flight-price-prediction-mixture-model

May 28, 2024

1 Airline Ticket Price Prediction

This project aims to predict airline ticket prices based on the class of service (Economy or Business). The dataset used in this analysis is loaded from a CSV file and various machine learning and statistical techniques are applied to model the ticket prices.

1.1 Libraries and Dependencies

We begin by importing the necessary libraries and setting up the environment for the analysis.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import torch
import pyro

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn as skl
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import mutual_info_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_validate

import scipy.stats as stats

# matplotlib options
plt.style.use('ggplot')
%matplotlib inline
```

```
plt.rcParams['figure.figsize'] = (12, 8)
```

[ ]:
```
from pyro.infer import MCMC, NUTS
```

Loading the Dataset The dataset is loaded from a CSV file and its dimensions are printed.

[ ]:
```
# load the transformed dataset
business_df = pd.read_csv('../datasets/Transformed_Dataset.csv')
```

[ ]:
```
business_df.shape
```

Exploratory Data Analysis We create histograms to visualize the distribution of ticket prices based on the class of service.

[ ]:
```
print('have histograms based if it is economy class')

business_df.hist('price', bins=50, by=business_df['Economy'], sharey=True,␣
 ↪sharex=True)
```

[ ]:
```
# let's see histogram of all data together
business_df.hist('price', bins=70)
```

Splitting the Data The dataset is split into training and testing sets. The features (Economy) and the target variable (price) are separated and converted to numpy arrays.

[ ]:
```
# split the train and test, where test has class and the y has the price
train_x, test_x, train_y,test_y = train_test_split(
    business_df['Economy'].to_numpy(dtype=np.int_),
    business_df['price'].values,
    test_size=0.5, random_state=488)

# we need it as torch tensor
x_train_clean = torch.tensor(train_x)
y_train_clean = torch.tensor(train_y)

x_test_clean = torch.tensor(test_x)
y_test_clean = torch.tensor(test_y)

# let's get ratio of business and economy class tickets
x_train_clean.count_nonzero(), len(x_train_clean)
```

Pyro Model - Normal Distribution We define a Pyro model with Normal distribution priors for the ticket prices based on the class of service. The model is then used to run inference using MCMC.

[ ]:
```
def pyro_model(x=None, y=None):
    # priors for components

    mu_loc = torch.tensor([55000.,4500.])
    mu_scale = torch.tensor([1., 1.])
```

```
    sigma_scale = torch.tensor([10., 10.])

    # the component is the type of class - economy/business
    with pyro.plate('components', 2):
        mu = pyro.sample('mu', pyro.distributions.Normal(mu_loc,mu_scale))
        sigma = pyro.sample('sigma', pyro.distributions.HalfCauchy(sigma_scale))

    with pyro.plate('data', len(y)):
        buss = x
        # index the correct distribution from which we draw
        price = pyro.sample('price', pyro.distributions.
↪Normal(mu[buss],sigma[buss]), obs=y)

    return price
```

[ ]:
```
# Run inference in Pyro
nuts_kernel = NUTS(pyro_model)
mcmc = MCMC(nuts_kernel, num_samples=400, warmup_steps=200, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
```

Pyro Model - Gamma Distribution We define another Pyro model with Gamma distribution priors for the ticket prices based on the class of service. The model is then used to run inference using MCMC.

[ ]:
```
def pyro_model(x=None, y=None):
    # priors for components
    alpha_scale = torch.tensor([3., 3.])
    beta_scale = torch.tensor([10., 10.])

    # component of different classes - business and economy
    with pyro.plate('components', 2):
        alpha = pyro.sample('alpha', pyro.distributions.HalfCauchy(alpha_scale))
        beta = pyro.sample('beta', pyro.distributions.HalfCauchy(beta_scale))

    with pyro.plate('data', len(y)):
        buss = x
        # draw from the right distribution from the mix
        price = pyro.sample('price', pyro.distributions.
↪Gamma(alpha[buss],beta[buss]), obs=y)

    return price
```

```
[ ]: # Run inference in Pyro
nuts_kernel = NUTS(pyro_model)
mcmc = MCMC(nuts_kernel, num_samples=400, warmup_steps=200, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
```

Posterior Samples Analysis Finally, we extract and analyze the posterior samples from the MCMC inference.

```
[ ]: posterior_samples = mcmc.get_samples()
print(posterior_samples.keys())
```

```
[ ]: posterior_samples['beta'].mean(dim=0)
```

2 Airline Ticket Price Prediction - First Model

This project aims to predict airline ticket prices using Bayesian inference with the Pyro library. The first model defines priors for mixture components and the components themselves, and then runs inference using MCMC.

2.1 First Model Definition

We define a Pyro model called `first_model`. This model includes priors for the mixture components and the price components (mean and standard deviation).

