Crypto Trade Analysis

Project Title: Bayesian Analysis of BTC/USDT Trades Using Gamma-Poisson, LogNormal-Poisson, and Joint Normal Models

Team:

1. Shaik Sattar Saif (MA23BTECH11023)
2. Chaitanya Nemmani (MA23BTECH11008)

You can find all our codes here: https://github.com/Zen-Nightshade/Crypto-Trade-Analysis

The relevant codes have also been mentioned in this report wherever necessary.

Abstract

This project analyzes the BTC/USDT (Bitcoin to Tether) trade dataset from Binance using Bayesian modeling. The **number of trades per minute** is modeled using Gamma-Poisson and LogNormal-Poisson distributions, while the **trade volume per minute** is modeled with a Joint Normal distribution with unknown mean and variance. Posterior distributions are estimated through **simulation**, and model validation is performed by **comparing observed trade counts and volumes with the posterior means**. The proportion of observed data falling within the **95% posterior confidence intervals** is also evaluated to assess model fit and uncertainty.

Problem Statement

Cryptocurrency trading, particularly in highly liquid pairs like BTC/USDT (Bitcoin to Tether), exhibits rapid and variable activity, making it challenging to accurately model trade dynamics. This analysis aims to study the **number of trades per minute** and the **volume of trades per minute** to understand the underlying market behavior.

For the **number of trades**, we have explored Bayesian count models: **Gamma-Poisson** and **LogNormal-Poisson** to account for over-dispersion and variability in trade frequency. For the **trade volume**, we model the **logarithm of trade costs** using a **Joint Normal distribution** with unknown mean and variance, enabling robust inference of volume fluctuations.

The goal is to leverage these Bayesian models to capture the probabilistic structure of trade activity, assess uncertainty, and provide insights into high-frequency trading patterns in the BTC/USDT market.

Data Collection & Preprocessing

Data Fetching

The trade data for the BTC/USDT pair was fetched using the **Binance API** via the `ccxt` Python library, covering the period from **2025-10-01 00:00:00** to **2025-10-08 00:00:00**. We retrieved all trades in this period, including timestamp, trade price, and trade volume.

```
import ccxt
import pandas as pd
import time
import os

binance = ccxt.binance()

symbol = 'BTC/USDT'
start_time = int(pd.Timestamp('2025-10-01 00:00:00').timestamp() * 1000)
end_time = int(pd.Timestamp('2025-10-08 00:00:00').timestamp() * 1000)

output_dir = "../data/raw/one_week"
os.makedirs(output_dir, exist_ok=True)

all_trades = []
since = start_time
chunk_size = 1000 # trades per API call
save_every = 10 # batches before saving

batch_count = 0
print(f"Starting from: {pd.to_datetime(since, unit='ms')}")

while since < end_time:
    try:
        trades = binance.fetch_trades(symbol, since=since, limit=chunk_size)
    except Exception as e:
```

```
        print(f"\nError fetching trades: {e}. Retrying in 10s...")
        time.sleep(10)
        continue

    if not trades:
        print("\nNo more trades returned. Exiting loop.")
        break

    all_trades.extend(trades)
    since = trades[-1]['timestamp'] + 1
    batch_count += 1

    if batch_count % save_every == 0:
        df_chunk = pd.DataFrame(all_trades)
        df_chunk['timestamp'] = pd.to_datetime(df_chunk['timestamp'], unit='ms', errors='coerce')

        df_chunk = df_chunk.dropna(subset=['timestamp'])

        df_chunk['date_time'] = df_chunk['timestamp'].dt.strftime("%Y-%m-%d_%H")

        for dt_str, group in df_chunk.groupby('date_time'):
            out_path = os.path.join(output_dir, f"trades_{dt_str}.csv")
            write_header = not os.path.exists(out_path)
            group.to_csv(out_path, mode='a', index=False, header=write_header)
            print(f"\nSaved {len(group)} trades → {out_path}")

        all_trades = []

    human_time = pd.to_datetime(since, unit='ms')
    print(f"\rFetched up to: {human_time}", end="", flush=True)

    # Avoid rate-limit issues
    time.sleep(0.5)

if all_trades:
    df_final = pd.DataFrame(all_trades)
    df_final['timestamp'] = pd.to_datetime(df_final['timestamp'], unit='ms', errors='coerce')
    df_final = df_final.dropna(subset=['timestamp'])
    df_final['date_time'] = df_final['timestamp'].dt.strftime("%Y-%m-%d_%H")

    for dt_str, group in df_final.groupby('date_time'):
        out_path = os.path.join(output_dir, f"trades_{dt_str}.csv")
        write_header = not os.path.exists(out_path)
        group.to_csv(out_path, mode='a', index=False, header=write_header)
        print(f"\nFinal save: {len(group)} trades → {out_path}")

print(f"\nDone! Trades saved to hourly CSV files in: {output_dir}")
```

Preprocessing

After fetching the trade data, we preprocessed it by **aggregating trades into 1-minute intervals**. For each interval, we have computed:

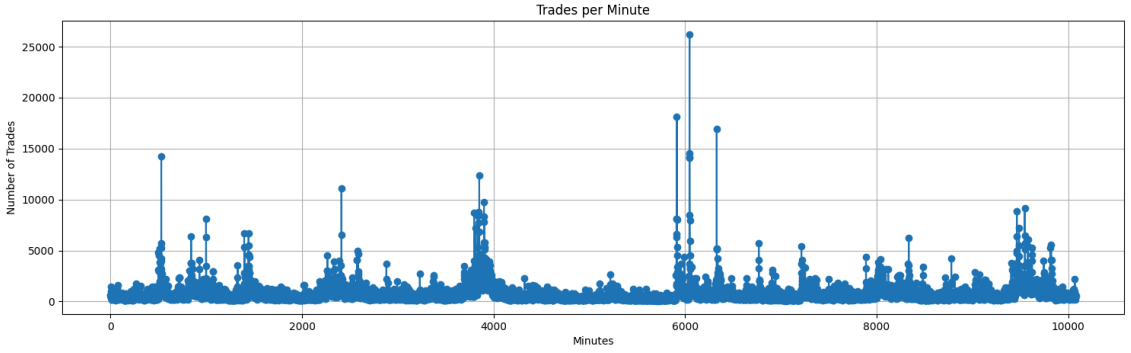
- **Aggregated bought and sold quantities**
- **Total traded volume and cost**
- **Number of trades occurring within the minute**
  This preprocessing ensured that the data is suitable for modeling both the **trade frequency** and **trade volume** in 1-minute intervals.

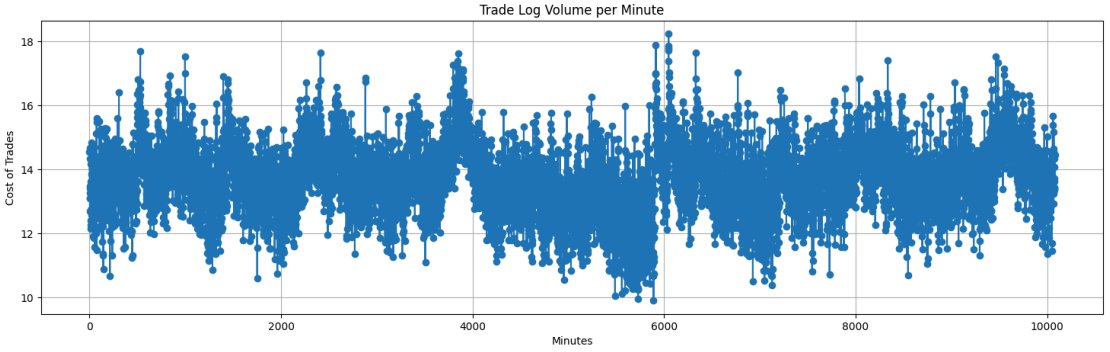
```
import pandas as pd
import numpy as np
import os

group = "10s" # Examples:- "10s", "1min"/"1T", "1H", "1D", "1W"

columns = ["timestamp", "symbol", "side", "price", "amount", "cost"]
input_dir = "../data/raw/one_week/"
output_dir = "../data/processed/trades_2025-10-01_to_2025-10-07_10s.csv"
```

```
file_paths = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith(".csv")]

trades = pd.DataFrame()
for file_path in file_paths:
    df = pd.read_csv(file_path)
    # print(file_path.split("/")[-1],len(df),sep="\t:\t")
    df = df[columns]
    df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce")

    df["bought"] = np.where(df["side"] == "buy", df["amount"], 0)
    df["sold"] = np.where(df["side"] == "sell", df["amount"], 0)

    df.drop(columns= ["side"], axis=1)
    # "timestamp", "symbol", "price", "amount", "bought", "sold"
    symbol_value = df["symbol"].iloc[0]

    df_agg = (
        df.set_index("timestamp")
        .resample(group)
        .apply(lambda x: pd.Series({
            "bought": x["bought"].sum(),
            "sold": x["sold"].sum(),
            "amount": x["amount"].sum(),
            "cost": x["cost"].sum(),
            "trade_count": len(x)

        }))
        .reset_index()
    )
    df_agg["symbol"] = symbol_value
    trades = pd.concat([trades, df_agg], ignore_index=True)

# trades["timestamp"] = pd.to_datetime(trades["timestamp"], errors="coerce")
trades = trades.sort_values(by="timestamp").reset_index(drop=True)

print(f"\nTrades\t:\t{len(trades)}\n")
trades.to_csv(output_dir, index=False)
```

Exploratory Data Analysis

We start our analysis by plotting the trade count per minute as well as the log of the trade costs per minute.





We observe that there are several sharp increases in the per-minute trade counts. These peaks quickly fall off and come back to a neutral point.

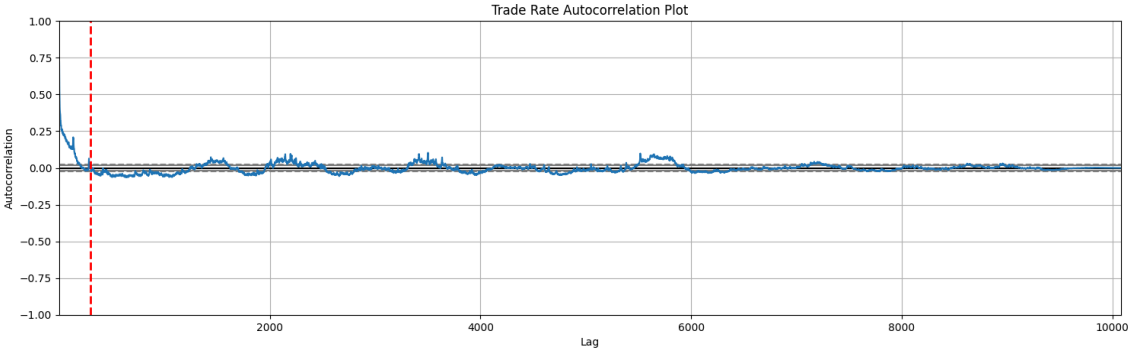
On the other hand, log of the trade costs (which will be referred to as trade volume for here on out) is extremely volatile, with very frequent rises and falls. The trade volume is thus highly dynamic.

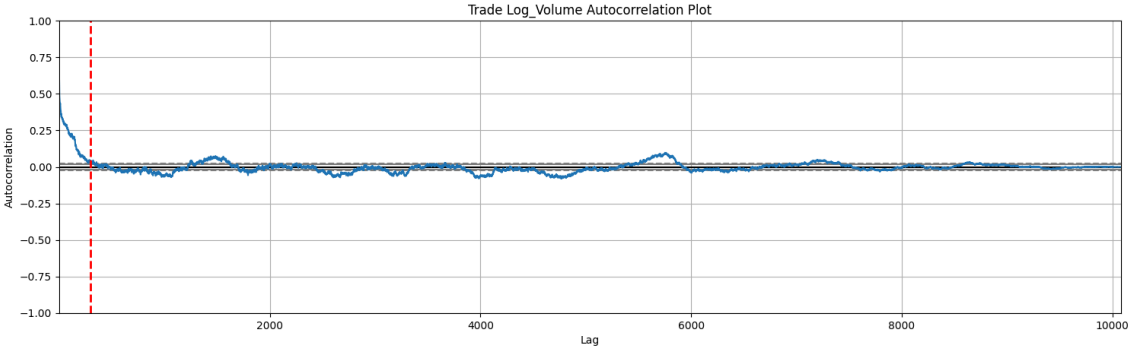
Since our data depends on time, computing the autocorrelation factor is informative. The autocorrelation factor for a given lag measures how values at time t are related to the values at times $t - lag$. The formula for the same is defined as follows:

$$\rho_{lag} = \frac{\sum_{t=lag+1}^{N}(X_t - \bar{X})(X_{t-lag} - \bar{X})}{\sum_{t=1}^{N}(X_t - \bar{X})^2}$$

A higher value of ρ_{lag} indicates higher dependance.

The plots of the Autocorrelation Factor for different values of lag for Trade Counts and Trade Volumes are given below.





We observe that the Autocorrelation factor in both the cases is highest for smaller values of lag. This indicates a dependence of the trade counts and volumes on more recent trades. Additionally note that the Autocorrelation factor is much larger for Trade Volumes. This result will be useful in future analysis.

Bayesian Modelling of Trade Rate

We aim to model the **number of trades occurring per minute**, denoted as Y, as a **Poisson random variable**. The Poisson distribution is a natural choice for modeling **count data** such as trade arrivals within a fixed time interval, assuming that trades occur independently and at a constant underlying rate.

$$Y_t \sim \text{Poisson}(\lambda_t)$$

where λ_t represents the **latent trade rate** (expected number of trades per minute) at time t.

To capture prior uncertainty in the trade rate, we explore two different prior distributions for λ_t:

- **Gamma Prior:**
  The conjugate prior for the Poisson likelihood, enabling analytical tractability of the posterior.

  $$\lambda_t \sim \text{Gamma}(\alpha, \beta)$$

- **Log-Normal Prior:**
  A more flexible, non-conjugate prior that allows for modeling heavier tails and potential asymmetry in trade rate variability.

  $$\lambda_t \sim \text{LogNormal}(\mu, \sigma^2)$$

  These two priors represent distinct modeling philosophies: the **Gamma–Poisson** model emphasizes count overdispersion through conjugacy, while the **LogNormal–Poisson** model provides greater flexibility for capturing skewed rate dynamics.

