  dataset[dataset['Vehicle Class']== 'SUV: Small']
midSize =  dataset[dataset['Vehicle Class']== 'Mid-size']
miniCompact =  dataset[dataset['Vehicle Class']== 'Minicompact']
suvStandard =  dataset[dataset['Vehicle Class']== 'SUV: Standard']
twoSeater =  dataset[dataset['Vehicle Class']== 'Two-seater']
subCompact =  dataset[dataset['Vehicle Class']== 'Subcompact']
stationWagonSmall =  dataset[dataset['Vehicle Class']== 'Station wagon: Small']
stationWagonMid =  dataset[dataset['Vehicle Class']== 'Station wagon: Mid-size']
fullSize =  dataset[dataset['Vehicle Class']== 'Full-size']
pickupSmall =  dataset[dataset['Vehicle Class']== 'Pickup truck: Small']
pickupStandard =  dataset[dataset['Vehicle Class']== 'Pickup truck: Standard']
minivan = dataset[dataset['Vehicle Class']== 'Minivan']
specialPurposeVehicle = dataset[dataset['Vehicle Class']== 'Special purpose vehi
```

Assign mean value of drag coefficient to every car in each vehicle class and creating the Drag coefficient column

In [ ]:
```
cx_compacts =(0.28+ 0.32)/2
cx_suv_small =(0.34 + 0.4)/2
cx_mid_size = (0.28 + 0.33)/2
cx_minicompact = (0.3 + 0.35)/2
cx_suv_standard =( 0.35 + 0.45)/2
cx_two_seater = (0.3 + 0.35)/2
cx_subcompact = (0.3 + 0.35)/2
cx_station_wagon_small = (0.3 + 0.35)/2
cx_station_wagon_mid_size = (0.32 + 0.36)/2
cx_full_size = (0.28 + 0.35)/2
cx_pickup_small = (0.38 + 0.45)/2
cx_pickup_standard = (0.4 + 0.5)/2
cx_minivan = (0.3 + 0.35)/2
cx_special_purpose = (0.33 + 0.37)/2
```

```
mapping = {
    'Compact': cx_compacts,
    'SUV: Small': cx_suv_small,
    'Mid-size': cx_mid_size,
    'Minicompact': cx_minicompact,
    'SUV: Standard': cx_suv_standard,
    'Two-seater': cx_two_seater,
    'Subcompact': cx_subcompact,
    'Station wagon: Small': cx_station_wagon_small,
    'Station wagon: Mid-size': cx_station_wagon_mid_size,
    'Full-size': cx_full_size,
    'Pickup truck: Small': cx_pickup_small,
    'Pickup truck: Standard': cx_pickup_standard,
    'Minivan': cx_minivan,
    'Special purpose vehicle': cx_special_purpose
}

dataset['Drag coefficient'] = dataset['Vehicle Class'].map(mapping).fillna(datas
dataset.to_csv('Data/processed_dataset.csv')
```

Dropping unnecessary data

The columns which are not needed to perform correct analysis are: "Model Year","Transmission","Fuel Type","Fuel Consumption (City (L/100 km)","Fuel Consumption(Comb (L/100 km))","Fuel Consumption(Comb (mpg))","CO2 Emissions(g/km)","CO2 Rating","Smog Rating" and "Vehicle Class". The unnecessary columns should be dropped to not blur our work.

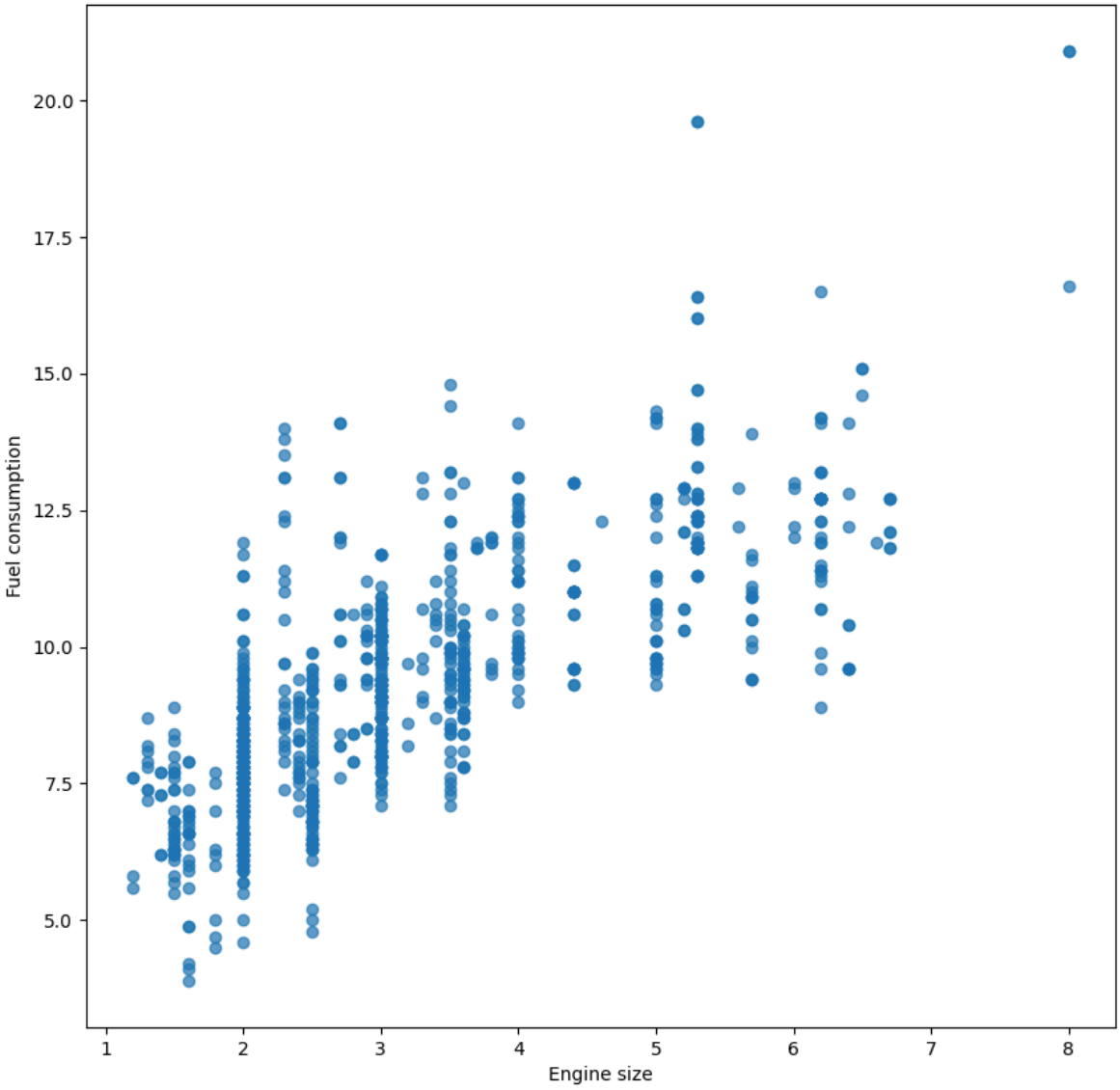
In [ ]:
```
dataset.drop(columns=["Model Year",'Transmission',"Fuel Type","Fuel Consumption
dataset.to_csv('Data/processed_dataset.csv')
```

Plotting dataset

Return to table of contents

It is important to see the data before commencing analysis, afterall we should check in case it's utter nonsense as demonstrated **here**.

In [ ]:
```
dataFrame = pd.read_csv("Data/processed_dataset.csv")
dataFrame = dataFrame.sort_values(by ='Engine Size(L)')
plt.figure(0,figsize=(10,10))
plt.scatter(dataFrame['Engine Size(L)'], dataFrame['Fuel Consumption(Hwy (L/100
plt.xlabel('Engine size')
plt.ylabel('Fuel consumption')
plt.show()
```



Data Analysis

Return to table of contents

For analysis we have created 2 bayesian models.

- Model 1 - uses 2 predictors: engine size and drag coefficient
- Model 2 - uses 3 predictors: engine size, drag coefficient and number of cylinders

Expanding the first model by increasing the number of predictors allows for a better fit of the model to the observations, in terms of the data, and for value prediction.

The equations, parameters and differences of individual models are presented in the corresponding chapters.

Model 1 - two predictors

Return to table of contents

Model has form:

$$price_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha + \beta_{engine\ size}[engine\ size_i] + \beta_{drag\ coefficient}[drag\ coefficient_i]$$

With parameter distributions set as follows:

$$\alpha \sim \text{Normal}(8.5, 3)$$

$$\beta_{engine\ size} \sim \text{Normal}(0, 1)$$

$$\beta_{drag\ coefficient} \sim \text{Normal}(0, 1)$$

$$\sigma \sim \text{Normal}(2, 1)$$

The required input data is the set of cars with information about engine size and drag coefficient for which the user wants to make a prediction.

Model 1 - Prior predictive check

Return to table of contents

First step is prior predicive check whether parameter values and distributions "make sense".

Parameters simulated from priors are a result of the model definition.

Priors were selected experimentally, starting with small, typical distributions (eg. Normal(0,10)) up to final values based on resultant plot.

On the basis of the obtained parameter values, it can be concluded that the prior selection was successful, the values are in line with the expectations.

Based on the shape of obtained cone which contains most of datapoints, it can be concluded that the prior predictive was successful. The obtained lines include points as expected.

PPC stan code:

```
data{
    int N;
    array[N] real<lower=1> engine_size;
    array[N] real<lower=0> drag_coeficient;
}

generated quantities {
    real alpha;
    real beta_engine_size;
    real beta_drag_coeficient;
    real  sigma;
    vector[N] fuel_consumption;
    alpha = normal_rng(8.5,3);
    beta_engine_size = normal_rng(0, 1);
    beta_drag_coeficient = normal_rng(0, 1);
    sigma = normal_rng(2, 1);
    for(i in 1:N)
    {
        fuel_consumption[i] = normal_rng(alpha+beta_engine_size*engine_size[i]+beta_drag_coeficient*drag_coeficient[i],sigma);
    }

