

Diamond Price Analysis

Authors:

- Marcin Bereźnicki
 - Jakub Burczyk
-

The problem:

The purpose of the project was to analyze diamond's pricing based on it's weight, taking into account its other characteristics such as cut and clarity.

The goal:

We hope, that after creating sufficient model it will be possible to predict a price for gem given it's weight also taking into account it's clarity and quality of the cut.

It may be possible to estimate a price without any trade specific knowledge, which could prevent getting ripped off by sellers/buyers.

Table of contents

We highly recommend using provided hyperlinks to sections.

- [Dataset](#)
- [Python modules](#)
- [Data Tidying](#)
 - [Dropping indexes](#)
 - [Relevant data extraction](#)
 - [Plotting dataset](#)
 - [Trimming dataset](#)
- [Data Analysis](#)
 - [Loading dataset](#)
 - [Model 1 - two predictors: weight and clarity](#)
 - [Prior predictive check](#)
 - [Comparing priors with data](#)
 - [Posterior analysis](#)
 - [Model parameters](#)
 - [Evaluation](#)
 - [Quantiles](#)
 - [Predictions and density](#)
 - [Model 2 - three predictors: weight, clarity and cut quality](#)

- [Prior predictive check](#)
 - [Comparing priors with data](#)
 - [Posterior analysis](#)
 - [Model parameters](#)
 - [Evaluation](#)
 - [Quantiles](#)
 - [Predictions and density](#)
 - [Model Comparison](#)
 - [PSIS-LOO Criterion](#)
 - [WAIC Criterion](#)
 - [Conclusions](#)
-

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 53 941 records containing description of 10 diamond properties.

The columns are as follows:

- **price** - in US dollars
- **carat** - weight of the gem
- **cut** - quality of the cut
- **color** - gem's color
- **clarity** - measurement how clear the gem is and it's defects
- **x** - length in milimeters
- **y** - width in milimeters
- **z** - depth in milimeters
- **table** width of top face of the diamond relative to widest point
- **depth** - depth percentage

$$depth = \frac{z}{mean(x, y)} \quad (1)$$

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
```

Data tidying

[Return to table of contents](#)

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

Dropping index

[Return to table of contents](#)

The first column contains record id without column name, but for our purposes it is not necessary thus it gets dropped after loading the dataset file.

```
In [ ]: df = pd.read_csv("data/diamonds.csv")
df.drop(columns=["Unnamed: 0"], inplace=True)
df.head()
```

```
Out[ ]:
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Data extraction

[Return to table of contents](#)

For our analysis we will consider up to three variables affecting gem pricing - it's mass, clarity and quality of the cut.

Because the values for clarity and cut quality are presented in descriptive classification, they have to be mapped to numeric classification values for processing.

```
In [ ]: cutRemap = {'Fair': 1, 'Good': 2, 'Very Good': 3, 'Premium': 4, 'Ideal': 5}
clarityRemap = {'I1': 1, 'SI2': 2, 'SI1': 3, 'VS2': 4, 'VS1': 5, 'VVS2': 6, 'VVS1': 7}
colorRemap = {'J': 1, 'I': 2, 'H': 3, 'G': 4, 'F': 5, 'E': 6, 'D': 7}

df=df.replace({"cut": cutRemap})
df=df.replace({"clarity": clarityRemap})
df=df.replace({"color": colorRemap})

df.head()
```

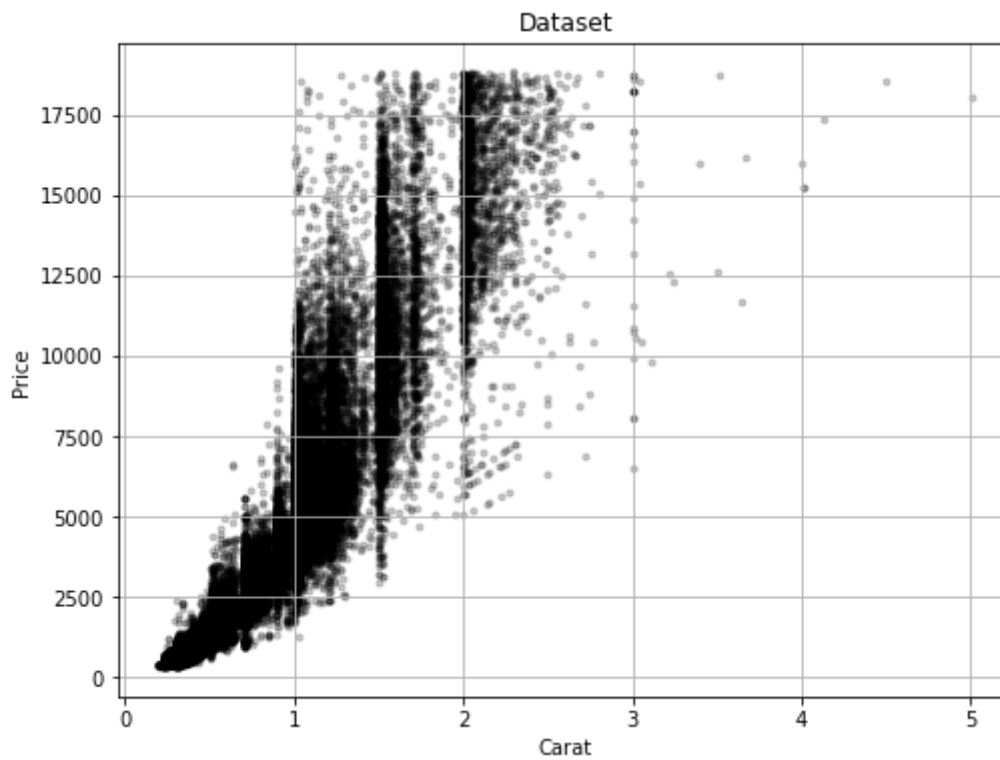
```
Out[ ]:   carat  cut  color  clarity  depth  table  price     x     y     z
0   0.23   5     6       2    61.5   55.0   326  3.95  3.98  2.43
1   0.21   4     6       3    59.8   61.0   326  3.89  3.84  2.31
2   0.23   2     6       5    56.9   65.0   327  4.05  4.07  2.31
3   0.29   4     2       4    62.4   58.0   334  4.20  4.23  2.63
4   0.31   2     1       2    63.3   58.0   335  4.34  4.35  2.75
```

Plotting dataset

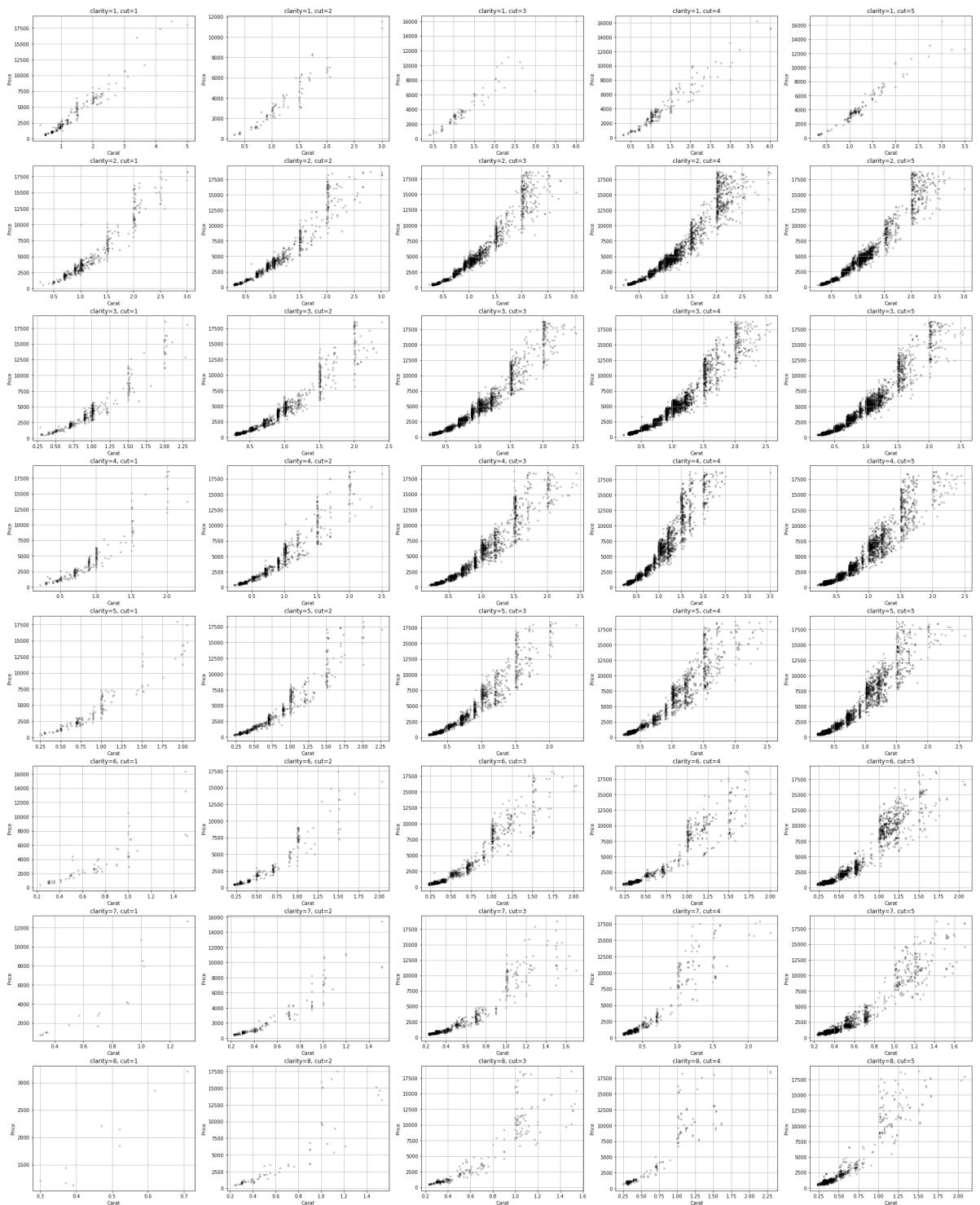
[Return to table of contents](#)