Gamma Poisson Modeling of Trade Rate

As the Gamma distribution acts as a conjugate prior for this Poisson distribution, we model the mean of the Poisson distribution (λ) as a Gamma Random Variable, i.e.,

$$\lambda \sim \text{Gamma}(\alpha, \beta)$$
$$Y \sim \text{Poisson}(\lambda)$$

Note that if we use have a sample $Y = (y_1, y_2, \ldots, y_n)$, then the posterior distribution is given by

$$P(\lambda|Y) \sim \text{Gamma}(\alpha + \sum_{i=1}^{n} y_i, \beta + n)$$

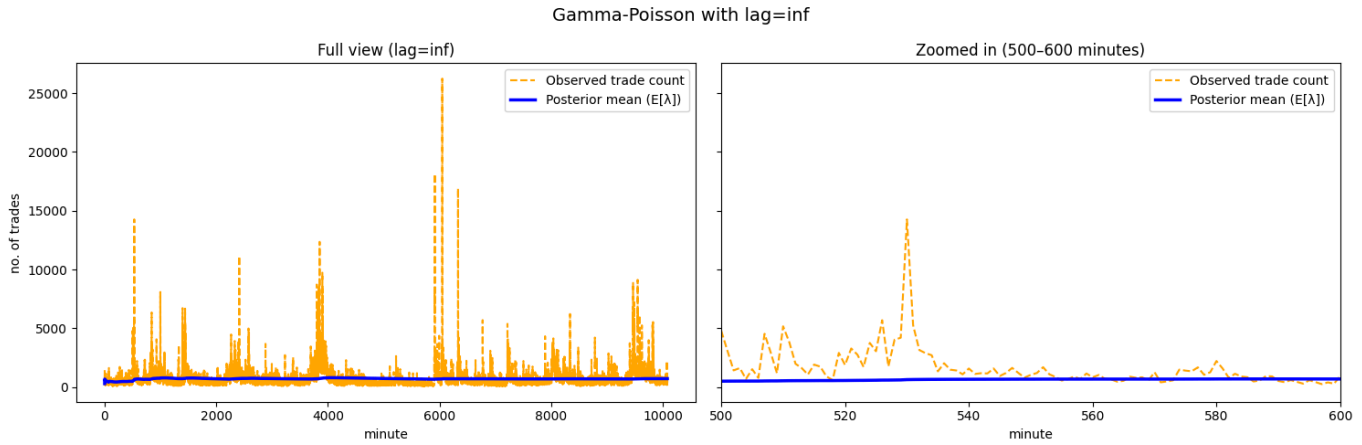
The following code computes the posterior distributions for the Gamma–Poisson model. Note that it is written in a way that can be adapted for lag-based modeling, which will be discussed later in this report.

```
class Gamma_Poisson:
    def __init__(self, alpha, beta, lag, df: pd.DataFrame):
        if alpha <= 0 or beta <= 0:
            raise ValueError("alpha and beta must be > 0")
        if lag <=0:
            raise ValueError("lag must be a Natural number")

        self.alpha = alpha
        self.beta = beta
        self.lag = (lag)
        self.df = df

        self.params = {"alpha":[alpha], "beta":[beta]}
        self.ppi = {}

    def generate_posterior(self):
        df = self.df
        df = df.sort_values(by="timestamp").reset_index(drop=True)
        array = df["trade_count"].tolist()
        alphas = []
        betas = []
        for i in range(len(array)):
            sum_counts = 0
            upd_time = 0
            for j in range(max(0, i - self.lag + 1), i):
                sum_counts += array[j]
                upd_time = upd_time + 1

            alphas.append(sum_counts + self.alpha)
            betas.append(self.beta + upd_time)

        self.params["alpha"] = alphas
        self.params["beta"] = betas

    def generate_ppi(self, percent):
        ppis = {"lower_bound":[], "upper_bound":[]}
        percent = 1 - percent
        for alpha, beta in zip(self.params["alpha"][0:], self.params["beta"][0:]):
            r = alpha
            p = beta / (beta + 1)

            lb = scipy.stats.nbinom.ppf(percent / 2, r, p)
            ub = scipy.stats.nbinom.ppf(1 - percent / 2, r, p)
            ppis["lower_bound"].append(lb)
            ppis["upper_bound"].append(ub)

        self.ppi[1 - percent] = ppis
```

Posterior Predictive Mean Plots

Now, after computing the posterior distribution, we have plotted the expectation of the posterior means of λ against the observed trades per minute data to see how well our Bayesian model mimics the actual data. We choose a weakly informative prior (α = 1, β = 1) so that no prior bias was introduced into the model. We perform a sequential update of the prior by number of trades at time t. The posterior

distribution was obtained with data at time t is used as the prior at time $t+1$. Using this, the following plot was obtained:



Note that this plot is very insensitive to sudden changes in the market, i.e., it doesn't respond to sudden rises/falls in the number of trades happening in a minute. In fact, we can see that the prediction remains almost constant throughout. This is because

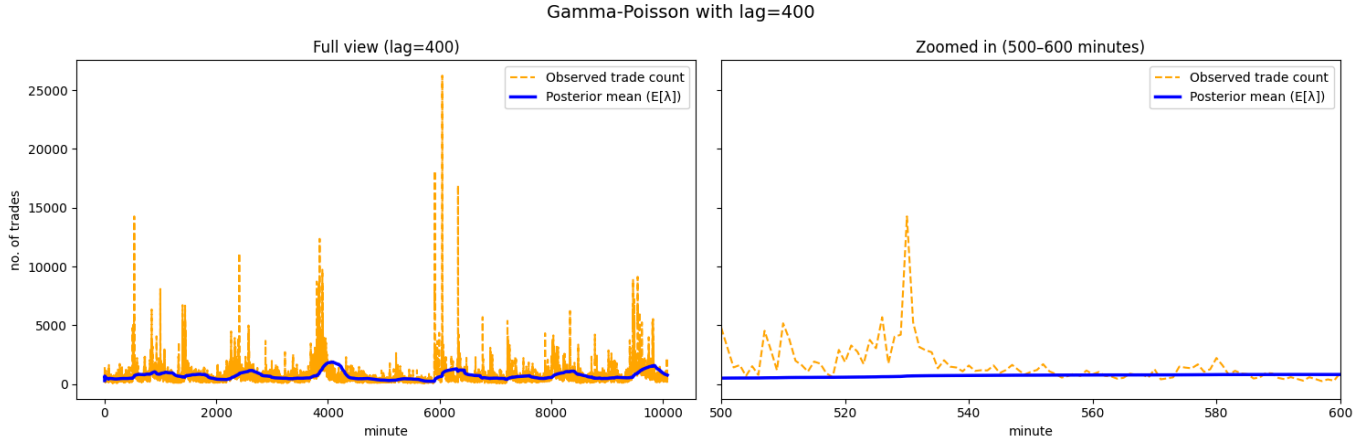
$$E[\lambda|Y] = \frac{\alpha + \sum_{i=1}^{n} y_i}{\beta + n}$$

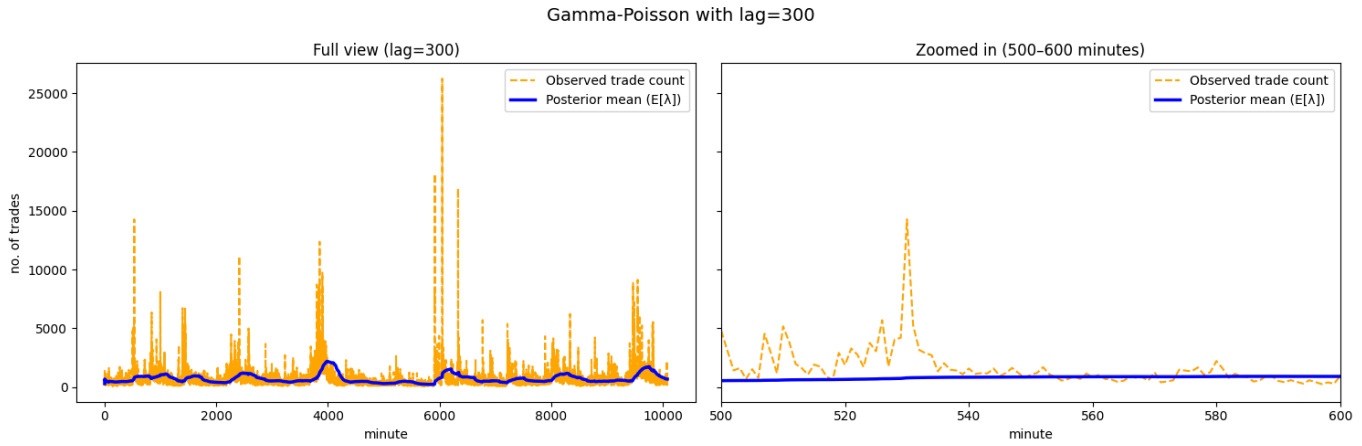
Since our prior assumes $\alpha = 1 \text{ and } \beta = 1$, as n increases, the posterior mean goes to $\overline{Y}$, the sample mean. The sample mean remains approximately constant throughout the data, as rise in number of trades is generally countered by a subsequent fall soon after. As such, using such a form of Bayesian Modelling seems inappropriate for this problem.

However, our data analysis revealed high autocorrelation at lower lags, indicating that the number of trades at time t is strongly influenced by the trades occurring shortly before t. As such, we aim to perform Bayesian Analysis using the data at lower lags.

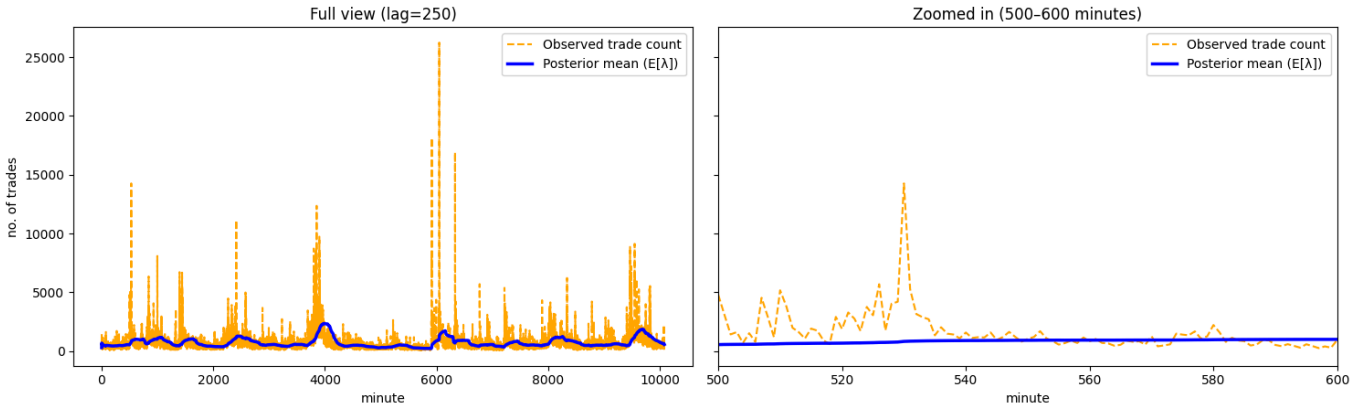
We again take the same weakly informative prior as before. However, at every time t, we update this same prior using trade counts in the time window from $t - lag$ to $t - 1$ (both inclusive), where lag is the number of recent minutes we aim to use (this is known as lag based modeling). Note that we are using the last $lag - 1$ datapoints to update our prior.

This analysis was performed by choosing various lag values. In particular, we choose lag = 400, 300, 250, 200, 150, 100, 50, 10, 5, 3, 2. The following plots were obtained for the corresponding lag values.