```
    with pyro.plate('cut_off', 2):
        cutoff_day = pyro.sample('cut_days', pyro.distributions.Normal(cutoff_days_mu,cutoff_days_sigma))
        cutoff_correlation = pyro.sample('cut_corr', pyro.distributions.Normal(cutoff_correlaiton_mu,cutoff_correlaiton_sigma))

    mu_scale = torch.tensor([3., 3.])
    sigma_scale = torch.tensor([10., 10.])
    with pyro.plate('components', 2):
        mu = pyro.sample('mu', pyro.distributions.HalfCauchy(mu_scale))#Normal(mu_loc,mu_scale))#torch.zeros(0), torch.ones(1)))
        sigma = pyro.sample('sigma', pyro.distributions.HalfCauchy(sigma_scale))#torch.ones(1)))

    price_sigma = pyro.sample('price_sigma',pyro.distributions.HalfCauchy(1))
    with pyro.plate('data', len(y)):
        #buss = pyro.sample('busi', pyro.distributions.Bernoulli(pi), infer={"enumerate":"parallel"}).long()
        buss = x[:,0]
        is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
        #price = pyro.sample('price', pyro.distributions.Chi2(mu[buss]), obs=y)
        base_price = pyro.sample('base_price', pyro.distributions.Gamma(mu[buss], sigma[buss]), obs=y)

        price = pyro.sample('price', pyro.distributions.Normal(base_price+is_bought_late*cutoff_correlation[buss]*(x[:,1]-cutoff_day[buss]),price_sigma), obs=y)

    return price
✓ 0.0s
```

```
# Run inference in Pyro
nuts_kernel = NUTS(first_model)
mcmc = MCMC(nuts_kernel, num_samples=100, warmup_steps=50, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
✓ 9.8s
```

```
Sample: 100%|██████████| 120/120 [00:09, 12.30it/s, step size=1.12e-03, acc. prob=0.062]


                 mean       std    median      5.0%     95.0%     n_eff     r_hat
  cut_corr[0]    1.87      0.00      1.87      1.87      1.87     58.61      1.00
  cut_corr[1]    0.69      0.00      0.69      0.69      0.69     18.32      1.04
  cut_days[0]    1.00      0.00      1.00      1.00      1.00     42.70      1.00
  cut_days[1]   -0.31      0.00     -0.31     -0.31     -0.31     36.16      0.99
        mu[0]    1.73      0.01      1.73      1.73      1.73     16.35      1.05
        mu[1]    3.86      0.00      3.86      3.86      3.86     15.84      1.06
  price_sigma    0.00      0.00      0.00      0.00      0.00     70.52      1.00
     sigma[0]    0.00      0.00      0.00      0.00      0.00     17.11      1.05
     sigma[1]    0.00      0.00      0.00      0.00      0.00     38.72      1.01

Number of divergences: 78
```

2.2 Second Model Definition

The second model defines priors for the price components and runs inference using MCMC with updated parameters.

```python

    mu_scale = torch.tensor([3., 3.])
    sigma_scale = torch.tensor([10., 10.])
    with pyro.plate('components', 2):
        mu = pyro.sample('mu', pyro.distributions.HalfCauchy(mu_scale))#Normal(mu_loc,mu_scale))#torch.zeros(0), torch.ones(1)))
        sigma = pyro.sample('sigma', pyro.distributions.HalfCauchy(sigma_scale))#torch.ones(1)))

    price_sigma = pyro.sample('price_sigma',pyro.distributions.HalfCauchy(1))
    with pyro.plate('data', len(y)):
        #buss = pyro.sample('busi', pyro.distributions.Bernoulli(pi), infer={"enumerate":"parallel"}).long()
        buss = x[:,0]
        is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
        #price = pyro.sample('price', pyro.distributions.Chi2(mu[buss]), obs=y)
        base_price = pyro.sample('base_price', pyro.distributions.Gamma(mu[buss], sigma[buss]), obs=y)

        price = pyro.sample('price', pyro.distributions.Normal(base_price+is_bought_late*cutoff_correlation[buss]*(x[:,1]-cutoff_day[buss]),price_sigma), obs=y)

    return price
```
✓ 0.0s

```python
# Run inference in Pyro
nuts_kernel = NUTS(first_model)
mcmc = MCMC(nuts_kernel, num_samples=80, warmup_steps=40, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
```
✓ 13m 32.7s

```
Sample: 100%|██████████| 120/120 [13:32,  6.77s/it, step size=3.15e-08, acc. prob=0.247]
```

	mean	std	median	5.0%	95.0%	n_eff	r_hat
cut_corr[0]	0.59	0.00	0.59	0.59	0.59	nan	nan
cut_corr[1]	1.29	0.00	1.29	1.29	1.29	nan	nan
cut_days[0]	-1.99	0.00	-1.99	-1.99	-1.99	nan	nan
cut_days[1]	-0.55	0.00	-0.55	-0.55	-0.55	nan	nan
mu[0]	12.05	0.00	12.05	12.05	12.05	nan	nan
mu[1]	3.73	0.00	3.73	3.73	3.73	nan	nan
price_sigma	0.00	0.00	0.00	0.00	0.00	2.99	1.89
sigma[0]	0.00	0.00	0.00	0.00	0.00	nan	nan
sigma[1]	0.00	0.00	0.00	0.00	0.00	nan	nan

Number of divergences: 0

2.3 Third Model Definition

In the third attempt, the model definition is similar to the second attempt but with further adjusted parameters for inference. It also provides the best results in terms of convergence and effective sample size, with adjustments in the number of samples and warmup steps enhancing the model's perfor-

