Lecture 5: Forecast Uncertainty, Anomaly Detection, and Imputation

Contents

DSCI 574
Spatial & Temporal Models

Lecture Outline

- Lecture Learning Objectives
- Imports
- 1. Forecast Uncertainty
- 2. Anomaly Detection
- 3. Imputation

Lecture Learning Objectives

Imports

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import plotly.express as px
import scipy.stats  as stats
import matplotlib.pyplot as plt
import torch
from torch import nn, optim
from torch.utils.data import DataLoader, TensorDataset
from lightgbm import LGBMRegressor
from statsmodels.tsa.api import ETSModel
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import STL
from statsmodels.nonparametric.smoothers_lowess import lowess
from sklearn.ensemble import IsolationForest
from pyod.models.knn import KNN
# Custom modules
from scripts.utils import *
from scripts.plotting import *
# Plot defaults
px.defaults.height = 400; px.defaults.width = 500
plt.style.use("bmh"); plt.rcParams.update({"font.size": 16, "figure.figsize":
```

- Explain different methods of generating prediction intervals for forecasts, including analytical methods, bootstrapping, and quantile regression
- Implement the above methods using `statsmodels` classes like `ETSModel()` and `ARIMA()`
- Explain what anomaly detection is
- Describe and implement simple methods for detecting anomalies, such as STL decomposition
- Explain what imputation is
- Describe and implement basic methods of imputation, such as mean imputation and polynomial interpolation

Recap

In univariate time-series forecasting, there are 3 main components that we usually model:

- Trend (STL decomposition, Holt)
- Seasonality (STL decomposition, Holt-Winter)
- Autocorrelation (ARIMA)

1. Probabilistic forecasting

- Up until now, we've been dealing mainly with **point forecasts**, we focused on modeling the 'averages'
- In many applications, we might want to estimate the uncertainty around our forecasting
- Or we want to predict the extreme (i.e., 90th quantile)
- Or we want to predict the variance (i.e., how volatile a metric will become in the future)

Real-life applications of probabilistic forecasting

- Electricity/enegery demand: We want to forecast the upper quantile of the electricity demand so we don't have to cut off electricity in the events of a spike.

- Natural disaster (earth quakes, floods, drought, fire): We want to forecast the extreme value, so that we can have a proper plan in place to respond to extreme weather events

Question

Which of the following is an analytical method for constructing prediction intervals?

A. Bootstrapping residuals

B. Quantile regression

C. Monte Carlo simulation

D. Assuming a normal distribution of residuals and using standard errors

1.1. Analytical

- If we assume the distribution of forecasts are normal, then the prediction interval at the h-step forecast is simply:

$$\hat{y}_{T+h|T} \pm c \times \hat{\sigma}_h$$

- Where $\hat{\sigma}_h$ is the estimated standard deviation of the h-step forecast distribution and c determines the coverage probability, usually we use 1.96 which gives us a 95% prediction interval (recall that 95% of the area under a standard normal curve lies within 1.96 standard deviations of the mean)
- So the focus is really on estimating $\hat{\sigma}_h$

Question

T/F: A 90% prediction interval will be wider than a 95% prediction interval

- A. TRUE
- B. **FALSE**

- For one-step ahead forecasts, it’s often reasonable to estimate $\hat{\sigma}_1$ from the fitted model residuals as:

$$\hat{\sigma}_1 = \sqrt{\frac{1}{T-K}\sum_{t=1}^{T} e_t^2}$$

- K is the number of parameters
- T is the total length of the time-series
- For multi-step forecasts, things get more difficult, and prediction intervals typically increase in length as the forecast horizon increases

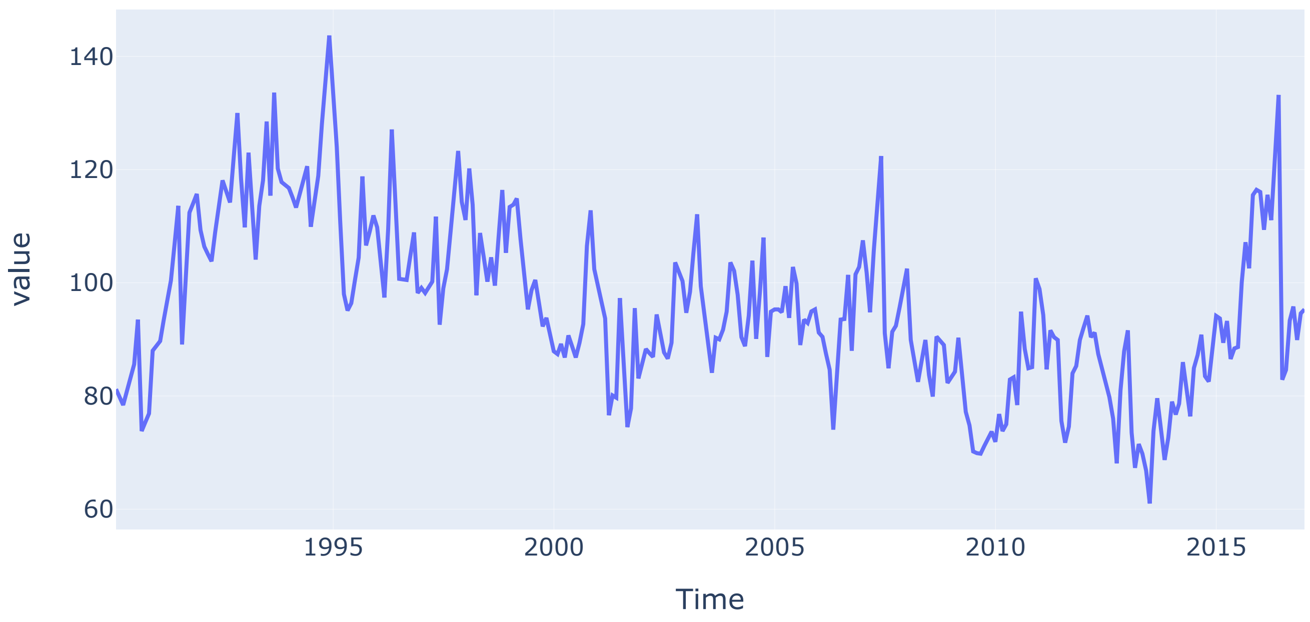
- Some methods have been derived mathematically:

Method	**Forecast standard deviation**
Mean	$\hat{\sigma_h} = \hat{\sigma}_1\sqrt{1+\frac{1}{T}}$
Naïve	$\hat{\sigma_h} = \hat{\sigma}_1\sqrt{h}$
Seasonal naïve	$\hat{\sigma_h} = \hat{\sigma}_1\sqrt{\frac{h-1}{m}+1}$
Drift	$\hat{\sigma_h} = \hat{\sigma}_1\sqrt{h(1+\frac{h}{T})}$
ETS	Chapter 6 of Forecasting with Exponential Smoothing
ARIMA	Section 6.4 of Introduction to Time Series and Forecasting

- Think about how $\hat{\sigma_h}$ change as T increases
- Think about how $\hat{\sigma_h}$ change as h increases

- Let's try forecasting some beach time series data:

```
# Load data
train = (pd.read_csv("data/beach_train.csv", index_col=0, parse_dates=True)
           .resample("1M").mean()
           .interpolate(method="linear")
           .loc["1990":]
        )
valid = (pd.read_csv("data/beach_valid.csv", index_col=0, parse_dates=True)
           .resample("1M").mean()
        )
# Forecast index
forecast_index = create_forecast_index(valid.index[0], len(valid.index), freq=
px.line(train, width=800)
```



A naïve forecast

Recall the formula for naive forecast SD:
$\hat{\sigma_h} = \hat{\sigma}_1\sqrt{h}$

- $\hat{\sigma}_1$ is the standard deviation of the residuals (from fitting model on training set!)

```
# coverage probability
c = 1.96

# fitted a naive
train['pred'] = train["Shoreline"].shift(1)

# residuals
train['resid'] = train['Shoreline'] - train['pred']
train.head()
```

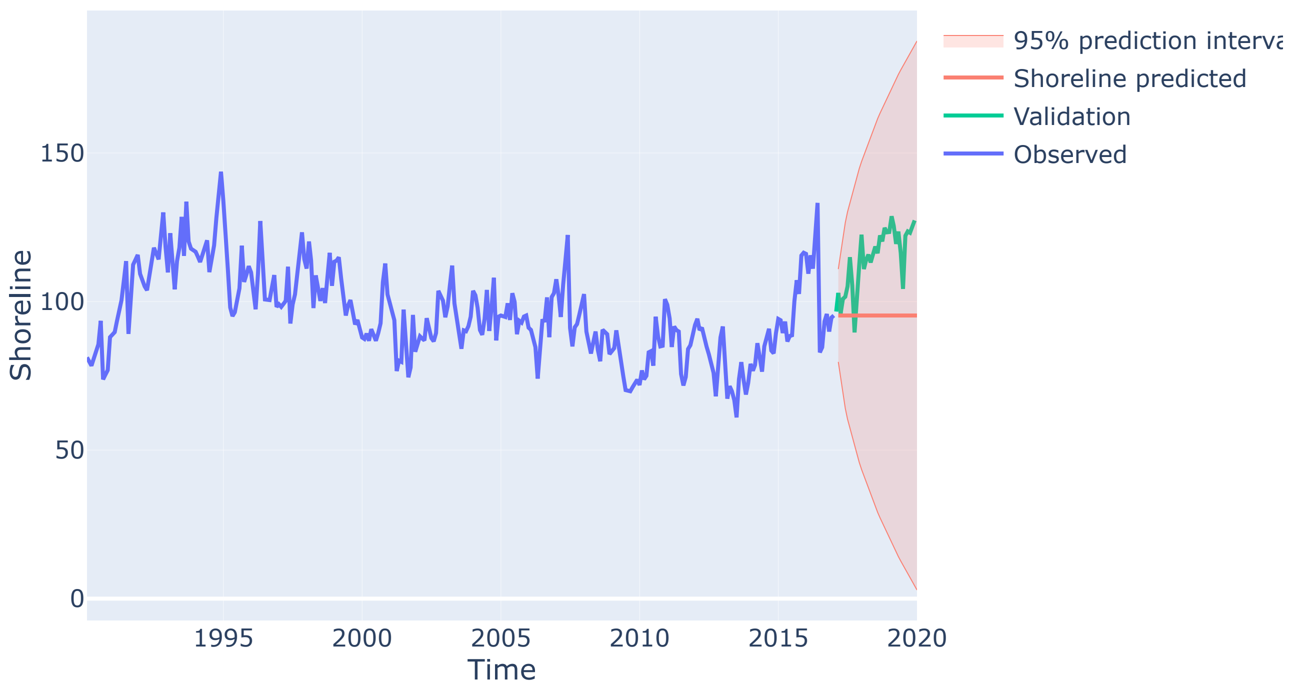
	Shoreline	pred	resid
Time			
1990-01-31	81.2	NaN	NaN
1990-02-28	79.8	81.2	-1.4
1990-03-31	78.4	79.8	-1.4
1990-04-30	80.8	78.4	2.4
1990-05-31	83.2	80.8	2.4

```
# SD of residuals
sigma = train['resid'].std()

# horizon h
horizon = np.arange(1, len(forecast_index) + 1)

# create naive forecast
naive_forecast = train["Shoreline"].iloc[-1]

# create lower and upper bound
naive = pd.DataFrame({"Shoreline": naive_forecast,
                      "pi_lower": naive_forecast - c * sigma * np.sqrt(horizon
                      "pi_upper": naive_forecast + c * sigma * np.sqrt(horizon
                      "Label": "Naive"},
                     index=forecast_index)
```