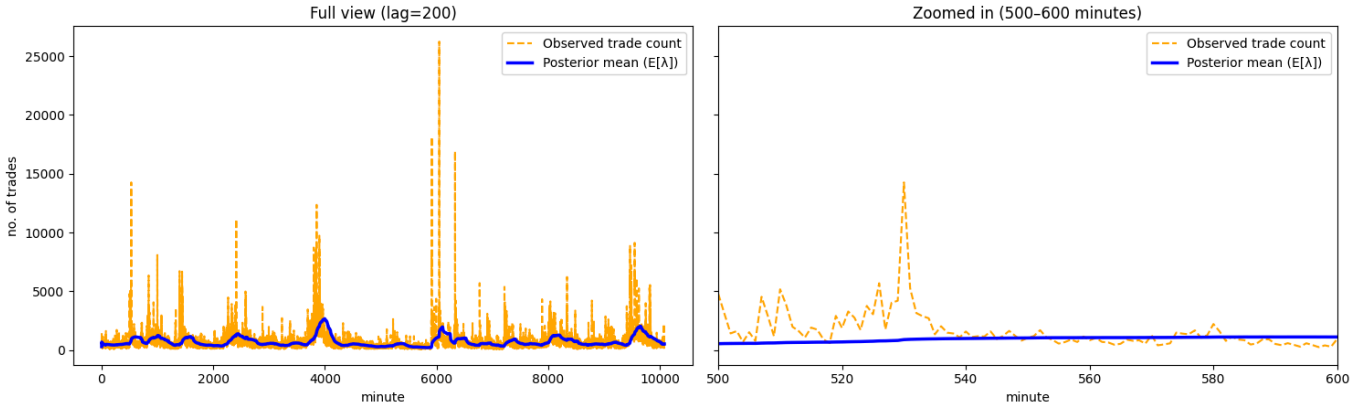




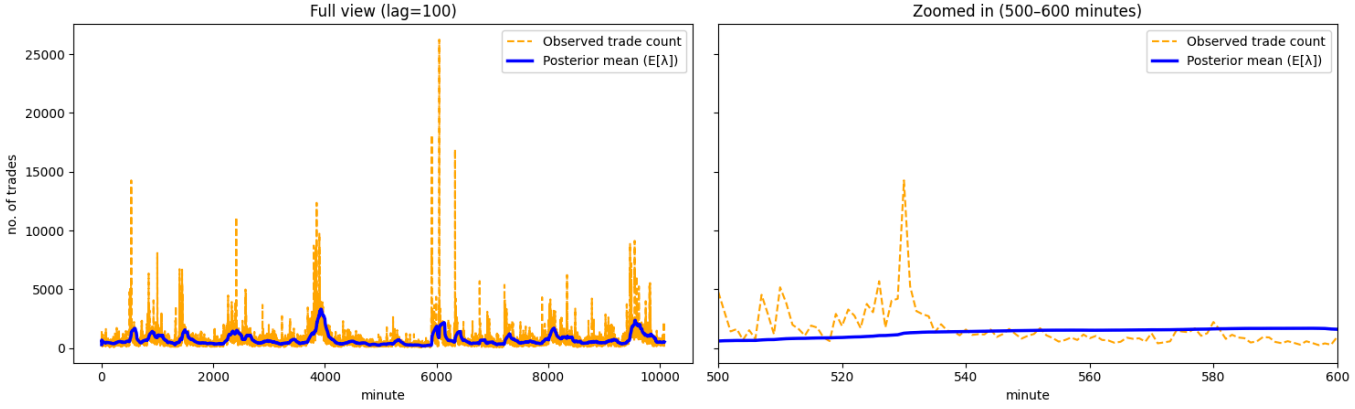
Gamma-Poisson with lag=250



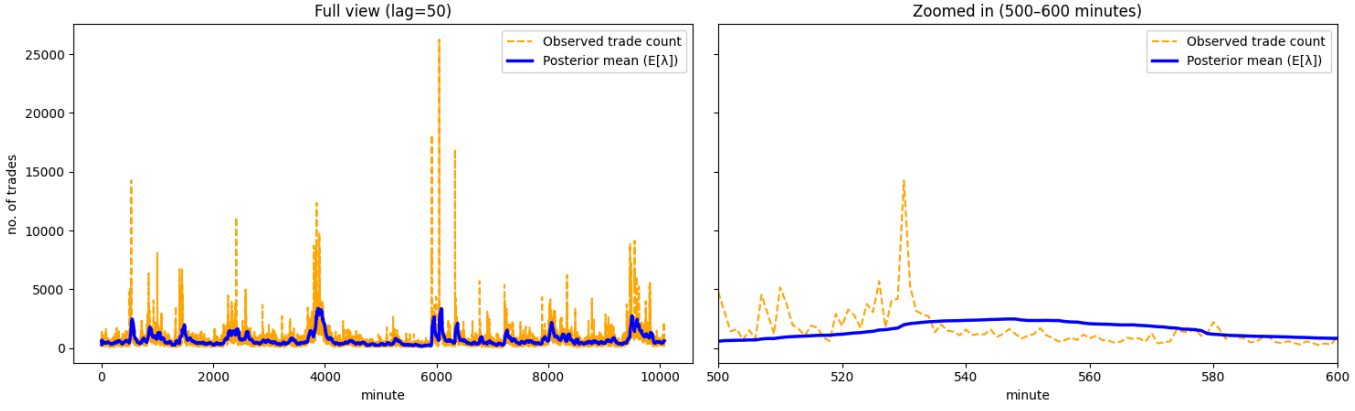
Gamma-Poisson with lag=200

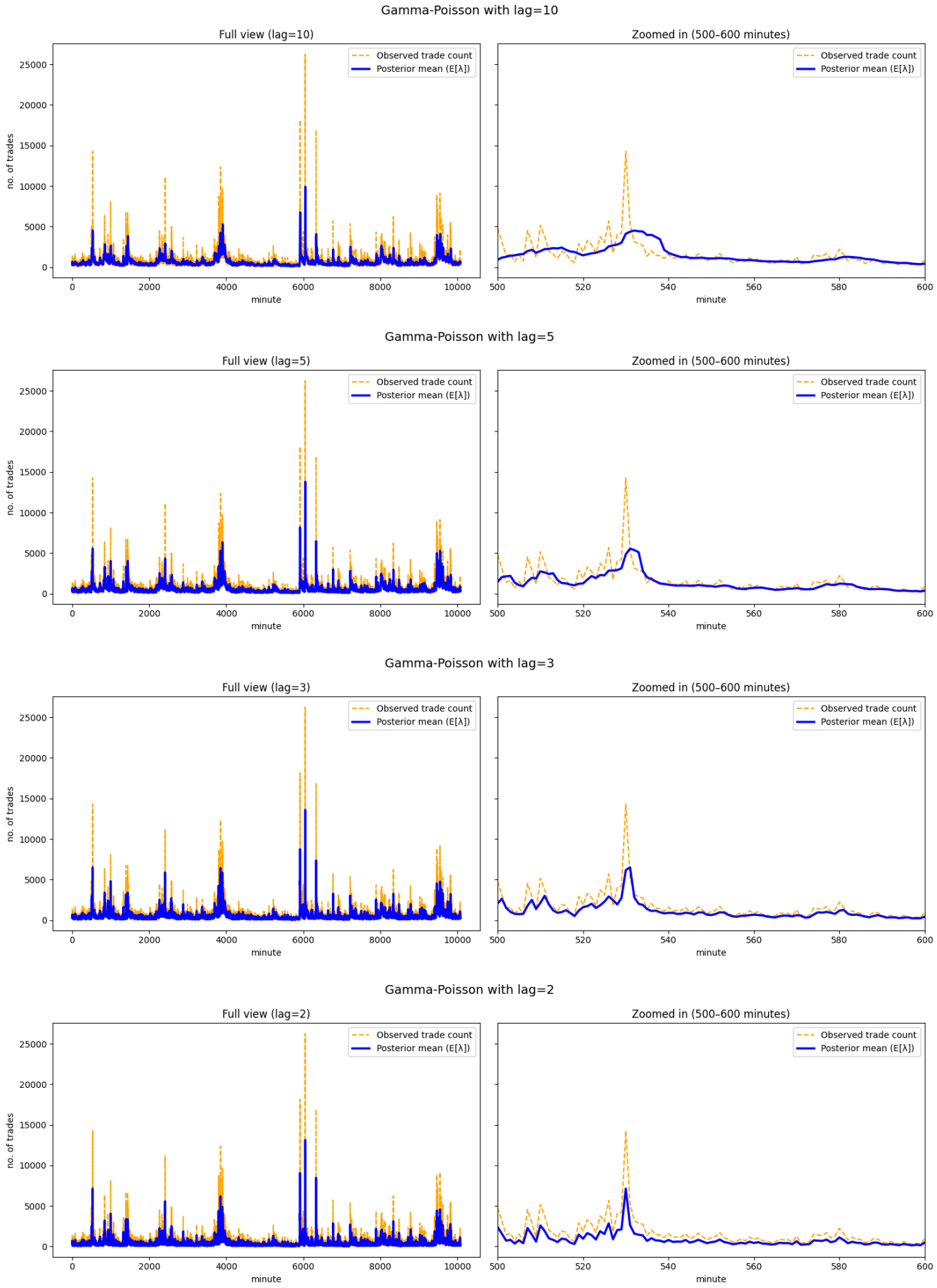


Gamma-Poisson with lag=100



Gamma-Poisson with lag=50





The plots indicate that lower values of lag give much better predictions. This is consistent with our results about the Autocorrelation factor. We also observe that these lower values of lag are much better at predicting steep rises/falls.

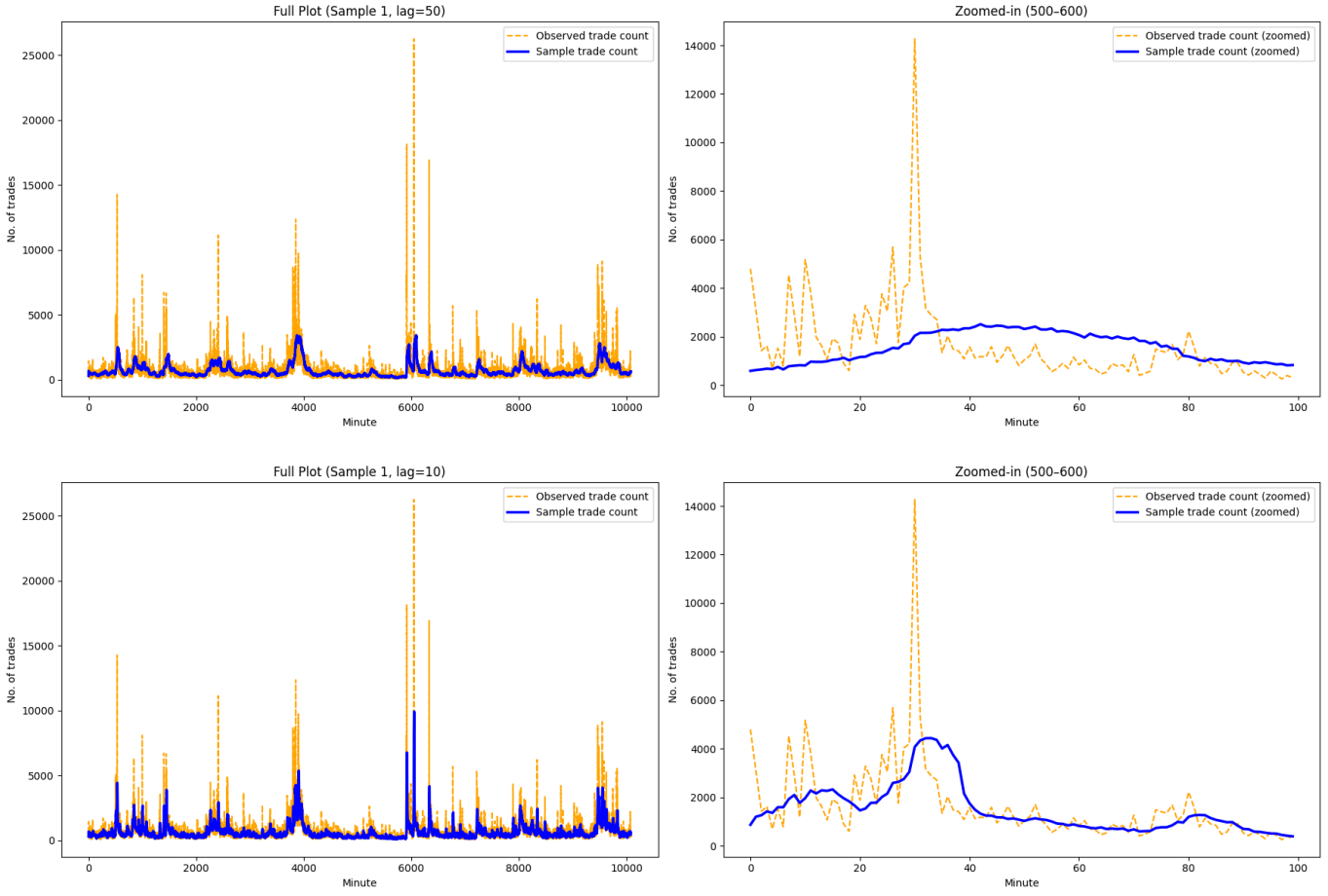
The correspnding code for computing the Posterior Expectation for the Gamma_Poisson model is as follows

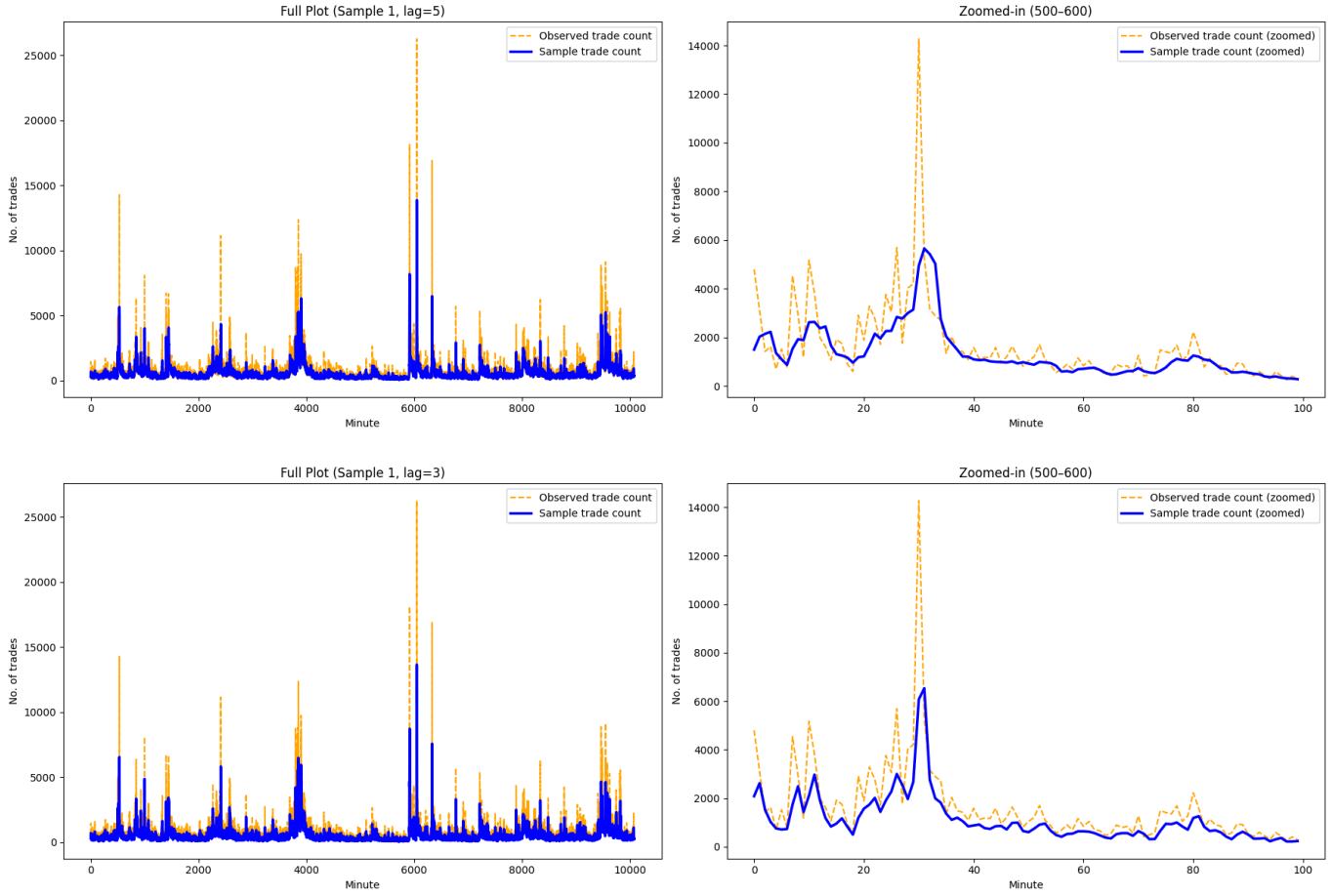
```
def expected_lambda(params, lag, zoom_start=0, zoom_end=200):
```

```
    fig, axs = plt.subplots(1, 2, figsize=(15, 5), sharey=True)

    lmda = [alpha / beta for alpha, beta in zip(params["alpha"][1:], params["beta"][1:])]
    x = range(len(df["trade_count"].values))

    # --- Left: Full plot ---
    axs[0].plot(df["trade_count"].values, color="orange", linestyle="--", label="Observed trade count")
    axs[0].plot(lmda, color="blue", linewidth=2.5, label="Posterior mean (E[λ])")
    axs[0].set_title(f"Full view (lag={lag})")
    axs[0].set_xlabel("minute")
    axs[0].set_ylabel("no. of trades")
    axs[0].legend()

    # --- Right: Zoomed-in plot ---
    axs[1].plot(df["trade_count"].values, color="orange", linestyle="--", label="Observed trade count")
    axs[1].plot(lmda, color="blue", linewidth=2.5, label="Posterior mean (E[λ])")
    axs[1].set_xlim(zoom_start, zoom_end)
    axs[1].set_title(f"Zoomed in ({zoom_start}–{zoom_end} minutes)")
    axs[1].set_xlabel("minute")
    axs[1].legend()

    plt.suptitle(f"Gamma-Poisson with lag={lag}", fontsize=14)
    plt.tight_layout()
    filename = f"GP_ExpMean_lag_{lag}.png"
    file_path = os.path.join("../figures/", filename)
    plt.savefig(file_path, format='png')
    plt.show()

for lag in [float("inf"), 400, 300, 250, 200, 150, 100, 50, 10, 5, 3, 2]:
    gp = Gamma_Poisson(1, 1, lag, df)
    gp.generate_posterior()
    expected_lambda(gp.params, lag, zoom_start=500, zoom_end=600)
```

Posterior Predictive Check

To validate the computed posterior distributions, we performed a Posterior Predictive Check (PPC), a method where we generate new simulated data from the posterior distribution and compare it with the observed data to see how well the model captures real behavior. This analysis was carried out for lag values of 50, 10, 5, and 3, and the resulting plots for these cases are shown below.





Clearly, the **smaller the lag value**, the **better the simulation performance**, as the model responds more quickly to sudden rises and falls in the number of trades, capturing short-term market dynamics more effectively. However, even with lower lag values, the model still **fails to fully capture the steep rises and sharp drops** seen in the observed data, indicating that it struggles to adapt to sudden, large market movements.

The corresponding code to plot the above graphs is give below

```
def gen_samples(params, n):

    samples = []
    for j in range(n):
        sample_j = []
        for i in range(1, len(params["alpha"])):
            lambda_i = np.random.gamma(params["alpha"][i], 1/params["beta"][i])
            x = np.random.poisson(lambda_i)
            sample_j.append(x)

        samples.append(sample_j)

    return samples

def plot_samples(samples, df, lag, zoom_start=None, zoom_end=None):
    i = 1
    for sample in samples:
        fig, axes = plt.subplots(1, 2, figsize=(18, 6))

        # --- Main Plot ---
        axes[0].plot(df["trade_count"].values, color="orange", linestyle="--", label="Observed trade count")
        axes[0].plot(sample, color="blue", linewidth=2.5, label="Sample trade count")
        axes[0].set_xlabel("Minute")
        axes[0].set_ylabel("No. of trades")
        axes[0].set_title(f"Full Plot (Sample {i}, lag={lag})")
        axes[0].legend()

        # --- Zoomed-in Plot ---
        if zoom_start is not None and zoom_end is not None:
            axes[1].plot(df["trade_count"].values[zoom_start:zoom_end], color="orange", linestyle="--", label="Ob
            axes[1].plot(sample[zoom_start:zoom_end], color="blue", linewidth=2.5, label="Sample trade count (zoo
```

```
            axes[1].set_xlabel("Minute")
            axes[1].set_ylabel("No. of trades")
            axes[1].set_title(f"Zoomed-in ({zoom_start}–{zoom_end})")
            axes[1].legend()
        else:
            axes[1].axis("off")

        plt.tight_layout()
        filename = f"GP_PPC_sample_{i}_lag_{lag}.png"
        file_path = os.path.join("../figures/", filename)
        plt.savefig(file_path, format='png')
        plt.show()
        i += 1


for lag in [50, 10, 5, 3]:
    gp = Gamma_Poisson(1, 1, lag, df)
    gp.generate_posterior()
    np.random.seed(42)
    samples = gen_samples(gp.params, 1)
    plot_samples(samples, df, lag, zoom_start=500, zoom_end = 600)
```

Posterior Probability Intervals

One other method to validate our Bayesian Model is through Posterior Probability Intervals. We find an interval (a, b) such that $P(a < y_{new} | y_{past} < b) = \alpha$, where α is the required confidence. In the Gamma-Poisson Distribution, the Posterior Predictive Distribution is given as follows:

$$P(y_{new}|y_{past}) = \int_{\Omega_\lambda} P(y_{new}|\lambda)P(\lambda|y_{past})d\lambda$$

Computing the integral above gives the kernel of a Negative Binomial Random Variable. In fact, we get

$$y_{new}|y_{past} \sim \text{Negative Binomial}(\alpha_{post}, \frac{\beta_{post}}{\beta_{post}+1})$$

where $\lambda|y_{past} \sim \text{Gamma}(\alpha_{post}, \beta_{post})$. We have found $\alpha_{post}, \beta_{post}$ via our Bayesian Analysis above.

In our analysis, we found the 95% Posterior Probability Interval. This interval is written to be $(\tau_{0.025}, \tau_{0.975})$ where τ_x is defined to be such that $P(Z < \tau_x) = x$, where Z follows the above defined Negative Binomial Distribution.

In general, if we want an interval of confidence α (0 < α < 1), we take $(\tau_{\alpha/2}, \tau_{1-\alpha/2})$.