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

```
In [ ]: plt.figure(figsize=[8, 6])
plt.scatter(df.carat, df.price, color='black', alpha=0.2, s=10)
plt.title('Dataset')
plt.xlabel("Carat")
plt.ylabel("Price")
plt.grid()
plt.show()
```



```
In [ ]: fig, axs = plt.subplots(8,5)
fig.set_size_inches(35, 45)
for i in range(0,8):
    for j in range(0,5):
        df_temp = df.loc[df['clarity'] == i+1]
        df_temp = df_temp.loc[df_temp['cut'] == j+1]
        axs[i][j].scatter(df_temp.carat, df_temp.price, color='black', alpha=0.2, s=10)
        axs[i][j].grid()
        axs[i][j].set_title(f'clarity={i+1}, cut={j+1}')
        axs[i][j].set_xlabel('Carat')
        axs[i][j].set_ylabel('Price')
plt.show()
```



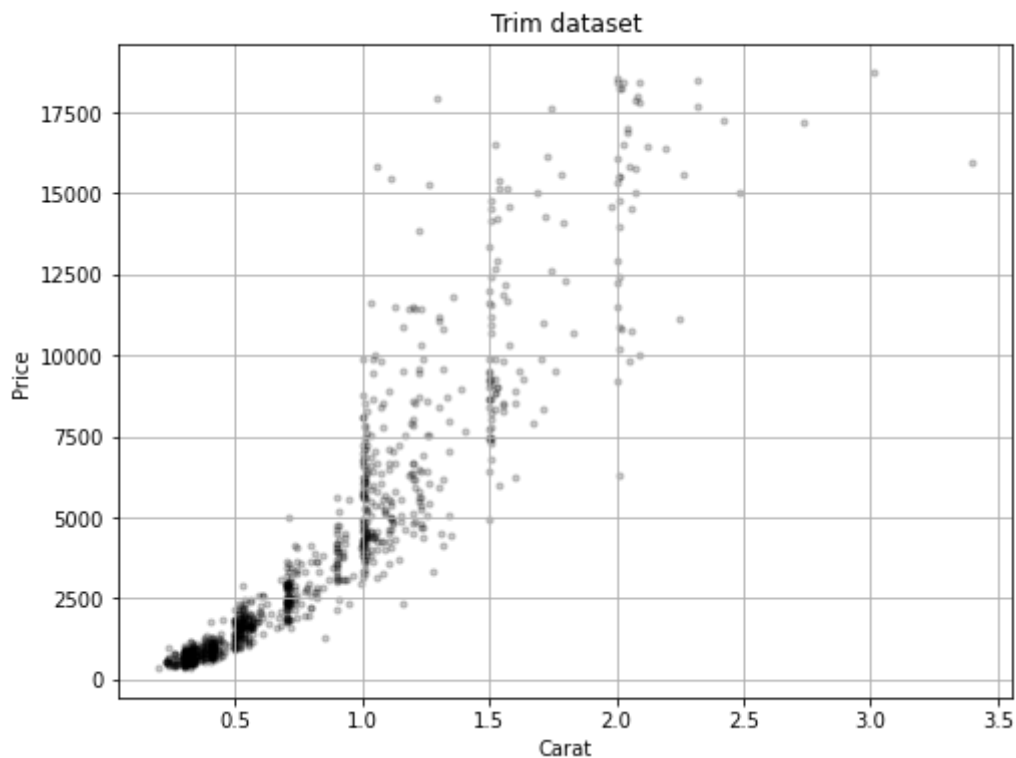
Trimming dataset

[Return to table of contents](#)

The dataset contains a lot of samples which negatively affects model sampling time exponentially. Due to that fact 1000 randoms samples are selected from the entire dataset for further analysis.

```
In [ ]: df_trim = df.sample(n = 1000)
df_trim.reset_index(drop=True, inplace=True)
```

```
plt.figure(figsize=[8, 6])
plt.scatter(df_trim.carat, df_trim.price, color='black', alpha=0.2, s=10)
plt.title('Trim dataset')
plt.xlabel("Carat")
plt.ylabel("Price")
plt.grid()
plt.show()
```



Data Analysis

[Return to table of contents](#)

For analysis we have created 2 bayesian models.

- Model 1 - uses 2 predictors: weight and clarity
- Model 2 - uses 3 predictors: weight, clarity and cut quality

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:

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

$$\mu_i = \alpha_{\text{clarity}}[\text{clarity}_i] + \beta_{\text{clarity}}[\text{clarity}_i] * \text{carat}_i$$

With parameter distributions set as follows:

$$\alpha_{\text{clarity}} \sim \text{Normal}(-1000, 10)$$

$$\beta_{\text{clarity}} \sim \text{Normal}(10000, 2000)$$

$$\sigma \sim \text{Exponential}(10)$$

The required input data is the set of diamonds with weight and clarity for which the user wants to make a prediction.

Model 1 - Prior predictive check

[Return to table of contents](#)

First step is prior predictive 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. $\text{Normal}(0,10)$) up to final values based on resultant plot. (See chapter ["Model 1 - Comparing margin prior values with data"](#))

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 Model:

```

1  data {
2    int N;
3    vector[N] carat;
4    array [N] int <lower=1, upper=8> clarity;
5  }
6
7  generated quantities {
8    vector[8] alpha_clarity;
9    vector[8] beta_clarity;
10   for (i in 1:8){
11     alpha_clarity[i] = normal_rng(-1000,10);
12     beta_clarity[i] = normal_rng(10000,2000);
13   }
14   real sigma = exponential_rng(10);
15   vector[N] price;
16   for (i in 1:N){
17     price[i] = normal_rng(alpha_clarity[clarity[i]] + beta_clarity[clarity[i]] * carat[i], sigma);
18   }
19 }

```

```
In [ ]: model_1_ppc = CmdStanModel(stan_file='stanfiles/model_1_ppc.stan')
```

INFO:cmdstanpy:found newer exe file, not recompiling


```
In [ ]: data_sim={'N':len(df_trim), 'carat': df_trim.carat, 'clarity': df_trim.clarity}
model_1_sim = model_1_ppc.sample(data=data_sim, iter_sampling=1000, iter_warmup=0,
```

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

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

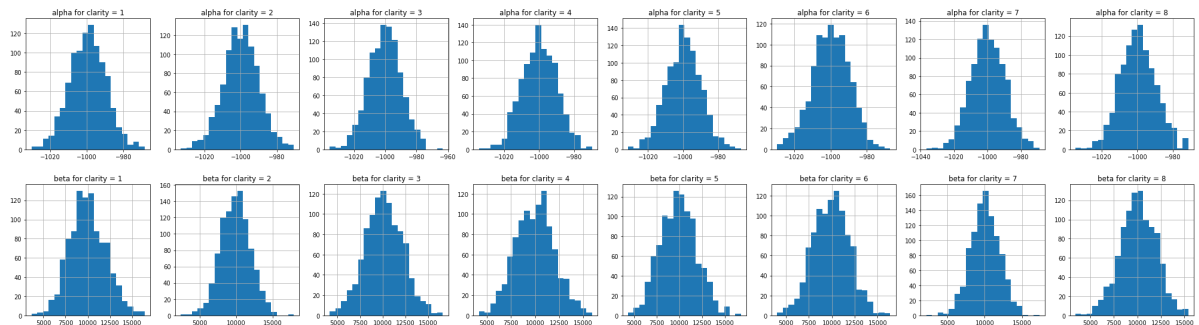
```
In [ ]: alpha_clarity_sim = pd.DataFrame(model_1_sim.stan_variable('alpha_clarity'))
beta_clarity_sim = pd.DataFrame(model_1_sim.stan_variable('beta_clarity'))
sigma_sim = model_1_sim.stan_variable('sigma')
price_sim = model_1_sim.stan_variable('price')

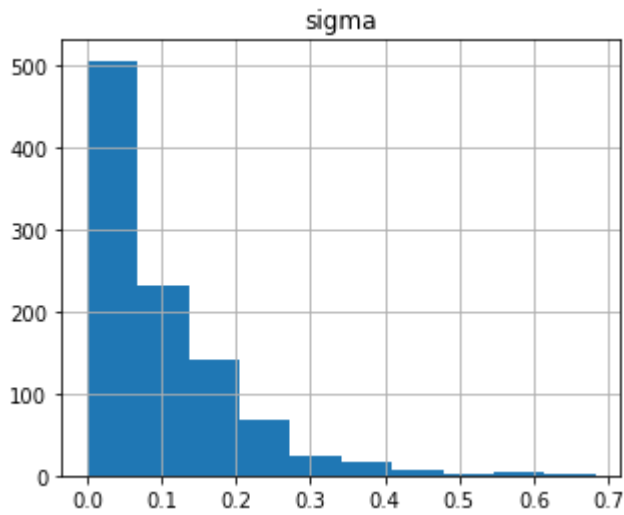
fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(alpha_clarity_sim[i], bins=20)
    axs[i].set_title(f"alpha for clarity = {i+1}")
    axs[i].grid()
plt.show()

fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(beta_clarity_sim[i], bins=20)
    axs[i].set_title(f"beta for clarity = {i+1}")
    axs[i].grid()
plt.show()

plt.figure(figsize=[5, 4])
plt.hist(sigma_sim)
plt.title('sigma')
plt.grid()
plt.show()

az.summary(model_1_sim, var_names=['alpha_clarity', 'beta_clarity', 'sigma'], round
```