```
naive.head()
```

	Shoreline	pi_lower	pi_upper	Label
Time				
2017-02-28	95.3	79.682534	110.917466	Naive
2017-03-31	95.3	73.213567	117.386433	Naive
2017-04-30	95.3	68.249755	122.350245	Naive
2017-05-31	95.3	64.065067	126.534933	Naive
2017-06-30	95.3	60.378284	130.221716	Naive

```
plot_prediction_intervals(train["Shoreline"], naive, "Shoreline", valid=valid[
```

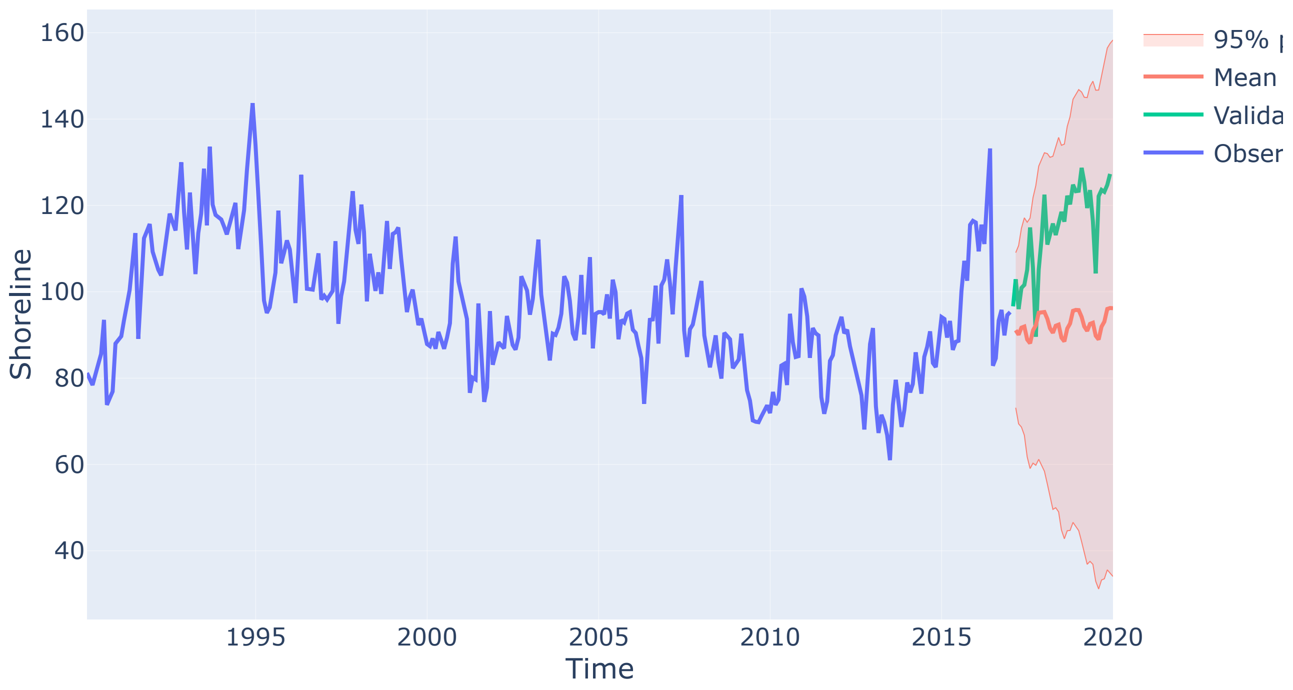


- `ETSModel` and `ARIMA` in `statsmodels` both have a method `get_prediction()`, which returns predicted mean values and prediction intervals
- Let's try `ETSModel`:

```
# Fit an ETS with additive trend, additive seasonality, additive error
model = ETSModel(train["Shoreline"], error="add", trend="add", seasonal="add")

# get prediction
ets = model.get_prediction(start=forecast_index[0], end=forecast_index[-1]).su
ets.head()
```

	mean	pi_lower	pi_upper
2017-02-28	91.100295	73.137273	109.063318
2017-03-31	90.056668	69.407704	110.705632
2017-04-30	91.693431	68.669314	114.717549
2017-05-31	91.952892	66.776295	117.129488
2017-06-30	88.945318	61.785922	116.104713

```
# plot
plot_prediction_intervals(train["Shoreline"], ets, "mean", valid=valid["Shorel
```

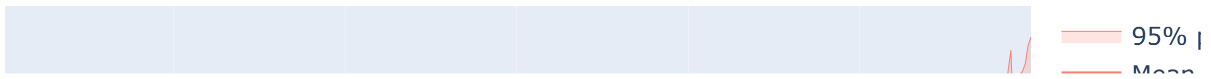


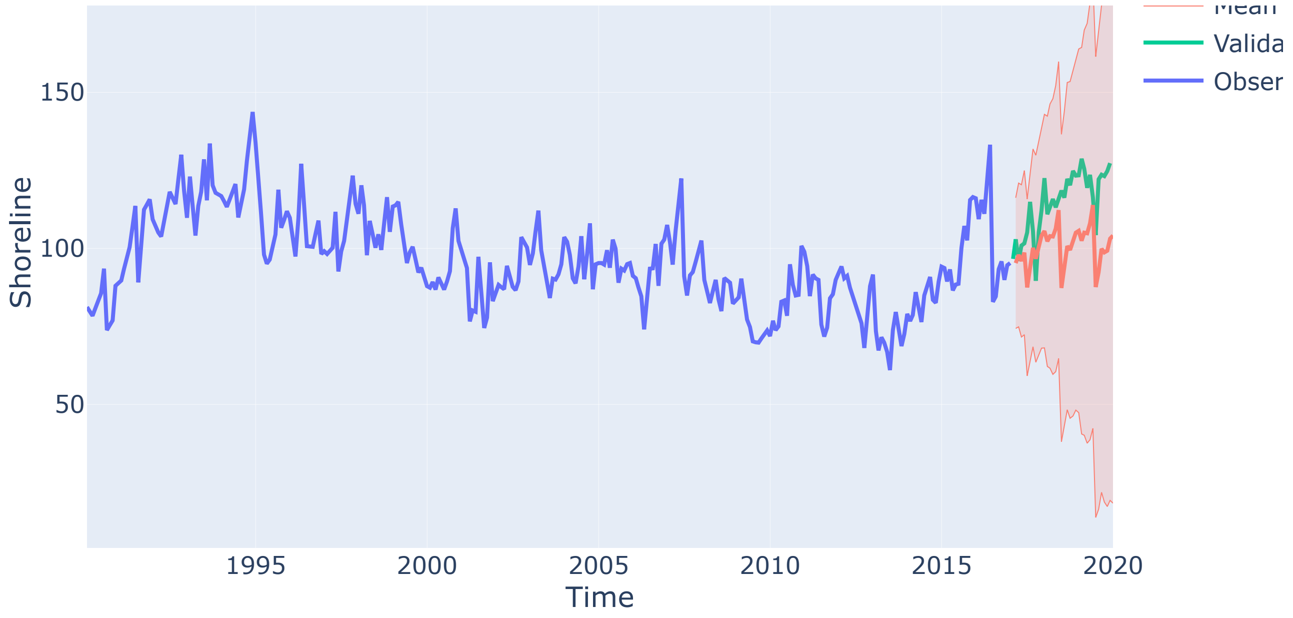
- And now, `ARIMA`:

```
# Fit a seasonal ARIMA
model = ARIMA(train["Shoreline"], order=(3, 1, 0), seasonal_order=(2, 1, 0, 12

# get prediction
arima = model.get_prediction(start=forecast_index[0], end=forecast_index[-1]).
arima.head()
```

Shoreline	mean	mean_se	mean_ci_lower	mean_ci_upper
2017-02-28	95.285870	10.676017	74.361262	116.210479
2017-03-31	97.885291	11.749448	74.856795	120.913787
2017-04-30	95.992893	12.444951	71.601237	120.384549
2017-05-31	98.632454	13.416434	72.336726	124.928181
2017-06-30	87.482282	14.455396	59.150227	115.814337

```
plot_prediction_intervals(train["Shoreline"], arima, "mean", valid=valid["Shor
```





1.2. Simulation & Bootstrapping

- If it’s unreasonable to assume a normal distribution for future forecasts and/or there are no analytical solutions, the most common alternative is to use bootstrapping to generate prediction intervals
- Assuming future errors will be similar to past errors, we can randomly draw from past errors and add them on to forecasts to simulate future values. We can simulate this over and over to generate many realisation of future scenarios and then calculate prediction intervals by calculating percentiles on these realisations.

For example, a forecast error is defined as: $$\epsilon_t = y_t - \hat{y}_{t|t-1}$$

We can re-write this as

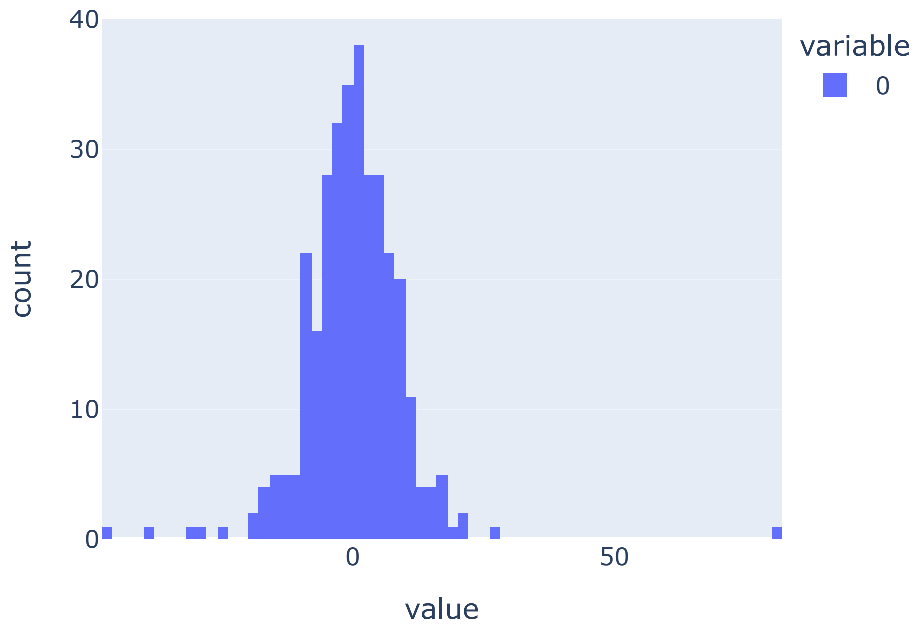
$$y_t = \hat{y}_{t|t-1} + \epsilon_t$$

We can simulate the next observation using $$y_{T+1} = \hat{y}_{T+1|T} + \epsilon_{T+1}$$

where $\hat{y}_{T+1|T}$ is the one-step forecast and ϵ_{T+1} is the unknown future error.

Assuming future errors will be similar to past errors, we can replace ϵ_{T+1} by sampling from the collection of errors we have seen in the past (i.e., the residuals)