}
```

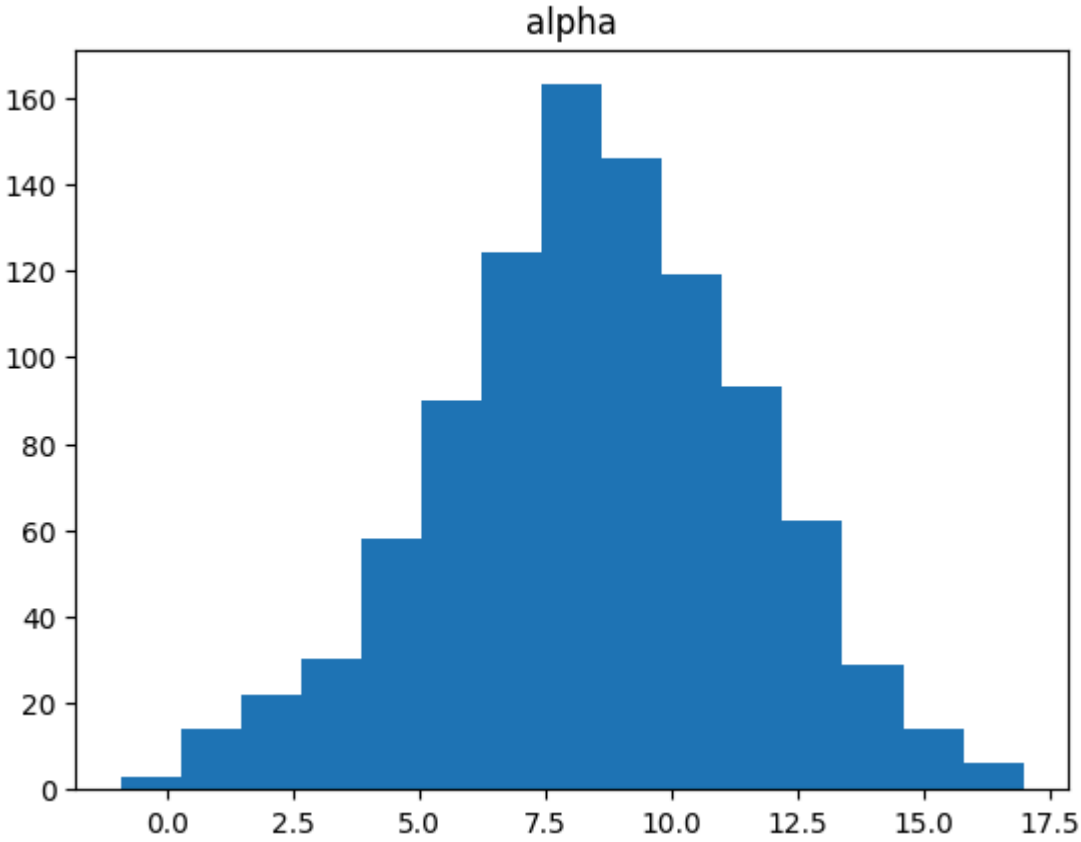
In [ ]:
```
model_1_prior=CmdStanModel(stan_file='Stan_files/model_1_prior.stan')
```

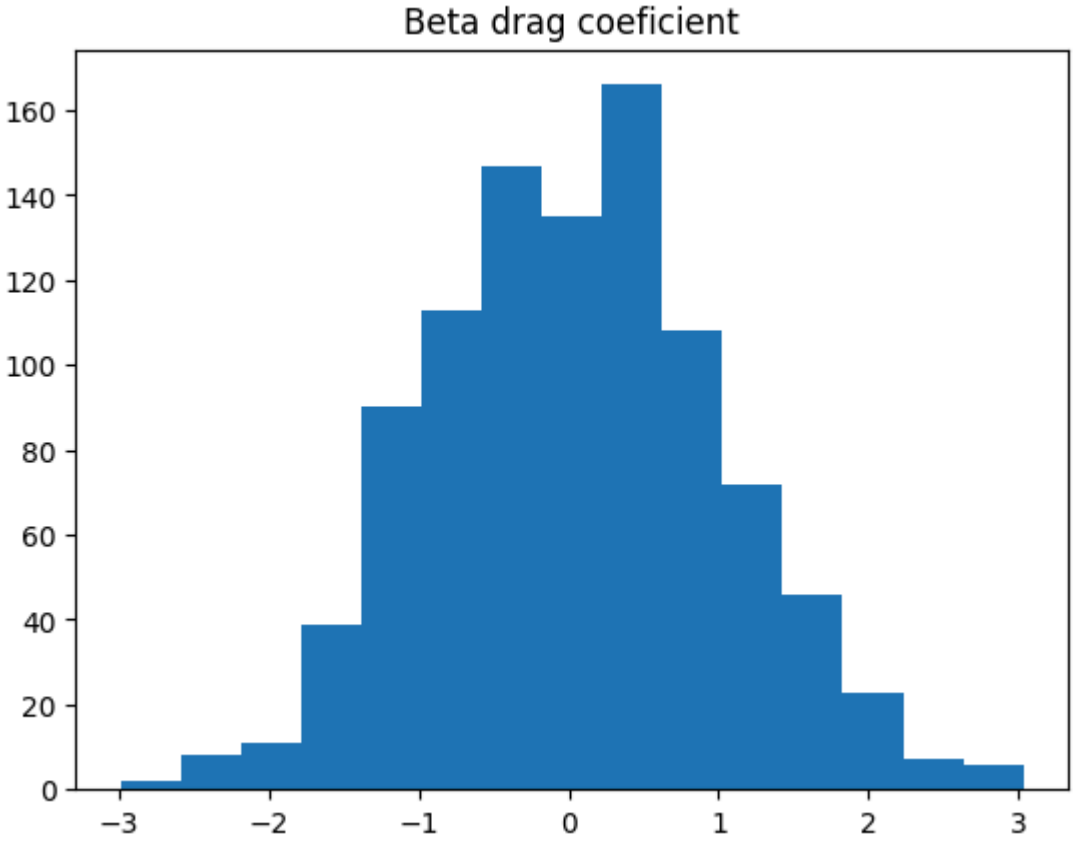
```
00:56:41 - cmdstanpy - INFO - compiling stan file C:\Users\jkwap\Desktop\Sem_I\DA
\Project\data_analitics\Stan_files\model_1_prior.stan to exe file C:\Users\jkwap
\Desktop\Sem_I\DA\Project\data_analitics\Stan_files\model_1_prior.exe
00:58:07 - cmdstanpy - INFO - compiled model executable: C:\Users\jkwap\Desktop\S
em_I\DA\Project\data_analitics\Stan_files\model_1_prior.exe
```

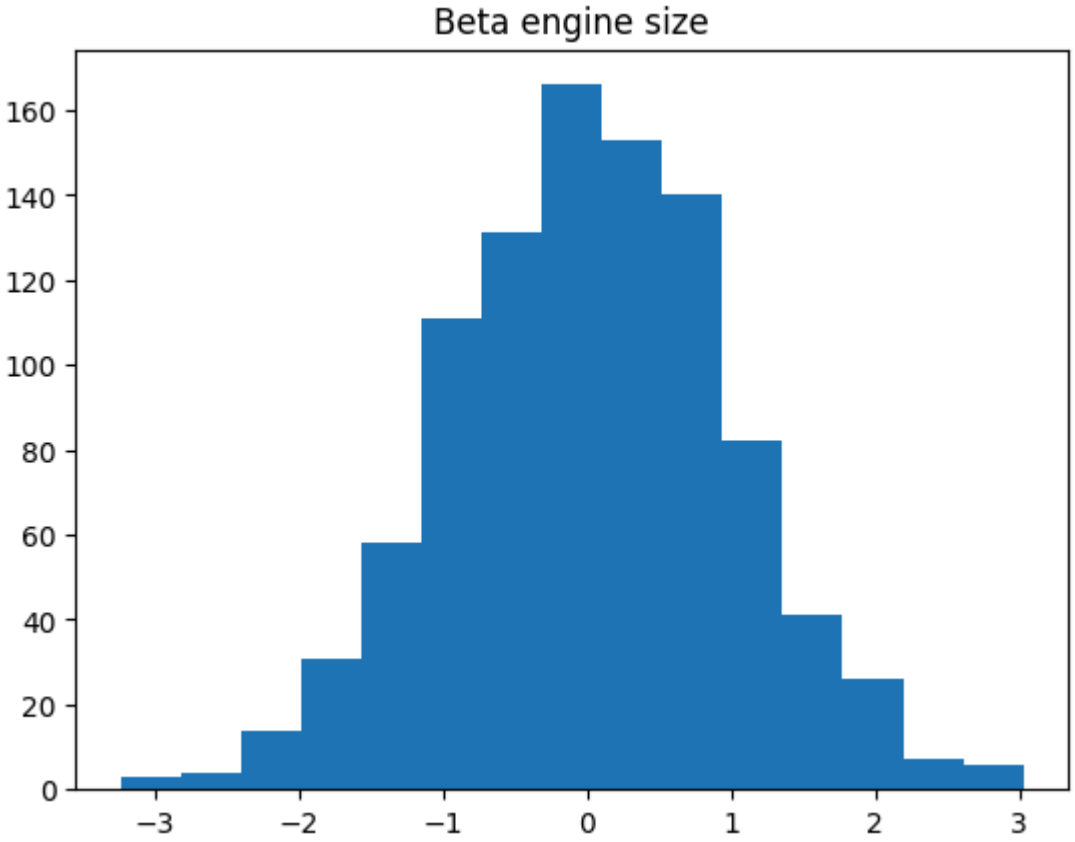
In [ ]:
```
X = 1000
N = dataFrame.shape[0]
engine_size_data = dataFrame['Engine Size(L)']
cylinders_data = dataFrame['Cylinders']
drag_coeficient_data = dataFrame['Drag coefficient']
fuel_consumption_data = dataFrame['Fuel Consumption(Hwy (L/100 km))']
```

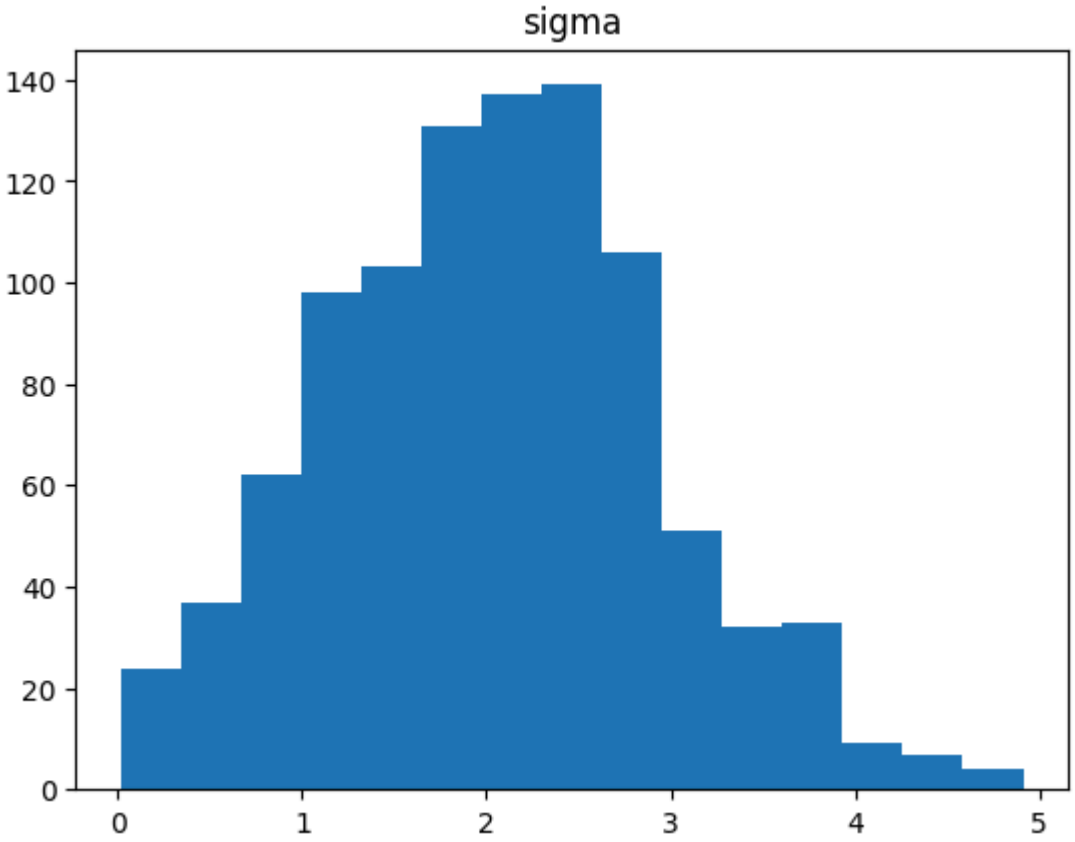
In [ ]:
```
samples_1_prior = model_1_prior.sample(data = {'N':N, 'engine_size':engine_size_
iter_sampling=X,
iter_warmup=2000,
chains=1,
seed=29042020,
refresh=N)
```

In [ ]:
```
dataFrame_model_1_prior = samples_1_prior.draws_pd()
plt.figure(0)
plt.hist(dataFrame_model_1_prior.alpha,bins=15)
plt.title('alpha')
plt.figure(1)
plt.hist(dataFrame_model_1_prior.beta_drag_coeficient,bins=15)
plt.title('Beta drag coeficient')
plt.figure(2)
plt.hist(dataFrame_model_1_prior.beta_engine_size,bins=15)
plt.title('Beta engine size')
plt.figure(3)
plt.hist(dataFrame_model_1_prior.sigma,bins=15)
plt.title('sigma')
plt.show()
```


alpha
160
140
120
100
80
60
40
20
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Beta drag coeficient
160
140
120
100
80
60
40
20
0
−3
−2
−1
0
1
2
3





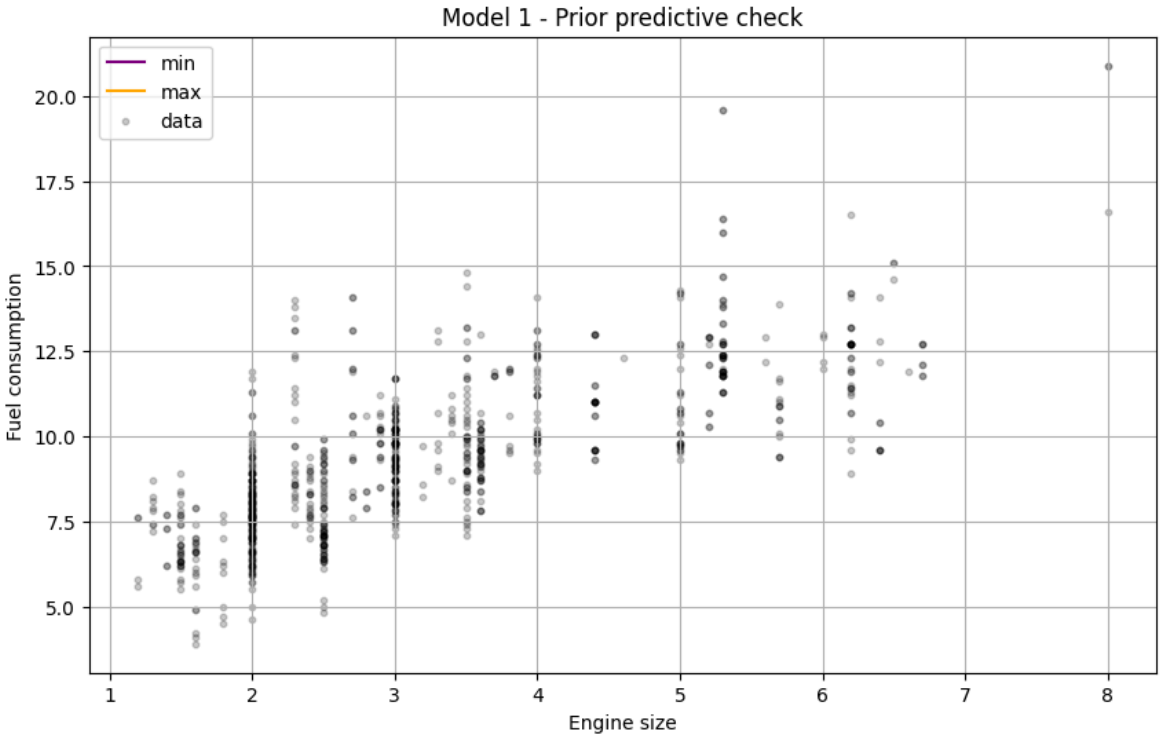
Model 1 - Comparing margin prior values with data

Return to table of contents

In [ ]:
```
def calcQuants(x, y):
    qlvls = [0, 1]
    quansList = [[], []]
    for i in range(y.shape[-1]):
        temp = y[:, i]
        for q, lvl in zip(quansList, qlvls):
            q.append(np.quantile(temp, lvl))
    return quansList

def quantsExtremes(df, y, q):
    engine_uq = df['Engine Size(L)'].unique()
    engine_uq = sorted(engine_uq)
    quansList = calcQuants(df['Engine Size(L)'], y)
    engineQuantDict = dict()
    for engine_val in engine_uq:
        engineList = np.array(df['Engine Size(L)'].tolist())
        idxs = np.where(engineList == engine_val)[0]
        qval = quansList[q][idxs[0]]
        for i in idxs:
            if q == 0 and quansList[q][i] < qval:
                qval = quansList[q][i]
            elif q == 1 and quansList[q][i] > qval:
                qval = quansList[q][i]
        engineQuantDict[engine_val] = qval
    return engineQuantDict
```

In [ ]:
```
plt.figure(figsize=[10, 6])

price_sim = samples_1_prior.stan_variable('fuel_consumption')

engQuantMinDict = quantsExtremes(dataFrame, price_sim, 0)
engMin = (engQuantMinDict.keys())
quantMin = (engQuantMinDict.values())

engQuantMaxDict = quantsExtremes(dataFrame, price_sim, 1)
engMax = (engQuantMaxDict.keys())
quantMax = (engQuantMaxDict.values())

plt.figure(figsize=[10, 6])
plt.plot(engMin, quantMin, color='purple')
plt.plot(engMax, quantMax, color='orange')
plt.scatter(dataFrame['Engine Size(L)'], dataFrame['Fuel Consumption(Hwy (L/100
plt.xlabel("Engine size")
plt.ylabel("Fuel consumption")
plt.title("Model 1 - Prior predictive check")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Based on the shape of obtained figure which contains most of datapoints, it can be concluded that the prior predictive was successful. The obtained lines include points as expected.