```
Sample: 100%|██████████| 150/150 [30:13, 12.09s/it, step size=1.60e-03, acc. prob=0.766]


                mean       std    median      5.0%     95.0%     n_eff     r_hat
  alpha[0,0]    2.30      0.08      2.28      2.19      2.44      5.80      1.01
  alpha[0,1]    3.46      0.09      3.47      3.41      3.56     16.70      1.10
  alpha[1,0]   16.38      0.17     16.36     16.16     16.63     88.82      0.99
  alpha[1,1]    4.25      0.03      4.25      4.21      4.31     23.23      1.06
  alpha[2,0]    3.11      0.15      3.09      2.90      3.39      6.26      1.05
  alpha[2,1]    6.34      0.08      6.33      6.22      6.45     32.61      0.99
  alpha[3,0]    0.14      0.00      0.14      0.13      0.14      3.55      1.91
  alpha[3,1]    3.42      0.03      3.41      3.38      3.46     58.30      1.02
  alpha[4,0]    1.38      0.12      1.42      1.18      1.54      2.81      2.21
  alpha[4,1]    5.14      0.59      5.31      4.36      5.65     10.36      1.10
  alpha[5,0]   15.23      0.10     15.24     15.05     15.40    116.54      0.99
  alpha[5,1]    4.78      0.03      4.77      4.74      4.82     91.47      0.99
   beta[0,0]    0.21      0.02      0.21      0.18      0.24      3.99      1.28
   beta[0,1]    0.00      0.00      0.00      0.00      0.00     18.48      1.08
   beta[1,0]    0.00      0.00      0.00      0.00      0.00     87.00      0.99
   beta[1,1]    0.00      0.00      0.00      0.00      0.00     26.03      1.06
   beta[2,0]    5.49      0.29      5.56      4.98      5.85      4.74      1.14
   beta[2,1]    0.00      0.00      0.00      0.00      0.00     37.63      0.99
   beta[3,0]    7.49      0.62      7.66      6.50      8.16      2.68      2.34
   beta[3,1]    0.00      0.00      0.00      0.00      0.00     62.82      1.02
   beta[4,0]    1.66      0.04      1.66      1.61      1.73      9.25      1.14
   beta[4,1]    0.00      0.00      0.00      0.00      0.00     10.33      1.10
   beta[5,0]    0.00      0.00      0.00      0.00      0.00    123.44      0.99
   beta[5,1]    0.00      0.00      0.00      0.00      0.00     86.39      0.99

Number of divergences: 0
```

mance.

2.4 Trying out Gamma

This model includes priors for the price components, but we use a Gamma distribution instead of

```
1  posterior_samples['beta'].mean(dim=0)
✓ 0.0s

tensor([[2.0850e-01, 8.4309e-04],
        [3.4713e-04, 5.8317e-04],
        [5.4865e+00, 1.1206e-03],
        [7.4890e+00, 6.4392e-04],
        [1.6631e+00, 8.2883e-04],
        [2.7456e-04, 6.1163e-04]])
```

the Normal distribution.

3 Make Predictions

3.0.1 Component Priors:

- **alpha** and **beta**: These are the shape and rate parameters for the Gamma distribution, respectively, modeled using a HalfCauchy distribution.

3.0.2 Data Likelihood:

- **buss**: Indicates the class of the ticket (economy or business) and is taken from the input x.
- **price**: The observed prices are modeled using a Gamma distribution, with parameters depending on the class of the ticket.

3.0.3 Prediction:

- **price_pred**: A predicted price sampled from the Gamma distribution using the inferred parameters.

[ ]:
```
def pyro_model(x=None, y=None):
    # priors for components
    alpha_scale = torch.tensor([3., 3.])
    beta_scale = torch.tensor([10., 10.])

    with pyro.plate('components', 2):
        alpha = pyro.sample('alpha', pyro.distributions.HalfCauchy(alpha_scale))
        beta = pyro.sample('beta', pyro.distributions.HalfCauchy(beta_scale))

    with pyro.plate('data', len(y)):
        buss = x

        pyro.sample('price', pyro.distributions.Gamma(alpha[buss],beta[buss]),␣
 ↪obs=y)

    # make a model for predicting
    price_pred = pyro.sample('price_pred', pyro.distributions.
 ↪Gamma(alpha[buss],beta[buss]))
    return price_pred
```

[ ]:
```
def pred(x, post_samples):
    return pyro.sample('prediction',pyro.distributions.
 ↪Gamma(post_samples['alpha'].mean(dim=0)[x],post_samples['beta'].
 ↪mean(dim=0)[x]))
```

[ ]:
```
posterior_samples = mcmc.get_samples()
```

[ ]:
```
# lets see what are the values for alpha and beta
print(posterior_samples['alpha'].mean(dim=0)[1], posterior_samples['beta'].
 ↪mean(dim=0)[1])
print(posterior_samples['alpha'].mean(dim=0)[0], posterior_samples['beta'].
 ↪mean(dim=0)[0])
```

[ ]:
```
# sample the ratio of business and economy so we can compare histogram from the␣
 ↪beggining
business = pyro.distributions.Gamma(torch.tensor(14.3093), torch.tensor(0.
 ↪0003)).sample_n(74938,)
```

```
economy = pyro.distributions.Gamma(torch.tensor(3.7298), torch.tensor(0.0006)).
↪sample_n(165184,)
```

[ ]:
```
# lets plot the mixing that we got
sns.histplot(business, bins=30, kde=True)
sns.histplot(economy, bins=30, kde=True)

plt.show()
```

[ ]:
```
# a nice method for calculating statistics form class
def compute_error(trues, predicted):
    corr = np.corrcoef(predicted, trues)[0,1]
    mae = np.mean(np.abs(predicted - trues))
    rae = np.sum(np.abs(predicted - trues)) / np.sum(np.abs(trues - np.
↪mean(trues)))
    rmse = np.sqrt(np.mean((predicted - trues)**2))
    r2 = max(0, 1 - np.sum((trues-predicted)**2) / np.sum((trues - np.
↪mean(trues))**2))
    return corr, mae, rae, rmse, r2
```

[ ]:
```
# get predictions
pred_y = pred(x_test_clean, posterior_samples)