```
px.histogram(model.resid)
```



- The `statsmodels` implementations of `ARIMA` and `ETS` have a `.simulate()` method for exactly this (although it would be easy to code up yourself too)
- Here's 100 simulations from an `ETS` model:

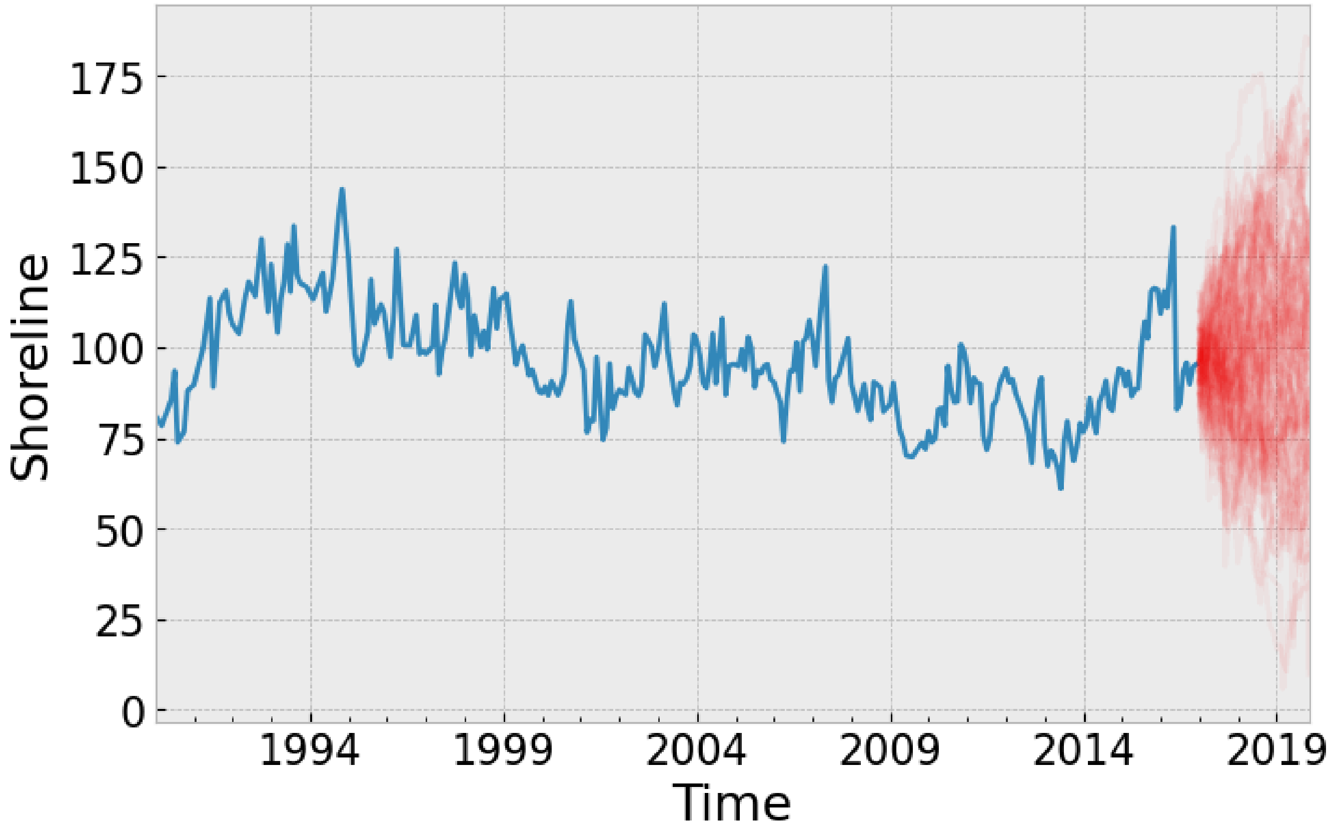
```
# number of simulations
n_simulations = 100

# Fit an ETS model
model = ETSModel(train["Shoreline"], error="add", trend="add").fit(disp=0)

# simulate predictions
ets = model.simulate(anchor="end", nsimulations=len(forecast_index),
                     repetitions=n_simulations,
                     random_errors="bootstrap")
ets.head()
```

	simulation.0	simulation.1	simulation.2	simulation.3	simulation.4	simula
2017-01-31	97.052376	93.606338	90.457546	89.959865	89.848554	91.
2017-02-28	100.514750	89.768987	93.562871	93.101119	99.962830	94.
2017-03-31	109.369340	86.646245	95.753347	84.083083	92.818643	77.
2017-04-30	103.056649	83.969551	89.074260	85.884254	103.913331	92.
2017-05-31	99.190012	88.670399	86.666531	63.767206	110.037863	91.

5 rows × 100 columns

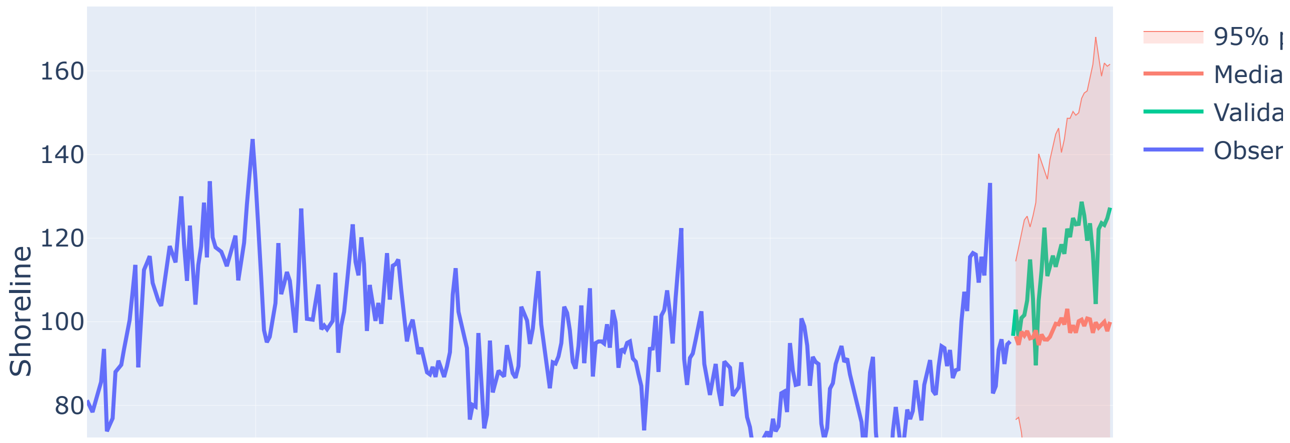
- Here's the realisation of all those 100 simulations:

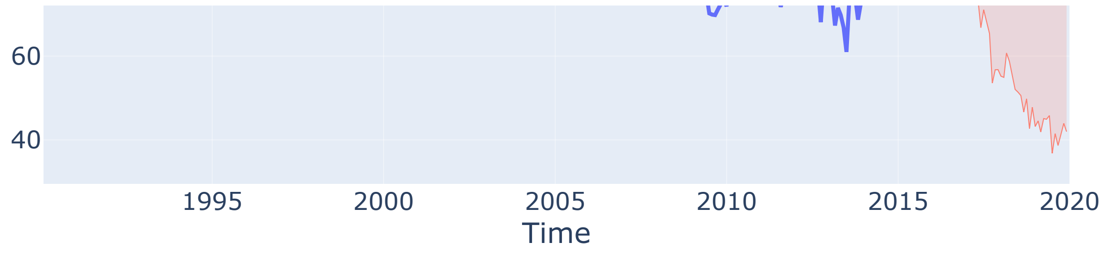
```
ax = train["Shoreline"].plot.line()
ets.plot.line(ax=ax, legend=False, color="r", alpha=0.05,
              xlabel="Time", ylabel="Shoreline", figsize=(8,5));
```



- We can use the dataframe `.quantile()` method to calculate desired quantiles from all these simulations:

```python
# we want 95% prediction intervals
upper_q = 0.975
lower_q = 1 - upper_q
ets = pd.DataFrame({"median": ets.median(axis=1),
                    "pi_lower": ets.quantile(lower_q, axis=1),
                    "pi_upper": ets.quantile(upper_q, axis=1)},
                   index=forecast_index)
plot_prediction_intervals(train["Shoreline"], ets, "median", valid=valid["Shor
```





- We could easily implement this for any ML model too

1.3. Quantile regression

- In quantile regression, we wish to predict a particular quantile of a distribution, rather than the usual mean (i.e., $E(Y|X)$)
- The idea here is that we are predicting some quantile of a future time step, say $q = 0.9$. That is, we expect the actual value to be below this quantile 90% of the time, and above it 10% of the time.

- We need to account for the fact that the true value is more likely to be below than above our prediction by weighting under/over prediction differently and that's what the quantile loss is doing
- For this, we often use the "quantile loss" (sometimes called "pinball loss"):

$$\mathcal{L} = \begin{cases} (1-q)(\hat{y}_{t,q} - y_t), & \text{if } y_t < \hat{y}_{t,q} \\ q(y_t - \hat{y}_{t,q}), & \text{if } y_t \geq \hat{y}_{t,q} \end{cases}$$

- Let's see how the loss varies for different quantiles (q) and residuals ($y_t - \hat{y}_{t,q}$):

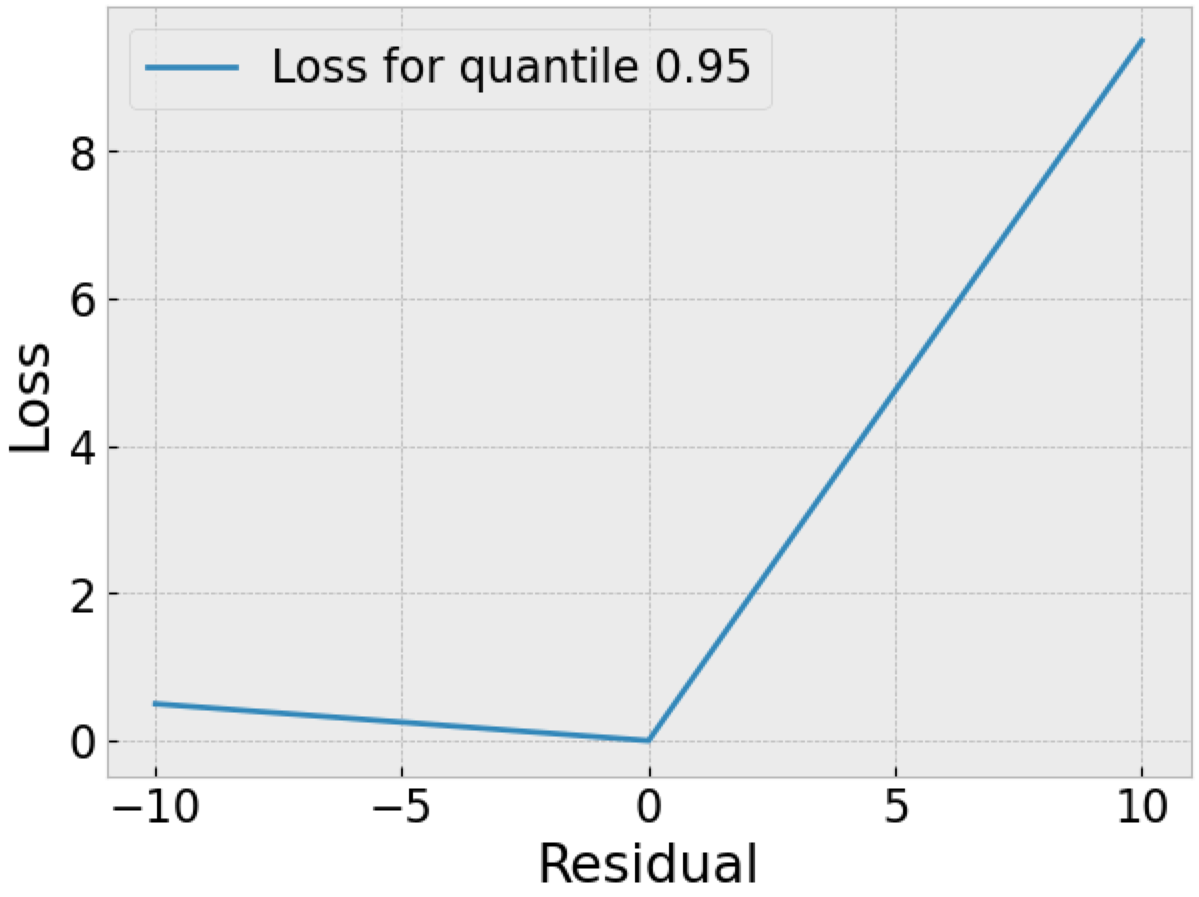
```python
def pinball_loss(residual, q):
    return np.maximum(q * residual, (q - 1) * residual)
```

Imagine we are predicting the 95th quantile of temperature (e.g., 95% of the data point is below this value)

When we say, "We forecast the temperature to have 95% of chances to be below 5°C," we expect the temperature to be lower than 5°C and would be surprised that the temperature would be 8°C.

In other words, this quantile forecast should get a higher penalty if it under forecasted the temperature than if it over predicted it.