Model 1 - Posterior analysis

Return to table of contents

After confirming that the priors values and trajectories are correct we can start a proper analysis.

As mentioned previously sampling time was heavily dependent on number of datapoints, but, according to diagnose result, no other issues were encountered.

Posterior stan code:

```
data {
    int<lower=0> N;
    array[N] real<lower=0> engine_size;
    array[N] real<lower=0> drag_coeficient;
    array[N] real <lower=0> fuel_consumption;
}

parameters {
    real alpha;
    real beta_engine_size;
    real beta_drag_coeficient;
    real<lower=0> sigma;
}

model {

    alpha ~ normal(8.5, 3);
    beta_engine_size ~ normal(0, 1);
    beta_drag_coeficient ~ normal(0, 1);
    sigma ~ normal(0, 1);


    for(i in 1:N)
    {
    fuel_consumption[i] ~ normal(alpha + beta_engine_size * engine_size[i] +
                                 beta_drag_coeficient * drag_coeficient[i], sigma);
    }
}

generated quantities {
   vector[N] y_out;
    vector[N] log_lik;
   for(i in 1:N)
    {
          log_lik[i] = normal_lpdf(fuel_consumption[i] | alpha + beta_engine_size * engine_size[i] + beta_drag_coeficient * drag_coeficient[i], sigma);

    y_out[i] = normal_rng(alpha + beta_engine_size * engine_size[i] + beta_drag_coeficient * drag_coeficient[i], sigma);
    }
}
```

In [ ]:
```
model_1_post=CmdStanModel(stan_file='Stan_files/model_1_post.stan')
```

In [ ]:
```
samples_1_post = model_1_post.sample(data = {'N':N, 'engine_size':engine_size_da
iter_sampling=X,
iter_warmup=200,
chains=1,
seed=29042020,
refresh=N)
```

```
02:41:24 - cmdstanpy - INFO - CmdStan start processing
chain 1 |██████████| 00:08 Sampling completed
```

```
02:41:32 - cmdstanpy - INFO - CmdStan done processing.
```

Model 1 - model parameters

Return to table of contents

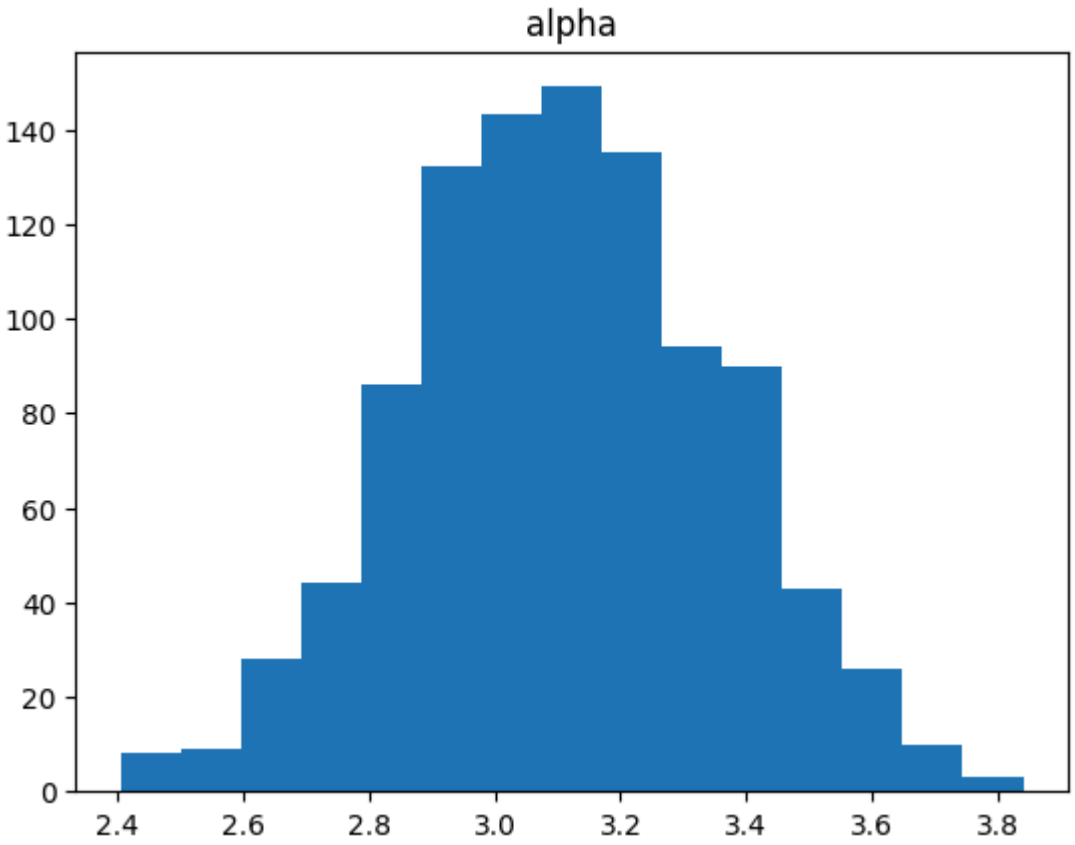
We can also extract stan variables that are used in the final fuel consumption prediction equation.

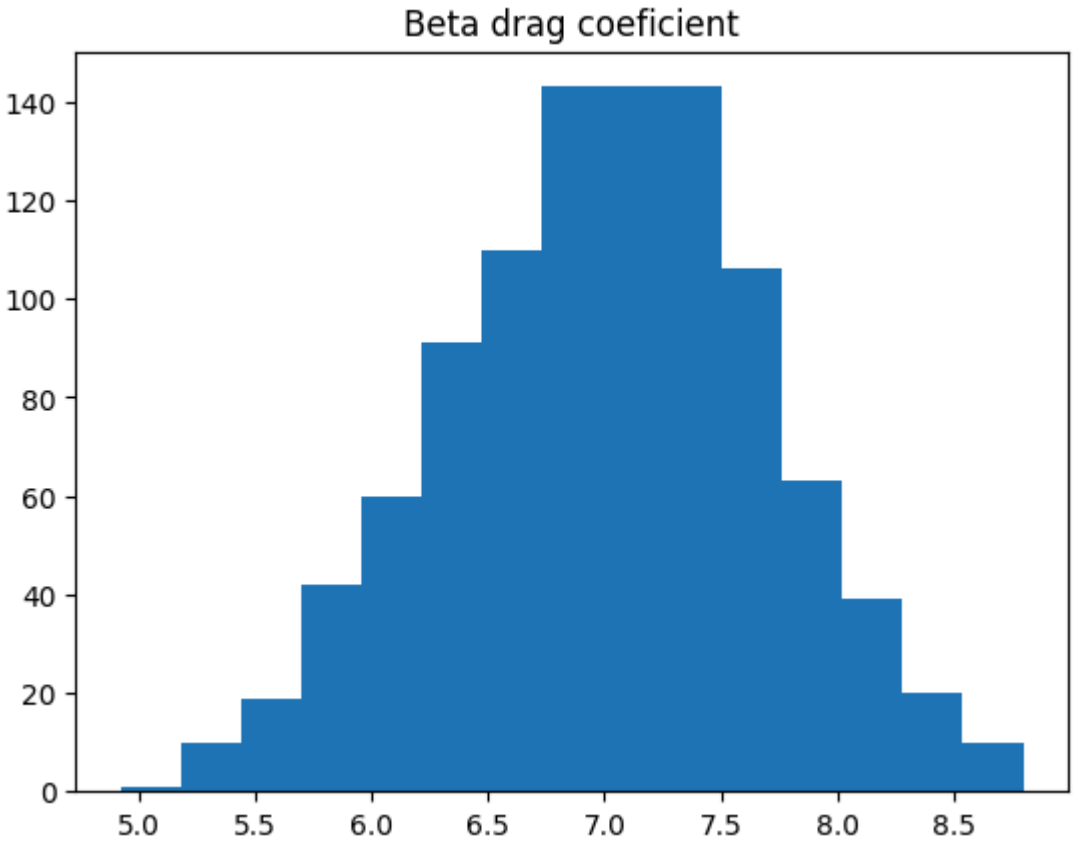
Based on the presented graphs and histograms of parameters, it can be concluded that parameter values are relatively concentrated.

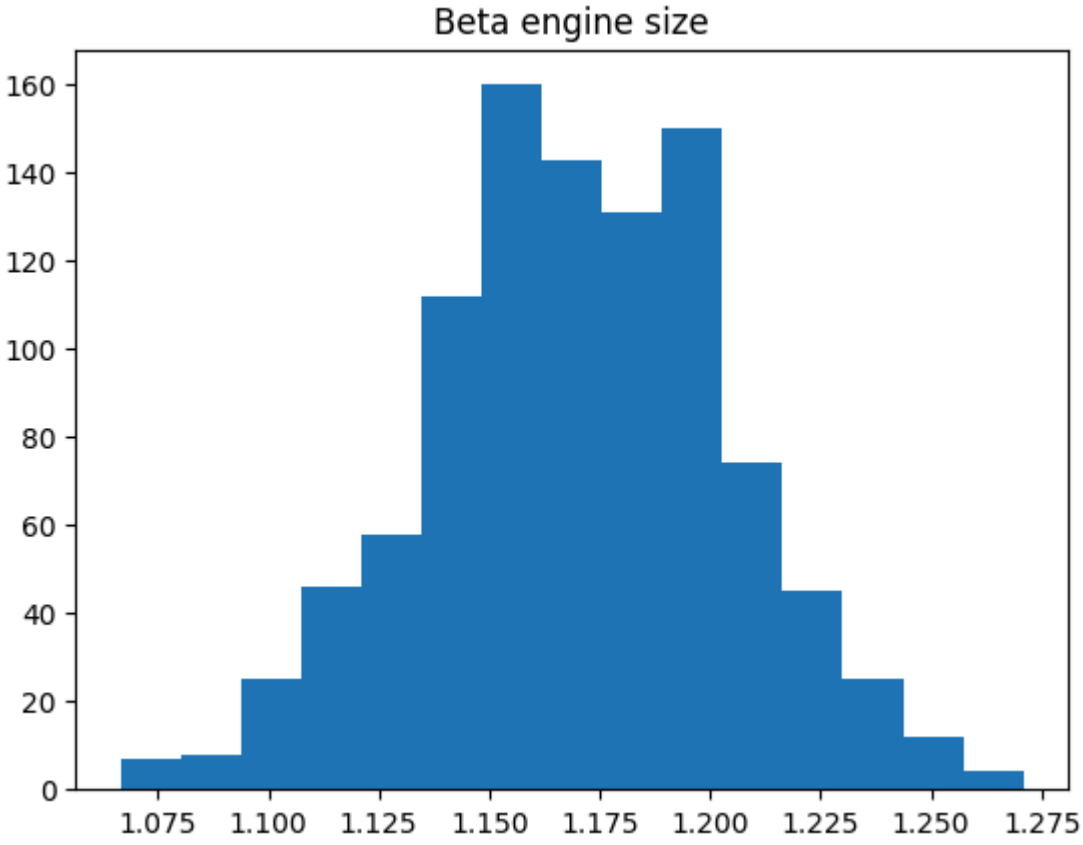
Their slight dispersion is indicative of the cars in the same class being slightly different from one another.

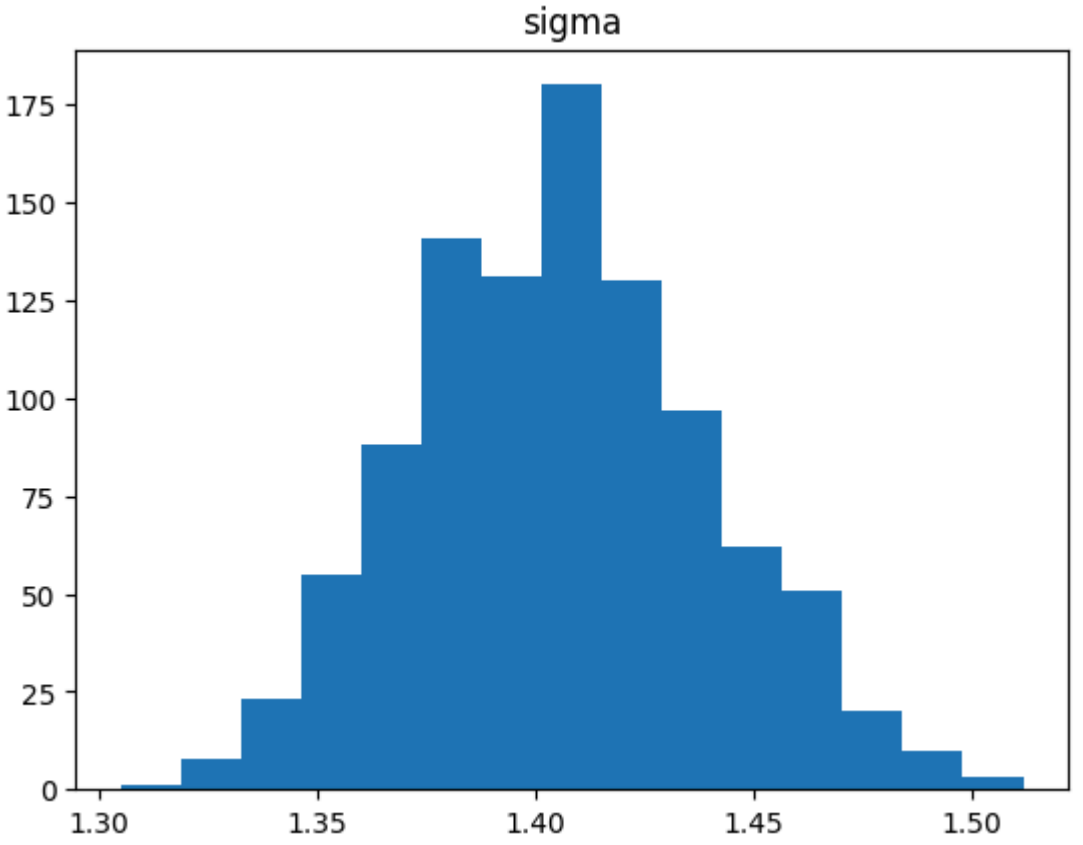
In [ ]:
```
dataFrame_model_1_post = samples_1_post.draws_pd()

plt.figure(0)
plt.hist(dataFrame_model_1_post.alpha,bins=15)
plt.title('alpha')
plt.figure(1)
plt.hist(dataFrame_model_1_post.beta_drag_coeficient,bins=15)
```

```
plt.title('Beta drag coeficient')
plt.figure(2)
plt.hist(dataFrame_model_1_post.beta_engine_size,bins=15)
plt.title('Beta engine size')
plt.figure(3)
plt.hist(dataFrame_model_1_post.sigma,bins=15)
plt.title('sigma')
plt.show()
```









Model 1 - evaluation

Return to table of contents

We can now observe the results.