# get statistics about errors
corr, mae, rae, rmse, r2 = compute_error(y_test_clean.numpy(),pred_y.numpy())

print("CorrCoef: %.3f\nMAE: %.3f\nRMSE: %.3f\nR2: %.3f" % (corr, mae, rmse, r2))
#type(pred_y.numpy()), type(y_test_clean.numpy())
```

4 Adding the airlines into mixing

Data Preparation: We start by splitting our dataset into training and testing sets. The dataset business_df contains features such as the class of travel (economy or business) and different airlines, which are one-hot encoded. The target variable is the price of the airline tickets.

[ ]:
```
# lets split the data into train and test, now the x contains class and airlines
train_x, test_x, train_y,test_y = train_test_split(
       business_df[['Economy', 'airline_AirAsia', 'airline_Air_India',␣
↪'airline_GO_FIRST',
       'airline_Indigo', 'airline_SpiceJet', 'airline_Vistara']].
↪to_numpy(dtype=np.int_),
       business_df['price'].values,
       test_size=0.2, random_state=488)

x_train_clean = torch.tensor(train_x) #business_df['Economy']
y_train_clean = torch.tensor(train_y)
```

```
x_test_clean = torch.tensor(test_x) #business_df['Economy']
y_test_clean = torch.tensor(test_y)
```

Defining the Pyro Model: The Pyro model is a hierarchical Bayesian model with Gamma distributions for the pricing. We define priors for the distribution parameters (alpha and beta) and then use these to model the observed prices.

```
[ ]: def pyro_model(x=None, y=None):
         airline_count = 6
         # priors for components
         alpha_scale = torch.tensor([14., 4.] * airline_count).
      ↪reshape(airline_count,2)
         beta_scale = torch.tensor([0.006, 0.003] * airline_count).
      ↪reshape(airline_count,2)

         with pyro.plate('components', 2): #component of class
             with pyro.plate('airlines',airline_count): # component of airlines
                 alpha = pyro.sample('alpha', pyro.distributions.
      ↪HalfCauchy(alpha_scale))
                 beta = pyro.sample('beta', pyro.distributions.
      ↪HalfCauchy(beta_scale))

         with pyro.plate('data', len(y)):
             buss = x[0] # get if it is business
             airline_index = x[1] # get the index of airline

             # draw from the corresponding distribution parameters
             price = pyro.sample('price', pyro.distributions.
      ↪Gamma(alpha[airline_index,buss],beta[airline_index,buss]), obs=y)

         return price
```

Data Transformation for Model Inference: To facilitate model handling, we transform the one-hot encoded airline data into indices.

```
[ ]: x_testing = torch.stack(
         (x_train_clean[:,0],
          torch.argmax(x_train_clean[:,1:], axis=1))) # for simple handling of␣
      ↪airlines, we take the airlines which are in one hot encoding and transform␣
      ↪that into indexing
     x_testing
```

```
[ ]: # lets see what bias is there based on number of flights per airline
     x_testing.unique(return_counts=True)
```

Running Inference with Pyro: We use the NUTS (No-U-Turn Sampler) kernel for MCMC (Markov Chain Monte Carlo) sampling to run our model and perform inference.

```
[ ]: # Run inference in Pyro
nuts_kernel = NUTS(pyro_model)
mcmc = MCMC(nuts_kernel, num_samples=100, warmup_steps=50, num_chains=1)
mcmc.run(x_testing,y_train_clean)

# Show summary of inference results
mcmc.summary()
```

Making Predictions: We define a prediction function that draws from the Gamma distribution based on the inferred posterior samples.

```
[ ]: def pred(x, post_samples): # define how we predict data - draw from Gamma based␣
  ↪on mean of parameters
    alpha = post_samples['alpha'].mean(dim=0)
    beta = post_samples['beta'].mean(dim=0)
    return pyro.sample('prediction', pyro.distributions.
  ↪Gamma(alpha[x[1],x[0]],beta[x[1],x[0]]))
```

```
[ ]: posterior_samples = mcmc.get_samples()
```

```
[ ]: # get precise values of the betas
posterior_samples['beta'].mean(dim=0)
```

```
[ ]: x = np.linspace (0, 30000, 500)

# for each of the airlines let's check how the distribution for economy class␣
  ↪price looks like
for alpha,beta,i in zip(posterior_samples['alpha'].mean(dim=0),␣
  ↪posterior_samples['beta'].mean(dim=0), range(6)):
    print('airline',i,'mean:', (alpha[1]/(beta[1])))
    print('alpha',alpha[1],'1/beta:', 1/(beta[1]))
    y = stats.gamma.pdf(x, a=alpha[1], scale=1/beta[1])
    plt.plot(x, y, label='airline '+str(i))

plt.legend()
plt.show()
```

```
[ ]: # transform test data the same way as we did for train data
x_validate = torch.stack((x_test_clean[:,0],torch.argmax(x_test_clean[:,1:],␣
  ↪axis=1)))
```

```
[ ]: pred_y = pred(x_validate, posterior_samples)

corr, mae, rae, rmse, r2 = compute_error(y_test_clean.numpy(),pred_y.numpy())

print("CorrCoef: %.3f\nMAE: %.3f\nRMSE: %.3f\nR2: %.3f" % (corr, mae, rmse, r2))
```

5 Using mixing for airlines as well