The following results indicate the number of minutes during which the actual trade counts fell outside the Posterior Probability Interval.

Lag	Count	Percentage
inf	9369	92.9372
400	9214	91.3997
300	9158	90.8442
250	9137	90.6358
200	9051	89.7828
150	8988	89.1578
100	8966	88.9396
50	8853	87.8187
10	8619	85.4975

Clearly, the actual trade counts lie outside the Posterior Probability Intervals an extremely large number of times. However, this can be explained by the fact that the trade counts show sudden large spikes/falls, which the posterior distribution is unable to capture precisely. This shows that our model may not be a good fit for this task. Our model fails to capture the variability in the data.

The code used to check how many trade counts lie inside the Posterior Probability Interval is given below. Please check the code related to the Gamma-Poisson class (pasted previously near the beginning of the Gamma-Poisson Model) for information regarding how the intervals were created.

```
def check_ppi_validity(gp, alpha):
```

```
  df = gp.df
  df = df.sort_values(by="timestamp").reset_index(drop=True)
  array = df["trade_count"].tolist()
  cnt = 0
  gp.generate_posterior()
  gp.generate_ppi(alpha)

  for i in range(len(array)):
    if array[i] < gp.ppi[alpha]["lower_bound"][i] or array[i] > gp.ppi[alpha]["upper_bound"][i]:
      cnt += 1

  return cnt

table = []
for lag in ([float("inf"), 400, 300, 250, 200, 150, 100, 50, 10]):
    sub = []
    sub.append(lag)
    gp = Gamma_Poisson(1, 1, lag, df)
    gp.generate_posterior()
    val = check_ppi_validity(gp, 0.95)
    sub.append(val)
    sub.append(100 * val / len(gp.df))
    table.append(sub)

html_table = tabulate(table, headers=["Lag", "Count", "Percentage"], tablefmt="html")
display(HTML(html_table))
```

Log Normal Poisson Modeling of Trade Rate

Following the Gamma Poisson modeling of Trade Rate. Next, we focus on modeling the trade rate using a Poisson likelihood with a Log-Normal prior. Unlike the Gamma prior, the Log-Normal is non-conjugate, allowing greater flexibility to capture skewed or heavy-tailed variations in trade activity that may occur in highly volatile markets.

We start with the prior

$$\begin{aligned} X &\sim \text{Poisson}(\lambda) \\ \log \lambda &\sim \mathcal{N}\left(\mu, \sigma^2\right) \end{aligned}$$

For which the kernel of posterior distribution is

$$p(\lambda \mid X) \propto \lambda^{x-1} \exp\left\{ -\lambda - \frac{(\log \lambda - \mu)^2}{2\sigma^2} \right\}$$

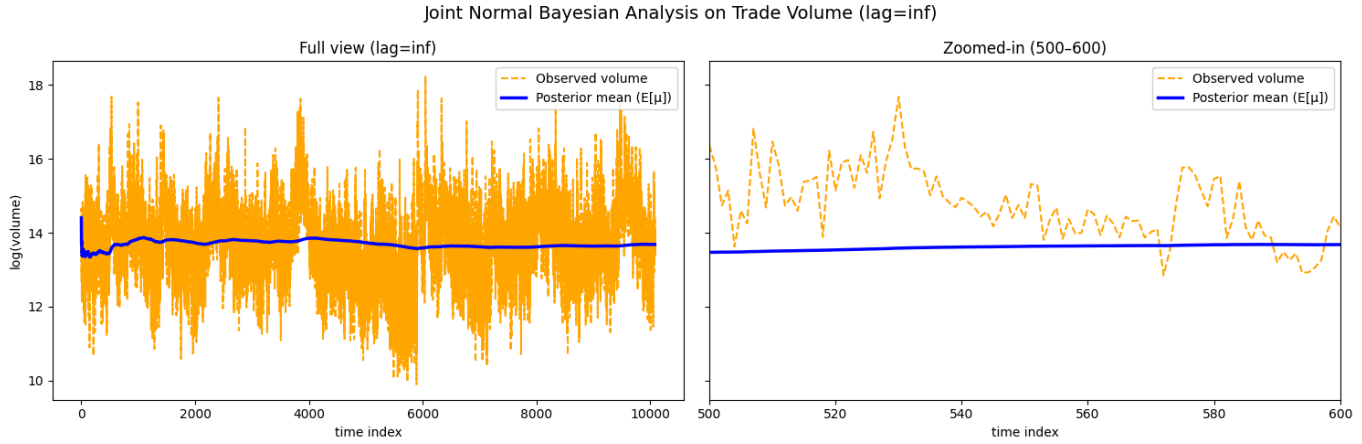
Since the LogNormal–Poisson posterior is non-conjugate, it cannot be integrated analytically. Therefore, further analysis is performed using Monte Carlo simulation. The following code defines the `log_normal_poisson` class, which computes the posterior mean and Posterior Probability Intervals (PPIs) for use in subsequent analysis.

```
class log_normal_poisson:
    def __init__(self, mean, variance, lag, df: pd.DataFrame):
        if variance < 0:
            raise ValueError("Variance must be positive")
        if lag <= 0:
            raise ValueError("Lag must be positive")
        self.mean = mean
        self.variance = variance
        self.lag = lag
        self.df = df
        self.posterior_means = [math.exp(mean + variance / 2.0)]
        self.ppi = {}

    def generate_posterior_means(self, sample_count=5000, device='cuda'):
        df = self.df.sort_values(by="timestamp").reset_index(drop=True)
        counts = torch.tensor(df["trade_count"].values, dtype=torch.float32, device=device)
        n = len(counts)

        posterior_means = []

        mean = torch.tensor(self.mean, dtype=torch.float32, device=device)
        std = torch.tensor(math.sqrt(self.variance), dtype=torch.float32, device=device)
```

```
        for i in range(n):
            means = torch.distributions.LogNormal(mean, std).sample((sample_count,))
            start_idx = max(0, i - self.lag + 1)
            past_counts = counts[start_idx:i]
            if len(past_counts) == 0:
                posterior_means.append(torch.mean(means).item())
                continue

            log_factorial = torch.lgamma(past_counts + 1)
            log_pmf_matrix = past_counts[None, :] * torch.log(means[:, None]) - means[:, None] - log_factorial[No

            log_weights = torch.sum(log_pmf_matrix, dim=1)
            log_weights = log_weights - torch.max(log_weights)
            weights = torch.exp(log_weights)
            weights = weights / torch.sum(weights)

            posterior_mean = torch.sum(weights * means)
            posterior_means.append(posterior_mean.item())

        self.posterior_means = posterior_means

    def generate_ppis(self, percent, sample_count=5000, device='cuda'):
        ppis = {"lower_bound":[], "upper_bound":[]}
        percent = 1 - percent
        df = self.df.sort_values(by="timestamp").reset_index(drop=True)
        counts = torch.tensor(df["trade_count"].values, dtype=torch.float32, device=device)
        n = len(counts)

        mean = torch.tensor(self.mean, dtype=torch.float32, device=device)
        std = torch.tensor(math.sqrt(self.variance), dtype=torch.float32, device=device)

        for i in range(n):
            means = torch.distributions.LogNormal(mean, std).sample((sample_count,))

            start_idx = max(0, i - self.lag + 1)
            past_counts = counts[max(0, i - self.lag + 1):i]

            log_factorial = torch.lgamma(past_counts + 1)
            log_pmf_matrix = past_counts[None, :] * torch.log(means[:, None]) - means[:, None] - log_factorial[No

            log_weights = torch.sum(log_pmf_matrix, dim=1)

            log_weights = log_weights - torch.max(log_weights)
            weights = torch.exp(log_weights)
            weights = weights / torch.sum(weights)

            sample_indices = torch.multinomial(weights, num_samples=sample_count, replacement=True)
            sample_lambdas = means[sample_indices]

            random_samples = torch.poisson(sample_lambdas)

            lower_bound = random_samples.quantile(percent / 2).item()
            upper_bound = random_samples.quantile((1 - percent / 2)).item()
            ppis["lower_bound"].append(lower_bound)
            ppis["upper_bound"].append(upper_bound)



        self.ppi[1 - percent] = ppis
```

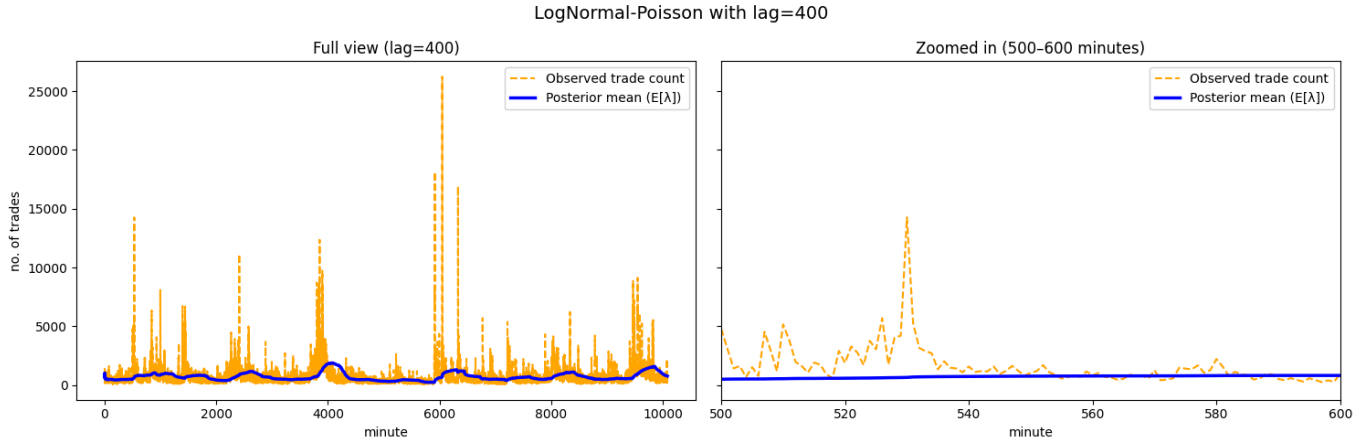
Posterior Predictive Mean Plots

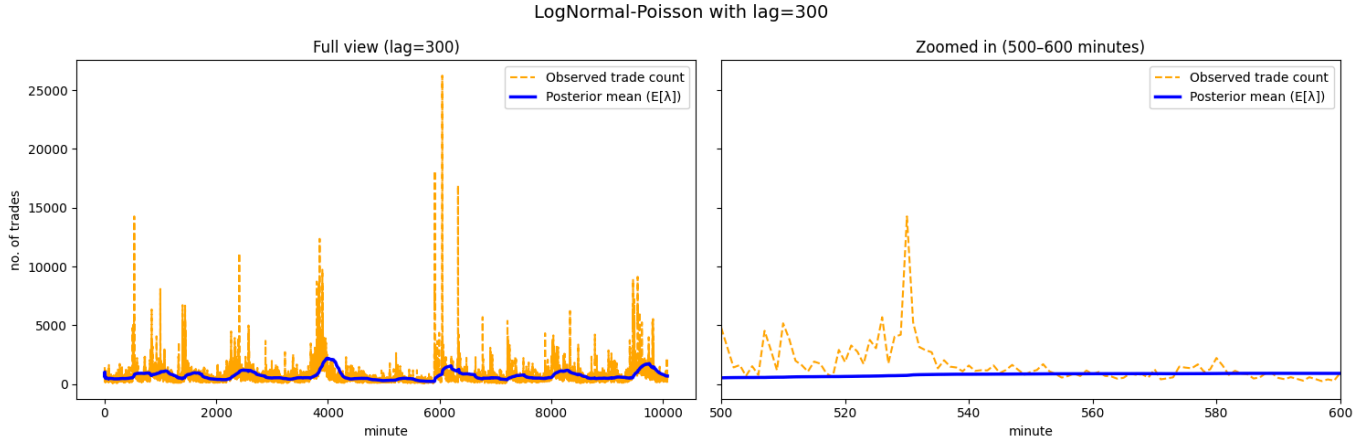
Following the approach used for Gamma–Poisson trade rate modeling, we examine how well the LogNormal–Poisson model captures observed trade volumes. To do this, we plot the expected posterior mean of λ against the observed trade counts per minute. The prior was set with μ equal to the population mean and σ^2 equal to the population variance, which we found experimentally to provide the best fit. The plot of the posterior mean for infinite lag is shown below:

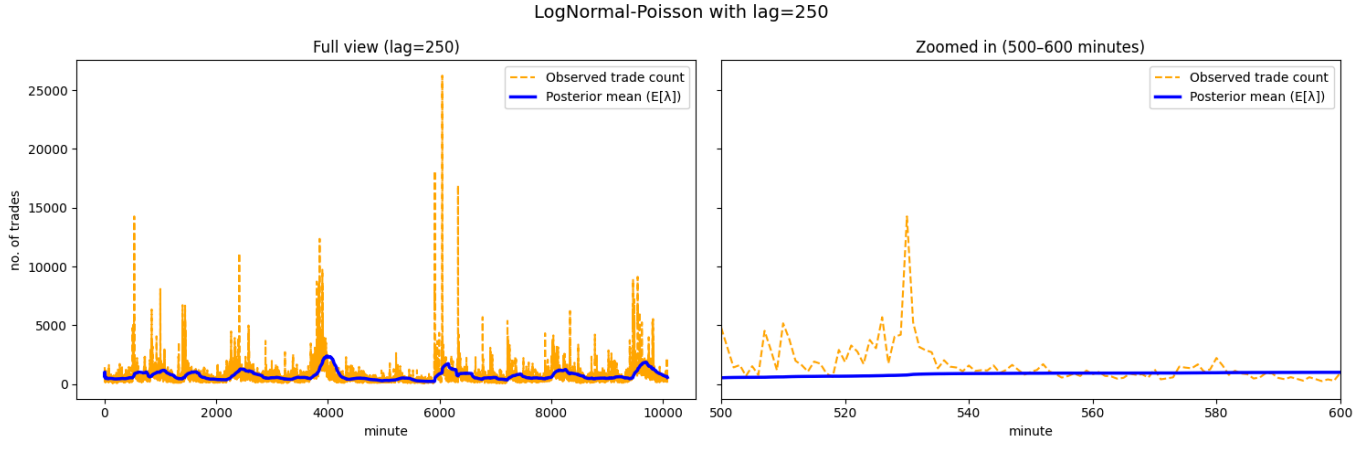


Similar to the Gamma–Poisson case, this plot responds slowly to sudden market changes and does not immediately reflect sharp rises or falls in trade volume. As the analysis of the data also revealed strong autocorrelation at short lags, suggesting that recent trading activity has a significant influence on current volumes.

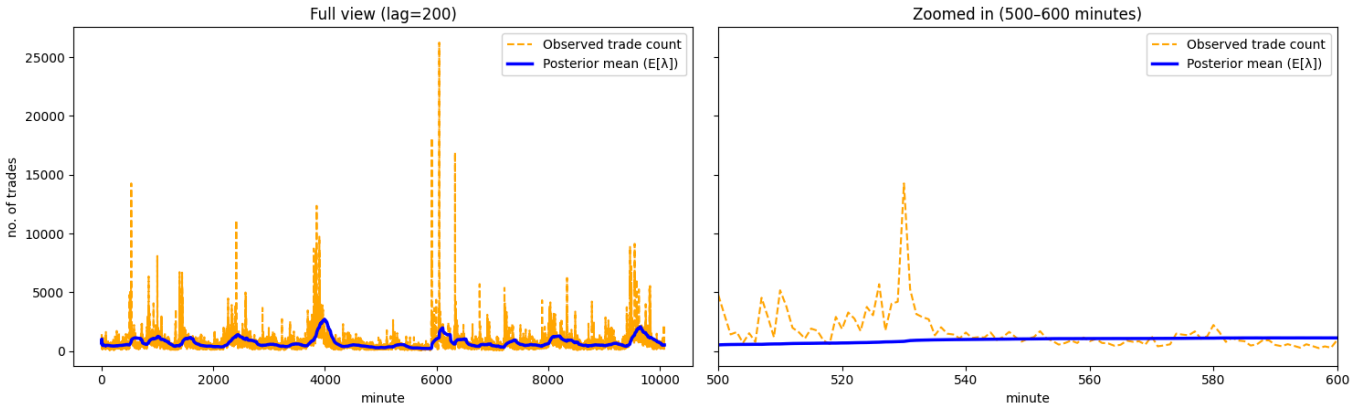
Hence, we have performed this analysis for several lag values: $lag = 400, 300, 250, 200, 100, 50, 10, 5, 3, 2$. The resulting plots show that smaller lag values improve the model's responsiveness, better capturing short-term fluctuations in trade rate.