Out[]:

	mean	sd	hdi_3%	hdi_97%
alpha_clarity[0]	-999.38	10.00	-1017.91	-980.58
alpha_clarity[1]	-999.87	9.83	-1018.40	-981.87
alpha_clarity[2]	-1000.26	10.41	-1020.43	-981.08
alpha_clarity[3]	-999.88	9.84	-1018.00	-982.33
alpha_clarity[4]	-999.79	9.81	-1019.38	-983.02
alpha_clarity[5]	-999.83	10.24	-1018.90	-980.53
alpha_clarity[6]	-999.73	10.21	-1019.24	-981.66
alpha_clarity[7]	-999.60	10.13	-1017.34	-979.50
beta_clarity[0]	10011.05	2031.60	6678.41	14201.80
beta_clarity[1]	9885.19	2018.80	6403.72	13764.90
beta_clarity[2]	10115.71	2059.66	6130.85	13808.60
beta_clarity[3]	9979.41	2029.27	5867.11	13429.00
beta_clarity[4]	9904.16	1994.63	6409.04	13660.40
beta_clarity[5]	9912.17	1965.22	6494.86	13811.70
beta_clarity[6]	10051.13	1954.61	6215.84	13514.80
beta_clarity[7]	10093.43	1896.18	6300.01	13397.50
sigma	0.10	0.10	0.00	0.27

```
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):
    carat_uq = df.carat.unique()
    carat_uq = sorted(carat_uq)
    quansList = calcQuants(df.carat, y)
    caratQuantDict = dict()
    for carat_val in carat_uq:
```

```

caratList = np.array(df.carat.tolist())
idxs = np.where(caratList == carat_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]
if q == 0:
    caratQuantDict[carat_val] = qval
elif q == 1:
    caratQuantDict[carat_val] = qval
return caratQuantDict

```

Model 1 - Comparing margin prior values with data

[Return to table of contents](#)

```

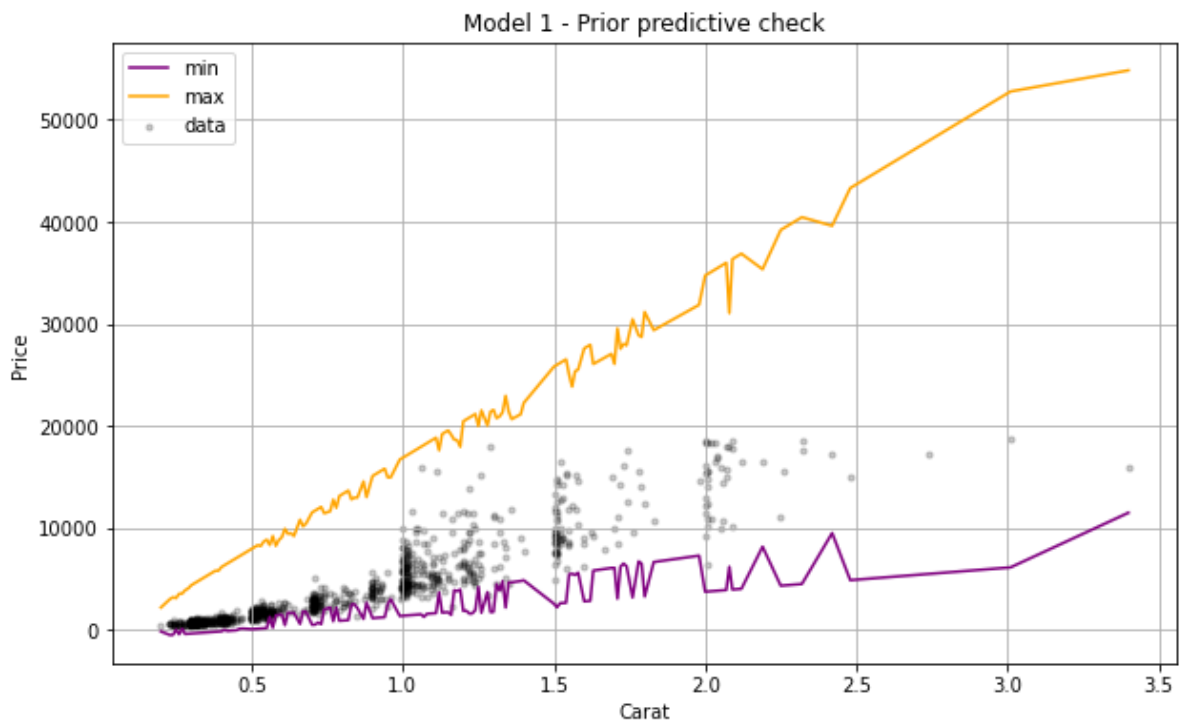
In [ ]: plt.figure(figsize=[10, 6])

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

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

plt.plot(caratMin, quantMin, color = 'purple')
plt.plot(caratMax, quantMax, color = 'orange')
plt.scatter(df_trim.carat, df_trim.price, color='black', alpha=0.2, s=10)
plt.xlabel("Carat")
plt.ylabel("Price")
plt.title("Model 1 - Prior predictive check")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()

```



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.

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.

First model stan code:

```

1  data {
2    int N;
3    vector[N] carat;
4    array [N] int <lower=1, upper=8> clarity;
5    vector[N] price;
6  }
7
8  parameters {
9    vector[8] alpha_clarity;
10   vector[8] beta_clarity;
11   real <lower=0> sigma;
12 }
13
14 transformed parameters {
15   array [N] real mu;
16   for (i in 1:N){
17     mu[i] = alpha_clarity[clarity[i]] + beta_clarity[clarity[i]] * carat[i];
18   }
19 }
20
21 model {
22   alpha_clarity ~ normal(-1000,10);
23   beta_clarity ~ normal(10000,2000);
24   sigma ~ exponential(10);
25
26   for (i in 1:N){
27     price[i] ~ normal(mu[i], sigma);
28   }
29 }
30
31 generated quantities {
32   vector[N] price_sim;
33   vector[N] log_lik;
34   for (i in 1:N){
35     price_sim[i] = normal_rng(mu[i], sigma);
36     log_lik[i] = normal_lpdf(price[i] | mu[i], sigma);
37   }
38 }

```

```
In [ ]: model_1 = CmdStanModel(stan_file='stanfiles/model_1.stan')
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
```

```
In [ ]: data_sim={'N':len(df_trim), 'carat': df_trim.carat, 'clarity': df_trim.clarity, 'price': df_trim.price}
model_1_fit = model_1.sample(data=data_sim)
```

```
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.
```

```
In [ ]: model_1_fit.diagnose()
```

```
Out[ ]: 'Processing csv files: C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt\\model_1-20220622095813_1.csv, C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt\\model_1-20220622095813_2.csv, C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt\\model_1-20220622095813_3.csv, C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt\\model_1-20220622095813_4.csv\\n\\nChecking sampler transitions treedepth.\\nTreedepth satisfactory for all transitions.\\n\\nChecking sampler transitions for divergences.\\nNo divergent transitions found.\\n\\nChecking E-BFMI - sampler transitions HMC potential energy.\\nE-BFMI satisfactory.\\n\\nEffective sample size satisfactory.\\n\\nSplit R-hat values satisfactory all parameters.\\n\\nProcessing complete, no problems detected.\\n'
```

Model 1 - model parameters

[Return to table of contents](#)

We can also extract stan variables that are used in the final price 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 diamonds in same class being slightly different from one another. The differences between distributions for each clarity class indicates that each class is modeled differently.

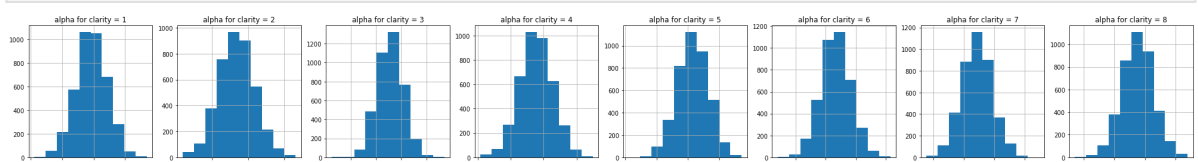
```
In [ ]: alpha_clarity_fit = pd.DataFrame(model_1_fit.stan_variable('alpha_clarity'))
beta_clarity_fit = pd.DataFrame(model_1_fit.stan_variable('beta_clarity'))
sigma_fit = model_1_fit.stan_variable('sigma')

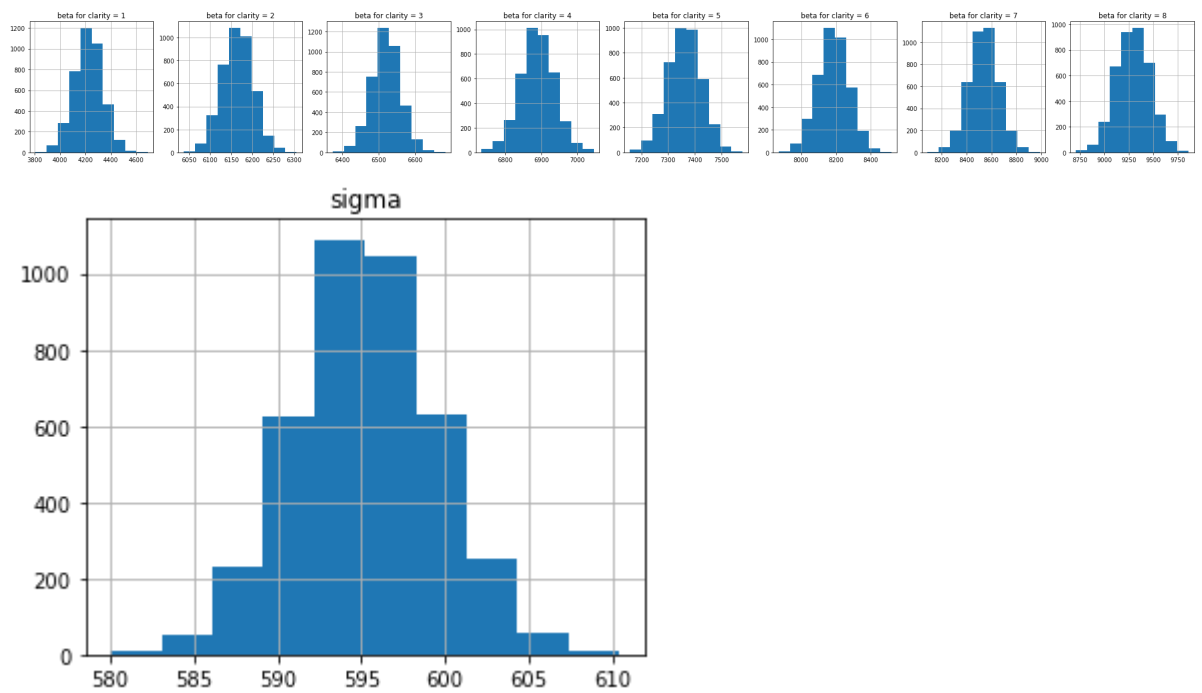
fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(alpha_clarity_fit[i])
    axs[i].set_title(f"alpha for clarity = {i+1}")
    axs[i].grid()
plt.show()

fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(beta_clarity_fit[i])
    axs[i].set_title(f"beta for clarity = {i+1}")
    axs[i].grid()
plt.show()

plt.figure(figsize=[5, 4])
plt.hist(sigma_fit)
plt.title('sigma')
plt.grid()
plt.show()

az.summary(model_1_fit, var_names=['alpha_clarity', 'beta_clarity', 'sigma'], round
```