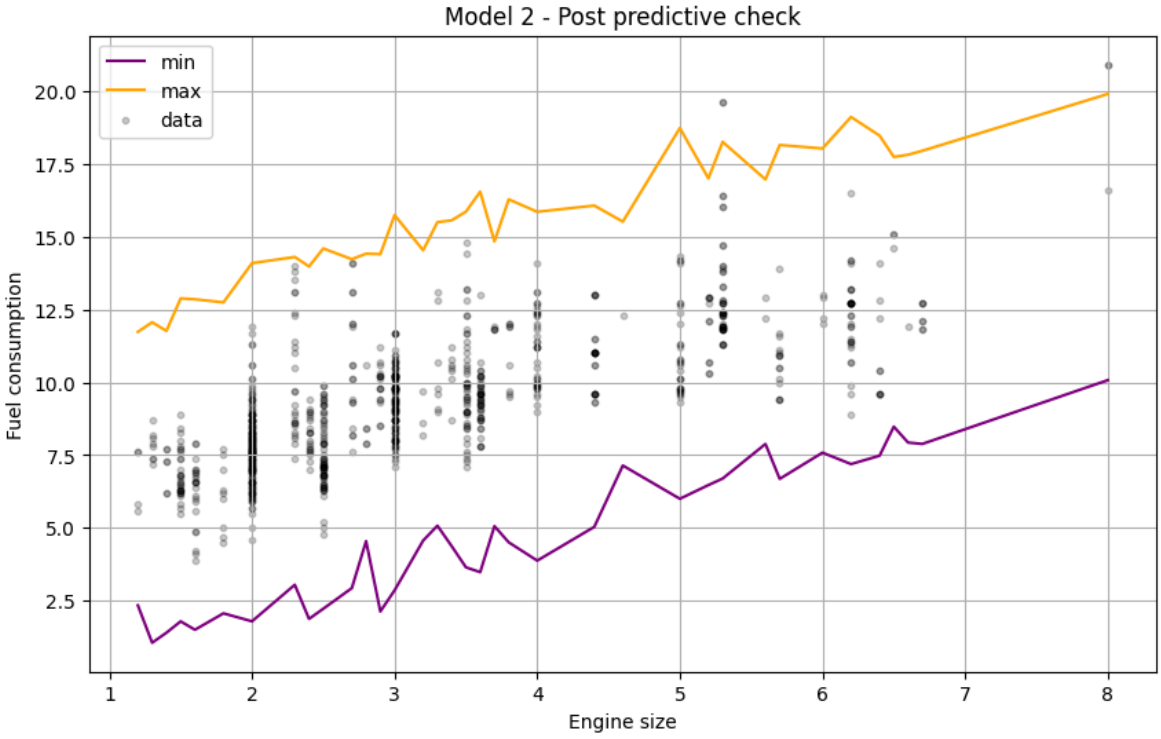
Model 1 - quantiles

Return to table of contents

After simulating we can analyze predictions. Most of the simulated cars fall within real data ranges.

In [ ]:
```
plt.figure(figsize=[10, 6])

price_sim = samples_1_post.stan_variable('y_out')

engQuantMinDict = quantsExtremes(dataFrame, price_sim, 0)
engMin = (engQuantMinDict.keys())
quantMin = (engQuantMinDict.values())

engQuantMaxDict = quantsExtremes(dataFrame, price_sim, 1)
engMax = (engQuantMaxDict.keys())
quantMax = (engQuantMaxDict.values())

plt.figure(figsize=[10, 6])
plt.plot(engMin, quantMin, color='purple')
plt.plot(engMax, quantMax, color='orange')
plt.scatter(dataFrame['Engine Size(L)'], dataFrame['Fuel Consumption(Hwy (L/100
plt.xlabel("Engine size")
plt.ylabel("Fuel consumption")
plt.title("Model 2 - Post predictive check")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



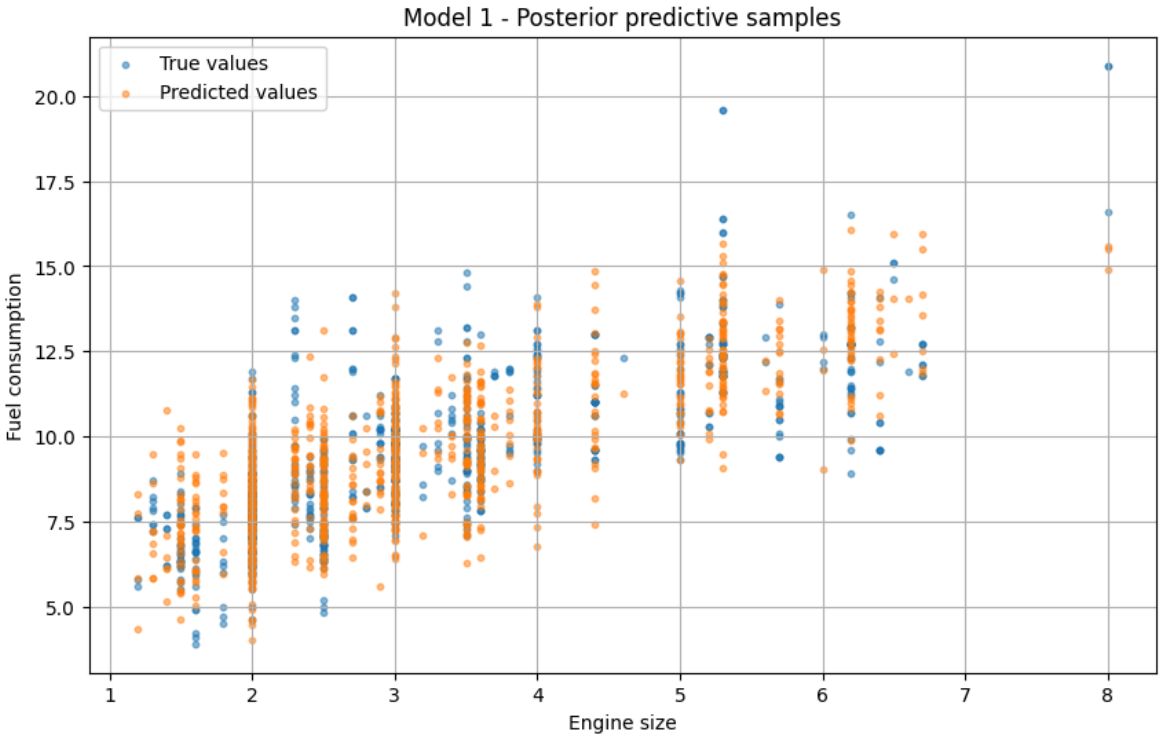
Model 1 - predictions and density plot

Return to table of contents

As seen on the plots the model is somewhat sufficient. It describes most accurately cars with common engine sizes. Despite this, fuel consumption figures are quite scattered.

Perhaps adding another predictor could solve the issue.

In [ ]:
```
price_sim = samples_1_post.stan_variable('y_out')
plt.figure(figsize=[10,6])
plt.scatter(engine_size_data, fuel_consumption_data, alpha=0.5, s=10)
plt.scatter(engine_size_data, price_sim[0], alpha=0.5, s=10)
plt.title("Model 1 - Posterior predictive samples")
plt.legend(["True values", "Predicted values"])
plt.xlabel("Engine size")
plt.ylabel("Fuel consumption")
plt.grid()
plt.show()
```

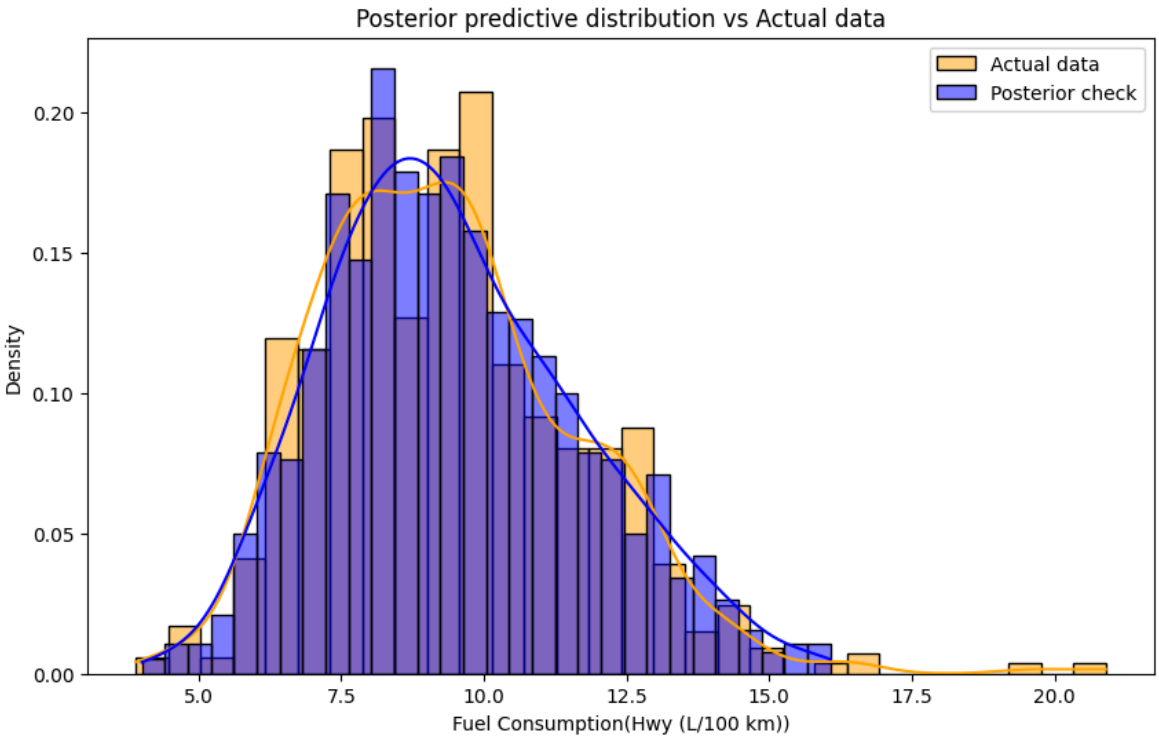


In [ ]:
```
dataFrame_model_1_post = samples_1_post.draws_pd()

list = []
for i in range(1,N):
    val = dataFrame_model_1_post['y_out[' + str(i)+ ']'].mean()
    list.append(val)

actual_data_flat = fuel_consumption_data
plt.figure(figsize=(10,6))
sns.histplot(actual_data_flat, bins =30, kde=True, stat='density', alpha=0.5,lab
sns.histplot(list, bins =30, kde=True, stat='density', alpha=0.5,label='Posterio
plt.legend()
plt.title('Posterior predictive distribution vs Actual data')
```

```
02:03:39 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:02 Sampling completed
```

```
02:03:41 - cmdstanpy - INFO - CmdStan done processing.
```

Out[ ]: Text(0.5, 1.0, 'Posterior predictive distribution vs Actual data')



Model 2 - three predictors

Return to table of contents

Model has form:

$$price_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha + \beta_{engine\ size}[engine\ size_i] + \beta_{drag\ coefficient}[drag\ coefficient_i] + \beta_{cylinders\ number}$$

With parameter distributions set as follows:

$$\alpha \sim \text{Normal}(8.5, 3)$$

$$\beta_{engine\ size} \sim \text{Normal}(0, 1)$$

$$\beta_{drag\ coefficient} \sim \text{Normal}(0, 1)$$

$$\beta_{cylinders\ number} \sim \text{Normal}(0, 1)$$

$$\sigma \sim \text{Normal}(2, 1)$$

The required input data is the set of cars with engine size, cylinders number and drag coefficient for which the user wants to make a prediction.

Model 2 - Prior predictive check

Return to table of contents

First step is prior predicive check whether parameter values and distributions "make sense".

Parameters simulated from priors are a result of the model definition.

Some priors are derived from the first model as the second one is it's expansion. Rest of which were selected experimentally following the same procedure, starting with small, typical distributions (eg. Normal(0,10)) up to final values based on resultant plot.

On the basis of the obtained parameter values, it can be concluded that the prior selection was successful, the values are in line with the expectations.

Based on the shape of obtained figure which contains most of datapoints, it can be concluded that the prior predictive was successful. The obtained lines include points as expected.

PPC stan code:

```
data{
    int N;
    array[N] real<lower=0> engine_size;
    array[N] real<lower=0> drag_coeficient;
    array[N] int<lower=0> cylinders;
}

generated quantities {
    real alpha;
    real beta_engine_size;
    real beta_cylinders;
    real beta_drag_coeficient;
    real  sigma;
    vector [N] fuel_consumption;

    alpha = normal_rng(9.36,2.29);
    beta_engine_size = normal_rng(0, 1);
    beta_cylinders = normal_rng(0, 1);
    beta_drag_coeficient = normal_rng(0, 1);
    sigma = normal_rng(2, 1);

        // sigma = exponential_rng(5);
    for(i in 1:N)
    {
        fuel_consumption[i] = normal_rng(alpha+beta_engine_size*engine_size[i]+beta_cylinders*cylinders[i]+beta_drag_coeficient*drag_coeficient[i],sigma);
    }

}
```

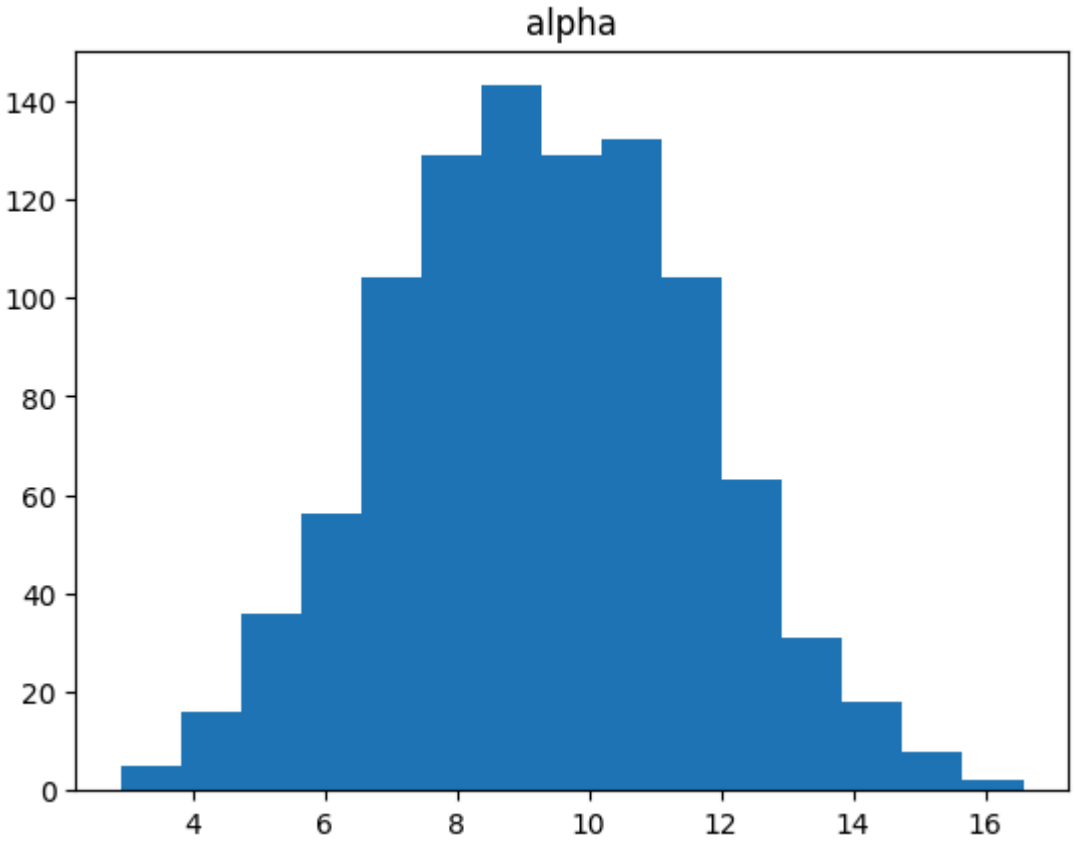
In [ ]:
```
model_2_prior=CmdStanModel(stan_file='Stan_files/model_2_prior.stan')
```

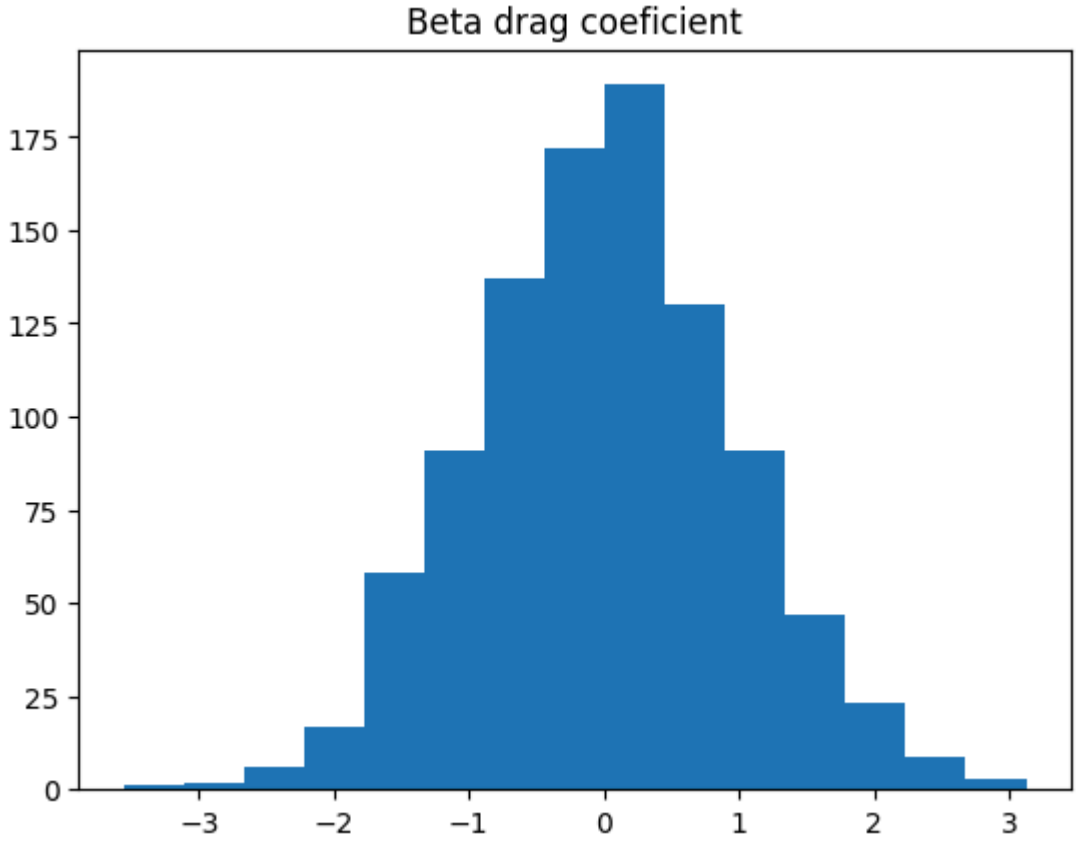
```
02:19:31 - cmdstanpy - INFO - compiling stan file C:\Users\jkwap\Desktop\Sem_I\DA
\Project\data_analitics\Stan_files\model_2_prior.stan to exe file C:\Users\jkwap
\Desktop\Sem_I\DA\Project\data_analitics\Stan_files\model_2_prior.exe
02:20:33 - cmdstanpy - INFO - compiled model executable: C:\Users\jkwap\Desktop\S
em_I\DA\Project\data_analitics\Stan_files\model_2_prior.exe
```

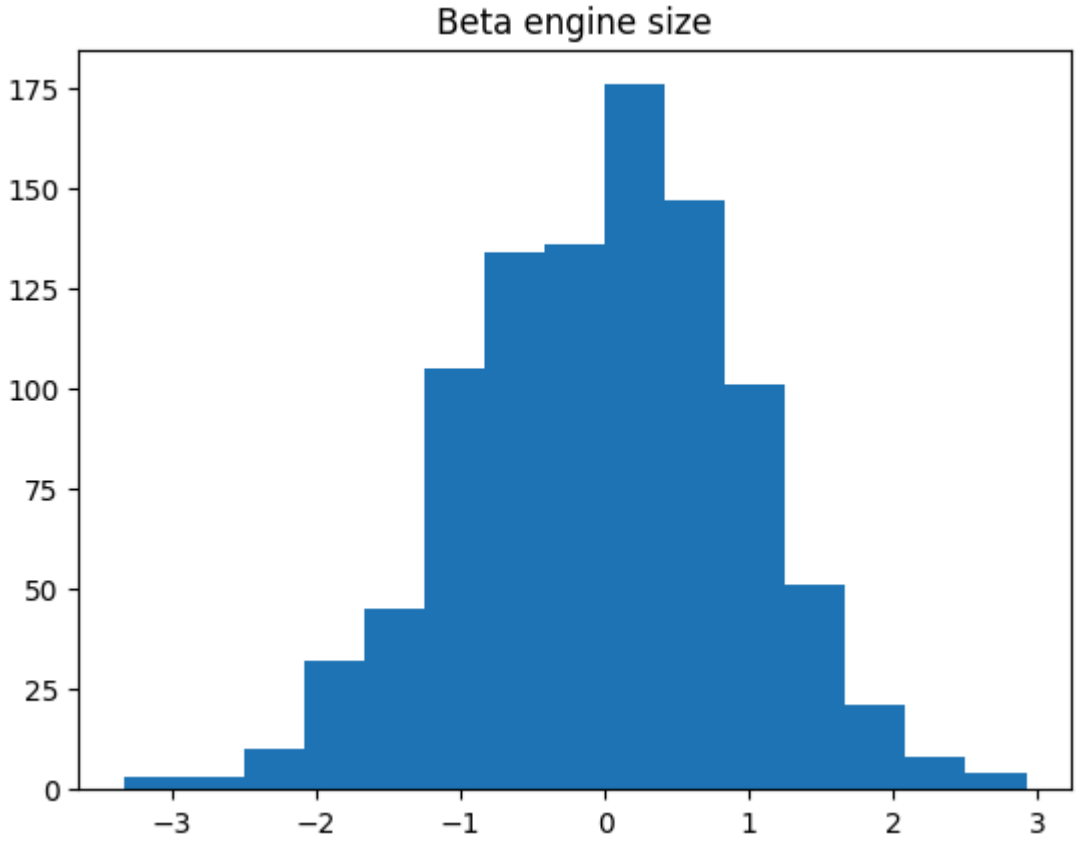
In [ ]:
```
samples_2_prior = model_2_prior.sample(data = {'N':N, 'engine_size':engine_size_
iter_sampling=X,
iter_warmup=2000,
chains=1,
seed=29042020,
refresh=N)
```

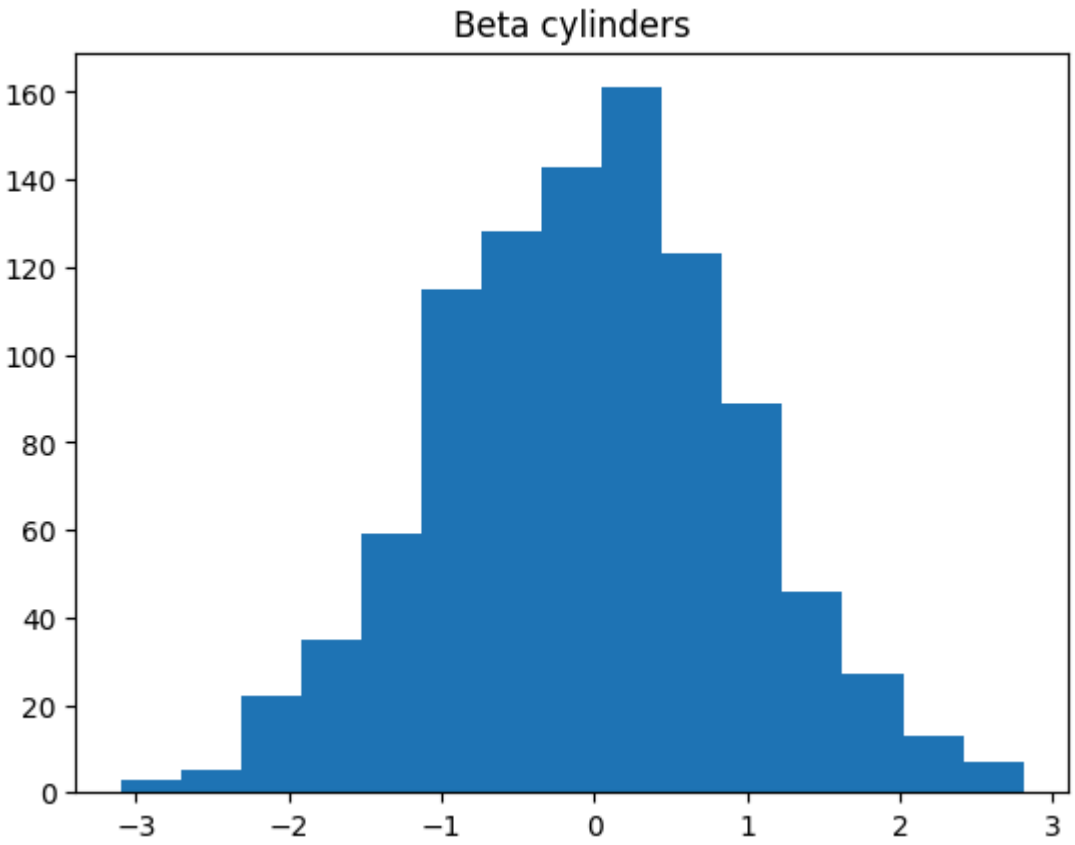
In [ ]:
```
dataFrame_model_2_prior = samples_2_prior.draws_pd()