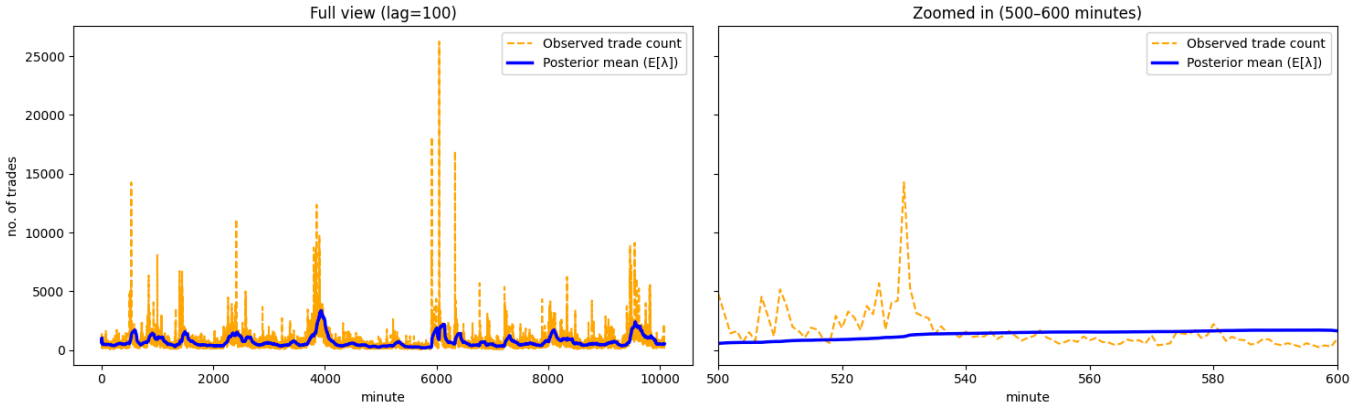




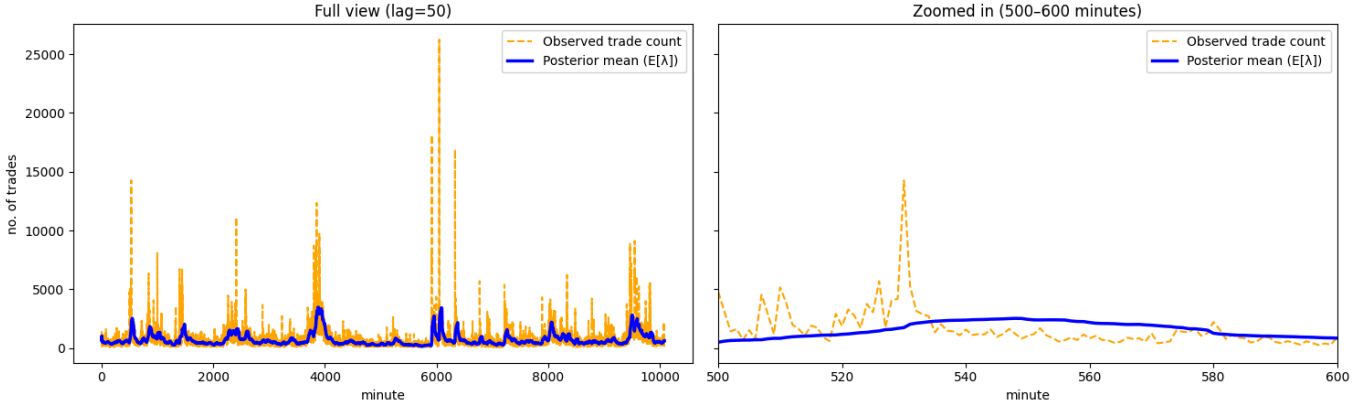
LogNormal-Poisson with lag=200



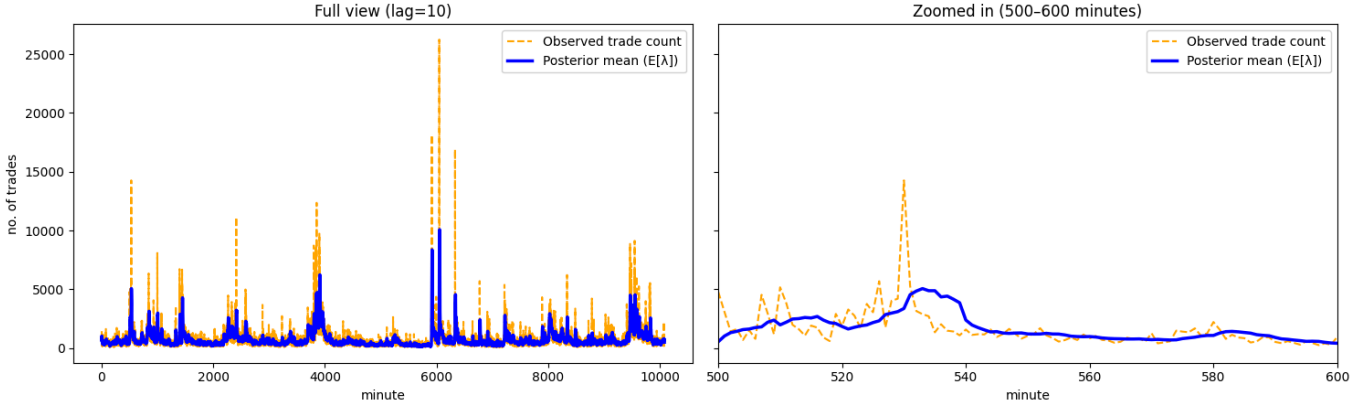
LogNormal-Poisson with lag=100

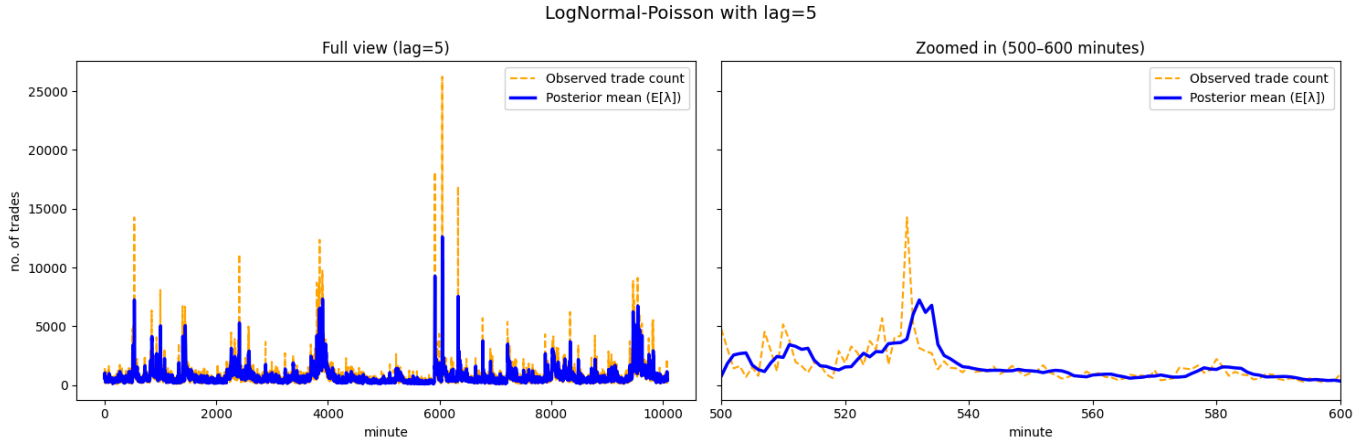


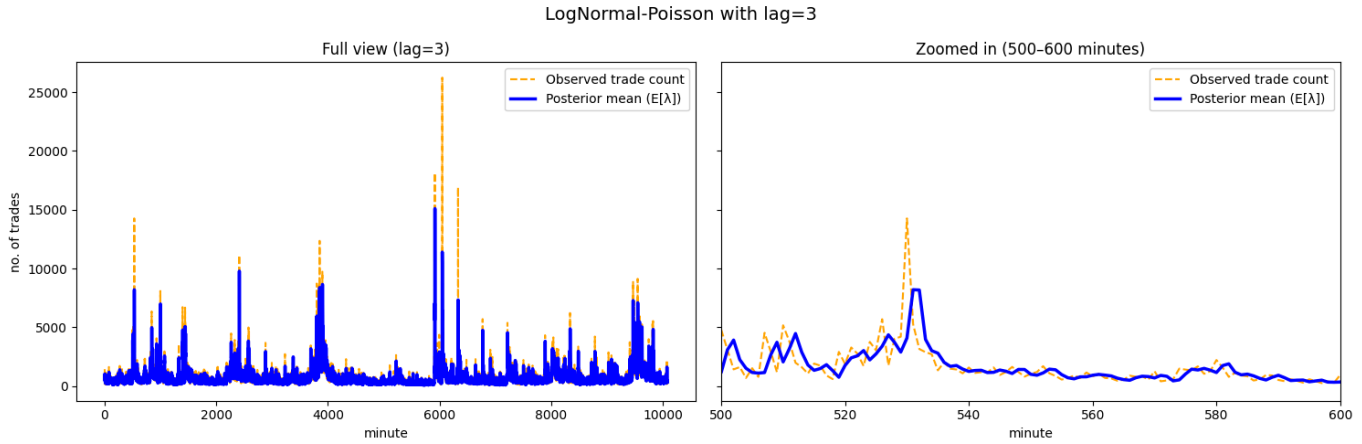
LogNormal-Poisson with lag=50

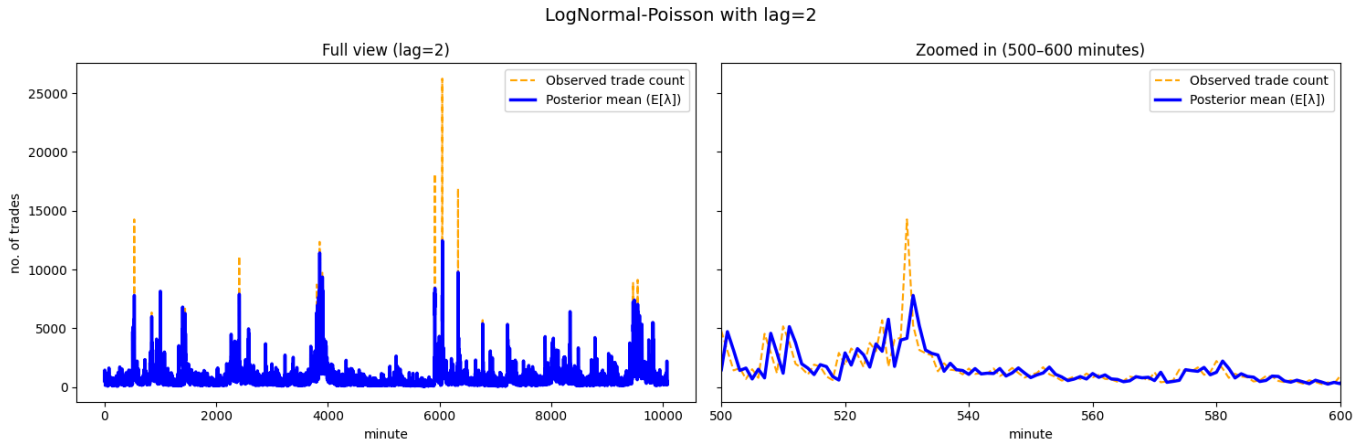


LogNormal-Poisson with lag=10









We computed this posterior mean via Monte Carlo simulation
as show in the previous code snippet. And the code for the ploting these plots is as follows

```
def expected_lambda(params, lag, zoom_start=0, zoom_end=200):
    fig, axs = plt.subplots(1, 2, figsize=(15, 5), sharey=True)

    lmda = params
    x = range(len(df["trade_count"].values))

    # --- Left: Full plot ---
    axs[0].plot(df["trade_count"].values, color="orange", linestyle="--", label="Observed trade count")
    axs[0].plot(lmda, color="blue", linewidth=2.5, label="Posterior mean (E[λ])")
    axs[0].set_title(f"Full view (lag={lag})")
    axs[0].set_xlabel("minute")
    axs[0].set_ylabel("no. of trades")
    axs[0].legend()

    # --- Right: Zoomed-in plot ---
    axs[1].plot(df["trade_count"].values, color="orange", linestyle="--", label="Observed trade count")
    axs[1].plot(lmda, color="blue", linewidth=2.5, label="Posterior mean (E[λ])")
    axs[1].set_xlim(zoom_start, zoom_end)
    axs[1].set_title(f"Zoomed in ({zoom_start}–{zoom_end} minutes)")
```

```
    axs[1].set_xlabel("minute")
    axs[1].legend()

    plt.suptitle(f"LogNormal-Poisson with lag={lag}", fontsize=14)
    plt.tight_layout()
    filename = f"LP_ExpMean_lag_{lag}.png"
    file_path = os.path.join("../figures/", filename)
    plt.savefig(file_path, format='png')
    plt.show()

for lag in [float('inf'), 400, 300, 250, 200, 100, 50, 10, 5, 3, 2]:
    prior_mean = np.log(df["trade_count"]).mean()
    prior_variance = np.log(df["trade_count"]).var()

    lnp = log_normal_poisson(prior_mean, prior_variance, lag, df)
    lnp.generate_posterior_means()
    expected_lambda(lnp.posterior_means, lag, zoom_start=500, zoom_end=600)
```

Posterior Probability Interval

To validate our LogNormal-Poisson model, lets compute the 95% PPI i.e. we find an interval (a, b) such that $P(a < X_{new}|X_{old} < b) = \alpha$, where α is the required confidence. For LogNormal-Poisson model, the Posterior Predictive Distribution is given as follows:

$$P(X_{new}|X_{old}) = \int_{\Omega_\lambda} P(X_{new}|\lambda)P(\lambda|X_{old})d\lambda$$

Computing the above integral cannot be solved analytically due to the non-conjugacy of the LogNormal prior. Hence, we use Monte Carlo simulation to approximate it.

The table below shows the number / percentage of observed trade rate points outside the Posterior Probability Interval (PPI) for different lag values, indicating how well the model captures variability.

Lag	Count	Percentage
∞	9369	92.94%
400	9217	91.43%
300	9145	90.72%
250	9120	90.47%
200	9054	89.81%
150	8988	89.16%
100	8967	88.95%
50	8881	88.10%
10	8689	86.19%
5	8515	84.47%
3	8329	82.62%
2	8197	81.31%

The corresponding code for the above computation is present in the `log_nomral_poisson` class and in the following code

```
def check_ppi_validity(model, alpha):
  df = model.df
  df = df.sort_values(by="timestamp").reset_index(drop=True)
  array = df["trade_count"].tolist()
  cnt = 0
  model.generate_ppis(alpha)

  for i in range(len(array)):
    if array[i] < model.ppi[alpha]["lower_bound"][i] or array[i] > model.ppi[alpha]["upper_bound"][i]:
      cnt += 1

  return cnt

table = []
prior_mean = np.log(df["trade_count"]).mean()
prior_variance = np.log(df["trade_count"]).var()
for lag in ([float("inf"), 400, 300, 250, 200, 150, 100, 50, 10, 5, 3, 2]):
```

```
    sub = []
    sub.append(lag)
    lnp = log_normal_poisson(prior_mean, prior_variance, lag, df)
    lnp.generate_posterior_means()
    val = check_ppi_validity(lnp, 0.95)
    sub.append(val)
    sub.append(100 * val / len(lnp.df))
    table.append(sub)

html_table = tabulate(table, headers=["Lag", "Count", "Percentage"], tablefmt="html")
display(HTML(html_table))
```

Gamma Poisson Vs Log Normal

We explored the Gamma–Poisson and Lognormal–Poisson models to see how well each could capture the variation in trade rate data. The comparison was based on how many observed trade counts fell outside the Posterior Probability Interval (PPI) for different lag values.