Out[]:

	mean	sd	hdi_3%	hdi_97%
alpha_clarity[0]	-1001.27	10.00	-1020.58	-983.03
alpha_clarity[1]	-1026.43	9.89	-1043.94	-1006.60
alpha_clarity[2]	-1028.49	10.18	-1046.95	-1009.30
alpha_clarity[3]	-1024.65	10.14	-1043.95	-1005.72
alpha_clarity[4]	-1016.85	9.78	-1035.13	-998.80
alpha_clarity[5]	-1013.17	10.01	-1031.98	-994.52
alpha_clarity[6]	-1011.46	9.96	-1030.83	-993.01
alpha_clarity[7]	-1004.73	10.34	-1022.88	-983.75
beta_clarity[0]	4223.81	113.74	4007.73	4432.05
beta_clarity[1]	6164.97	37.03	6096.62	6235.63
beta_clarity[2]	6520.36	39.63	6445.84	6595.05
beta_clarity[3]	6888.84	47.96	6798.19	6981.89
beta_clarity[4]	7361.18	62.37	7244.50	7476.14
beta_clarity[5]	8185.09	89.93	8022.01	8357.88
beta_clarity[6]	8536.79	119.23	8312.73	8753.54
beta_clarity[7]	9293.83	172.89	8977.69	9614.81
sigma	595.22	4.25	586.70	602.64

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 diamonds fall within real data ranges.

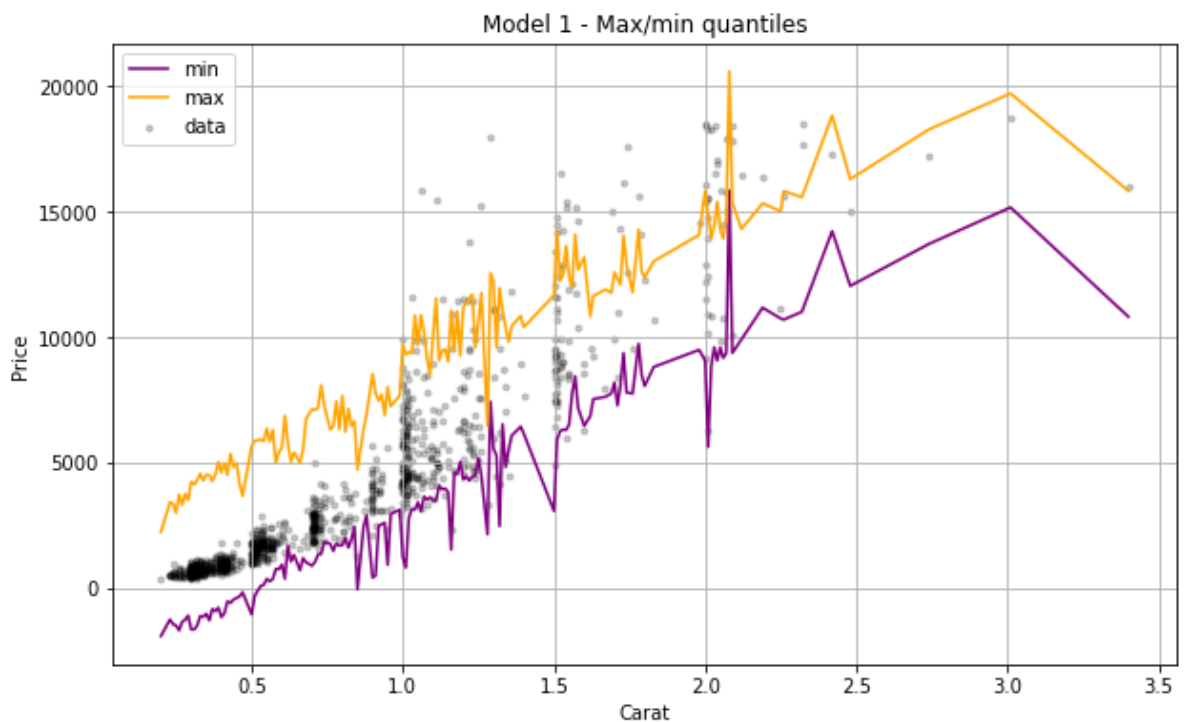
```
In [ ]: data = model_1_fit.draws_pd()
price_sims = data[data.columns[len(df_trim)+18:len(df_trim)+1018]]
#print(price_sims)

In [ ]: price_sim = model_1_fit.stan_variable('price_sim')
plt.figure(figsize=[10, 6])

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

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

plt.plot(caratMin, quantMin, color = 'purple')
plt.plot(caratMax, quantMax, color = 'orange')
plt.scatter(df_trim.carat, df_trim.price, color='black', alpha=0.2, s=10)
plt.xlabel("Carat")
plt.ylabel("Price")
plt.title("Model 1 - Max/min quantiles")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()
```



Model 1 - predictions and density plot

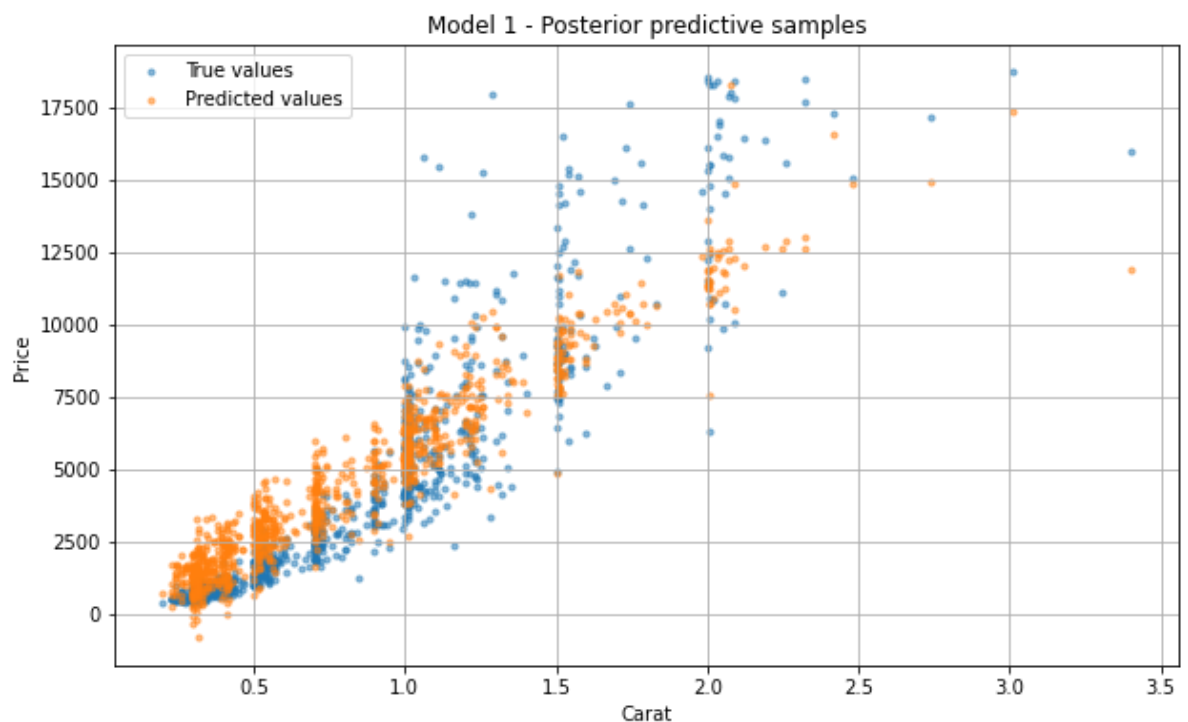
[Return to table of contents](#)