plt.figure(0)
plt.hist(dataFrame_model_2_prior.alpha,bins=15)
plt.title('alpha')
plt.figure(1)
plt.hist(dataFrame_model_2_prior.beta_drag_coeficient,bins=15)
```

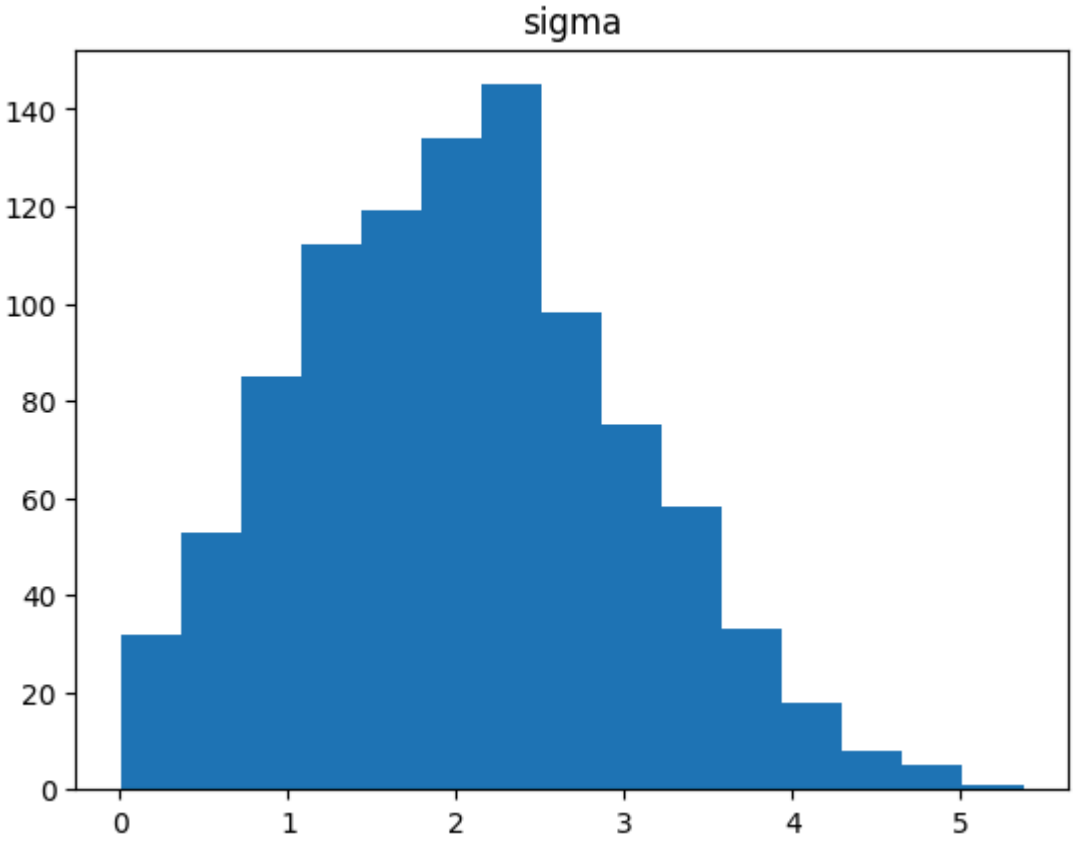
```
plt.title('Beta drag coeficient')
plt.figure(2)
plt.hist(dataFrame_model_2_prior.beta_engine_size,bins=15)
plt.title('Beta engine size')
plt.figure(3)
plt.hist(dataFrame_model_2_prior.beta_cylinders,bins=15)
plt.title('Beta cylinders')
plt.figure(4)
plt.hist(dataFrame_model_2_prior.sigma,bins=15)
plt.title('sigma')
plt.show()
```




Beta drag coeficient
175
150
125
100
75
50
25
0
−3
−2
−1
0
1
2
3


Beta engine size
175
150
125
100
75
50
25
0
−3
−2
−1
0
1
2
3



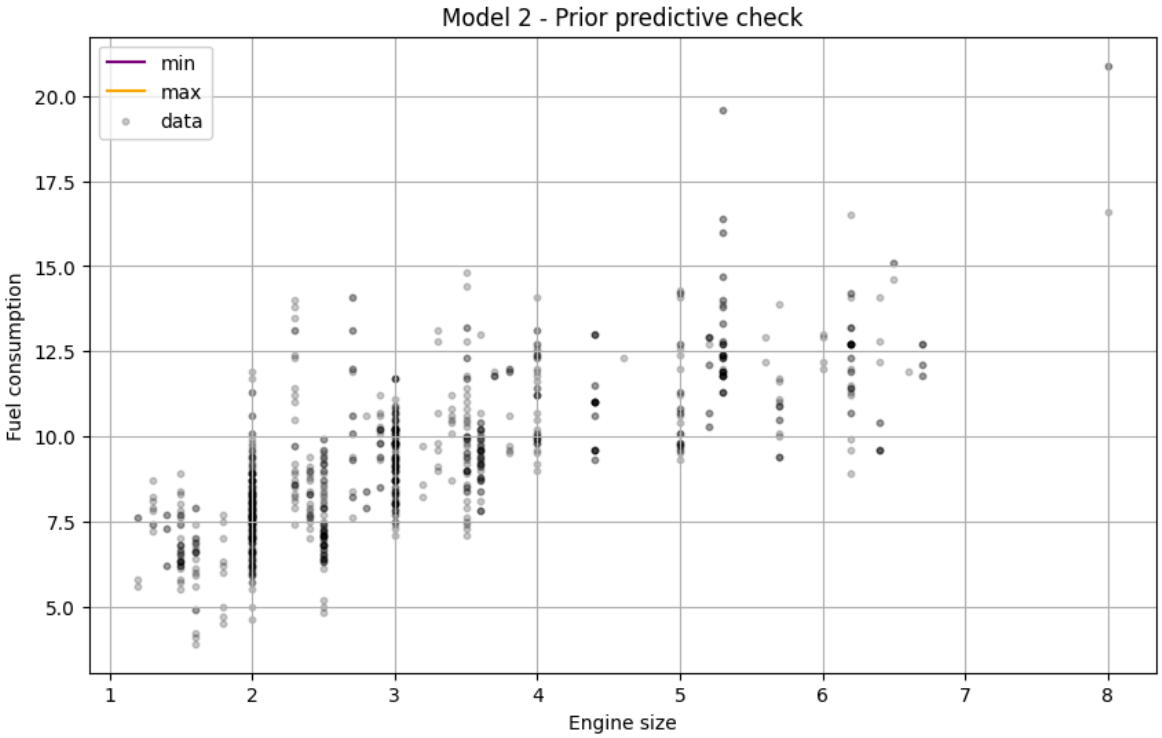


Model 2 - Comparing margin prior values with data

Return to table of contents

In [ ]:
```
plt.figure(figsize=[10, 6])

price_sim = samples_2_prior.stan_variable('fuel_consumption')

caratQuantMinDict = quantsExtremes(dataFrame, price_sim, 0)
caratMin = (caratQuantMinDict.keys())
quantMin = (caratQuantMinDict.values())

caratQuantMaxDict = quantsExtremes(dataFrame, price_sim, 1)
caratMax = (caratQuantMaxDict.keys())
quantMax = (caratQuantMaxDict.values())

plt.figure(figsize=[10, 6])
plt.plot(caratMin, quantMin, color='purple')
plt.plot(caratMax, quantMax, color='orange')
plt.scatter(dataFrame['Engine Size(L)'], dataFrame['Fuel Consumption(Hwy (L/100
plt.xlabel("Engine size")
plt.ylabel("Fuel consumption")
plt.title("Model 2 - Prior predictive check")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Based on the shape of obtained figure which contains most of datapoints, it can be concluded that the prior predictive was successful. The obtained lines include points as expected.

Model 2 - Posterior analysis

Return to table of contents

After confirming that the priors values and trajectories are correct we can start a proper analysis.

As mentioned previously sampling time was heavily dependent on number of datapoints, but, according to diagnose result, no other issues were encountered.

Posterior stan code:

```
data {
    int<lower=0> N;
    array[N] real<lower=0> engine_size;
    array[N] real<lower=0> drag_coeficient;
    array[N] int<lower=0> cylinders;
    array[N] real <lower=0> fuel_consumption;
}
parameters {
    real alpha;
    real beta_engine_size;
    real beta_cylinders;
    real beta_drag_coeficient;
    real<lower=0> sigma;
}
model {
    alpha ~ normal(9.36, 2.29);
    beta_engine_size ~ normal(0, 1);
    beta_cylinders ~ normal(0, 1);
    beta_drag_coeficient ~ normal(0, 1);
    sigma ~ normal(0, 1);
    for(i in 1:N)
    {
    fuel_consumption[i] ~ normal(alpha+ beta_engine_size * engine_size[i] +
                                 beta_cylinders * cylinders[i] +
                                 beta_drag_coeficient * drag_coeficient[i], sigma);
    }
}
generated quantities {
   array[N] real y_out;
    vector[N] log_lik;
   for(i in 1:N)
    {
          log_lik[i] = normal_lpdf(fuel_consumption[i] | alpha+beta_engine_size*engine_size[i]+beta_cylinders*cylinders[i]+beta_drag_coeficient*drag_coeficient[i], sigma);
    y_out[i] = normal_rng(alpha + beta_engine_size * engine_size[i] +
                                 beta_cylinders * cylinders[i] +
                                 beta_drag_coeficient * drag_coeficient[i], sigma);

    }
}
```

In [ ]:
```
model_2_post=CmdStanModel(stan_file='Stan_files/model_2_post.stan')
```

In [ ]:
```
samples_2_post = model_2_post.sample(data = {'N':N, 'engine_size':engine_size_da
iter_sampling=X,
iter_warmup=200,
chains=1,
seed=29042020,
refresh=N)
```

```
02:39:31 - cmdstanpy - INFO - CmdStan start processing
chain 1 |██████████| 00:36 Sampling completed
```

```
02:40:08 - cmdstanpy - INFO - CmdStan done processing.
```

Model 2 - model parameters

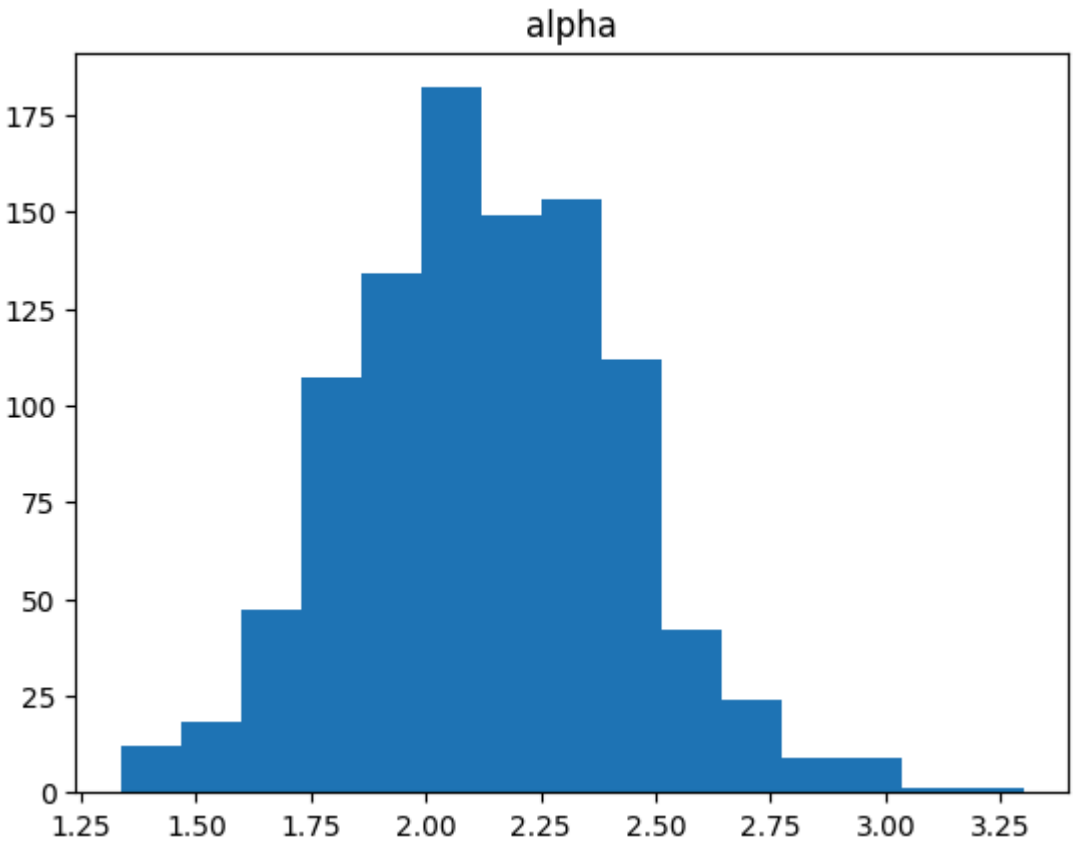
Return to table of contents

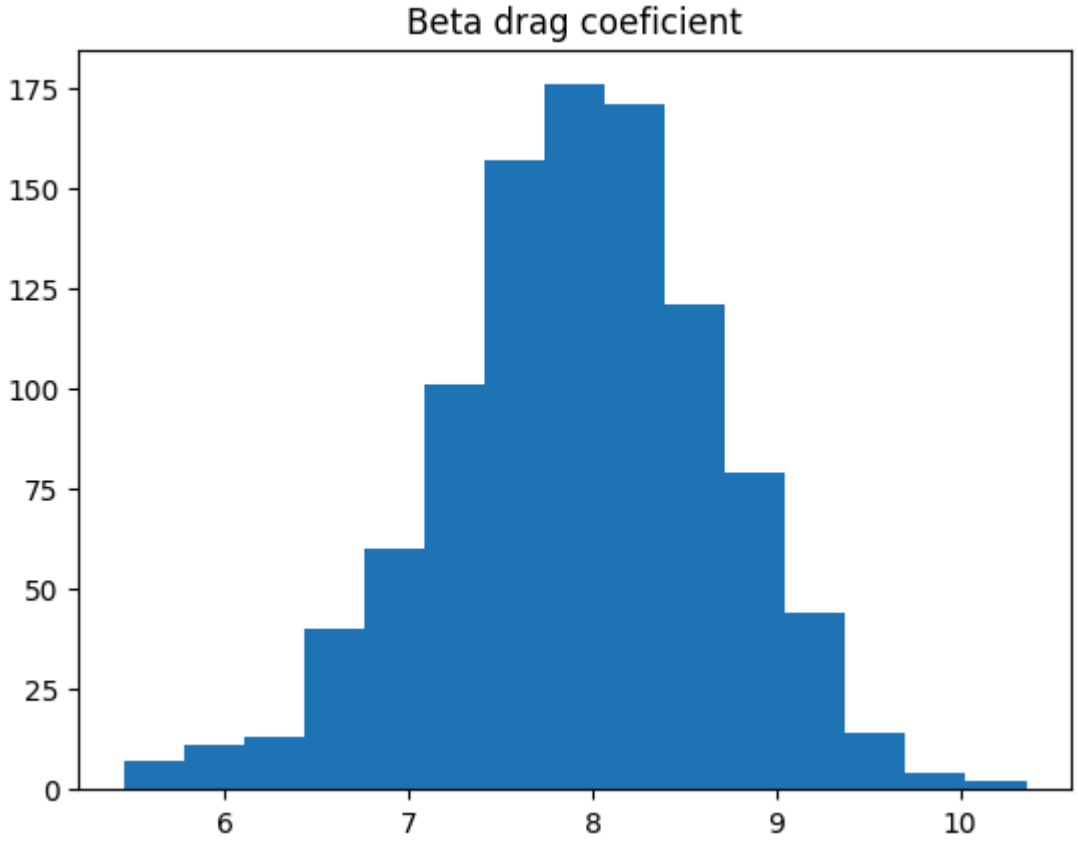
We can also extract stan variables that are used in the final fuel consumption prediction equation.