The results show that both models struggle to capture the observed variability, as a large majority of data points lie outside their respective PPIs across all lags. This suggests that neither model provides an adequate fit to the underlying trade rate dynamics.

However, Lognormal–Poisson model does a bit better when the lag is small (like 5 or 3), but the improvement is very small. It still doesn't match the real data well. The Gamma–Poisson model also struggles at very small lags (like 3 or 2), where it becomes unstable because the prior starts affecting the results too much.

In short, both models do a poor job of capturing how trade rates actually change. Most data points still fall outside their predicted range. The Lognormal–Poisson reacts a bit better to quick market changes, but overall, neither model fits the data well.

Bayesian Modeling of Trade Volume

In this analysis, we aim to model the **logarithm of trade cost per minute**, i.e., the total cash flow in 1-minute intervals. Let V_t denote the **total trade cost** in minute t. Since trade volumes are typically **positively skewed** and can vary over several orders of magnitude, we model the **log-transformed volume**:

$$X_t = \log(V_t)$$

Using the Joint Normal Bayesian Model with unknown mean and variance.

Motivation for Log Transfomation:

The log transformation is applied to trade volumes to improve model fit and support more reliable Bayesian inference.

It helps stabilize variance and reduce skewness by compressing extreme values. This is because the log transformation converts multiplicative effects into additive ones.

Overall, this makes the data more suitable for Gaussian-based modeling and leads to more stable and interpretable posterior estimates of trade volume dynamics.

Joint Normal Model (Unknown μ and σ^2)

We use the observed log-volumes $X_t = x_1, \ldots, x_n$ over a period of t minutes to estimate the unknown mean μ and variance σ^2. As we have conjugate priors, we can compute the posterior distributions using simple formula based on the sample mean $(\overline{x})$ and variance (s^2). This lets us update our prior about the log transformed trade volume.

We have modeled **Joint normal Bayesain Model** for X_t, with **unknown mean** (μ) and **unknown variance** (σ^2):

$$X_t \sim \mathcal{N}(\mu, \sigma^2)$$

Priors

We start with prior assumption that

$$\mu|\sigma \sim \mathcal{N}\left(\mu_0, \frac{\sigma^2}{k_0}\right), \quad \sigma^2 \sim \text{Inverse-Gamma}\left(\frac{r_0}{2}, \frac{r_0}{2}\sigma_0^2\right)$$

where $\mu_0, k_0, \sigma_0, r_0$ are pre-defined hyperparameters

We know that the Normal distribution serves as a conjugate prior for mean (μ) of a Normal Distribution where variance was known. We also know that the Inverse Gamma Distribution serves as a conjugate prior for the case of unknown variance (σ^2) of a Normal Distribution where mean is known.

It is this quite natural to combine both these distributions to perform a Joint Bayesian Analysis for the Normal Distribution. It helps simplify posterior calculations a lot.

Posterior Inference

Now, the **posterior distribution** of (μ, σ^2) can be computed analytically as

$$\mu|\sigma \sim \mathcal{N}\left(\mu_n, \frac{\sigma^2}{k_n}\right), \quad \sigma^2 \sim \text{Inverse-Gamma}\left(\frac{r_n}{2}, \frac{r_n}{2}\sigma_n^2\right)$$

And the corresponding posterior updates are:

$$\begin{aligned}
k_n &= k_0 + n \\
\mu_n &= \frac{k_0}{k_n}\mu_0 + \frac{n}{k_n}\overline{x} \\
r_n &= r_0 + n \\
\sigma_n^2 &= \frac{1}{r_n}\left(r_0\sigma_0^2 + (n-1)s^2 + \frac{k_0}{k_n}(\overline{x}^2 - \mu_0^2)\right)
\end{aligned}$$

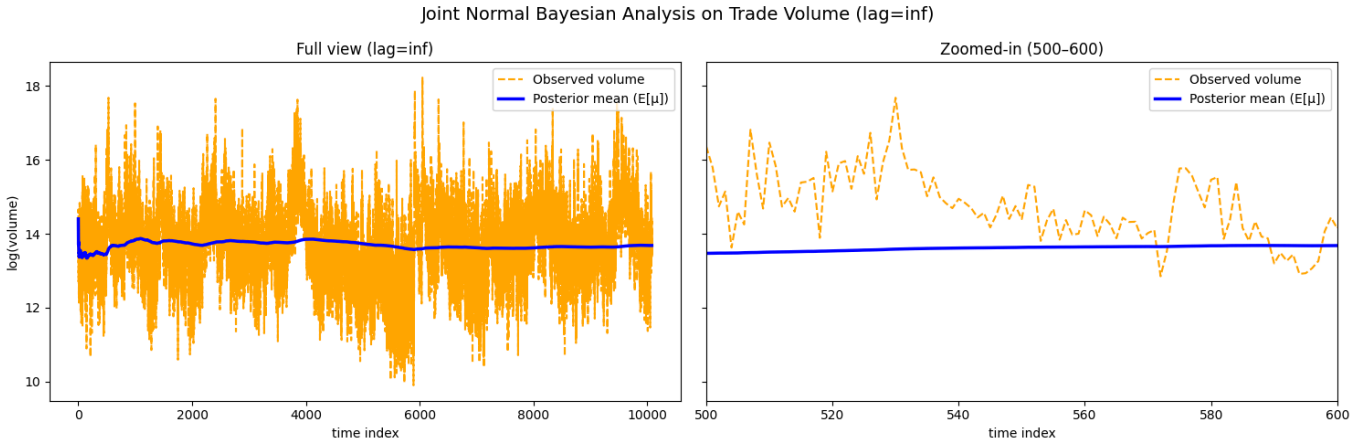
These posterior distributions give us updated estimates for the mean and variance of the log-trade volumes. The posterior updates above summarize the analytical computation, and the code for posterior distribution computation is as follows:

```
class Joint_Normal:
    def __init__(self, mu, k, sigma2, r, lag, df: pd.DataFrame):
        if min(sigma2, k ,r) <= 0:
            raise ValueError("sigma2, k, r must be > 0")
        if lag <=0:
            raise ValueError("lag must be a Natural number")

        self.lag = (lag)
        self.df = df

        self.params = {"mu":[mu], "k": [k], "sigma2":[sigma2], "r": [r]}
        self.ppi = {}

    def generate_posterior(self):
        df = self.df
        df = df.sort_values(by="timestamp").reset_index(drop=True)
        array = np.log(df["cost"].tolist())

        for i in range(len(array)):
            n = 0
            # sum_volume =0
            # sum2_volume =0

            window = array[max(0, i - self.lag):i]

            if len(window) < 2:
                continue

            n = len(window)

            # sample mean and sample variance calculation
            ybar = np.mean(window)
            s2 = np.var(window, ddof=1)

            # prior parameters
            mu0, k0, sigma20, r0 = self.params["mu"][0], self.params["k"][0], self.params["sigma2"][0], self.para

            # posterior updates
            kn = k0 + n
            rn = r0 + n

            mun = (k0 * mu0 + n * ybar) / kn
            sigma2n = (r0 * sigma20  + (n-1) * s2 + k0 * n * ((ybar - mu0)**2) / kn ) / rn

            self.params["mu"].append(mun)
            self.params["k"].append(kn)
            self.params["sigma2"].append(sigma2n)
            self.params["r"].append(rn)
```

```
def generate_ppi(self, percent):
    ppis = {"lower_bound": [], "upper_bound": []}
    alpha = 1 - percent

    for i in range(1, len(self.params["mu"])):
        mu_n = self.params["mu"][i]
        k_n = self.params["k"][i]
        sigma2_n = self.params["sigma2"][i]
        r_n = self.params["r"][i]

        if np.isnan(sigma2_n) or np.isnan(k_n):
            print(f"NaN encountered: sigma2_n={sigma2_n}, k_n={k_n}, i={i}")
        elif sigma2_n < 0:
            print(f"Negative sigma2_n={sigma2_n}, i={i}")
        elif k_n == 0:
            print(f"k_n==0 at i={i}")

        # degrees of freedom for posterior predictive distribution
        df = 2 * r_n
        # scale for predictive distribution
        scale = (sigma2_n * (1 + 1/k_n)) ** 0.5

        # credible interval bounds using t-distribution
        lower = scipy.stats.t.ppf(alpha / 2, df, loc=mu_n, scale=scale)
        upper = scipy.stats.t.ppf(1 - alpha / 2, df, loc=mu_n, scale=scale)

        ppis["lower_bound"].append(lower)
        ppis["upper_bound"].append(upper)

    self.ppi[percent] = ppis
```

Posterior Predictive Mean Plots

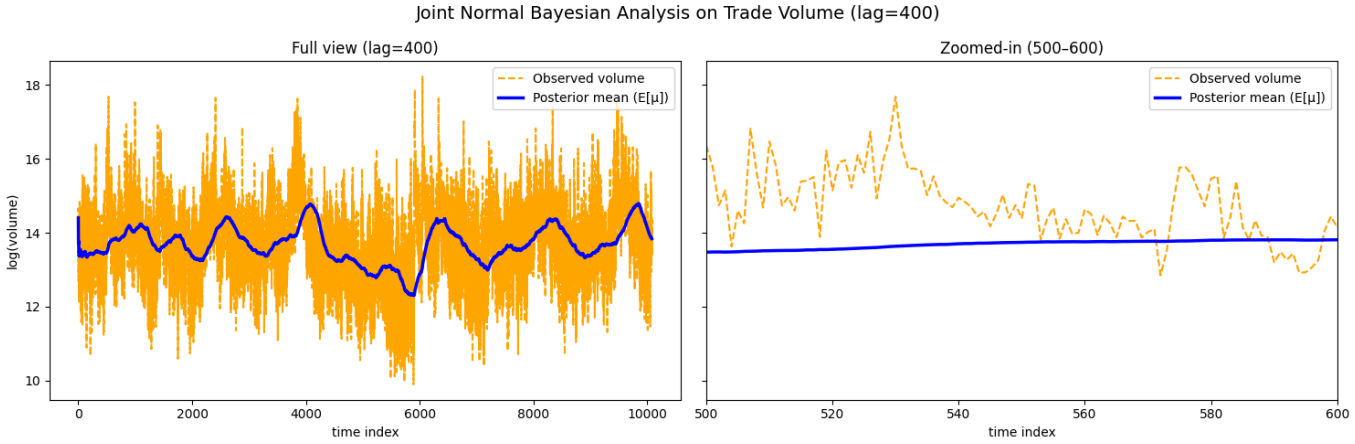
Similar to the case of Gamma-Poisson Modeling for Trade Rate we have ploted the expectation of the posterior means of μ against the observed trade volume per minute data to see how well our Bayesian model mimics the observed data. We have started with the prior that $\mu_0 = 14, k_0 = 1, \sigma_0^2 = 4, r_0 = 1$, which is something we have found expermentally to provide the best fit. And as for the plot of posterior mean obtained with infinite lag is as follows.

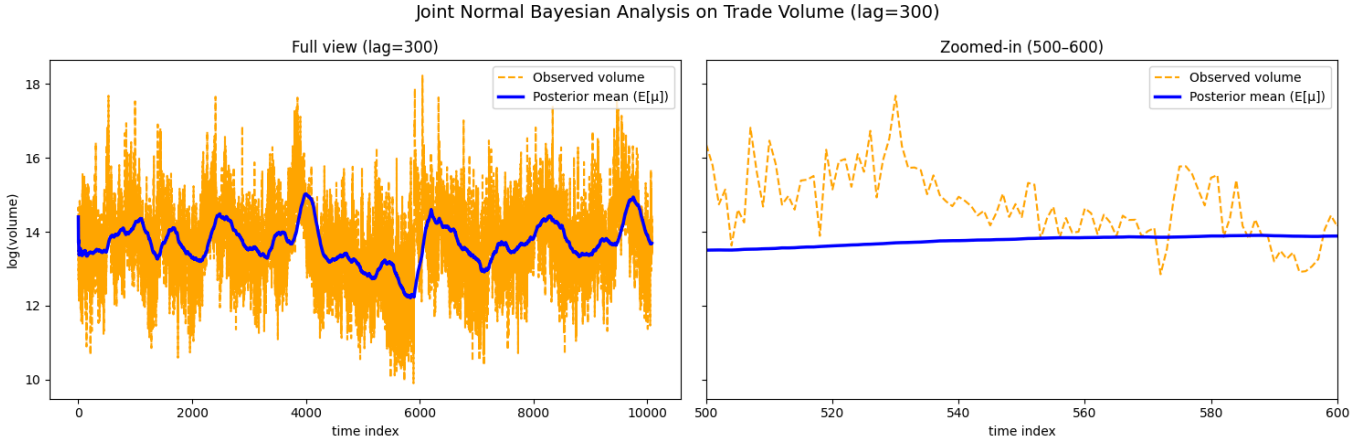


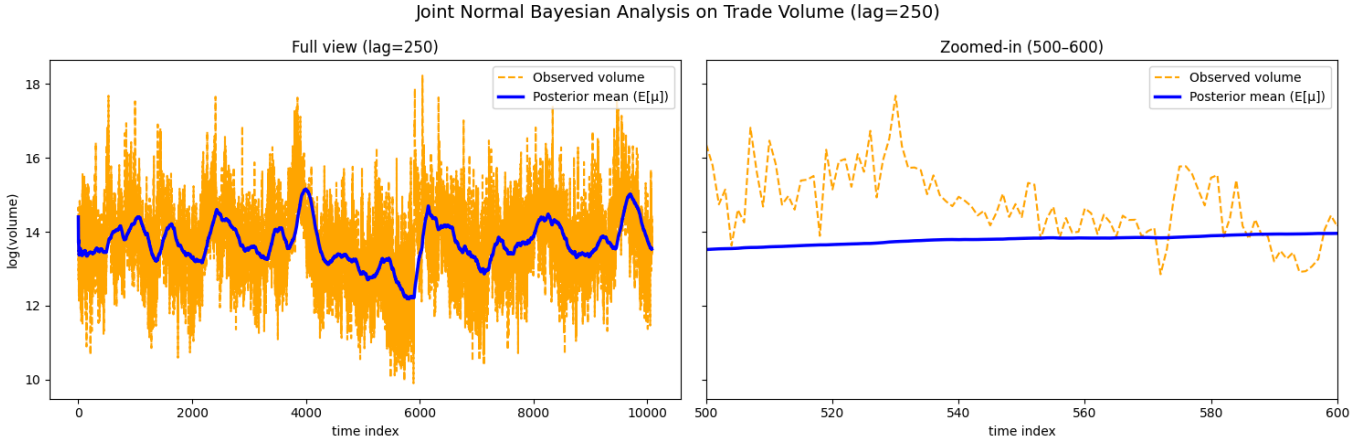
As with the Gamma–Poisson case, this plot also is relatively insensitive to sudden market changes, failing to immediately capture sharp rises or drops in trade volume. Our analysis also revealed high autocorrelation at lower lags, indicating that recent trading activity strongly influences current volume.

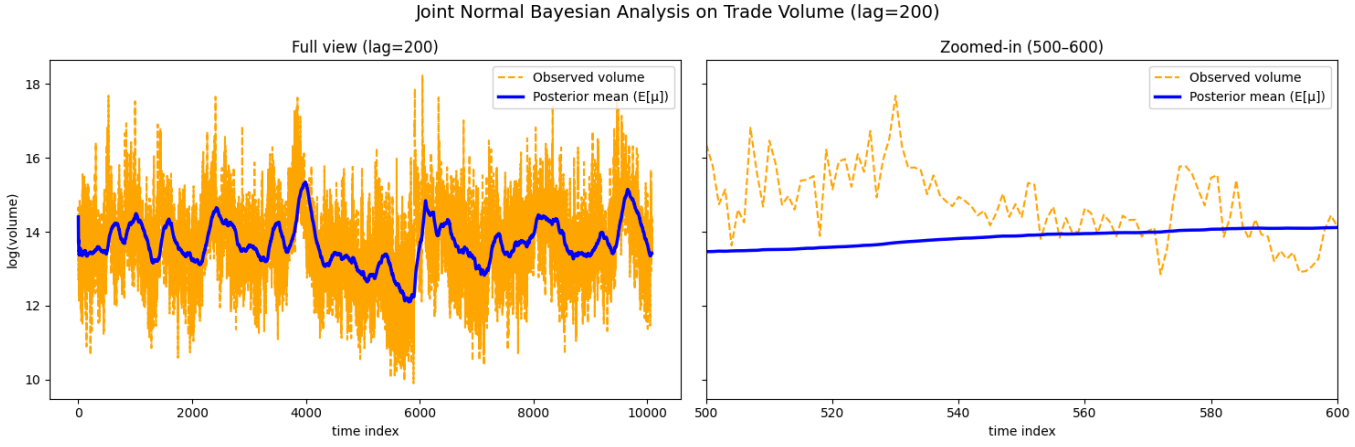
To address this, we adopt the same informative prior, updating it at each time t using data from $t - \text{lag}$ to $t - 1$. In other words, the last lag - 1 observations are used to update the prior dynamically.