```python
    with pyro.plate('cut_off', 2):
        cutoff_day = pyro.sample('cut_days', pyro.distributions.Normal(cutoff_days_mu,cutoff_days_sigma))
        cutoff_correlation = pyro.sample('cut_corr', pyro.distributions.Normal(cutoff_correlaiton_mu,cutoff_correlaiton_sigma))

    mu_scale = torch.tensor([3., 3.])
    sigma_scale = torch.tensor([10., 10.])
    with pyro.plate('components', 2):
        mu = pyro.sample('mu', pyro.distributions.HalfCauchy(mu_scale))#Normal(mu_loc,mu_scale))#torch.zeros(0), torch.ones(1)))
        sigma = pyro.sample('sigma', pyro.distributions.HalfCauchy(sigma_scale))#torch.ones(1)))

    price_sigma = pyro.sample('price_sigma',pyro.distributions.HalfCauchy(1))
    with pyro.plate('data', len(y)):
        #buss = pyro.sample('busi', pyro.distributions.Bernoulli(pi), infer={"enumerate":"parallel"}).long()
        buss = x[:,0]
        is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
        #price = pyro.sample('price', pyro.distributions.Chi2(mu[buss]), obs=y)
        base_price = pyro.sample('base_price', pyro.distributions.Gamma(mu[buss], sigma[buss]), obs=y)

        price = pyro.sample('price', pyro.distributions.Normal(base_price+is_bought_late*cutoff_correlation[buss]*(x[:,1]-cutoff_day[buss]),price_sigma), obs=y)

    return price
```

✓ 0.0s

```python
# Run inference in Pyro
nuts_kernel = NUTS(first_model)
mcmc = MCMC(nuts_kernel, num_samples=100, warmup_steps=50, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
```

✓ 9.8s

```
Sample: 100%|██████████| 120/120 [00:09, 12.30it/s, step size=1.12e-03, acc. prob=0.062]

                 mean       std    median      5.0%     95.0%     n_eff     r_hat
  cut_corr[0]    1.87      0.00      1.87      1.87      1.87     58.61      1.00
  cut_corr[1]    0.69      0.00      0.69      0.69      0.69     18.32      1.04
  cut_days[0]    1.00      0.00      1.00      1.00      1.00     42.70      1.00
  cut_days[1]   -0.31      0.00     -0.31     -0.31     -0.31     36.16      0.99
        mu[0]    1.73      0.01      1.73      1.73      1.73     16.35      1.05
        mu[1]    3.86      0.00      3.86      3.86      3.86     15.84      1.06
  price_sigma    0.00      0.00      0.00      0.00      0.00     70.52      1.00
     sigma[0]    0.00      0.00      0.00      0.00      0.00     17.11      1.05
     sigma[1]    0.00      0.00      0.00      0.00      0.00     38.72      1.01

Number of divergences: 78
```

5.1 Using both parameter as latent

```python

    mu_scale = torch.tensor([3., 3.])
    sigma_scale = torch.tensor([10., 10.])
    with pyro.plate('components', 2):
        mu = pyro.sample('mu', pyro.distributions.HalfCauchy(mu_scale))#Normal(mu_loc,mu_scale))#torch.zeros(0), torch.ones(1)))
        sigma = pyro.sample('sigma', pyro.distributions.HalfCauchy(sigma_scale))#torch.ones(1)))

    price_sigma = pyro.sample('price_sigma',pyro.distributions.HalfCauchy(1))
    with pyro.plate('data', len(y)):
        #buss = pyro.sample('busi', pyro.distributions.Bernoulli(pi), infer={"enumerate":"parallel"}).long()
        buss = x[:,0]
        is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
        #price = pyro.sample('price', pyro.distributions.Chi2(mu[buss]), obs=y)
        base_price = pyro.sample('base_price', pyro.distributions.Gamma(mu[buss], sigma[buss]), obs=y)

        price = pyro.sample('price', pyro.distributions.Normal(base_price+is_bought_late*cutoff_correlation[buss]*(x[:,1]-cutoff_day[buss]),price_sigma), obs=y)

    return price
```

✓ 0.0s

```python
# Run inference in Pyro
nuts_kernel = NUTS(first_model)
mcmc = MCMC(nuts_kernel, num_samples=80, warmup_steps=40, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
```

✓ 13m 32.7s

```
Sample: 100%|██████████| 120/120 [13:32,  6.77s/it, step size=3.15e-08, acc. prob=0.247]

                 mean       std    median      5.0%     95.0%     n_eff     r_hat
  cut_corr[0]    0.59      0.00      0.59      0.59      0.59       nan       nan
  cut_corr[1]    1.29      0.00      1.29      1.29      1.29       nan       nan
  cut_days[0]   -1.99      0.00     -1.99     -1.99     -1.99       nan       nan
  cut_days[1]   -0.55      0.00     -0.55     -0.55     -0.55       nan       nan
        mu[0]   12.05      0.00     12.05     12.05     12.05       nan       nan
        mu[1]    3.73      0.00      3.73      3.73      3.73       nan       nan
  price_sigma    0.00      0.00      0.00      0.00      0.00      2.99      1.89
     sigma[0]    0.00      0.00      0.00      0.00      0.00       nan       nan
     sigma[1]    0.00      0.00      0.00      0.00      0.00       nan       nan