```python
residuals = np.arange(-10, 11)
(pd.DataFrame({"Loss for quantile 0.95": pinball_loss(residuals, 0.95)},
               index=residuals)
   .plot.line(xlabel="Residual", ylabel="Loss", figsize=(7, 5)));
```



- Let’s consider the quantile 0.95 above:
  - Recall that residuals are $y_t - \hat{y}_{t,q}$ (true - observed)
  - For positive residuals (under-predictions) our loss is high. We get heavily penalised because our prediction of the 0.95 quantile is *below* the true value, that’s bad, because we expect the true value to be *below* the 0.95 quantile with 95% probability.
- For negative residuals (over-predictions) our loss is lower for the opposite reason
- As you can see, the pinball loss L_α is highly asymmetrical: it doesn’t increase at the same speed if you over forecast (low penalty) or under forecast (high penalty).
- For example, if our q95 forecast = 5°C and true value = 0°C. Our residual = 0 - 5 = -5. The penalty = 0.25.

- If our q95 forecast = 5°C and true value = 10°C. Our residual = 10 - 5 = 5. The penalty = 4.75

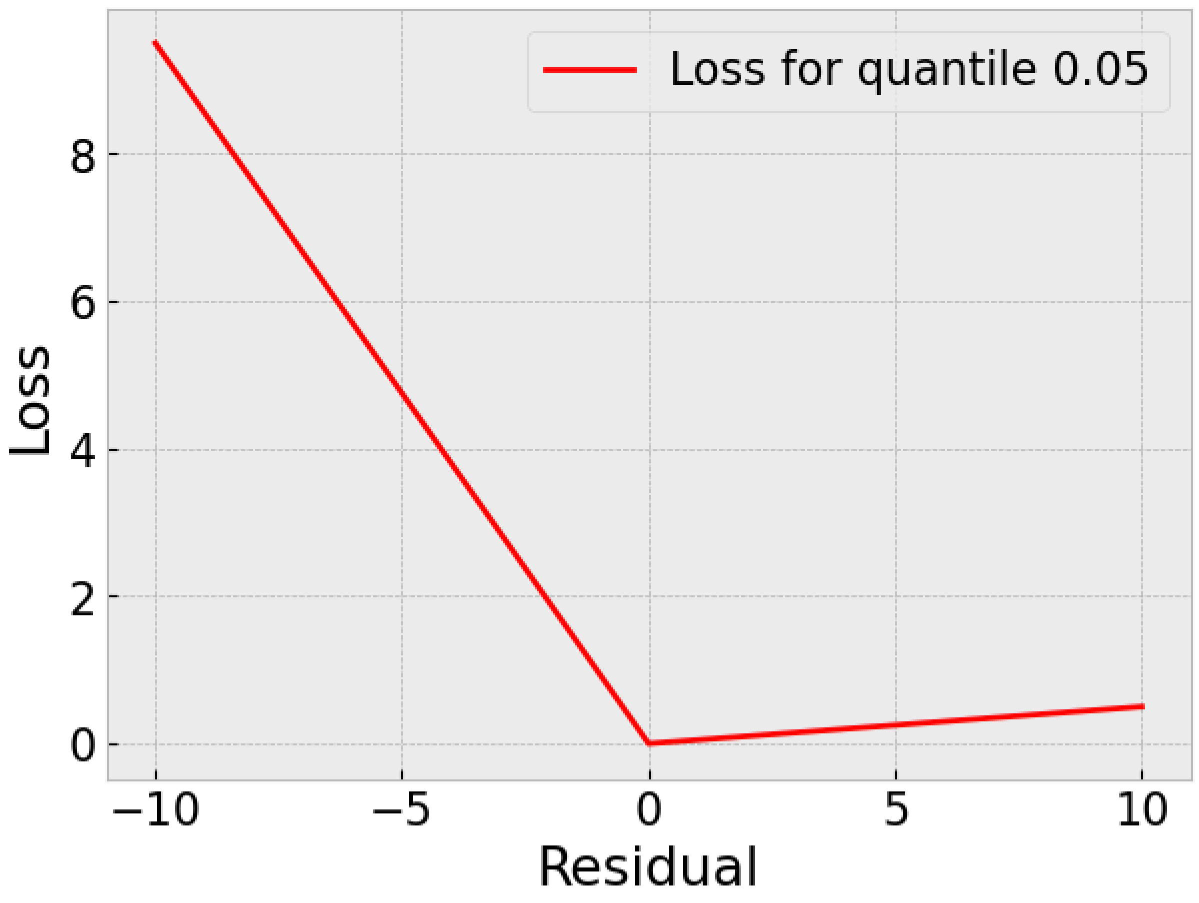
iClicker Question:

True or False: The 0.05 quantile places higher penalty on under-forecasting then over-forecasting.

A. TRUE

B. FALSE

```
residuals = np.arange(-10, 11)
(pd.DataFrame({"Loss for quantile 0.05": pinball_loss(residuals, 0.05)},
              index=residuals)
   .plot.line(xlabel="Residual", ylabel="Loss", figsize=(7, 5), color="r"));
```



- The 0.05 quantile is just the mirror of the above.

- Different packages have quantile loss implemented in various forms, e.g., the `statsmodels` class `QuantReg` or the `sklearn` class `GradientBoostingRegressor`. `lightgbm` is commonly used for quantile regression:

```
# Load data
train = (pd.read_csv("data/beach_train.csv", index_col=0, parse_dates=True)
           .resample("1M").mean()
           .interpolate(method="linear")
           .loc["1990":]
        )
valid = (pd.read_csv("data/beach_valid.csv", index_col=0, parse_dates=True)
           .resample("1M").mean()
        )
# Forecast index
forecast_index = create_forecast_index(valid.index[0], len(valid.index), freq=
```

```
# Prep data
lagged_train = lag_df(train.copy(), lag=12).dropna()

y_train = lagged_train["Shoreline"]
X_train = lagged_train.drop(columns="Shoreline")

# Prediction intervals
models = {}
for q in [0.025, 0.5, 0.975]:
    models[str(q)] = LGBMRegressor(objective="quantile", alpha=q).fit(X_train,
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of test
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1245
[LightGBM] [Info] Number of data points in the train set: 312, number of used
[LightGBM] [Info] Start training from score 70.132500
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testi
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1245
[LightGBM] [Info] Number of data points in the train set: 312, number of used
[LightGBM] [Info] Start training from score 94.850006
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

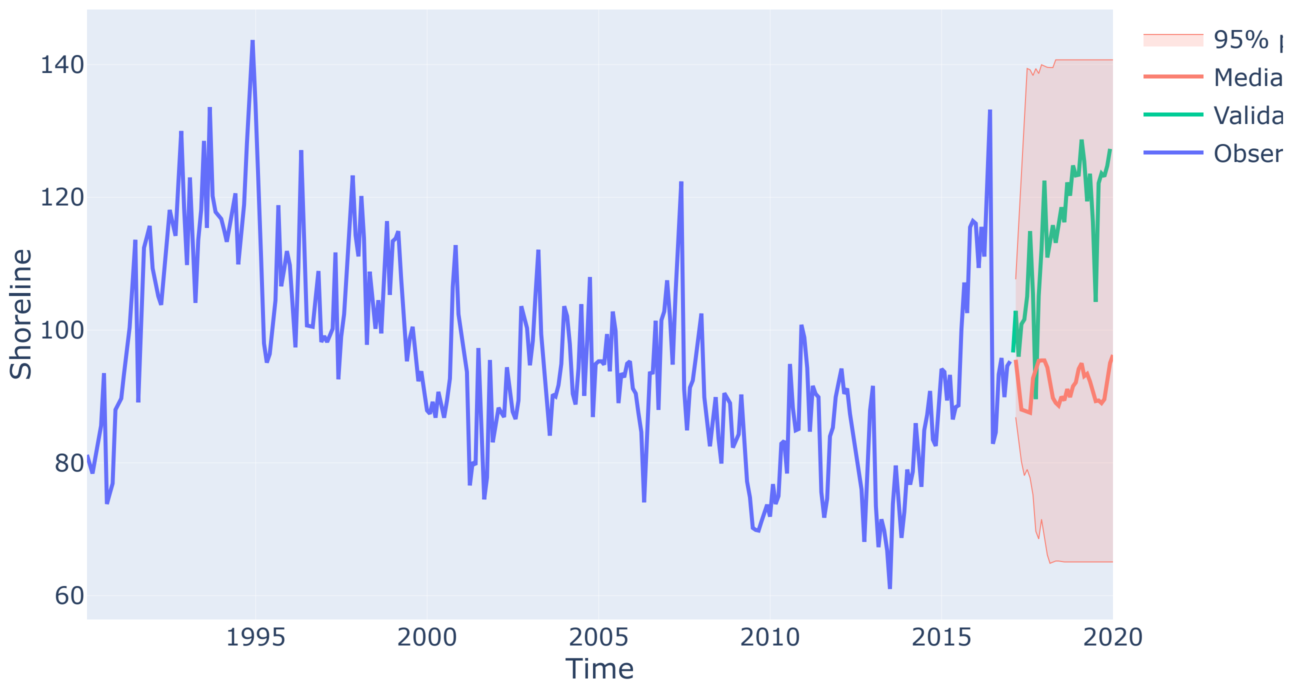
```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of test
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1245
```

```
[LightGBM] [Info] Number of data points in the train set: 312, number of used
[LightGBM] [Info] Start training from score 127.325005
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```python
input_data = X_train.iloc[-1].copy().to_numpy()
lgbm = pd.DataFrame({"median": recursive_forecast(input_data, models["0.5"], n
                     "pi_lower": recursive_forecast(input_data, models["0.025"
                     "pi_upper": recursive_forecast(input_data, models["0.975"
                    index=forecast_index)
plot_prediction_intervals(train["Shoreline"], lgbm, "median", valid=valid["Sho
```



- We could also implement this loss easily in a neural network framework like `pytorch`
- Quantile loss is not currently a supported criterion in `pytorch` but it's easy to define ourselves. We really have two options:
  1. Train a network for each quantile we want to predict; or
  2. Train a network to output multiple quantiles at once
- I'm going to opt for 2 below:

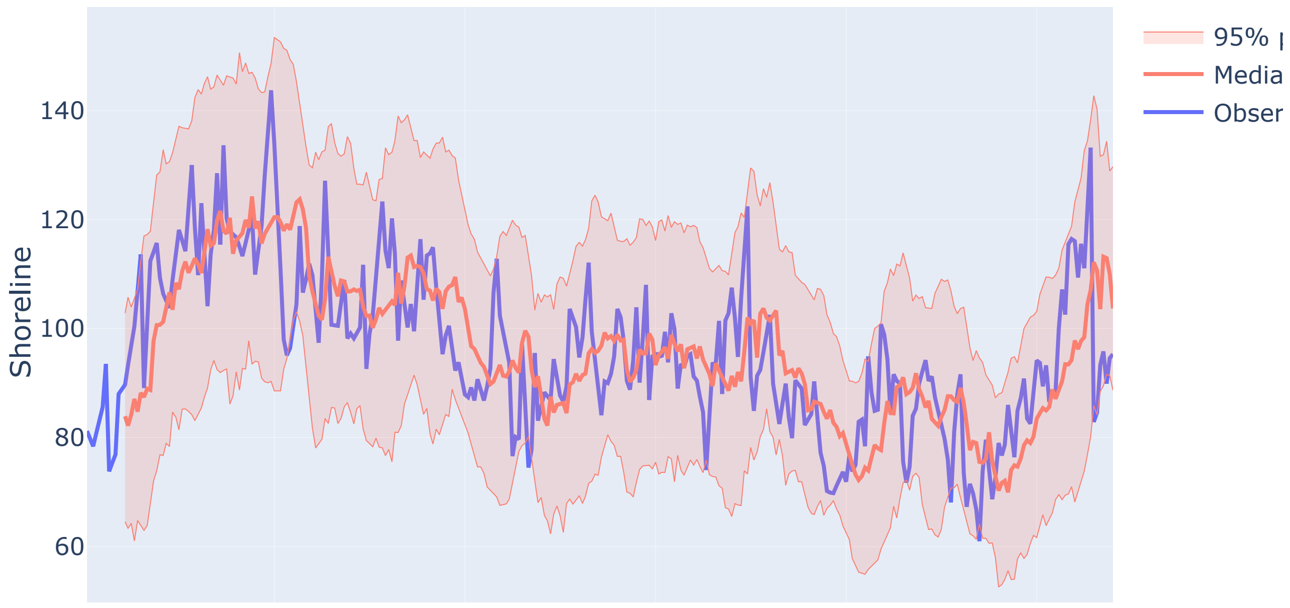
```
# define network
class quantile_nn(nn.Module):
    def __init__(self):
        super().__init__()
        self.main = nn.Sequential(
            nn.Linear(12, 30),
            nn.ReLU(),
            nn.Linear(30, 3)
        )

    def forward(self, x):
        return self.main(x)

# custom quantile loss
def quantile_loss(y_hat, y_true, alpha):
    q = [alpha, 0.5, 1 - alpha]  # quantiles: alpha, 0.5, 1 - alpha
    error = y_hat - y_true       # errors
    loss = 0
    for n in range(error.shape[1]):
        loss += torch.max(q[n] * error[:, n], (q[n] - 1) * error[:, n])  # qua
    return loss.mean()

# make tensors
dataset = TensorDataset(torch.tensor(X_train.to_numpy(), dtype=torch.float32),
                        torch.tensor(y_train.to_numpy(), dtype=torch.float32))
dataloader = DataLoader(dataset, batch_size=50, shuffle=True)
```

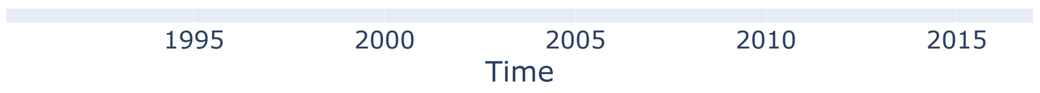
```
# define model framework
model = quantile_nn()
optimizer = optim.Adam(model.parameters())
epochs = 20

# train
for epoch in range(epochs):  # for each epoch
    for X, y in dataloader:  # for each batch
        optimizer.zero_grad()  # Zero all the gradients w.r.t. parameters
        y_hat = model(X)       # Forward pass to get output
        loss = quantile_loss(y_hat, y.unsqueeze(1), alpha=0.975)  # alpha=0.97
        loss.backward()  # Calculate gradients w.r.t. parameters
        optimizer.step()  # Update parameters
    print(f"Epoch {epoch + 1}. Loss = {loss.item():.2f}")
```

```
Epoch 1. Loss = 94.19
Epoch 2. Loss = 59.36
Epoch 3. Loss = 34.01
Epoch 4. Loss = 10.23
Epoch 5. Loss = 7.87
Epoch 6. Loss = 8.03
Epoch 7. Loss = 6.03
Epoch 8. Loss = 7.31
Epoch 9. Loss = 4.94
Epoch 10. Loss = 6.42
Epoch 11. Loss = 6.57
Epoch 12. Loss = 5.35
Epoch 13. Loss = 6.72
Epoch 14. Loss = 5.75
Epoch 15. Loss = 6.14
Epoch 16. Loss = 4.83
Epoch 17. Loss = 5.95
Epoch 18. Loss = 6.95
Epoch 19. Loss = 6.00
Epoch 20. Loss = 5.54
```

- I could then make forecasts using a recursive or direct method, but I'll just show the prediction on the train set for now:

```
qnn = model(torch.tensor(X_train.to_numpy(), dtype=torch.float32)).detach().nu
qnn = pd.DataFrame({"median": qnn[:, 1],
                    "pi_lower": qnn[:, 2],
                    "pi_upper": qnn[:, 0]},
                   index=X_train.index)
plot_prediction_intervals(train["Shoreline"], qnn, "median", width=800)
```





1.4. Evaluating distributional forecast accuracy

- Note that prediction intervals for time series forecats are almost always too narrow - this is a well known problem
- There are 4 main sources of uncertainty:
  1. Random error term;
  2. Uncertainty of model parameter estimates;
  3. Uncertainty introduced by choice of model;
  4. Uncertainty about the consistency of the data generating process into the future.
- Most of the methods above only account for the first source. Unlike OLS regression, there's no closed form solutions available to include 2. Simulation tries to account for 2 and 3 somewhat. 4 is practically impossible to know and can only be guessed at.
- We often focus on evaluating our point forecasts, but sometimes we want to evaluate our distributional forecasts as well!
- For this, we typically focus on the "coverage" (do observations fall within the prediction intervals?), how far inside/outside observations are, and how wide the intervals themselves are
- I'm not going to talk much about this here, but here are a few common evaluation metrics (see this for more details):
  - Quantile loss (this was used in the M5 competition)
  - Winkler score
  - Mean Interval Score (this was used in the M4 competition)
  - Continuous Ranked Probability Score

2. Anomaly Detection

- Outliers are observations that are very different from the majority of the observations in the time series

- They may be errors (human error, measurement error, machine error, etc.), or they may just be unusual or unique observations
- There are countless methods for detecting outliers and I wanted to briefly show you some examples here
- But outlier detection is very field-specific and it takes some expert knowledge and interpretation to identify an outlier in your data!
- Let’s use the amazon aws ec2 latency dataset from the Numenta Anomaly Benchmark (NAB). CPU usage data from a server in Amazon’s East Coast datacenter. The dataset ends with complete system failure resulting from a documented failure of AWS API servers

```
url='https://raw.githubusercontent.com/numenta/NAB/master/data/realKnownCause/
df = pd.read_csv(url, parse_dates=['timestamp'], index_col='timestamp')
df.head()
```

	value
timestamp	
2014-03-07 03:41:00	45.868
2014-03-07 03:46:00	47.606
2014-03-07 03:51:00	42.580
2014-03-07 03:56:00	46.030
2014-03-07 04:01:00	44.992

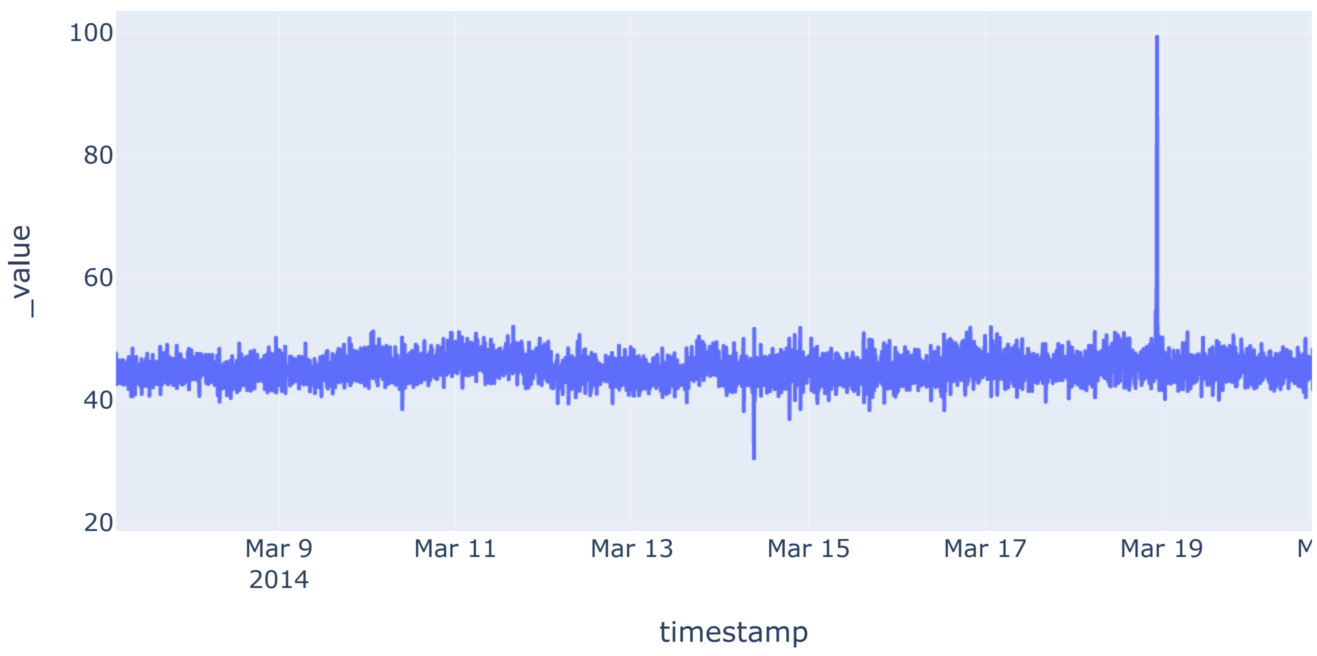
```
# print out day 1 of data
day1 = df[df.index.date==pd.to_datetime('2014-03-18')]
day1.shape[0]
```

```
0
```

```
# descriptive info
print('Number of observations: %d'  %(df.shape[0]))
print('Number of observations per day: %d ' %(day1.shape[0]))
print('Number of days: %d' %(df.index.map(pd.Timestamp.date).nunique()))
```

```
Number of observations: 4032
Number of observations per day: 0
Number of days: 15
```