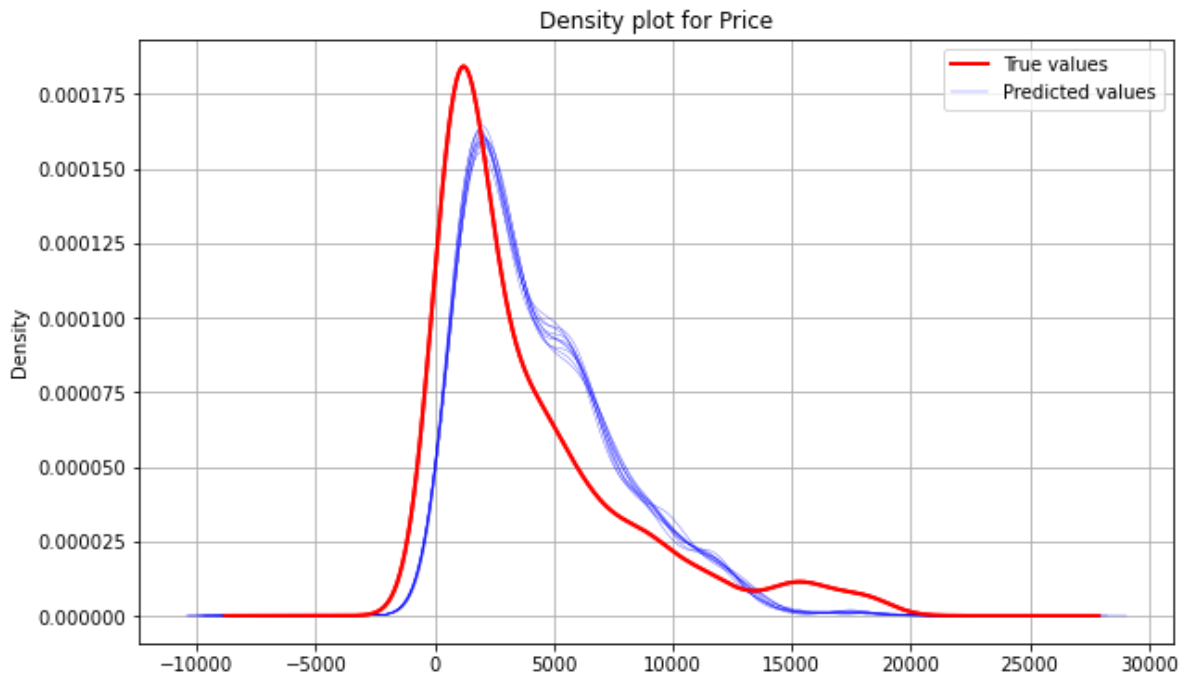
As seen on the plots the model is somewhat sufficient. It describes most accurately diamonds of weights between 0.75 to 1.75 carats. It overestimates the deviation of low mass diamonds and underestimates for high weight ones.

Perhaps adding another predictor could solve the issue.

```
In [ ]: price_sim = model_1_fit.stan_variable('price_sim')
plt.figure(figsize=[10,6])
plt.scatter(df_trim.carat, df_trim.price, alpha=0.5, s=10)
plt.scatter(df_trim.carat, price_sim[0], alpha=0.5, s=10)
plt.title("Model 1 - Posterior predictive samples")
plt.legend(["True values", "Predicted values"])
plt.xlabel("Carat")
plt.ylabel("Price")
plt.grid()
plt.show()

df_trim.price.plot.density(figsize=(10,6), linewidth=2, color='red')
for i in range(0,10):
    price_sims.iloc[i].plot.density(linewidth=0.25, color='blue')
df_trim.price.plot.density(figsize=(10,6), linewidth=2, color='red')
plt.title('Density plot for Price')
plt.legend(["True values", "Predicted values"])
plt.grid()
plt.show()
```





Model 2 - three predictors

[Return to table of contents](#)

Model has form:

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

$$\mu_i = \alpha_{cut}[cut_i] + \alpha_{clarity}[clarity_i] + (\beta_{cut}[cut_i] + \beta_{clarity}[clarity_i]) * carat_i$$

With parameter distributions set as follows:

$$\alpha_{clarity} \sim \text{Normal}(-1000, 10)$$

$$\beta_{clarity} \sim \text{Normal}(10000, 2000)$$

$$\alpha_{cut} \sim \text{Normal}(-1000, 10)$$

$$\beta_{cut} \sim \text{Normal}(500, 100)$$

$$\sigma \sim \text{Exponential}(10)$$

The required input data is the set of diamonds with weight, clarity and cut quality for which the user wants to make a prediction.

Model 2 - Prior predictive check

[Return to table of contents](#)

First step is prior predictive 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. (See chapter ["Model 2 - Comparing margin prior values with data"](#))

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 Model:

```
1 data {
2   int N;
3   vector[N] carat;
4   array [N] int <lower=1, upper=5> cut;
5   array [N] int <lower=1, upper=8> clarity;
6 }
7
8 generated quantities {
9   vector[5] alpha_cut;
10  vector[5] beta_cut;
11  for (i in 1:5){
12    alpha_cut[i] = normal_rng(-1000,10);
13    beta_cut[i] = normal_rng(500,100);
14  }
15
16  vector[8] alpha_clarity;
17  vector[8] beta_clarity;
18  for (i in 1:8){
19    alpha_clarity[i] = normal_rng(-1000,10);
20    beta_clarity[i] = normal_rng(10000,2000);
21  }
22
23  real sigma = exponential_rng(10);
24
25  vector[N] price;
26  for (i in 1:N){
27    price[i] = normal_rng(alpha_cut[cut[i]] + alpha_clarity[clarity[i]] + (beta_cut[cut[i]] + beta_clarity[clarity[i]]) * carat[i], sigma);
28  }
29 }
```

```
In [ ]: model_2_ppc = CmdStanModel(stan_file='stanfiles/model_2_ppc.stan')
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
```

```
In [ ]: data_sim={'N':len(df_trim), 'carat': df_trim.carat, 'cut': df_trim.cut, 'clarity':
model_2_sim = model_2_ppc.sample(data=data_sim, iter_sampling=1000, iter_warmup=0,
```

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

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

```
In [ ]: alpha_cut_sim = pd.DataFrame(model_2_sim.stan_variable('alpha_cut'))
alpha_clarity_sim = pd.DataFrame(model_2_sim.stan_variable('alpha_clarity'))
beta_cut_sim = pd.DataFrame(model_2_sim.stan_variable('beta_cut'))
beta_clarity_sim = pd.DataFrame(model_2_sim.stan_variable('beta_clarity'))
sigma_sim = model_2_sim.stan_variable('sigma')
price_sim = model_2_sim.stan_variable('price')

fig, axs = plt.subplots(1,5)
fig.set_size_inches(25, 4)
for i in range(0,5):
    axs[i].hist(alpha_cut_sim[i])
    axs[i].set_title(f"alpha for cut = {i+1}")
    axs[i].grid()
plt.show()
```

```

fig, axs = plt.subplots(1,5)
fig.set_size_inches(25, 4)
for i in range(0,5):
    axs[i].hist(beta_cut_sim[i])
    axs[i].set_title(f"beta for cut = {i+1}")
    axs[i].grid()
plt.show()

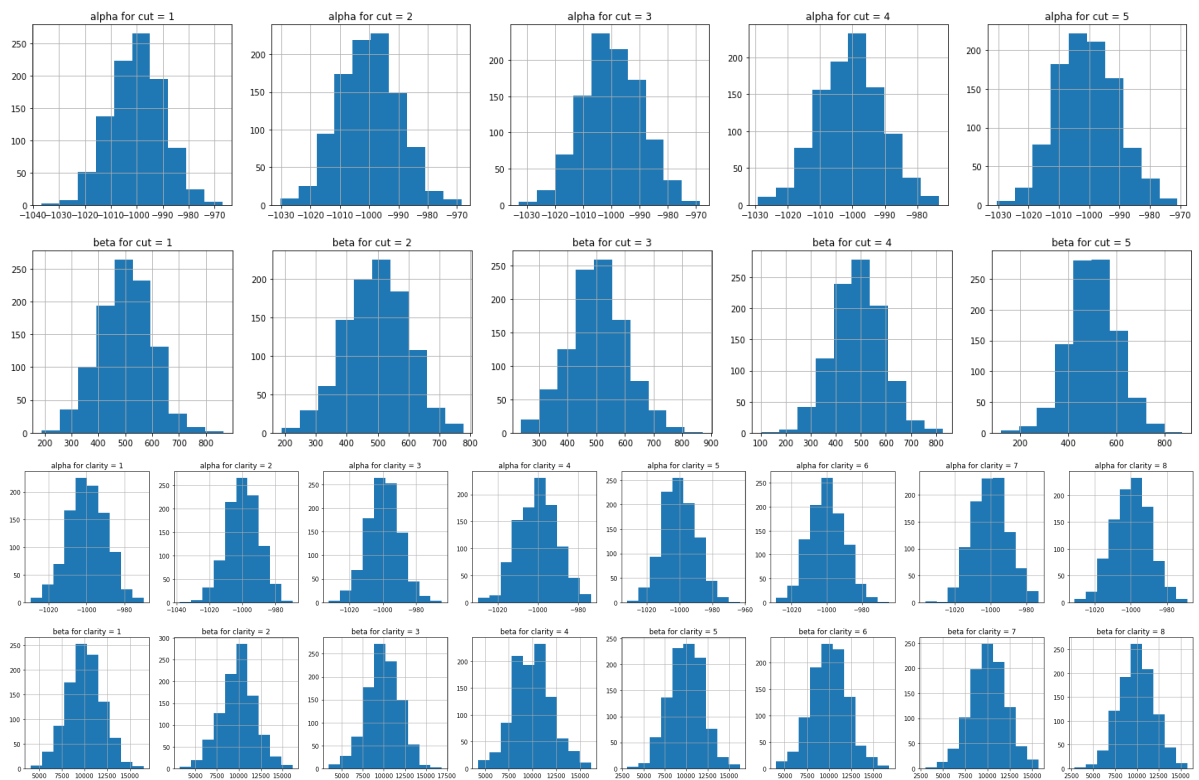
fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(alpha_clarity_sim[i])
    axs[i].set_title(f"alpha for clarity = {i+1}")
    axs[i].grid()
plt.show()

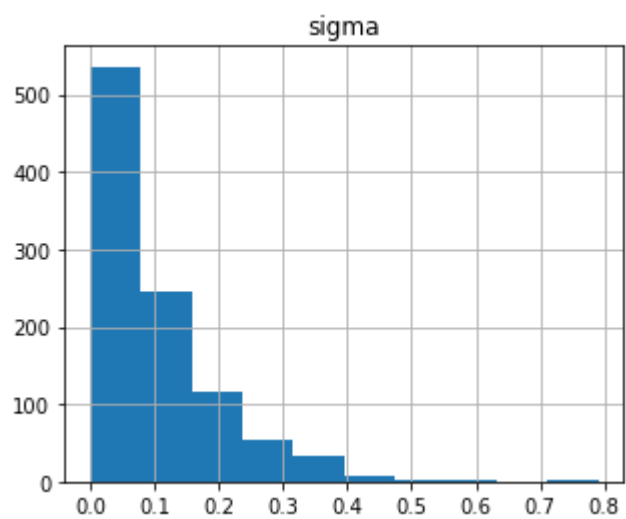
fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(beta_clarity_sim[i])
    axs[i].set_title(f"beta for clarity = {i+1}")
    axs[i].grid()
plt.show()

plt.figure(figsize=[5, 4])
plt.hist(sigma_sim)
plt.title('sigma')
plt.grid()
plt.show()

az.summary(model_2_sim, var_names=['alpha_cut', 'beta_cut', 'alpha_clarity', 'beta_