Number of divergences: 0
```

5.2 Last try

```
Sample: 100%|██████████| 150/150 [30:13, 12.09s/it, step size=1.60e-03, acc. prob=0.766]
```

	mean	std	median	5.0%	95.0%	n_eff	r_hat
alpha[0,0]	2.30	0.08	2.28	2.19	2.44	5.80	1.01
alpha[0,1]	3.46	0.09	3.47	3.41	3.56	16.70	1.10
alpha[1,0]	16.38	0.17	16.36	16.16	16.63	88.82	0.99
alpha[1,1]	4.25	0.03	4.25	4.21	4.31	23.23	1.06
alpha[2,0]	3.11	0.15	3.09	2.90	3.39	6.26	1.05
alpha[2,1]	6.34	0.08	6.33	6.22	6.45	32.61	0.99
alpha[3,0]	0.14	0.00	0.14	0.13	0.14	3.55	1.91
alpha[3,1]	3.42	0.03	3.41	3.38	3.46	58.30	1.02
alpha[4,0]	1.38	0.12	1.42	1.18	1.54	2.81	2.21
alpha[4,1]	5.14	0.59	5.31	4.36	5.65	10.36	1.10
alpha[5,0]	15.23	0.10	15.24	15.05	15.40	116.54	0.99
alpha[5,1]	4.78	0.03	4.77	4.74	4.82	91.47	0.99
beta[0,0]	0.21	0.02	0.21	0.18	0.24	3.99	1.28
beta[0,1]	0.00	0.00	0.00	0.00	0.00	18.48	1.08
beta[1,0]	0.00	0.00	0.00	0.00	0.00	87.00	0.99
beta[1,1]	0.00	0.00	0.00	0.00	0.00	26.03	1.06
beta[2,0]	5.49	0.29	5.56	4.98	5.85	4.74	1.14
beta[2,1]	0.00	0.00	0.00	0.00	0.00	37.63	0.99
beta[3,0]	7.49	0.62	7.66	6.50	8.16	2.68	2.34
beta[3,1]	0.00	0.00	0.00	0.00	0.00	62.82	1.02
beta[4,0]	1.66	0.04	1.66	1.61	1.73	9.25	1.14
beta[4,1]	0.00	0.00	0.00	0.00	0.00	10.33	1.10
beta[5,0]	0.00	0.00	0.00	0.00	0.00	123.44	0.99
beta[5,1]	0.00	0.00	0.00	0.00	0.00	86.39	0.99

```
Number of divergences: 0
```

with corresponding betas:

```
1  posterior_samples['beta'].mean(dim=0)
✓ 0.0s
```

```
tensor([[2.0850e-01, 8.4309e-04],
        [3.4713e-04, 5.8317e-04],
        [5.4865e+00, 1.1206e-03],
        [7.4890e+00, 6.4392e-04],
        [1.6631e+00, 8.2883e-04],
        [2.7456e-04, 6.1163e-04]])
```

both of them gives about 0.767 R^2

6 Now lets make it dependant based on time - days left until departure

[ ]:
```
# lets try to make it based on the days left, so x contain class and days left
train_x, test_x, train_y,test_y = train_test_split(
    business_df[['Economy','days_left']].to_numpy(dtype=np.int_),
    business_df['price'].values,
    test_size=0.3, random_state=488)

x_train_clean = torch.tensor(train_x) #business_df['Economy']
y_train_clean = torch.tensor(train_y)

x_test_clean = torch.tensor(test_x) #business_df['Economy']
y_test_clean = torch.tensor(test_y)

x_train_clean.count_nonzero(), len(x_train_clean)
```

[ ]:
```
x_train_clean, y_train_clean
```

[ ]:
```
def pyro_model(x=None, y=None):
    # priors for days
    cutoff_days_mu = torch.zeros(2)
    cutoff_days_sigma = torch.ones(2)
    cutoff_correlaiton_mu = torch.zeros(2)
    cutoff_correlaiton_sigma = torch.ones(2)
    with pyro.plate('cut_off', 2):
        cutoff_day = pyro.sample('cut_days', pyro.distributions.
↪Normal(cutoff_days_mu,cutoff_days_sigma))
        cutoff_correlation = pyro.sample('cut_corr', pyro.distributions.
↪Normal(cutoff_correlaiton_mu,cutoff_correlaiton_sigma))

    # priors for class - business/ economy
    alpha_scale = torch.tensor([3., 3.])
    beta_scale = torch.tensor([10., 10.])
    with pyro.plate('components', 2):
        alpha = pyro.sample('alpha', pyro.distributions.HalfCauchy(alpha_scale))
        beta = pyro.sample('beta', pyro.distributions.HalfCauchy(beta_scale))

    # prior for noise of final price
    price_sigma = pyro.sample('price_sigma',pyro.distributions.HalfCauchy(1))
    with pyro.plate('data', len(y)):

        buss = x[:,0]
        is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
        price = pyro.sample('price', pyro.distributions.Gamma(alpha[buss],␣
↪beta[buss]+is_bought_late*cutoff_correlation[buss]*(x[:
↪,1]-cutoff_day[buss])), obs=y)
```

```
        # get price as before to have the base one
        #base_price = pyro.sample('base_price', pyro.distributions.
↪Gamma(alpha[buss], beta[buss]))

        # if people buy late it is more costly - the base price is increased by␣
↪the extra cost
        #price = pyro.sample('price', pyro.distributions.
↪Normal(base_price+is_bought_late*cutoff_correlation[buss]*(x[:
↪,1]-cutoff_day[buss]),price_sigma), obs=y)

    return price
```

[ ]:
```
from pyro.contrib.autoguide import AutoDiagonalNormal, AutoMultivariateNormal
from pyro.infer import MCMC, NUTS, HMC, SVI, Trace_ELBO
from pyro.optim import Adam, ClippedAdam
```

[ ]:
```
%%time