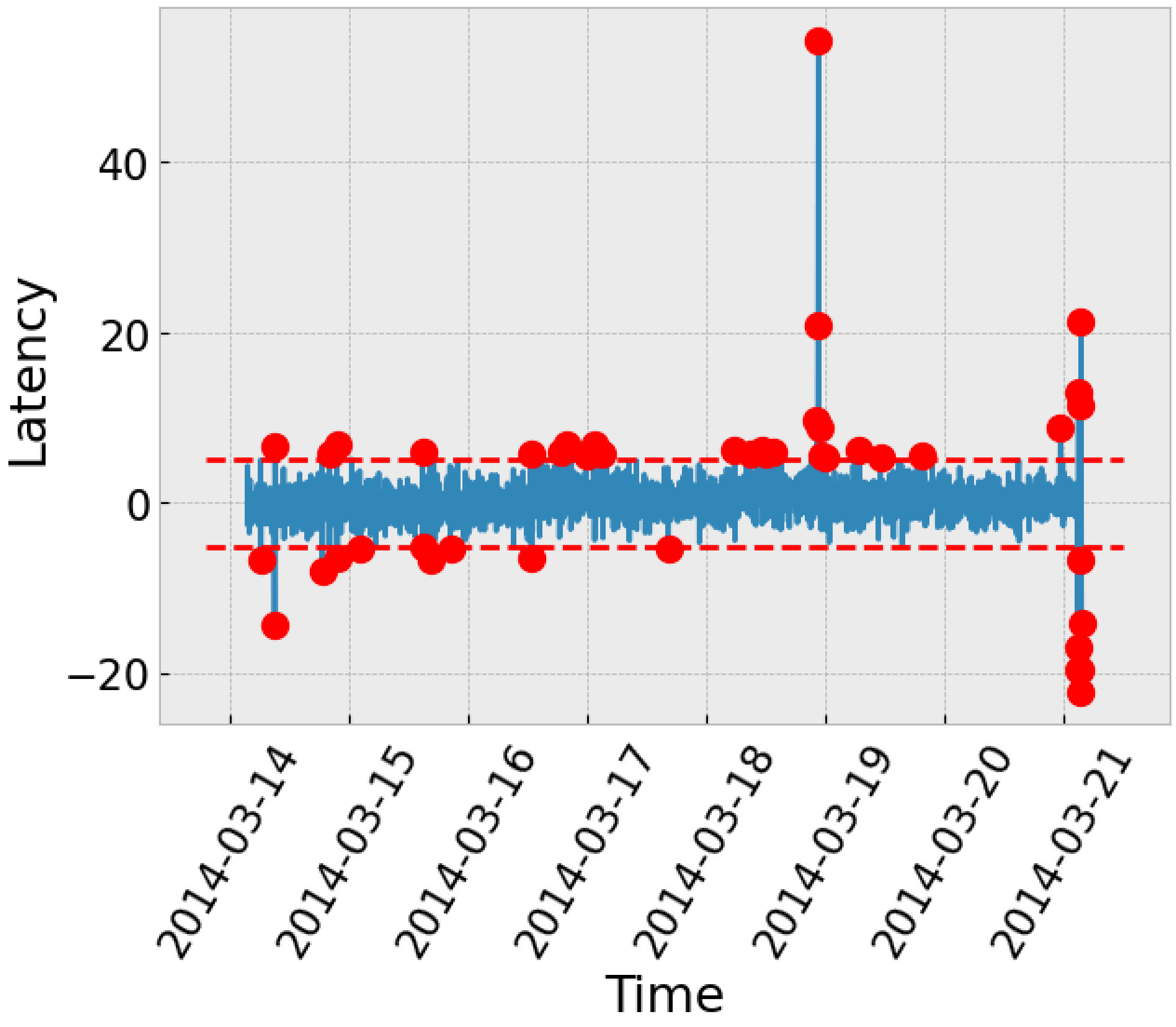
```
px.line(df, width=800)
```



2.1. Rolling median

- Methodology:
  1. Subtract a rolling median from the data (with suitable window size)
  2. Calculate standard deviation of residuals ($\hat{\sigma}_r$)
  3. Assume normally distributed residuals, identify outliers as $\pm 1.96 \times \hat{\sigma}_r$ (outside the 95% confidence interval)

```python
# Find outliers
resid = df - df.rolling(12*24*7).median() #7days rolling
stdev = resid.std().to_numpy()
upper_lim = 1.96 * stdev
lower_lim = -1.96 * stdev
outliers = np.where((resid > upper_lim) | (resid < lower_lim))[0]
# Plot
def outlier_plot(df, outliers, lower_lim, upper_lim):
    fig = plt.figure(figsize=(7, 5))
    plt.plot(df)
    xlim = plt.xlim()
    plt.plot(xlim, [lower_lim] * 2, "--r")
    plt.plot(xlim, [upper_lim] * 2, "--r")
    plt.plot(df.iloc[outliers], 'ro', ms=10)
    plt.xlabel("Time"), plt.xticks(rotation=60), plt.ylabel("Latency")
outlier_plot(resid, outliers, lower_lim, upper_lim)
```



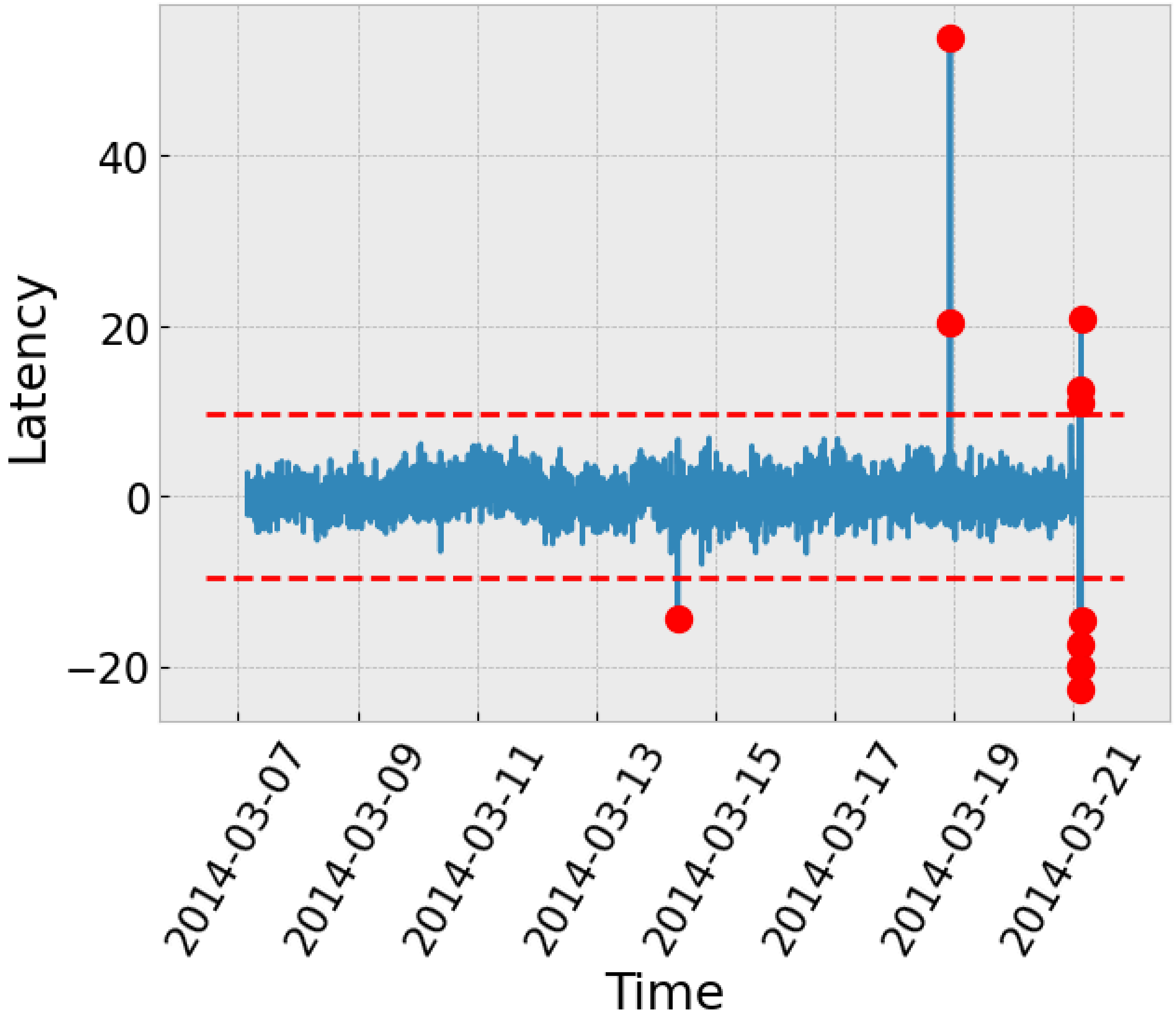
Question:

Why don’t we use rolling mean for outlier detection?

2.2. STL decomposition

- This approach mimics what happens is the R package `tsoutliers()` function of the R `forecast` package
- Methodology:
    1. Decompose data to find residuals:
        1. Non-seasonal data: use a loess curve
        2. Seasonal data: do STL decomposition
    2. Calculate $q_{0.1}$ and $q_{0.9}$ of the residuals
    3. Identify outliers as $\pm 2 \times (q_{0.9} - q_{0.1})$

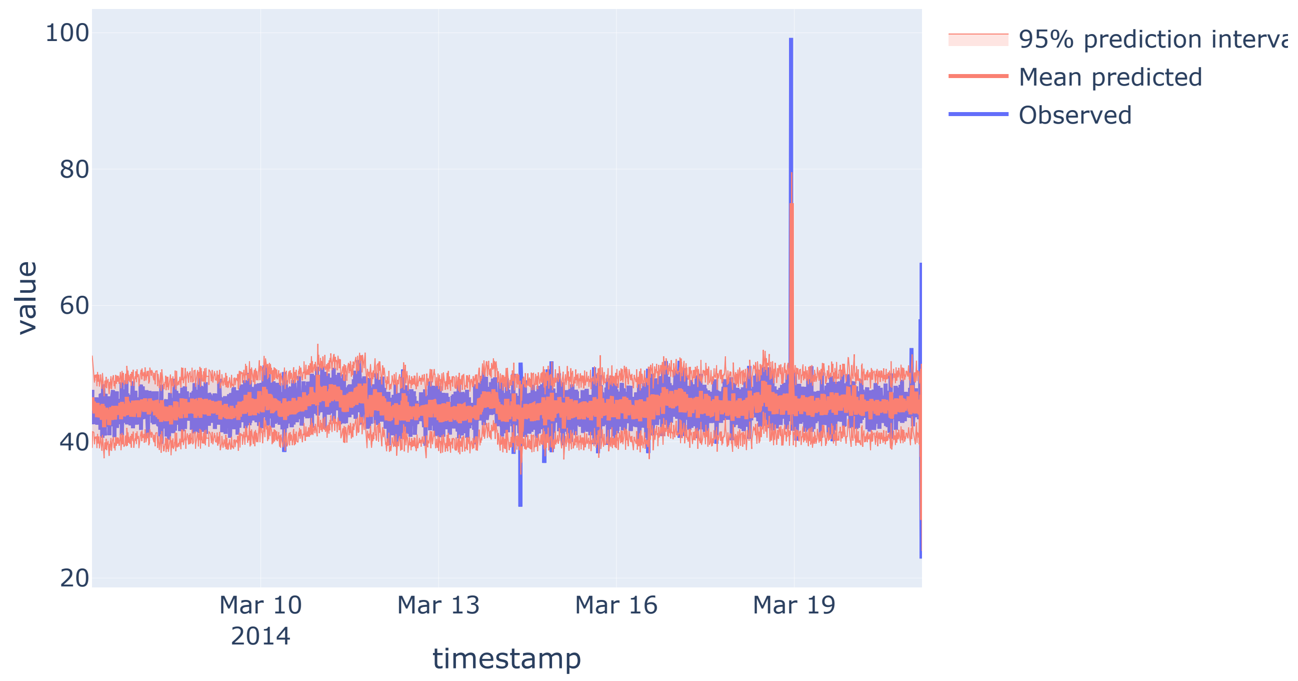
```
# Find outliers
resid = df - lowess(df["value"], df.index, return_sorted=False)[:, None]
q_01 = resid.quantile(0.1)
q_09 = resid.quantile(0.9)
upper_lim = 2 * (q_09 - q_01)
lower_lim = -2 * (q_09 - q_01)
outliers = np.where((resid > upper_lim) | (resid < lower_lim))[0]
# Plot
outlier_plot(resid, outliers, lower_lim, upper_lim)
```



2.3. Model-based

- Methodology:
  1. Fit a model to your data (e.g., ARIMA, ETS, machine learning, etc)
  2. Identify outliers as significant deviations from model predictions
- Below I'll plot the 95% confidence intervals for in-sample predictions from an ARIMA model:

```python
import warnings
warnings.filterwarnings("ignore")

model = ARIMA(df["value"], order=(3, 1, 0)).fit()
arima = model.get_prediction(start=1).summary_frame()
plot_prediction_intervals(df["value"], arima, "mean")
```