```





```
Out[ ]:
```

	mean	sd	hdi_3%	hdi_97%
alpha_cut[0]	-999.93	10.15	-1018.83	-981.37
alpha_cut[1]	-1000.15	10.11	-1016.77	-980.53
alpha_cut[2]	-999.82	10.31	-1018.27	-979.77
alpha_cut[3]	-1000.62	9.66	-1017.00	-981.90
alpha_cut[4]	-1000.67	9.84	-1017.88	-981.41
beta_cut[0]	501.41	98.19	301.54	659.84
beta_cut[1]	495.03	100.69	307.68	688.09
beta_cut[2]	503.49	97.83	314.95	692.28
beta_cut[3]	488.05	101.48	300.60	666.09
beta_cut[4]	502.83	100.88	325.59	713.06
alpha_clarity[0]	-999.79	9.96	-1019.51	-982.27
alpha_clarity[1]	-999.96	9.64	-1017.77	-981.90
alpha_clarity[2]	-999.41	9.80	-1017.83	-980.40
alpha_clarity[3]	-1000.24	9.90	-1018.66	-982.65
alpha_clarity[4]	-1000.09	10.28	-1019.45	-981.82
alpha_clarity[5]	-1000.03	9.93	-1017.89	-981.63
alpha_clarity[6]	-999.96	9.91	-1017.72	-981.34
alpha_clarity[7]	-999.58	10.18	-1016.99	-979.84
beta_clarity[0]	10014.42	1933.71	6415.00	13693.90
beta_clarity[1]	9946.57	2002.38	6007.45	13595.00
beta_clarity[2]	9890.90	1993.80	6086.85	13567.20
beta_clarity[3]	9969.74	2049.59	6481.63	14352.40
beta_clarity[4]	9957.57	1971.66	6017.95	13284.40
beta_clarity[5]	10020.39	2047.86	6086.38	13623.30
beta_clarity[6]	10101.14	2019.57	6588.03	14390.10
beta_clarity[7]	9961.38	1996.85	6029.36	13483.20
sigma	0.10	0.10	0.00	0.29

Model 2 - Comparing margin prior values with data

[Return to table of contents](#)

```
In [ ]: price_sim = model_2_sim.stan_variable('price')
plt.figure(figsize=[10, 6])

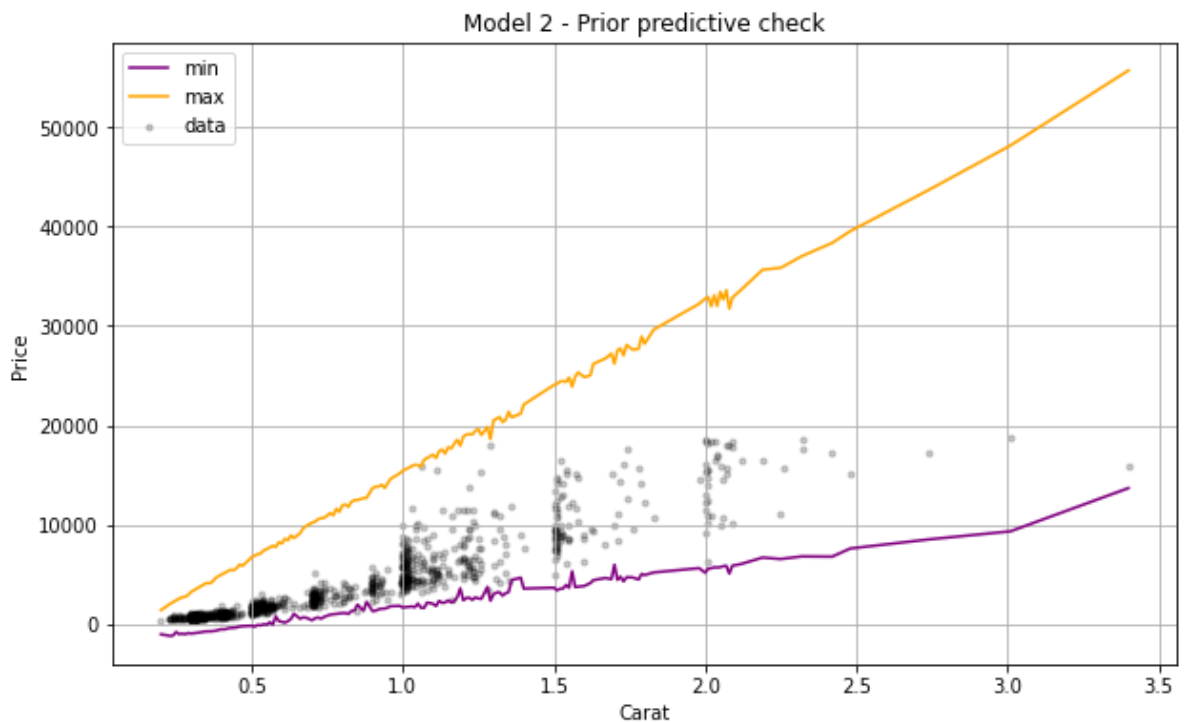
caratQuantMinDict = quantsExtremes(df_trim, price_sim, 0)
caratMin = list(caratQuantMinDict.keys())
quantMin = list(caratQuantMinDict.values())
```

```

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

plt.plot(caratMin, quantMin, color = 'purple')
plt.plot(caratMax, quantMax, color = 'orange')
plt.scatter(df_trim.carat, df_trim.price, color='black', alpha=0.2, s=10)
plt.xlabel("Carat")
plt.ylabel("Price")
plt.title("Model 2 - Prior predictive check")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()

```



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.

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.

Second model stan code:

```

1  data {
2    int N;
3    vector[N] carat;
4    array [N] int <lower=1, upper=5> cut;
5    array [N] int <lower=1, upper=8> clarity;
6    vector[N] price;
7  }
8
9  parameters {
10   vector[5] alpha_cut;
11   vector[8] alpha_clarity;
12   vector[5] beta_cut;
13   vector[8] beta_clarity;
14   real <lower=0> sigma;
15 }
16
17 transformed parameters {
18   array [N] real mu;
19   for (i in 1:N){
20     mu[i] = alpha_cut[cut[i]] + alpha_clarity[clarity[i]] + (beta_cut[cut[i]] + beta_clarity[clarity[i]]) * carat[i];
21   }
22 }
23
24 model {
25   alpha_cut ~ normal(-1000,10);
26   alpha_clarity ~ normal(-1000,10);
27   beta_cut ~ normal(500,100);
28   beta_clarity ~ normal(10000,2000);
29   sigma ~ exponential(10);
30
31   for (i in 1:N){
32     price[i] ~ normal(mu[i], sigma);
33   }
34 }
35
36 generated quantities {
37   vector[N] price_sim;
38   vector[N] log_lik;
39   for (i in 1:N){
40     price_sim[i] = normal_rng(mu[i], sigma);
41     log_lik[i] = normal_lpdf(price[i] | mu[i], sigma);
42   }
43 }

```

```
In [ ]: model_2 = CmdStanModel(stan_file='stanfiles/model_2.stan')
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
```

```
In [ ]: data_sim={'N':len(df_trim), 'carat': df_trim.carat, 'cut': df_trim.cut, 'clarity':
model_2_fit = model_2.sample(data=data_sim)
```

```
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.
```

```
In [ ]: model_2_fit.diagnose()
```

```
Out[ ]: 'Processing csv files: C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt\\model
_2-20220622095943_1.csv, C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt\\mod
el_2-20220622095943_2.csv, C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt\\m
odel_2-20220622095943_3.csv, C:\\Users\\Marcin\\AppData\\Local\\Temp\\tmpihm1etnt
\\model_2-20220622095943_4.csv\\n\\nChecking sampler transitions treedepth.\\nTreedep
th satisfactory for all transitions.\\n\\nChecking sampler transitions for divergenc
es.\\nNo divergent transitions found.\\n\\nChecking E-BFMI - sampler transitions HMC
potential energy.\\nE-BFMI satisfactory.\\n\\nEffective sample size satisfactory.\\n\\n
Split R-hat values satisfactory all parameters.\\n\\nProcessing complete, no problem
s detected.\\n'
```

Model 2 - model parameters

[Return to table of contents](#)