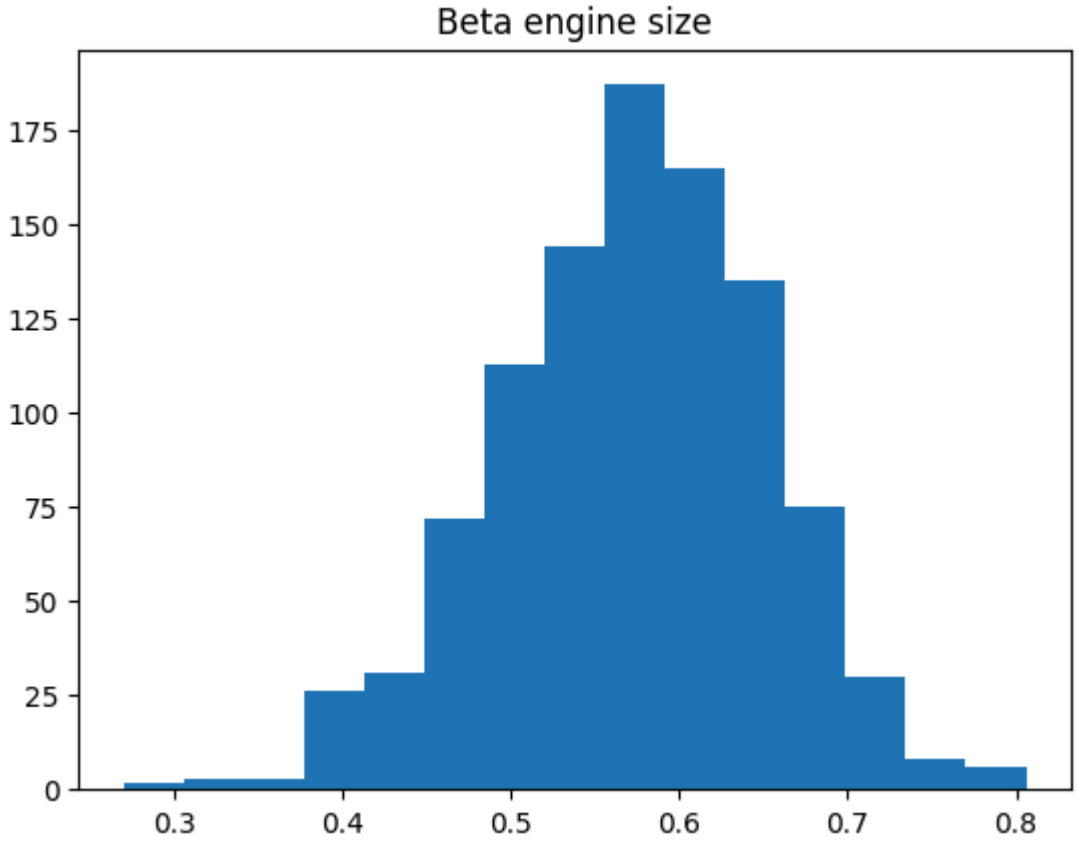
Based on the presented graphs and histograms of parameters, it can be concluded that parameter values are relatively concentrated.

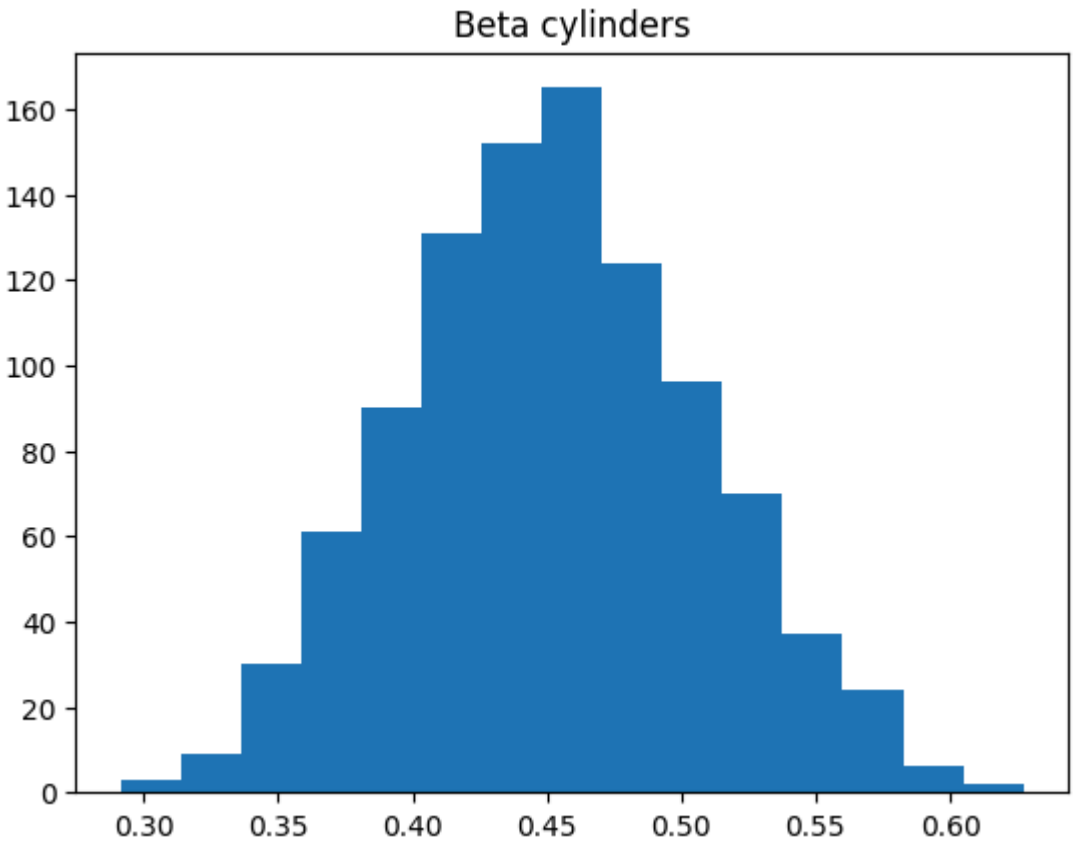
Their slight dispersion is indicative of the cars in the same class being slightly different from one another.

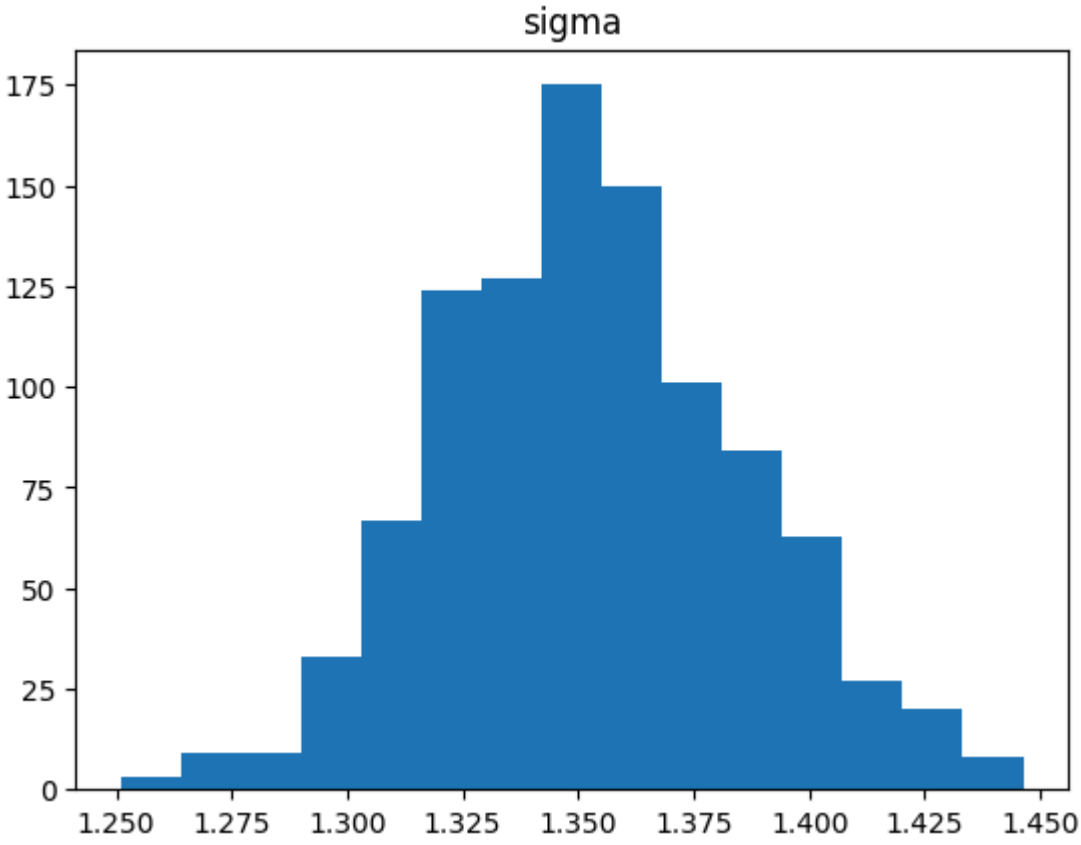
In [ ]:
```
dataFrame_model_2_post = samples_2_post.draws_pd()

plt.figure(0)
plt.hist(dataFrame_model_2_post.alpha,bins=15)
plt.title('alpha')
plt.figure(1)
plt.hist(dataFrame_model_2_post.beta_drag_coeficient,bins=15)
plt.title('Beta drag coeficient')
plt.figure(2)
plt.hist(dataFrame_model_2_post.beta_engine_size,bins=15)
plt.title('Beta engine size')
plt.figure(3)
plt.hist(dataFrame_model_2_post.beta_cylinders,bins=15)
plt.title('Beta cylinders')
plt.figure(4)
plt.hist(dataFrame_model_2_post.sigma,bins=15)
plt.title('sigma')
plt.show()
```




Beta drag coeficient
175
150
125
100
75
50
25
0
6
7
8
9
10


Beta engine size
175
150
125
100
75
50
25
0
0.3
0.4
0.5
0.6
0.7
0.8





Model 2 - evaluation

Return to table of contents

We can now observe the results.

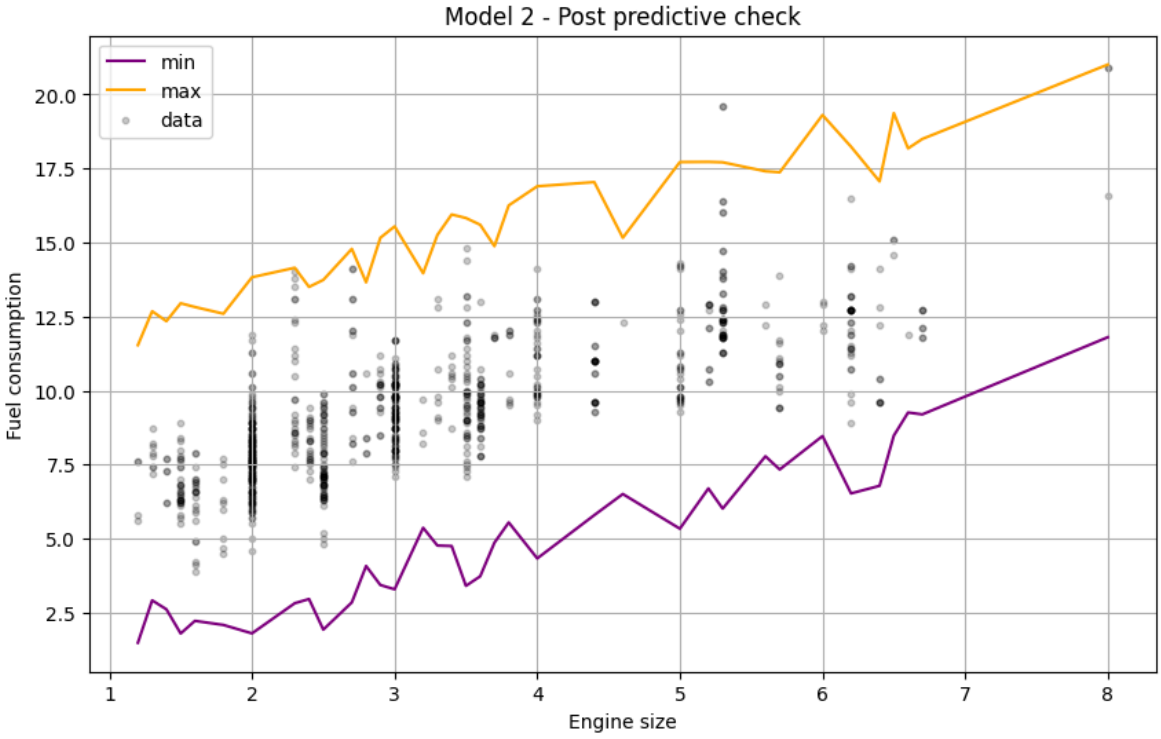
Model 2 - quantiles

Return to table of contents

After simulating we can analyze predictions. Most of the simulated values fall within real data ranges.