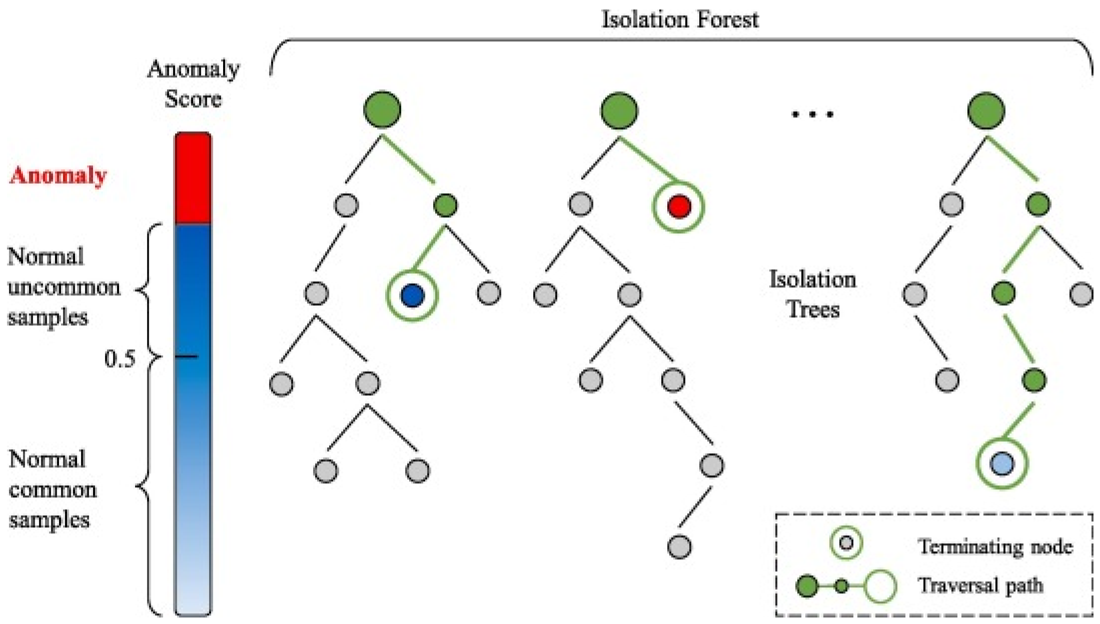
2.4. Machine learning approaches

- A unique approach to outlier detection is to train an ML model to predict outliers (if we have labelled examples)
- There are plenty of unsupervised and supervised approaches available for you to use. Here's a few common packages:
  - pyod
  - sklearn
  - luminaire
  - sklyline
  - And many more...

2.4.1 Isolation forest

- Here's an example of sklearn's IsolationForest (a tree-based algorithm). Note that the `contamination` function affects how many outliers will be detected, and is something that's usually based on historical data or tuned with a train/valid/test split (I'm not doing that here):

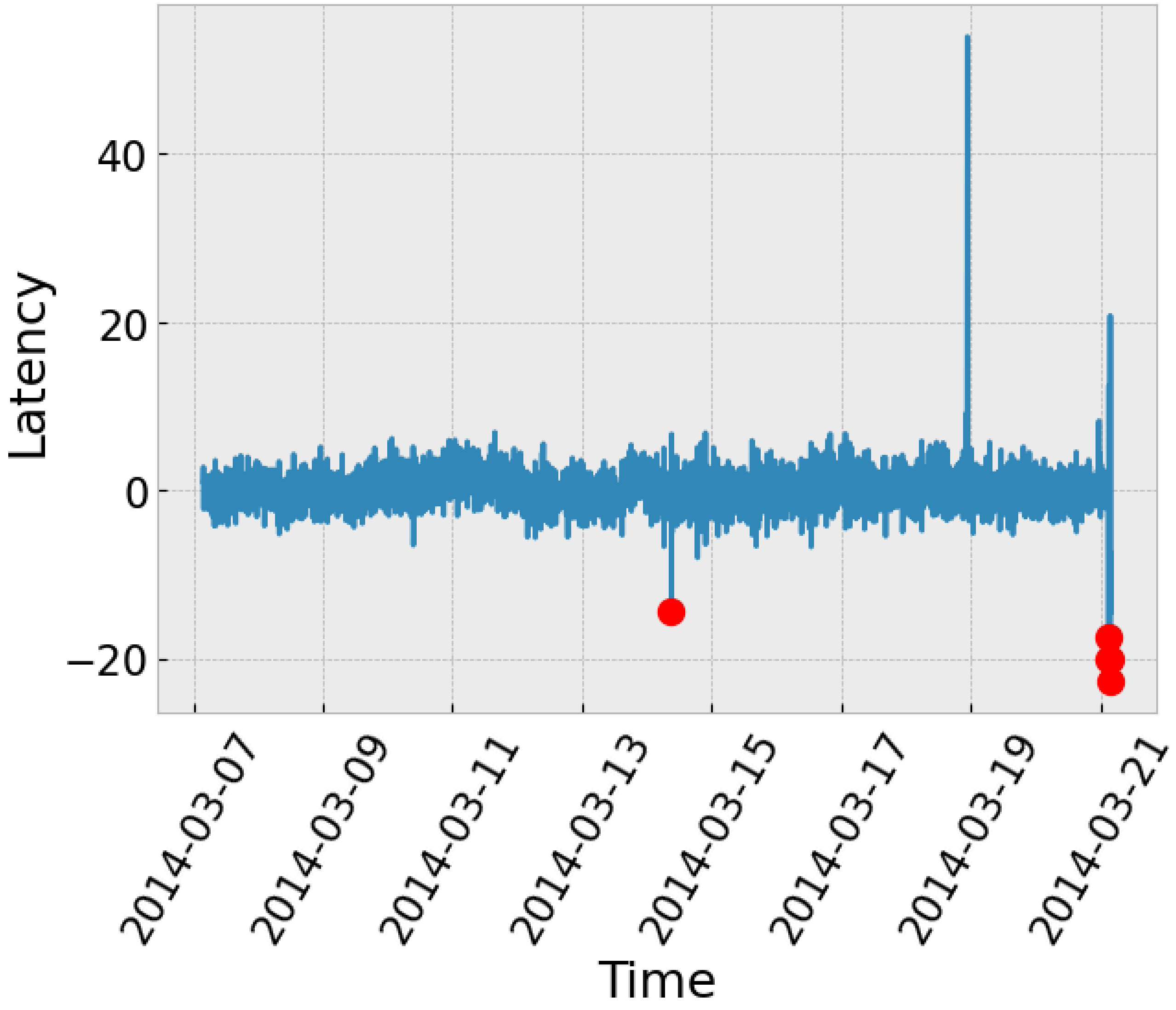
- Isolation Forest, like any tree ensemble method, is built on the basis of decision trees. In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature.
- Outliers are less frequent than regular observations and have more extreme values. Hence, when you use random forest to partition data points, outliers should be identified closer to the root of the tree (shorter average path length)



Source: https://www.sciencedirect.com/science/article/abs/pii/S1474034620301105

- A score close to 1 indicates anomalies
- Score much smaller than 0.5 indicates normal observations
- If all scores are close to 0.5 then the entire sample does not seem to have clearly distinct anomalies

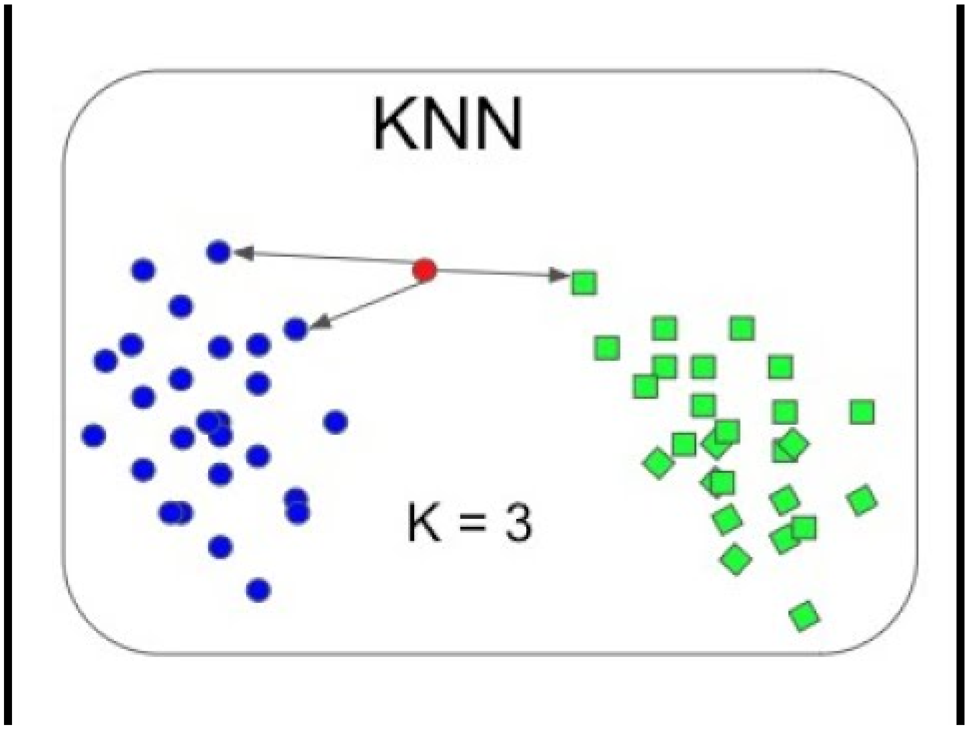
```
url='https://raw.githubusercontent.com/numenta/NAB/master/data/realKnownCause/
df = pd.read_csv(url, parse_dates=['timestamp'], index_col='timestamp')

# contamination parameter specifies the percentage of data points to be anomal
outliers = IsolationForest(contamination=0.001).fit_predict(df) == -1
# Plot
outlier_plot(resid, outliers, None, None)
```



This video provides an intuitive explanation of how isolation forests work for anomaly detection.

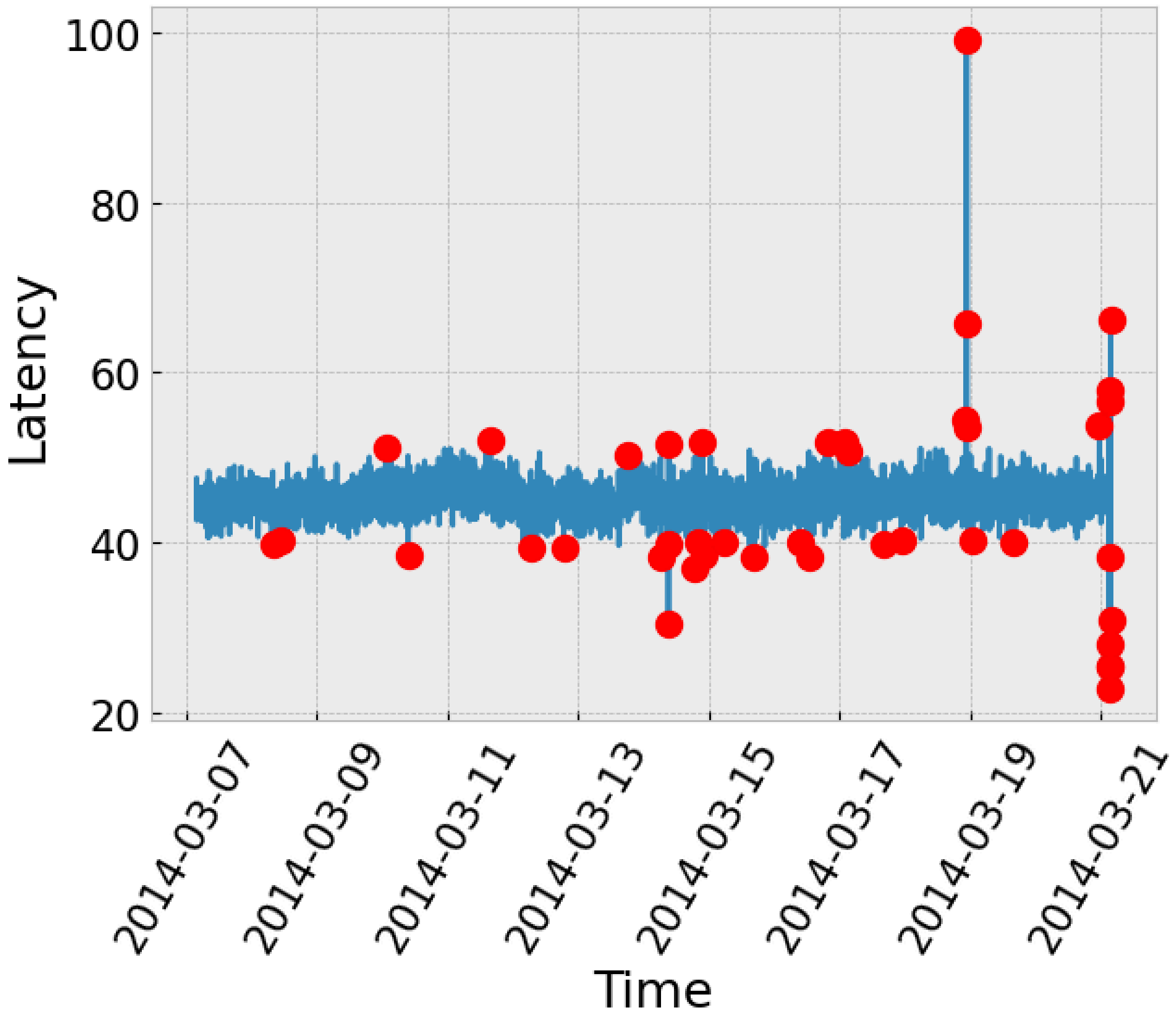
2.4.2 K-nearest neighbor

- Now `pyod`'s `KNN()` outlier detector:
- For each observation, compute the average distance to its knn
- The largest average distance values = outliers