# Define guide function
guide = AutoDiagonalNormal(pyro_model)

# Reset parameter values
pyro.clear_param_store()

# Define the number of optimization steps
n_steps = 1500

# Setup the optimizer
adam_params = {"lr": 0.009}
optimizer = ClippedAdam(adam_params)

# Setup the inference algorithm
elbo = Trace_ELBO(num_particles=3)
svi = SVI(pyro_model, guide, optimizer, loss=elbo)

# Do gradient steps
for step in range(n_steps):
    elbo = svi.step(x_train_clean, y_train_clean)
    if step % 100 == 0:
        print("[%d] ELBO: %.1f" % (step, elbo))
```

[ ]:
```
# Run inference in Pyro
nuts_kernel = NUTS(pyro_model)
mcmc = MCMC(nuts_kernel, num_samples=100, warmup_steps=50, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
```

```
mcmc.summary()
```

```
[ ]: def pred(x, post_samples):
         return pyro.sample('prediction',pyro.distributions.
      ↪Gamma(post_samples['alpha'].mean(dim=0)[x],post_samples['beta'].
      ↪mean(dim=0)[x]))
```

```
[ ]: def pred(x, post_samples): # define how we predict data - draw from Gamma based␣
      ↪on mean of parameters
         alpha = post_samples['alpha'].mean(dim=0)
         beta = post_samples['beta'].mean(dim=0)
         cutoff_day = post_samples['cut_days'].mean(dim=0)
         cutoff_correlation = post_samples['cut_corr'].mean(dim=0)

         buss = x[:,0]
         is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
         return pyro.sample('prediction', pyro.distributions.Gamma(alpha[buss],␣
      ↪beta[buss]+is_bought_late*cutoff_correlation[buss]*(x[:
      ↪,1]-cutoff_day[buss])))
```

```
[ ]: posterior_samples = mcmc.get_samples()
```

```
[ ]: pred_y = pred(x_test_clean, posterior_samples)

     corr, mae, rae, rmse, r2 = compute_error(y_test_clean.numpy(),pred_y.numpy())

     print("CorrCoef: %.3f\nMAE: %.3f\nRMSE: %.3f\nR2: %.3f" % (corr, mae, rmse, r2))
```

7 Results

7.1 First

```
    with pyro.plate('cut_off', 2):
        cutoff_day = pyro.sample('cut_days', pyro.distributions.Normal(cutoff_days_mu,cutoff_days_sigma))
        cutoff_correlation = pyro.sample('cut_corr', pyro.distributions.Normal(cutoff_correlaiton_mu,cutoff_correlaiton_sigma))

    mu_scale = torch.tensor([3., 3.])
    sigma_scale = torch.tensor([10., 10.])
    with pyro.plate('components', 2):
        mu = pyro.sample('mu', pyro.distributions.HalfCauchy(mu_scale))#Normal(mu_loc,mu_scale))#torch.zeros(0), torch.ones(1)))
        sigma = pyro.sample('sigma', pyro.distributions.HalfCauchy(sigma_scale))#torch.ones(1)))

    price_sigma = pyro.sample('price_sigma',pyro.distributions.HalfCauchy(1))
    with pyro.plate('data', len(y)):
        #buss = pyro.sample('busi', pyro.distributions.Bernoulli(pi), infer={"enumerate":"parallel"}).long()
        buss = x[:,0]
        is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
        #price = pyro.sample('price', pyro.distributions.Chi2(mu[buss]), obs=y)
        base_price = pyro.sample('base_price', pyro.distributions.Gamma(mu[buss], sigma[buss]), obs=y)

        price = pyro.sample('price', pyro.distributions.Normal(base_price+is_bought_late*cutoff_correlation[buss]*(x[:,1]-cutoff_day[buss]),price_sigma), obs=y)

    return price
```
✓ 0.0s

```
# Run inference in Pyro
nuts_kernel = NUTS(first_model)
mcmc = MCMC(nuts_kernel, num_samples=100, warmup_steps=50, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
```
✓ 9.8s

```
Sample: 100%|██████████| 120/120 [00:09, 12.30it/s, step size=1.12e-03, acc. prob=0.062]
```

	mean	std	median	5.0%	95.0%	n_eff	r_hat
cut_corr[0]	1.87	0.00	1.87	1.87	1.87	58.61	1.00
cut_corr[1]	0.69	0.00	0.69	0.69	0.69	18.32	1.04
cut_days[0]	1.00	0.00	1.00	1.00	1.00	42.70	1.00
cut_days[1]	-0.31	0.00	-0.31	-0.31	-0.31	36.16	0.99
mu[0]	1.73	0.01	1.73	1.73	1.73	16.35	1.05
mu[1]	3.86	0.00	3.86	3.86	3.86	15.84	1.06
price_sigma	0.00	0.00	0.00	0.00	0.00	70.52	1.00
sigma[0]	0.00	0.00	0.00	0.00	0.00	17.11	1.05
sigma[1]	0.00	0.00	0.00	0.00	0.00	38.72	1.01

Number of divergences: 78

Sec-