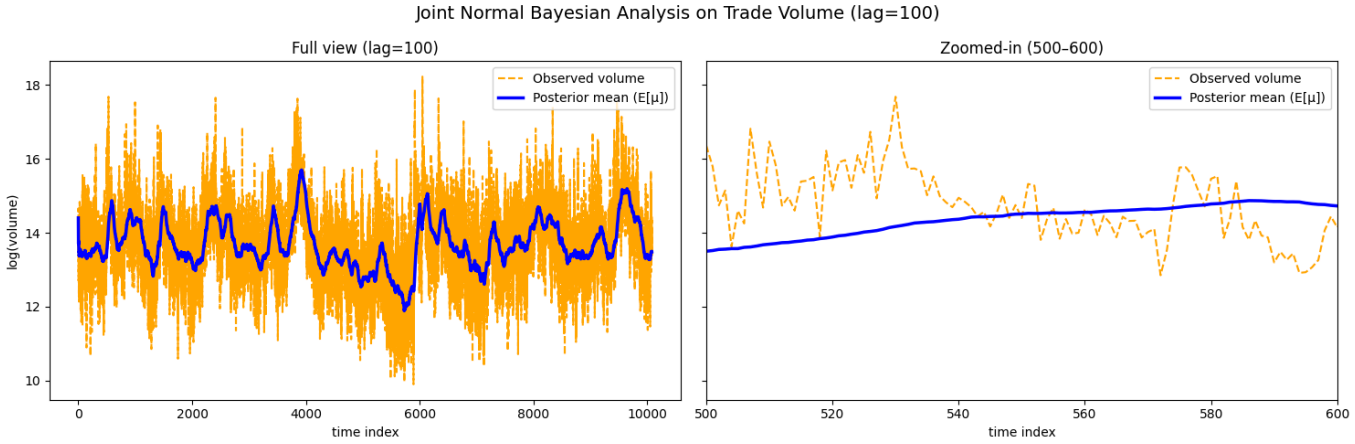
We performed this analysis for several lag values: $lag = 400, 300, 250, 200, 150, 100, 50, 30, 10, 5, 3, 2$. The corresponding plots illustrate how smaller lag values make the model more responsive to recent market activity.

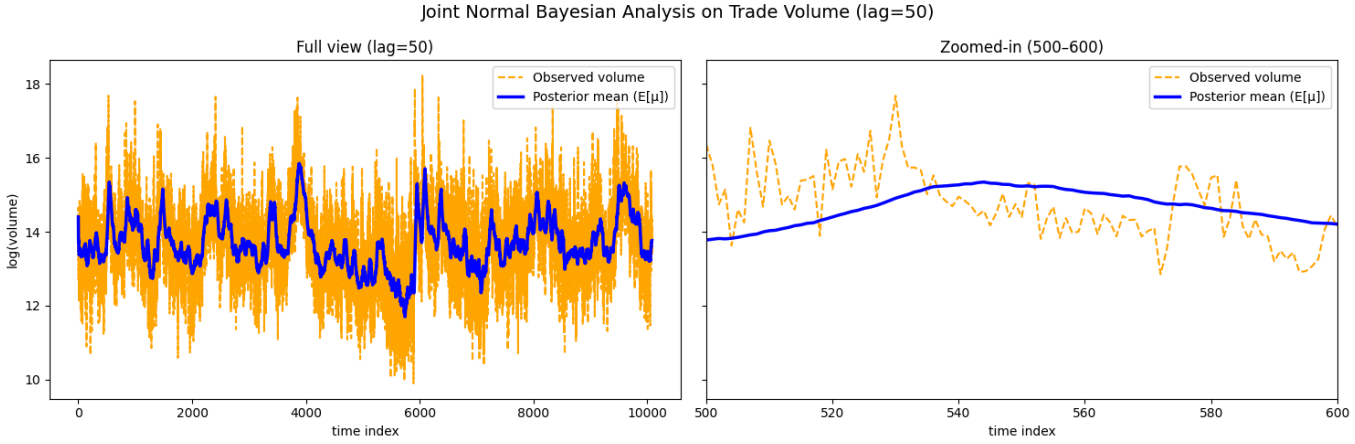

Joint Normal Bayesian Analysis on Trade Volume (lag=400)
Full view (lag=400)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index

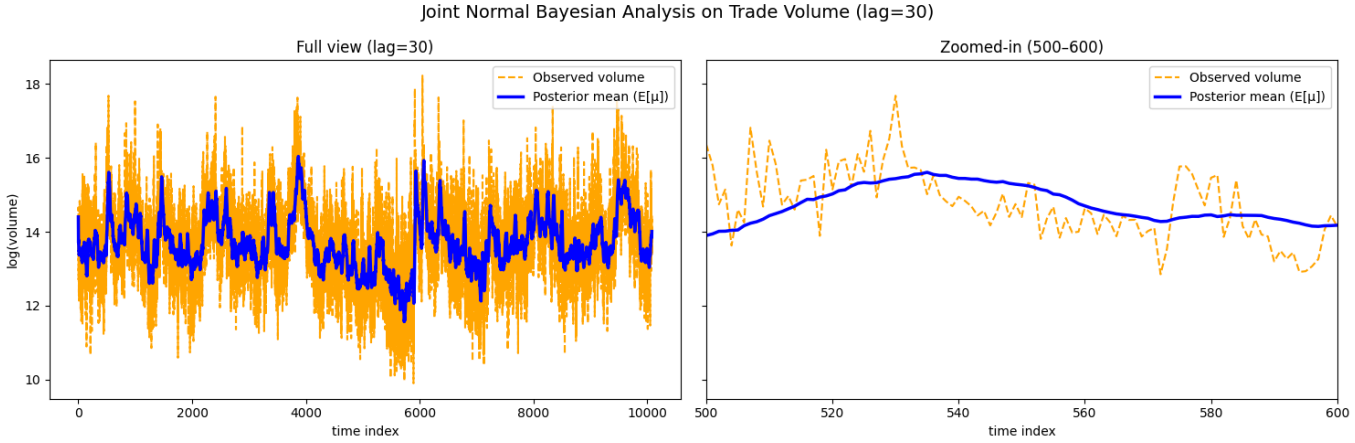

Joint Normal Bayesian Analysis on Trade Volume (lag=300)
Full view (lag=300)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index

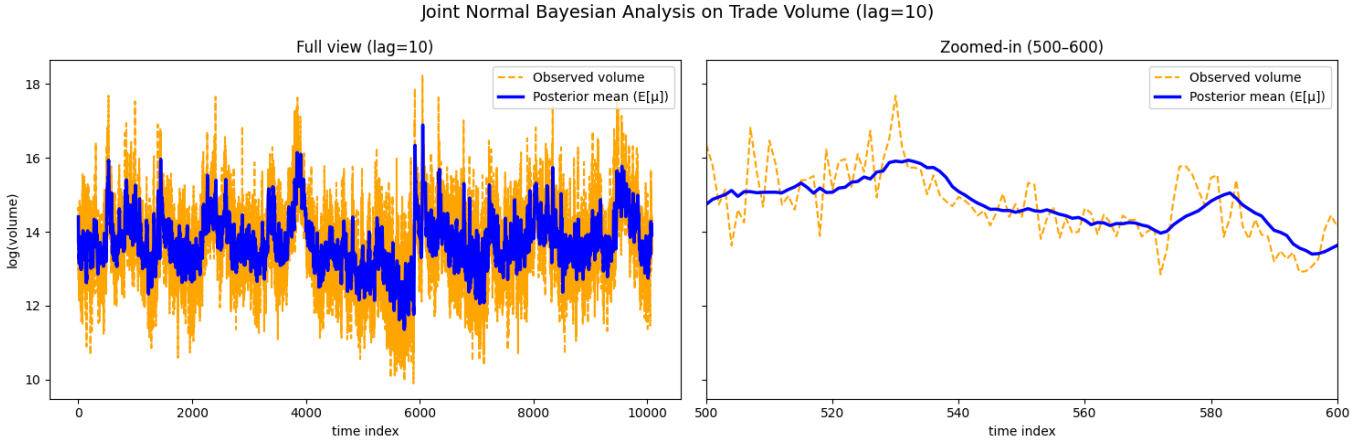

Joint Normal Bayesian Analysis on Trade Volume (lag=250)
Full view (lag=250)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index

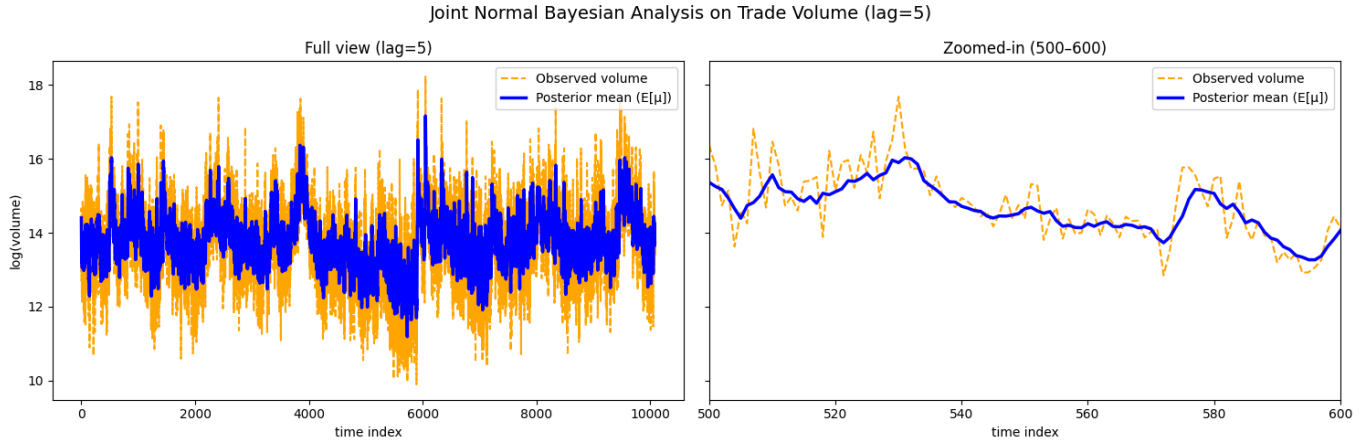

Joint Normal Bayesian Analysis on Trade Volume (lag=200)
Full view (lag=200)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index

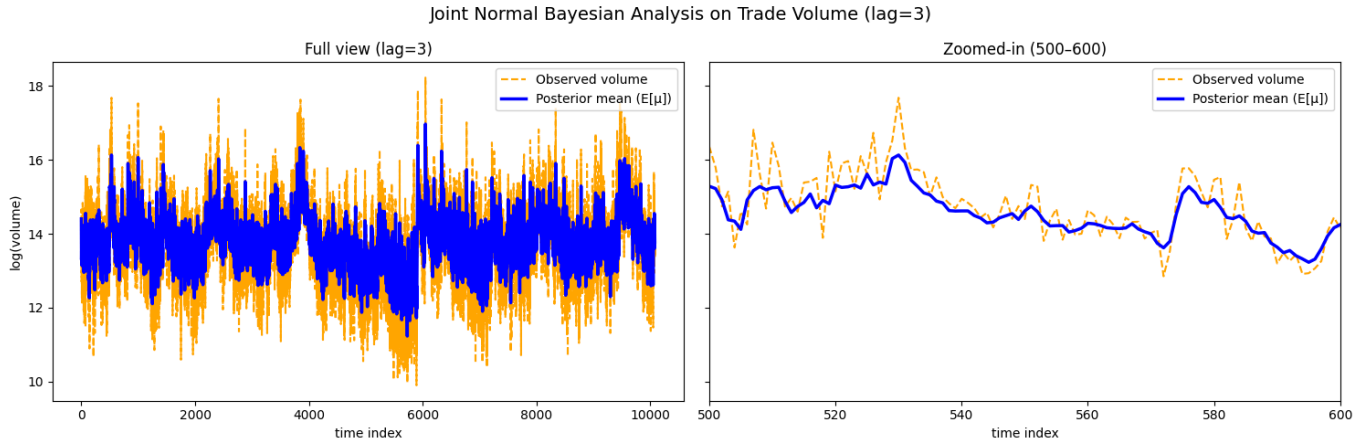

Joint Normal Bayesian Analysis on Trade Volume (lag=100)
Full view (lag=100)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index

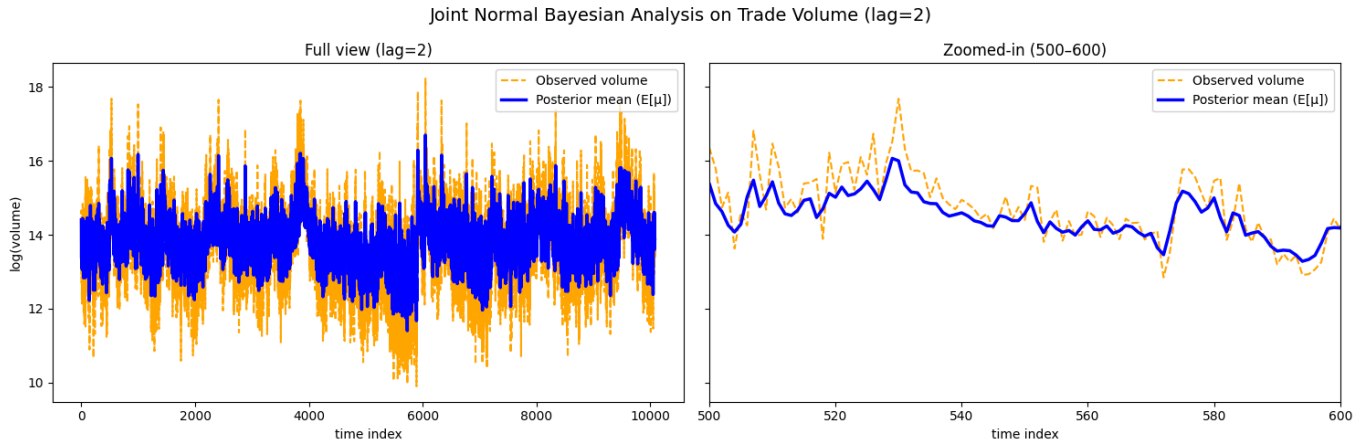

Joint Normal Bayesian Analysis on Trade Volume (lag=50)
Full view (lag=50)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index


Joint Normal Bayesian Analysis on Trade Volume (lag=30)
Full view (lag=30)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index


Joint Normal Bayesian Analysis on Trade Volume (lag=10)
Full view (lag=10)
Zoomed-in (500–600)
Observed volume
Posterior mean (E[μ])
log(volume)
time index







These plots show that smaller lag values lead to more accurate predictions, which is consistent with the observed autocorrelation in the data analysis. In particular, lower lags are better at capturing steep rises and sharp drops in trade volume, making the model more responsive to recent market fluctuations.