We can also extract stan variables that are used in the final price 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 diamonds in same class being slightly different from one another. The differences between distributions for each clarity class indicates that each class is modeled differently.

```
In [ ]: alpha_cut_fit = pd.DataFrame(model_2_fit.stan_variable('alpha_cut'))
alpha_clarity_fit = pd.DataFrame(model_2_fit.stan_variable('alpha_clarity'))
beta_cut_fit = pd.DataFrame(model_2_fit.stan_variable('beta_cut'))
beta_clarity_fit = pd.DataFrame(model_2_fit.stan_variable('beta_clarity'))
sigma_fit = model_2_fit.stan_variable('sigma')

fig, axs = plt.subplots(1,5)
fig.set_size_inches(25, 4)
for i in range(0,5):
    axs[i].hist(alpha_cut_fit[i])
    axs[i].set_title(f"alpha for cut = {i+1}")
    axs[i].grid()
plt.show()

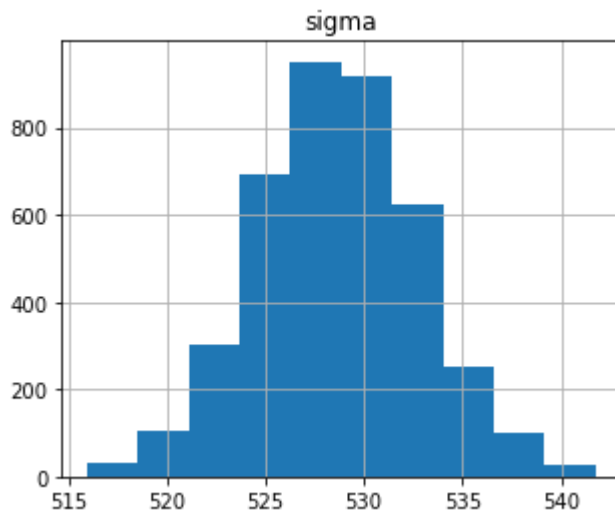
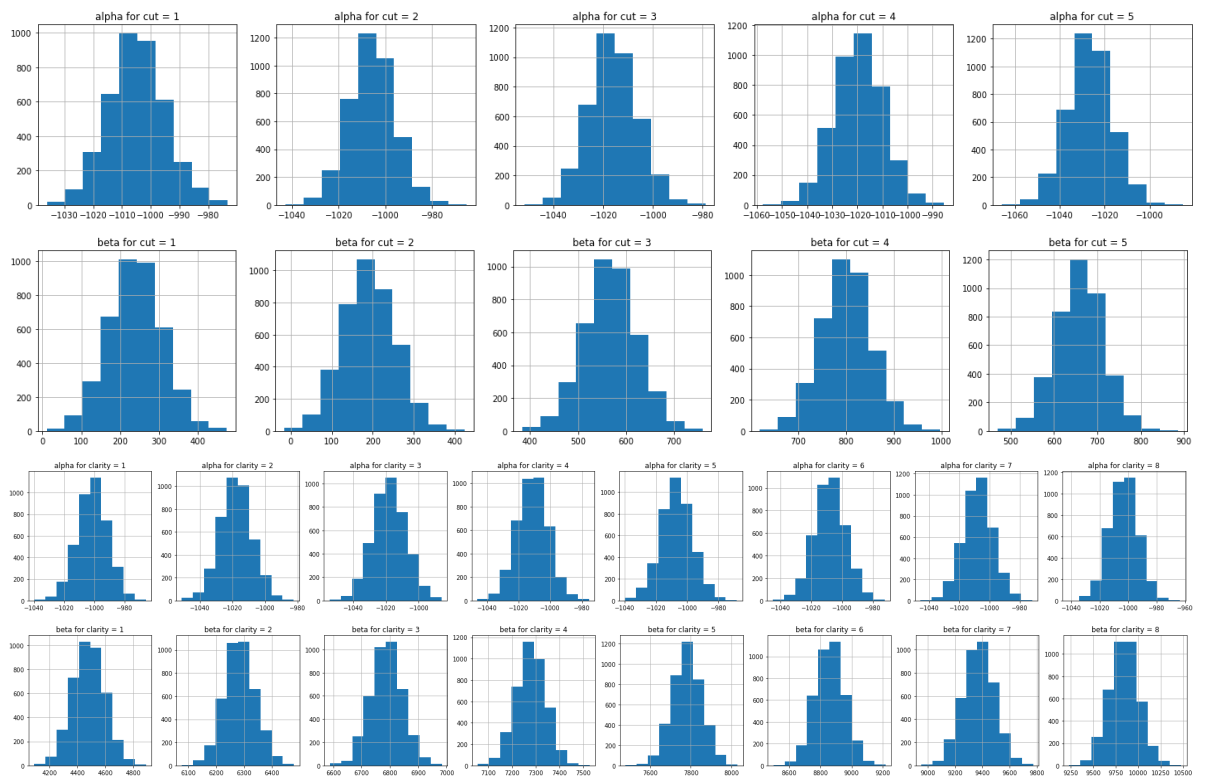
fig, axs = plt.subplots(1,5)
fig.set_size_inches(25, 4)
for i in range(0,5):
    axs[i].hist(beta_cut_fit[i])
    axs[i].set_title(f"beta for cut = {i+1}")
    axs[i].grid()
plt.show()

fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(alpha_clarity_fit[i])
    axs[i].set_title(f"alpha for clarity = {i+1}")
    axs[i].grid()
plt.show()

fig, axs = plt.subplots(1,8)
fig.set_size_inches(35, 4)
for i in range(0,8):
    axs[i].hist(beta_clarity_fit[i])
    axs[i].set_title(f"beta for clarity = {i+1}")
    axs[i].grid()
plt.show()

plt.figure(figsize=[5, 4])
plt.hist(sigma_fit)
plt.title('sigma')
plt.grid()
plt.show()

az.summary(model_2_fit, var_names=['alpha_cut', 'beta_cut', 'alpha_clarity', 'beta_
```



Out[]:

	mean	sd	hdi_3%	hdi_97%
alpha_cut[0]	-1004.95	9.82	-1023.32	-985.97
alpha_cut[1]	-1005.75	9.71	-1024.29	-987.47
alpha_cut[2]	-1016.04	9.77	-1034.04	-997.56
alpha_cut[3]	-1019.52	9.69	-1038.58	-1002.19
alpha_cut[4]	-1026.65	9.88	-1044.08	-1007.05
beta_cut[0]	239.13	68.92	100.59	359.74
beta_cut[1]	190.85	64.39	68.85	308.51
beta_cut[2]	567.33	56.16	468.88	681.72
beta_cut[3]	800.68	53.08	701.96	902.04
beta_cut[4]	660.13	54.60	558.52	763.41
alpha_clarity[0]	-1001.25	10.21	-1020.31	-982.05
alpha_clarity[1]	-1017.84	9.87	-1036.16	-999.62
alpha_clarity[2]	-1017.98	10.18	-1036.24	-997.71
alpha_clarity[3]	-1011.18	9.99	-1031.66	-994.12
alpha_clarity[4]	-1006.93	10.13	-1025.92	-987.07
alpha_clarity[5]	-1007.89	10.24	-1026.24	-987.54
alpha_clarity[6]	-1007.62	9.91	-1026.60	-989.51
alpha_clarity[7]	-1003.13	9.95	-1020.38	-983.38
beta_clarity[0]	4481.01	115.80	4260.21	4689.82
beta_clarity[1]	6284.66	56.50	6177.38	6389.73
beta_clarity[2]	6787.84	56.76	6678.46	6891.14
beta_clarity[3]	7279.83	62.95	7161.99	7397.82
beta_clarity[4]	7780.68	72.02	7649.66	7920.15
beta_clarity[5]	8861.82	94.80	8690.47	9043.69
beta_clarity[6]	9371.55	116.43	9154.40	9590.88
beta_clarity[7]	9842.54	162.63	9543.08	10145.70
sigma	528.67	4.12	520.86	536.66

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 diamonds fall within real data ranges.

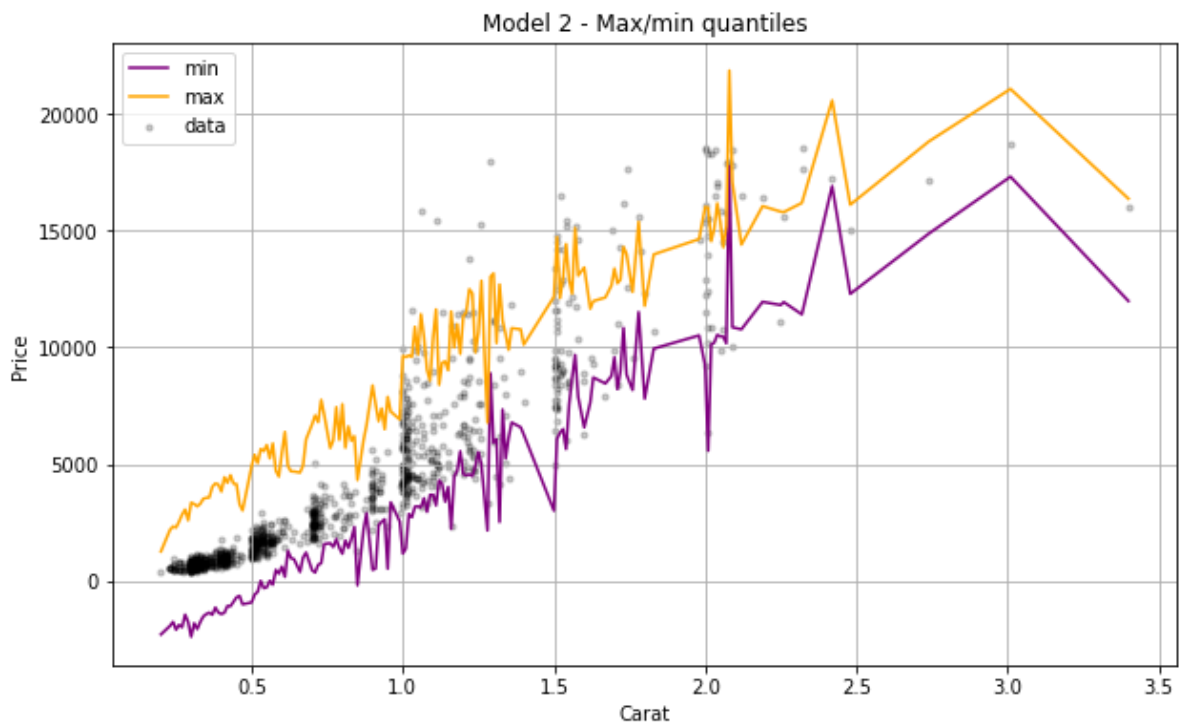
```
In [ ]: data = model_2_fit.draws_pd()
price_sims = data[data.columns[len(df_trim)+34:len(df_trim)+1034]]
#print(price_sims)

In [ ]: price_sim = model_2_fit.stan_variable('price_sim')
plt.figure(figsize=[10, 6])

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

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

plt.plot(caratMin, quantMin, color = 'purple')
plt.plot(caratMax, quantMax, color = 'orange')
plt.scatter(df_trim.carat, df_trim.price, color='black', alpha=0.2, s=10)
plt.xlabel("Carat")
plt.ylabel("Price")
plt.title("Model 2 - Max/min quantiles")
plt.legend(['min', 'max', 'data'])
plt.grid()
plt.show()
```