```

    mu_scale = torch.tensor([3., 3.])
    sigma_scale = torch.tensor([10., 10.])
    with pyro.plate('components', 2):
        mu = pyro.sample('mu', pyro.distributions.HalfCauchy(mu_scale))#Normal(mu_loc,mu_scale))#torch.zeros(0), torch.ones(1)))
        sigma = pyro.sample('sigma', pyro.distributions.HalfCauchy(sigma_scale))#torch.ones(1)))

    price_sigma = pyro.sample('price_sigma',pyro.distributions.HalfCauchy(1))
    with pyro.plate('data', len(y)):
        #buss = pyro.sample('busi', pyro.distributions.Bernoulli(pi), infer={"enumerate":"parallel"}).long()
        buss = x[:,0]
        is_bought_late = torch.lt(x[:,1],cutoff_day[buss]).type(torch.int8)
        #price = pyro.sample('price', pyro.distributions.Chi2(mu[buss]), obs=y)
        base_price = pyro.sample('base_price', pyro.distributions.Gamma(mu[buss], sigma[buss]), obs=y)

        price = pyro.sample('price', pyro.distributions.Normal(base_price+is_bought_late*cutoff_correlation[buss]*(x[:,1]-cutoff_day[buss]),price_sigma), obs=y)

    return price
```
✓ 0.0s

```
# Run inference in Pyro
nuts_kernel = NUTS(first_model)
mcmc = MCMC(nuts_kernel, num_samples=80, warmup_steps=40, num_chains=1)
mcmc.run(x_train_clean,y_train_clean)

# Show summary of inference results
mcmc.summary()
```
✓ 13m 32.7s

```
Sample: 100%|██████████| 120/120 [13:32,  6.77s/it, step size=3.15e-08, acc. prob=0.247]
```

	mean	std	median	5.0%	95.0%	n_eff	r_hat
cut_corr[0]	0.59	0.00	0.59	0.59	0.59	nan	nan
cut_corr[1]	1.29	0.00	1.29	1.29	1.29	nan	nan
cut_days[0]	-1.99	0.00	-1.99	-1.99	-1.99	nan	nan
cut_days[1]	-0.55	0.00	-0.55	-0.55	-0.55	nan	nan
mu[0]	12.05	0.00	12.05	12.05	12.05	nan	nan
mu[1]	3.73	0.00	3.73	3.73	3.73	nan	nan
price_sigma	0.00	0.00	0.00	0.00	0.00	2.99	1.89
sigma[0]	0.00	0.00	0.00	0.00	0.00	nan	nan
sigma[1]	0.00	0.00	0.00	0.00	0.00	nan	nan

Number of divergences: 0

ond ##
Testing of the second one

8 Adding more specific features on top of class

[ ]:
```
business_df.columns
```

[ ]:
```
business_df
just_b_df = business_df[business_df['Economy'] == 0]
just_e_df = business_df[business_df['Economy'] == 1]
```

[ ]:
```
# have a function that can see if there is some correlation in the attribute␣
↪and the price, for business and economy class
def show_if_att_correlated(column):
    ax1 = plt.subplot(121)
    ax1.hist2d(just_b_df[column], just_b_df['price'], bins=10)
    ax2 = plt.subplot(122)#, sharey=ax1)
    ax2.hist2d(just_e_df[column], just_e_df['price'], bins=10)
    plt.show()
```

[ ]:
```
# check distribution of the price for each airline separately
for airline in ['airline_AirAsia', 'airline_Air_India', 'airline_GO_FIRST',
       'airline_Indigo', 'airline_SpiceJet', 'airline_Vistara']:
    just_e_df[just_e_df[airline]>0].hist('price', bins=20)
```

[ ]:
```
np.argmax(just_e_df[['airline_AirAsia', 'airline_Air_India', 'airline_GO_FIRST',
       'airline_Indigo', 'airline_SpiceJet', 'airline_Vistara']].values, axis=1)
```

[ ]:
```
show_if_att_correlated('days_left')
```

[ ]:
```
show_if_att_correlated('duration')
```

[ ]:
```
show_if_att_correlated('destination_size')
```

[ ]:
```
show_if_att_correlated('source_size')
```

[ ]:
```
def show_hist_2d(columnx,columny, ylim=None , xlim=None ,bins=10, title=None):
    plt.hist2d(business_df[columnx], business_df[columny], bins=bins)
    axes = plt.gca()
    axes.set_ylim(ylim)
    axes.set_xlim(xlim)
    plt.title(title)
    plt.show()
```

[ ]:
```
show_hist_2d('duration','price', title='Price and duration 2d␣
↪histogram',ylim=[0,80000], xlim=[1,30],bins=50)
```

[ ]:
```
show_hist_2d('days_left', 'price', title='days left and price 2d histogram'␣
↪,bins=30, ylim=[0,80000])
```

9 Running PCA to get ideas of what is important

[ ]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
pca = PCA(n_components=2)
```

[ ]:
```
X = business_df.drop('price', inplace=False, axis=1)
```

[ ]:
```
X_s = StandardScaler().fit_transform(X)
pcas = PCA(n_components=2)
ress = pcas.fit_transform(X_s)
print(pcas.explained_variance_ratio_)
print('sum:', np.sum(pcas.explained_variance_ratio_))
```

[ ]:
```
plt.matshow(pcas.components_, cmap='viridis')
plt.yticks([0, 1], ["First component", "Second component"])
plt.colorbar()
plt.xticks(range(len(X.columns)),
           X.columns, rotation=60, ha='left')
plt.xlabel("Feature")
plt.ylabel("Principal components")
plt.show()
```

[ ]:
```
# lets see the correlation matrix
corr_m = business_df.corr()

mask = np.triu(corr_m)
plt.figure(figsize=(17,17))
sns.heatmap(corr_m, annot=True, mask=mask)
plt.show()
```