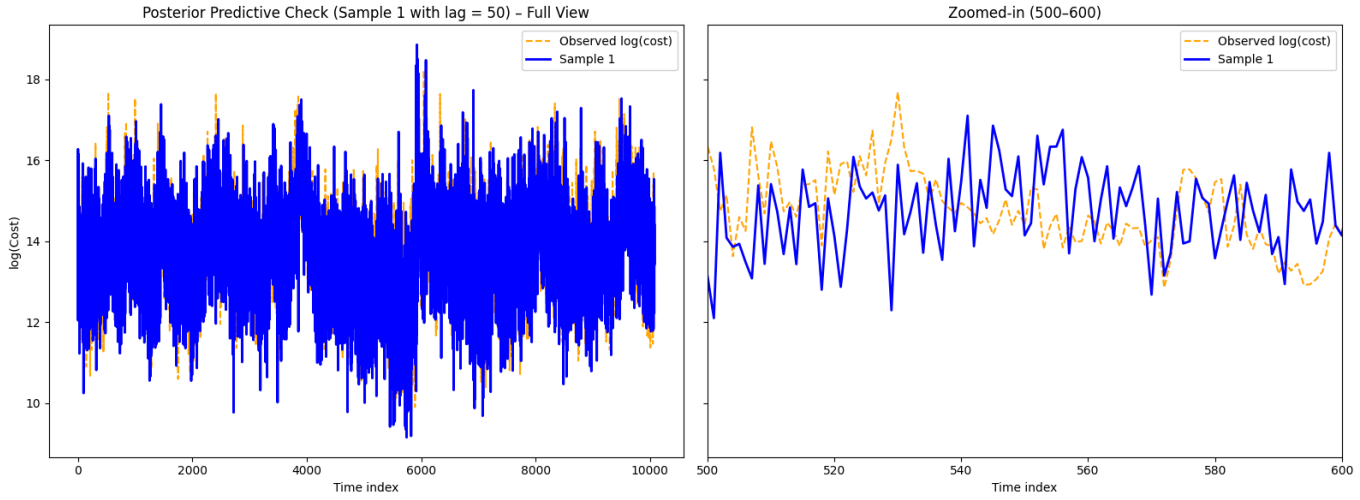
The code for computing the posterior distributions for the Gamma–Poisson model is as follows:

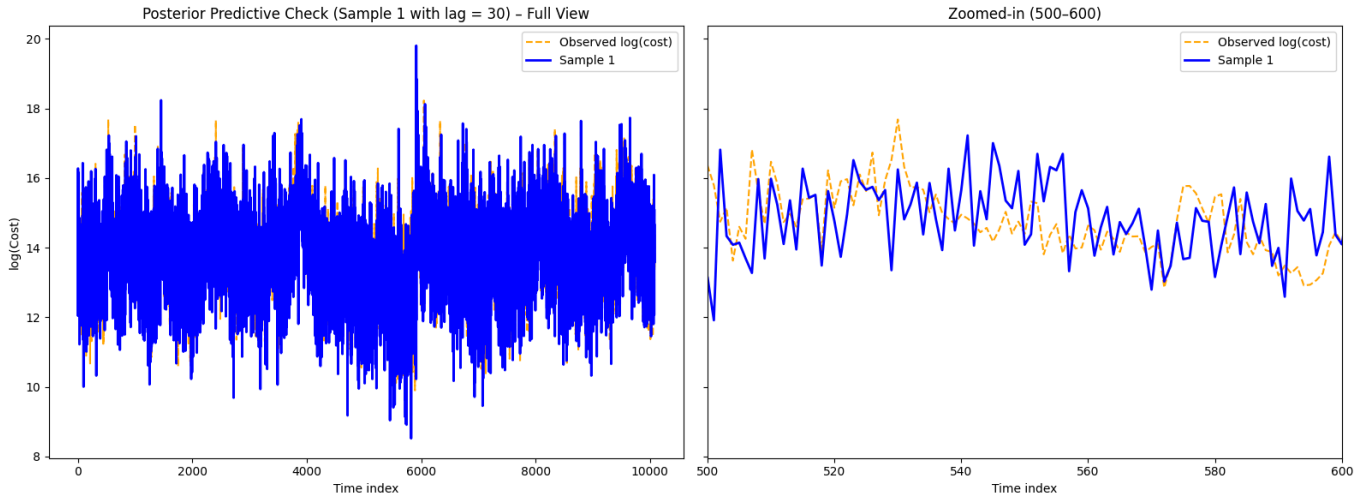
```
def expected_mu(model, lag, zoom_start=0, zoom_end=200, log_scale=True):
    df = model.df
    mu_values = model.params["mu"][1:]
    sigma2_values = model.params["sigma2"][1:]
    k_values = model.params["k"][1:]
    r_values = model.params["r"][1:]


    observed = np.log(df["cost"].values) if log_scale else df["cost"].values

    if not log_scale:
        mu_values = np.exp(mu_values)
        if lower is not None:
            lower = np.exp(lower)
            upper = np.exp(upper)
```

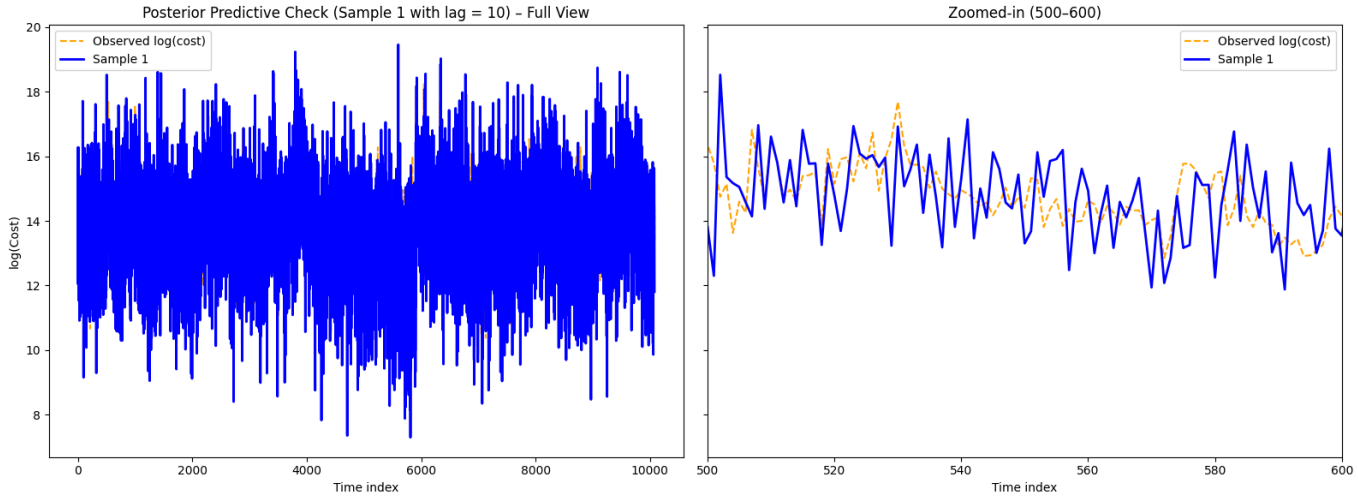
```
    fig, axs = plt.subplots(1, 2, figsize=(15, 5), sharey=True)
    x = np.arange(len(observed))

    # --- Left: Full view ---
    axs[0].plot(observed, color="orange", linestyle="--", label="Observed volume")
    axs[0].plot(mu_values, color="blue", linewidth=2.5, label="Posterior mean (E[μ])")
    axs[0].set_title(f"Full view (lag={lag})")
    axs[0].set_xlabel("time index")
    axs[0].set_ylabel("log(volume)" if log_scale else "trade volume")
    axs[0].legend()

    # --- Right: Zoomed-in view ---
    axs[1].plot(observed, color="orange", linestyle="--", label="Observed volume")
    axs[1].plot(mu_values, color="blue", linewidth=2.5, label="Posterior mean (E[μ])")
    axs[1].set_xlim(zoom_start, zoom_end)
    axs[1].set_title(f"Zoomed-in ({zoom_start}–{zoom_end})")
    axs[1].set_xlabel("time index")
    axs[1].legend()

    plt.suptitle(f"Joint Normal Bayesian Analysis on Trade Volume (lag={lag})", fontsize=14)
    plt.tight_layout()
    filename = f"JN_ExpMean_lag_{lag}.png"
    file_path = os.path.join("../figures/", filename)
    plt.savefig(file_path, format='png')
    plt.show()


for lag in [float("inf"), 400, 300, 250, 200, 100, 50, 30, 10, 5, 3, 2]:
    jn = Joint_Normal(mu=14, k=1, sigma2=4, r=1, lag=lag, df=df)
    jn.generate_posterior()
    # jn.generate_ppi(0.95)
    expected_mu(jn, lag, zoom_start=500, zoom_end=600, log_scale=True)
```

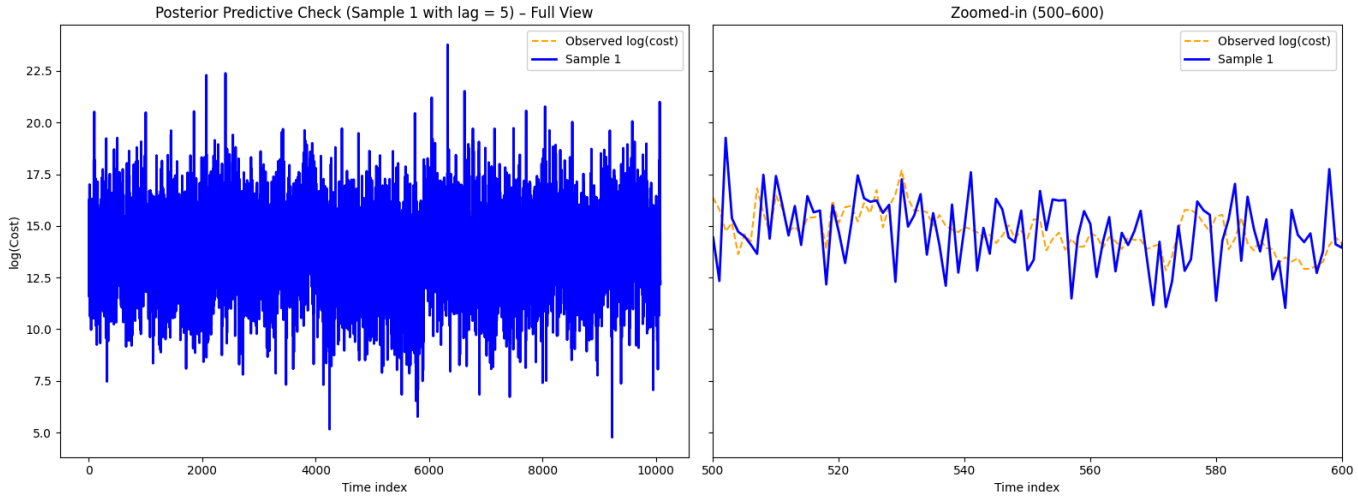
Posterior Predictive Check

The validation of the posterior distribution has been done by simulating the posterior distribution with lag values 50, 30, 10, 5 and the corresponding plots for it are









Clearly, lower lag values provide a better fit. Note that, unlike the Gamma–Poisson case, we did not plot results for $lag = 3$, as the posterior in this case was heavily influenced by the prior and did not reflect the observed data accurately.

The corresponding code is as follows:

```
def plot_samples_joint_normal(samples, df, lag, zoom_start=None, zoom_end=None):
    observed = np.log(df["cost"].values)

    for i, sample in enumerate(samples, start=1):
        fig, axes = plt.subplots(1, 2, figsize=(16, 6), sharey=True)

        # --- Main plot ---
        axes[0].plot(observed, color="orange", linestyle="--", label="Observed log(cost)")
        axes[0].plot(sample, color="blue", linewidth=2.0, label=f"Sample {i}")
        axes[0].set_title(f"Posterior Predictive Check (Sample {i} with lag = {lag}) – Full View")
        axes[0].set_xlabel("Time index")
```

```
        axes[0].set_ylabel("log(Cost)")
        axes[0].legend()

        # --- Zoomed-in plot ---
        if zoom_start is not None and zoom_end is not None:
            axes[1].plot(observed, color="orange", linestyle="--", label="Observed log(cost)")
            axes[1].plot(sample, color="blue", linewidth=2.0, label=f"Sample {i}")
            axes[1].set_xlim(zoom_start, zoom_end)
            axes[1].set_title(f"Zoomed-in ({zoom_start}–{zoom_end})")
            axes[1].set_xlabel("Time index")
            axes[1].legend()
        else:
            axes[1].axis("off")

        plt.tight_layout()
        filename = f"JN_PPC_sample_{i}_lag_{lag}.png"
        file_path = os.path.join("../figures/", filename)
        plt.savefig(file_path, format='png')
        plt.show()

for lag in [50, 30, 10, 5]:
    jn = Joint_Normal(mu=14, k=1, sigma2=10, r=1, lag=lag, df=df)
    jn.generate_posterior()
    np.random.seed(42)
    samples = gen_samples_joint_normal(jn.params, 1)
    plot_samples_joint_normal(samples, jn.df, lag, zoom_start = 500, zoom_end = 600)
```

Posterior Probability Intervals

Finally for the epilog of our Joint Normal Analysis of Log volume we compute the Posterior Probability Intervals. We find an interval (a, b) such that $P(a < X_{new}|X_{old} < b) = \alpha$, where α is the required confidence. In the Joint Normal Distribution, the Posterior Predictive Distribution is given as follows:

$$P(X_{new}|X_{old}) = \int_{\Omega_{\sigma^2}} \int_{\Omega_\mu} P(X_{new}|\mu, \sigma^2) P(\mu, \sigma^2|X_{old}) d\mu \, d\sigma^2$$

Computing the integral above gives the **posterior predictive distribution** of a new log-volume observation as

$$X_{\text{new}} \mid X_{\text{old}} \sim t_{r_n}\Big(\mu_n, \sigma_n \sqrt{1 + \frac{1}{k_n}}\Big)$$

where $(\mu_n, \sigma_n^2, k_n, r_n)$ are the posterior parameters computed from the observed data (X_{old}) as described above. This Student's t-distribution accounts for **uncertainty in both the mean and variance** of the log-transformed trade volumes.

Similar to Gamma-Poisson in this analysis too, we found the 95% Posterior Probability Interval. This interval is written to be $(\tau_{0.025}, \tau_{0.975})$ where τ_x is defined to be such that $P(Z < \tau_x) = x$, where Z follows the above defined $t-$Distribution.

In general, if we want an interval of confidence α (0 < α < 1), we take $(\tau_{\alpha/2}, \tau_{1-\alpha/2})$.

The table below shows the number / percentage of observed log-volume points outside the Posterior Probability Interval (PPI) for different lag values, indicating how well the model captures variability.

Lag	Count Outside	Percentage Outside
inf	665	6.60%
400	711	7.05%
300	680	6.75%
250	653	6.48%
200	624	6.19%
150	579	5.74%
100	519	5.15%
50	414	4.11%
10	39	0.39%

The corresponding code for this computations is as follows

```
def check_ppi_validity(model, alpha):
```

```
    df = model.df.sort_values(by="timestamp").reset_index(drop=True)
    y = np.log(df["cost"].to_numpy())

    lower = model.ppi[alpha]["lower_bound"]
    upper = model.ppi[alpha]["upper_bound"]

    n_valid = len(lower)
    y = y[-n_valid:]

    cnt_outside = np.sum((y < np.array(lower)) | (y > np.array(upper)))
    return int(cnt_outside), 100 * cnt_outside / n_valid


table = []
for lag in [float("inf"), 400, 300, 250, 200, 150, 100, 50, 10]:
    jn = Joint_Normal(mu=14, k=1, sigma2=10, r=1, lag=lag, df=df)
    jn.generate_posterior()
    jn.generate_ppi(0.95)

    count, percent = check_ppi_validity(jn, 0.95)
    table.append([lag, count, f"{percent:.2f}%"])

html_table = tabulate(table, headers=["Lag", "Count Outside", "Percentage Outside"], tablefmt="html")
display(HTML(html_table))
```

Conclusion

In this project, we explored Bayesian approaches to model cryptocurrency trading activity by analyzing both trade rate and trade volume at a one-minute resolution.

We modeled trade counts/rate using both a Gamma–Poisson (conjugate) and a Lognormal–Poisson (non-conjugate) framework. While both models captured overall trading trends, but they struggled with sudden spikes and drops, as seen from the high percentage of observations falling outside the Posterior Predictive Interval (PPI). The lag-based updating improved adaptability i.e. smaller lags made the models more responsive to recent changes, though overly small lags caused the prior to dominate.

For trade volume, we used a Joint Normal model, which performed well, with PPIs covering most of the observed data. Similar to trade rate, smaller lag values led to better responsiveness and prediction accuracy.

Overall, this study demonstrates the potential of Bayesian modeling to capture the underlying dynamics of cryptocurrency trading. While the models effectively characterize general trends in trade rate and volume, their ability to respond to abrupt market changes is limited, highlighting the importance of choosing appropriate lag values and priors.