In [ ]:
```
plt.figure(figsize=[10, 6])

price_sim = samples_2_post.stan_variable('y_out')

caratQuantMinDict = quantsExtremes(dataFrame, price_sim, 0)
caratMin = (caratQuantMinDict.keys())
quantMin = (caratQuantMinDict.values())

caratQuantMaxDict = quantsExtremes(dataFrame, price_sim, 1)
caratMax = (caratQuantMaxDict.keys())
quantMax = (caratQuantMaxDict.values())

plt.figure(figsize=[10, 6])
plt.plot(caratMin, quantMin, color='purple')
plt.plot(caratMax, quantMax, color='orange')
plt.scatter(dataFrame['Engine Size(L)'], dataFrame['Fuel Consumption(Hwy (L/100
plt.xlabel("Engine size")
plt.ylabel("Fuel consumption")
plt.title("Model 2 - Post predictive check")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



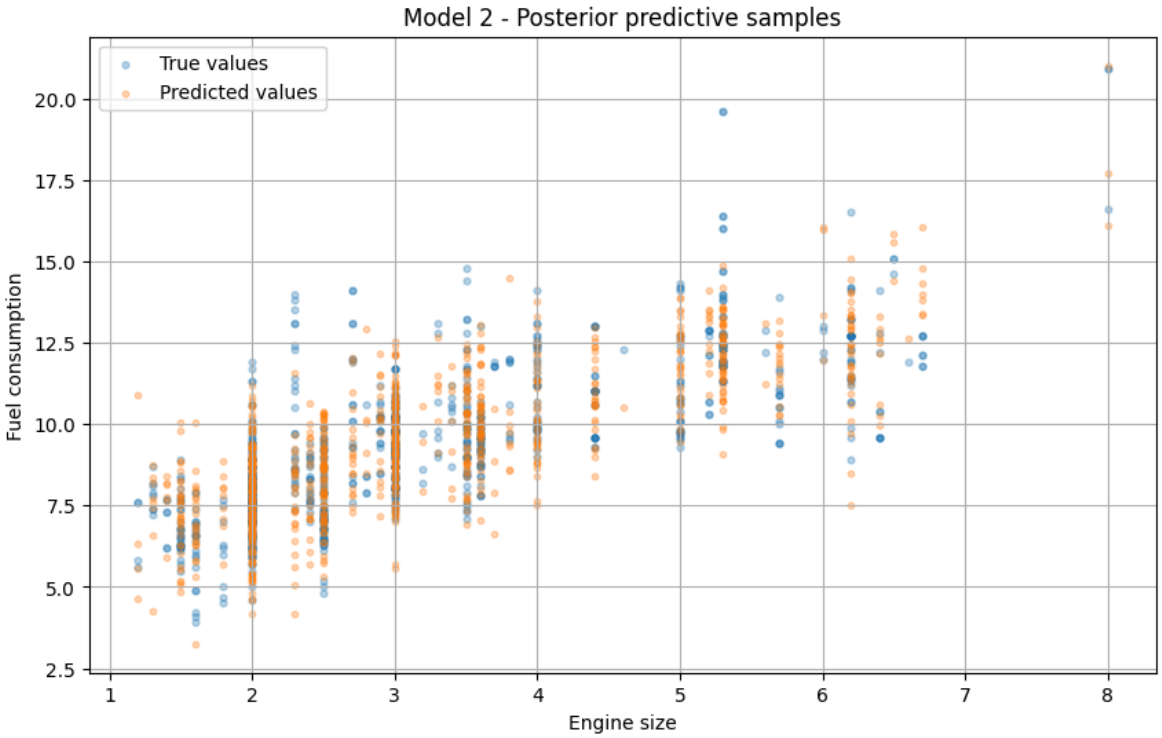
Model 2 - predictions and density plot

Return to table of contents

As we can see the model is slightly better than the first one.

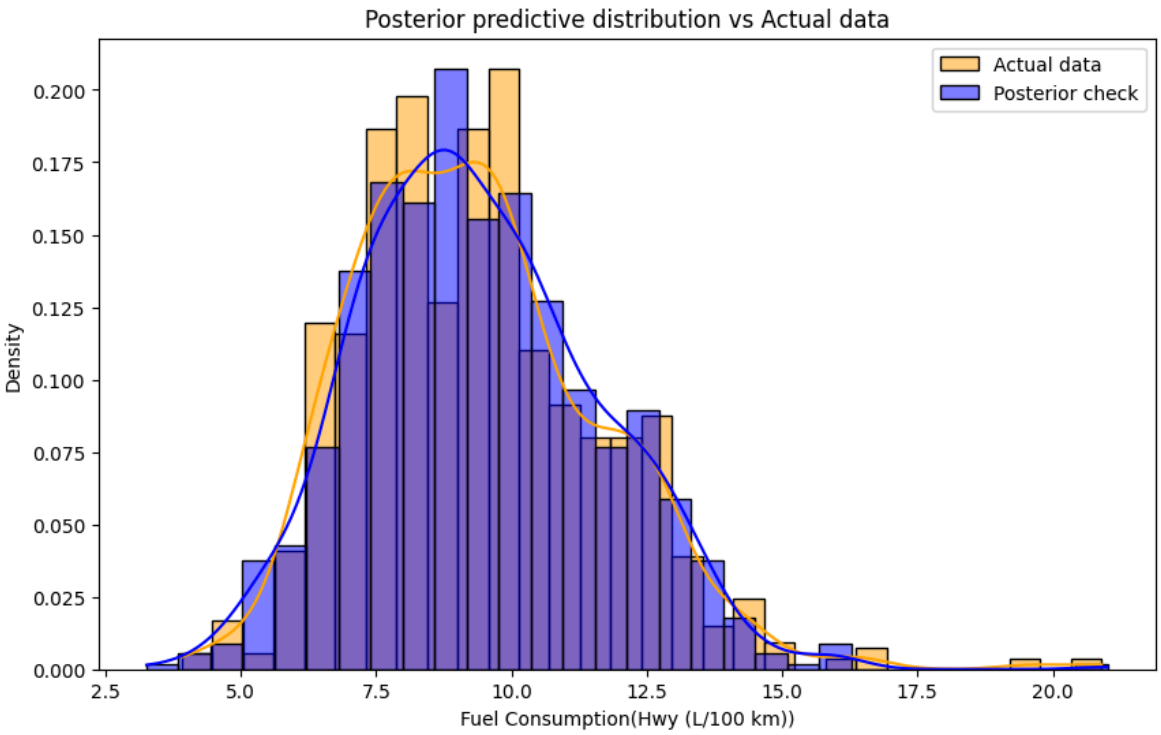
Predictions seem to be more concentrated around real values while having more true-to-dataset deviation.

In [ ]:
```
price_sim = samples_2_post.stan_variable('y_out')
plt.figure(figsize=[10,6])
plt.scatter(engine_size_data, fuel_consumption_data, alpha=0.3, s=12)
plt.scatter(engine_size_data, price_sim[0], alpha=0.3, s=10)
plt.title("Model 2 - Posterior predictive samples")
plt.legend(["True values", "Predicted values"])
plt.xlabel("Engine size")
plt.ylabel("Fuel consumption")
plt.grid()
plt.show()
```



In [ ]:
```
dataFrame_model_2_post = samples_2_post.draws_pd()

list = []
for i in range(1,N):
    val = dataFrame_model_2_post['y_out[' + str(i)+ ']'].mean()
    list.append(val)

actual_data_flat = fuel_consumption_data
plt.figure(figsize=(10,6))
sns.histplot(actual_data_flat, bins =30, kde=True, stat='density', alpha=0.5,lab
sns.histplot(list, bins =30, kde=True, stat='density', alpha=0.5,label='Posterio
plt.legend()
plt.title('Posterior predictive distribution vs Actual data')
```

Out[ ]:
```
Text(0.5, 1.0, 'Posterior predictive distribution vs Actual data')
```



Model Comparison

Return to table of contents

Leave-one-out cross-validation (LOO) and the widely applicable information criterion (WAIC) are methods for estimating pointwise out-of-sample prediction accuracy from a fitted Bayesian model using the log-likelihood evaluated at the posterior simulations of the parameter values. The comparison function used allows the models to be assessed against each of these criteria, ordering them from best to worst.

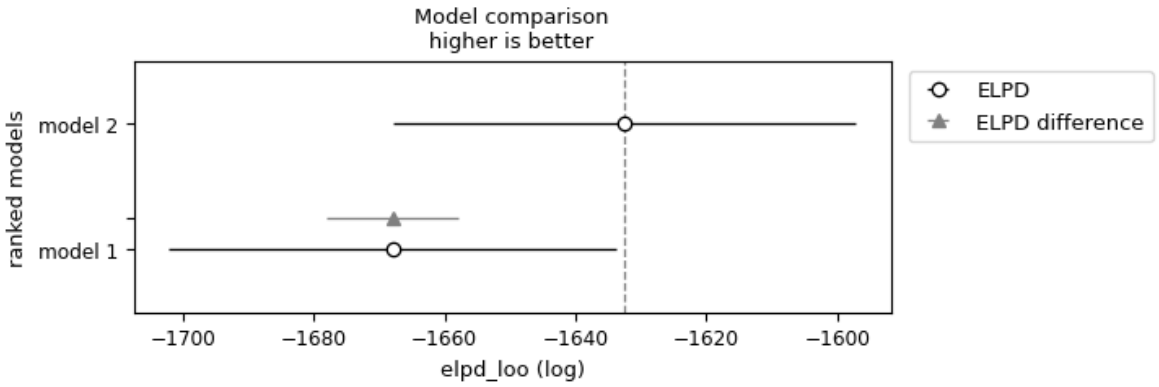
PSIS-LOO Criterion

Return to table of contents

Based on the LOO log mean values, Model 2 appears to be better in terms of predicting new data. However, the wider confidence interval for Model 2 indicates greater uncertainty compared to the more stable Model 1. The final choice of the model may depend on whether better average predictions (Model 2) or lower uncertainty and stability (Model 1) are more valued.

In [ ]:
```
compare=az.compare({'model 1':samples_1_post,'model 2':samples_2_post}, ic='loo'
az.plot_compare(compare)
```

Out[ ]:
```
<AxesSubplot: title={'center': 'Model comparison\nhigher is better'}, xlabel='e
lpd_loo (log)', ylabel='ranked models'>
```



WAIC Criterion

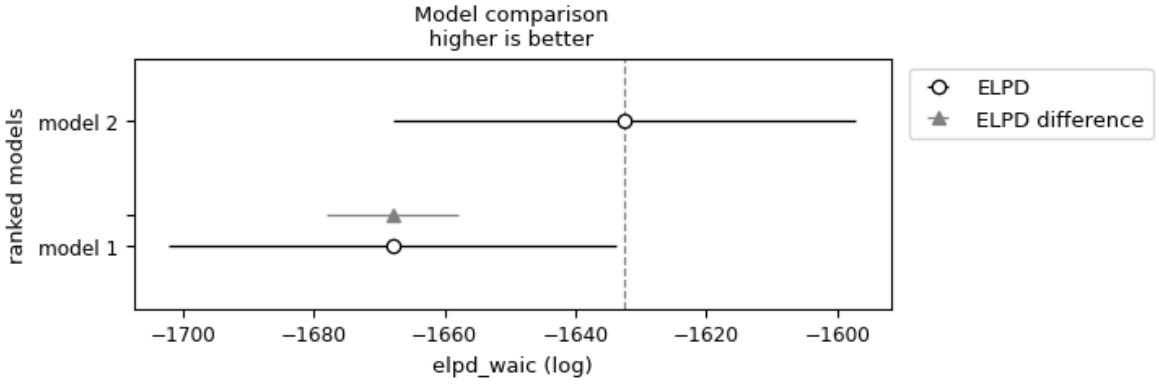
Return to table of contents

Based on the WAIC values, Model 1 appears to be better because it has a lower WAIC value. However, Model 2 has a higher weight, indicating a high probability of being the better model when considering model uncertainty. The final choice of the model may depend on whether a lower WAIC value or higher model weight is more important for the specific application.

In [ ]:
```
compare=az.compare({'model 1':samples_1_post,'model 2':samples_2_post}, ic='waic
az.plot_compare(compare)
```

```
c:\Users\jkwap\AppData\Local\Programs\Python\Python310\lib\site-packages\arviz\st
ats\stats.py:1632: UserWarning: For one or more samples the posterior variance of
the log predictive densities exceeds 0.4. This could be indication of WAIC starti
ng to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
c:\Users\jkwap\AppData\Local\Programs\Python\Python310\lib\site-packages\arviz\st
ats\stats.py:1632: UserWarning: For one or more samples the posterior variance of
the log predictive densities exceeds 0.4. This could be indication of WAIC starti
ng to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
```

Out[ ]:
```
<AxesSubplot: title={'center': 'Model comparison\nhigher is better'}, xlabel='e
lpd_waic (log)', ylabel='ranked models'>
```



Model Comparison - conclusions

Return to table of contents

Comparing the models, both visually and by criteria, we can agree that both models can be used in different applications. Differences are rather small and that may be caused by strong connections between two of predictors. Overall accuracy of predictions may be improved by increasing number of independent predictors.