Model 2 - predictions and density plot

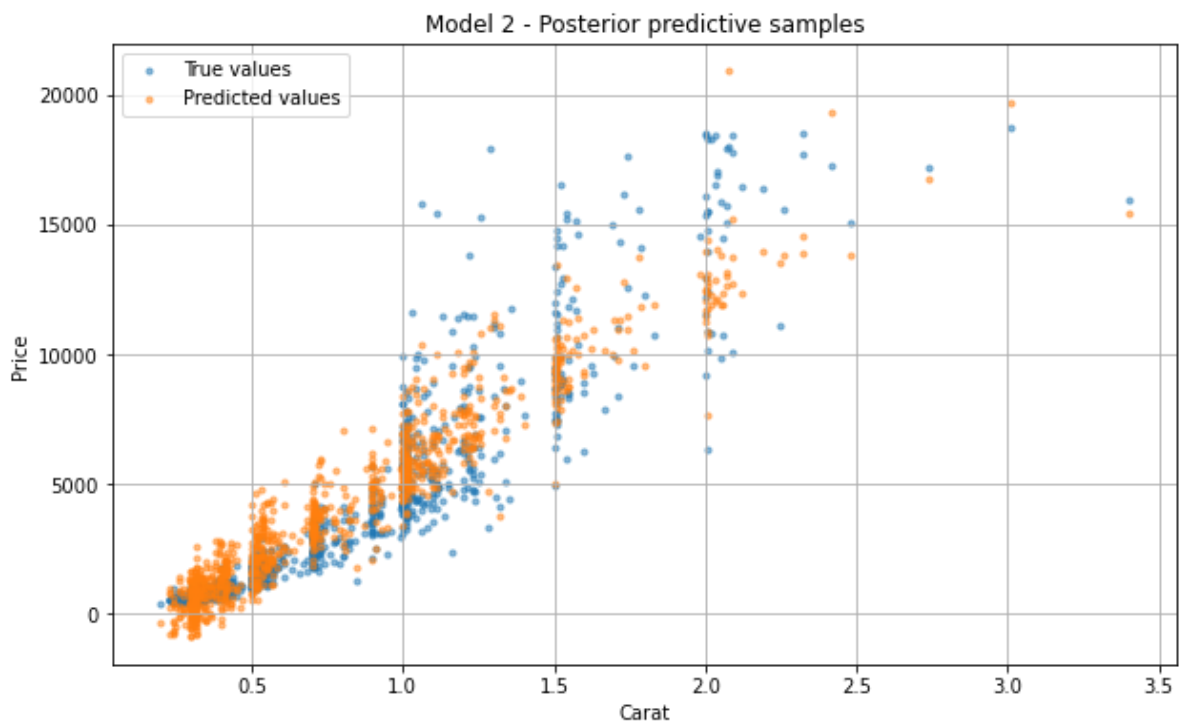
[Return to table of contents](#)

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 = model_2_fit.stan_variable('price_sim')
plt.figure(figsize=[10,6])
plt.scatter(df_trim.carat, df_trim.price, alpha=0.5, s=10)
plt.scatter(df_trim.carat, price_sim[1], alpha=0.5, s=10)
plt.title("Model 2 - Posterior predictive samples")
plt.legend(["True values", "Predicted values"])
plt.xlabel("Carat")
plt.ylabel("Price")
plt.grid()
plt.show()

df_trim.price.plot.density(figsize=(10,6), linewidth=2, color='red')
for i in range(0,10):
    price_sims.iloc[i].plot.density(linewidth=0.25, color='blue')
df_trim.price.plot.density(figsize=(10,6), linewidth=2, color='red')
plt.title('Density plot for Price')
plt.legend(["True values", "Predicted values"])
plt.grid()
plt.show()
```




```

fit_2 = az.from_cmdstanpy(posterior=model_2_fit,
                        log_likelihood="log_lik",
                        posterior_predictive="price_sim",
                        observed_data=df_trim['price'])

compare_dict = {"Model_1": fit_1, "Model_2": fit_2}

comp_loo = az.compare(compare_dict, ic = "loo")
print('\n')
print(comp_loo)
az.plot_compare(comp_loo)

```

```

e:\Programowanie\Anaconda\envs\marcinbereznicki\lib\site-packages\arviz\stats\stat
s.py:145: UserWarning: The default method used to estimate the weights for each mo
del, has changed from BB-pseudo-BMA to stacking
  warnings.warn(
e:\Programowanie\Anaconda\envs\marcinbereznicki\lib\site-packages\arviz\stats\stat
s.py:655: UserWarning: Estimated shape parameter of Pareto distribution is greater
than 0.7 for one or more samples. You should consider using a more robust model, t
his is because importance sampling is less likely to work well if the marginal pos
terior and LOO posterior are very different. This is more likely to happen with a
non-robust model and highly influential observations.
  warnings.warn(
e:\Programowanie\Anaconda\envs\marcinbereznicki\lib\site-packages\arviz\stats\stat
s.py:655: UserWarning: Estimated shape parameter of Pareto distribution is greater
than 0.7 for one or more samples. You should consider using a more robust model, t
his is because importance sampling is less likely to work well if the marginal pos
terior and LOO posterior are very different. This is more likely to happen with a
non-robust model and highly influential observations.
  warnings.warn(

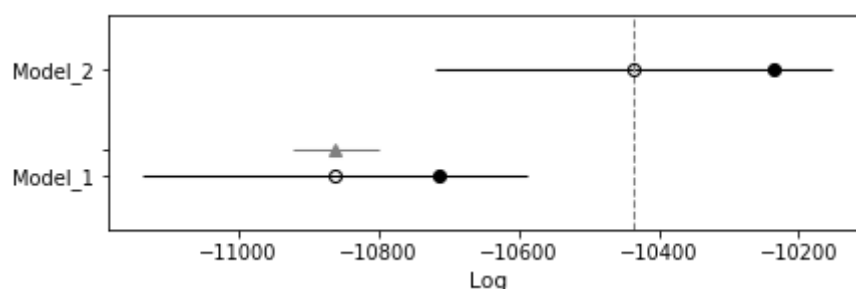
```

	rank	loo	p_loo	d_loo	weight	se	\
Model_2	0	-10435.866209	201.801427	0.000000	0.86503	283.488747	
Model_1	1	-10862.271012	149.030768	426.404803	0.13497	274.561077	

	dse	warning	loo_scale
Model_2	0.000000	True	log
Model_1	61.939052	True	log

<AxesSubplot:xlabel='Log'>

Out[]:



WAIC Criterion

[Return to table of contents](#)

Based on the comparison of the models using the WAIC criterion, it can be concluded that the conclusions are identical to the previous criterion, i.e.:

- Second model - three predictors - has lower rank (which means the best model).
- It has higher out-of-sample predictive fit ('waic' column).

- Has higher probability of the correctness of the model ('weight' column).
- Standard error of the difference information criteria between each model and the top ranked model ('dse' column) shows that the differences between the models are small.
- For both models there is a warning that indicates that the computation of the information criteria may not be reliable.

```
In [ ]: comp_waic = az.compare(compare_dict, ic = "waic")
print('\n')
print(comp_waic)
az.plot_compare(comp_waic)
```

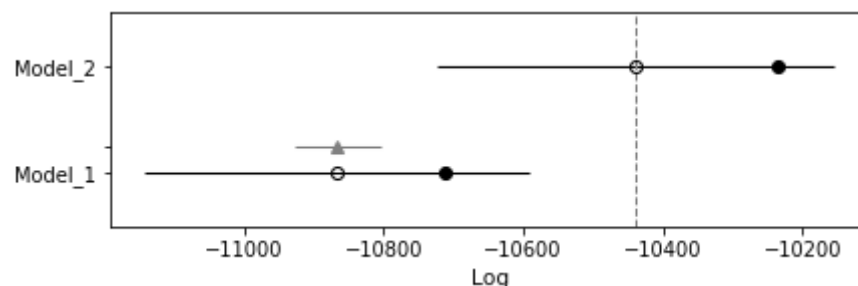
```
e:\Programowanie\Anaconda\envs\marcinbereznicki\lib\site-packages\arviz\stats\stat
s.py:145: UserWarning: The default method used to estimate the weights for each mo
del, has changed from BB-pseudo-BMA to stacking
  warnings.warn(
e:\Programowanie\Anaconda\envs\marcinbereznicki\lib\site-packages\arviz\stats\stat
s.py:1405: 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 fai
l.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
```

	rank	waic	p_waic	d_waic	weight	se \
Model_2	0	-10439.216019	205.151237	0.000000	0.865006	284.672670
Model_1	1	-10866.709288	153.469044	427.49327	0.134994	275.738111

	dse	warning	waic_scale
Model_2	0.000000	True	log
Model_1	61.844685	True	log

```
e:\Programowanie\Anaconda\envs\marcinbereznicki\lib\site-packages\arviz\stats\stat
s.py:1405: 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 fai
l.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
```

```
Out[ ]: <AxesSubplot:xlabel='Log'>
```



Model Comparison - conclusions

[Return to table of contents](#)

Comparing the models, both visually and by criteria, we can agree that the model with more predictors fits the real data better, although the differences seem to be small but not insignificant. It is most likely an issue caused by predictor values. As mentioned the clarity

and quality were provided merely as a classification that falls within a range of numeric values or subjective grading. Having precise measurements for these predictors could vastly improve the overall accuracy and provide an insight on predictor's importance in gem pricing.