- Three kNN detectors are supported:
  - largest: use the distance to the kth neighbor as the outlier score
  - mean: use the average of all k neighbors as the outlier score
  - median: use the median of the distance to k neighbors as the outlier score

```
outliers = KNN(contamination=0.01).fit(df).labels_ == 1
# Plot
outlier_plot(df, outliers == 1, None, None)
```



Global vs local outliers

- Global outliers: A data point with its value is far outside of the entirety of the data set (e.g., billionaires)
- Local/Contextual outliers: A data point is considered a contextual outlier if its value significantly deviates from the rest the data points in the same context. (e.g., earning 50K income in a developing countries)

2.5. Final thoughts

- We’ve mostly focussed on outlier detection here (i.e., identifying single anomalous points)

- But with time series, sometimes we often care about changes in the characteristics of the series, like changes in the level (data shift), trend, or seasonality. Libraries like luminaire, skyline, etc. can help with that. Such libraries can also help with multivariate outlier detection too. But as I mentioned, this kind of work is often very field-specific, and you make end up just making your own customs pipelines.
- Here are a few interesting reads from big tech companies concerned with outlier detection:
    - Netflix
    - Twitter
    - Alibaba

3. Imputation

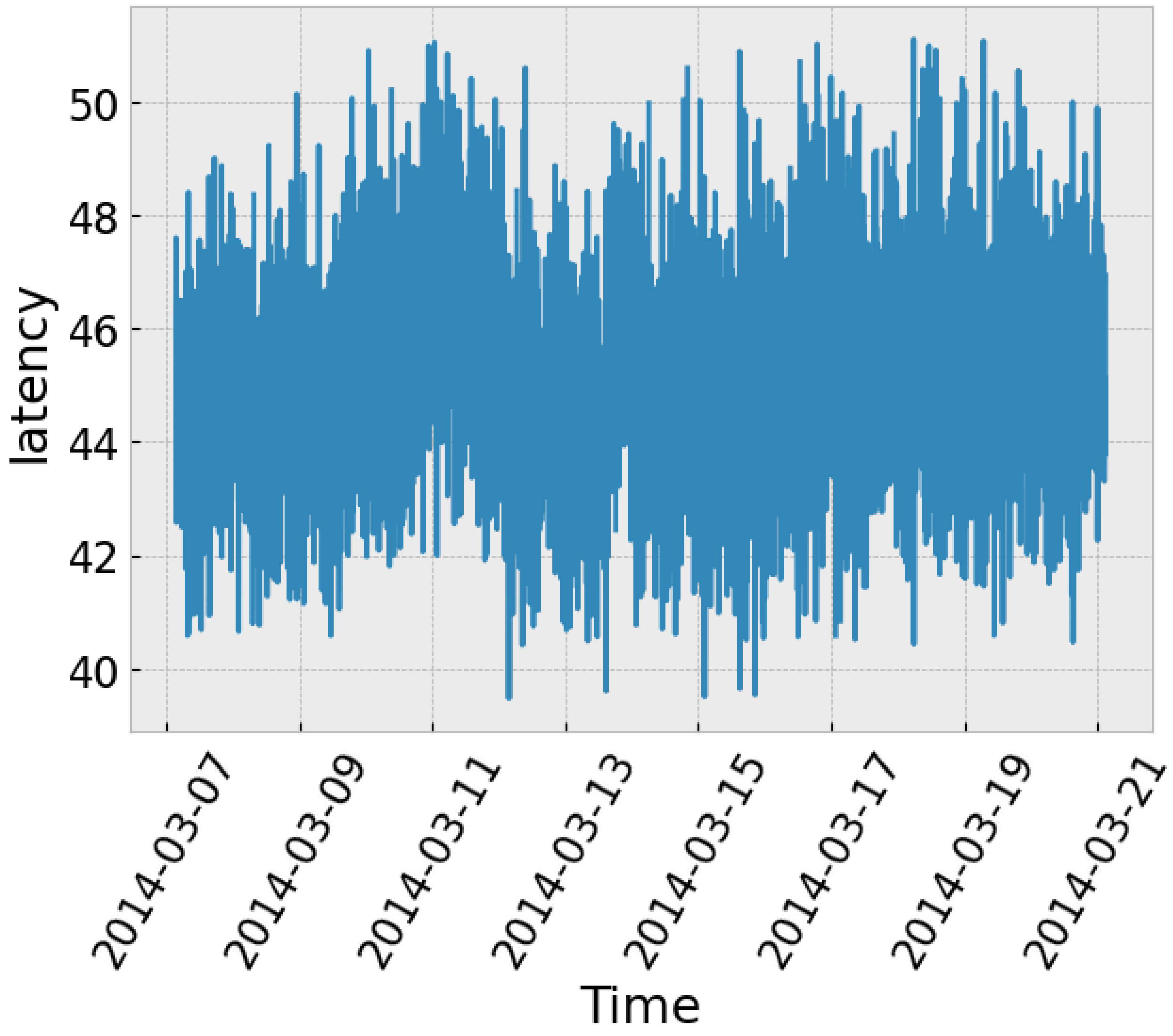
- Imputation refers to the filling of missing values or outliers
- Sometimes missing values/outlier actually provide useful information about the process that produced the data, so you should be careful when imputing them
- However, if you do have genuine errors in the data then imputing them can make the model-building process and forecasting better ea to drop NA in time-series data?

- There are a few over-arching techniques we can use for imputation:
    1. Remove (`.dropna()`)
    2. Fill manually based on some expert-interpreted values (`.fillna()`)
    3. Fill naively using, e.g., mean, median, last observed value, random value
    4. Fill based on a rolling statistic, e.g., moving mean or median
    5. Polynomial interpolation (linear is often a good place to start unless you have a good reason to use a higher order)
    6. Fill based on temporal dependence ("temporal imputation"), i.e., use the same value from the same period last season, or the average of all periods in past seasons (e.g., to fill a January, use last January, or the average of all past January's)
    7. Fill with model fitted values
    8. More advanced methods leveraging multiple imputation, expectation maximization, etc. `fancyimpute` was the go-to package for advanced imputation in Python for many years, but now much of its core functionality has been included in `sklearn`,

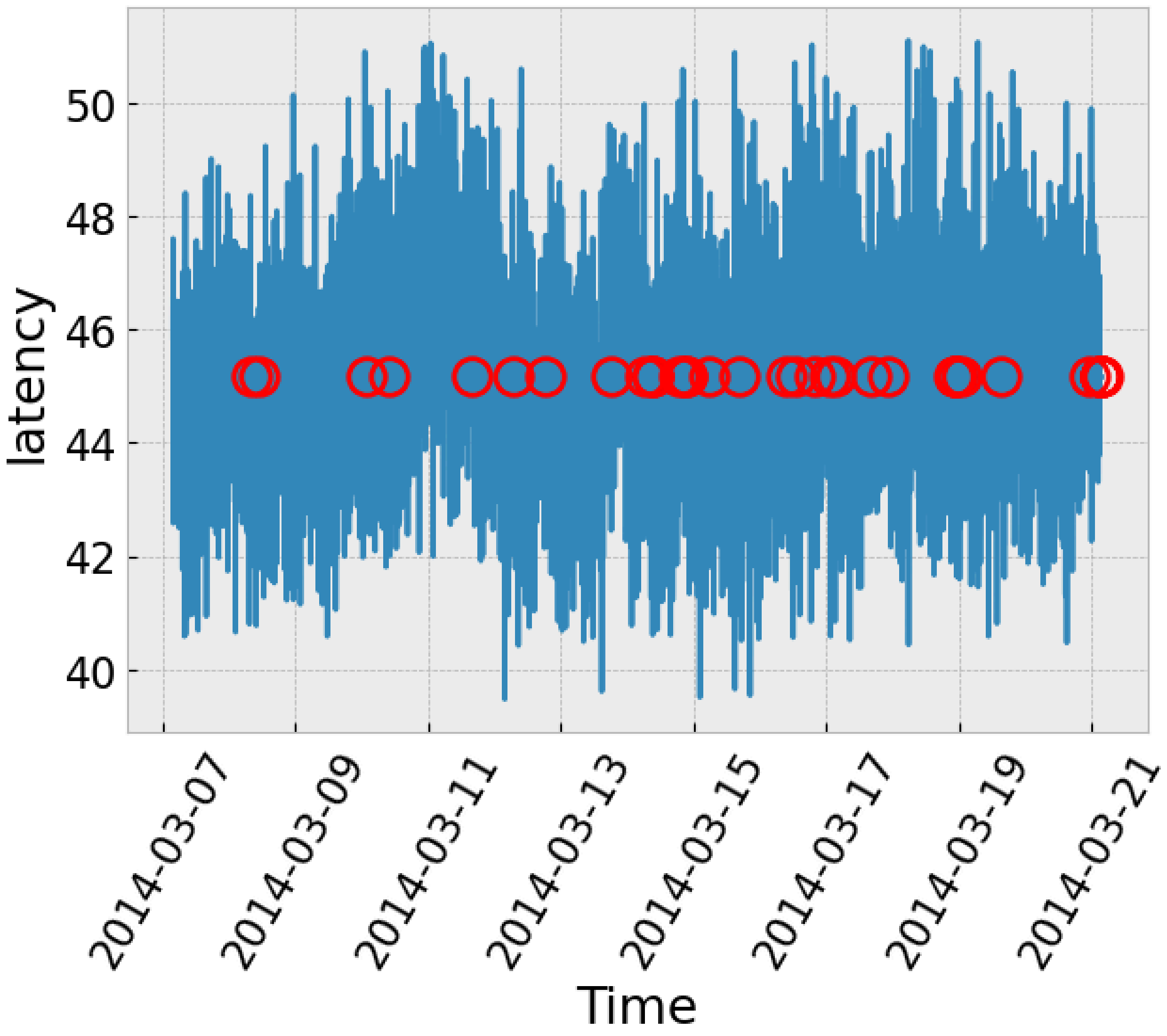
mostly in the `IterativeImputer` class. A common algorithm `MICE` is also available in `statsmodels`

- For advanced reading on imputation, see Flexible Imputation of Missing Data. For the most part, I find imputation to be a field and data set specific task.
- Let's see a few quick example:

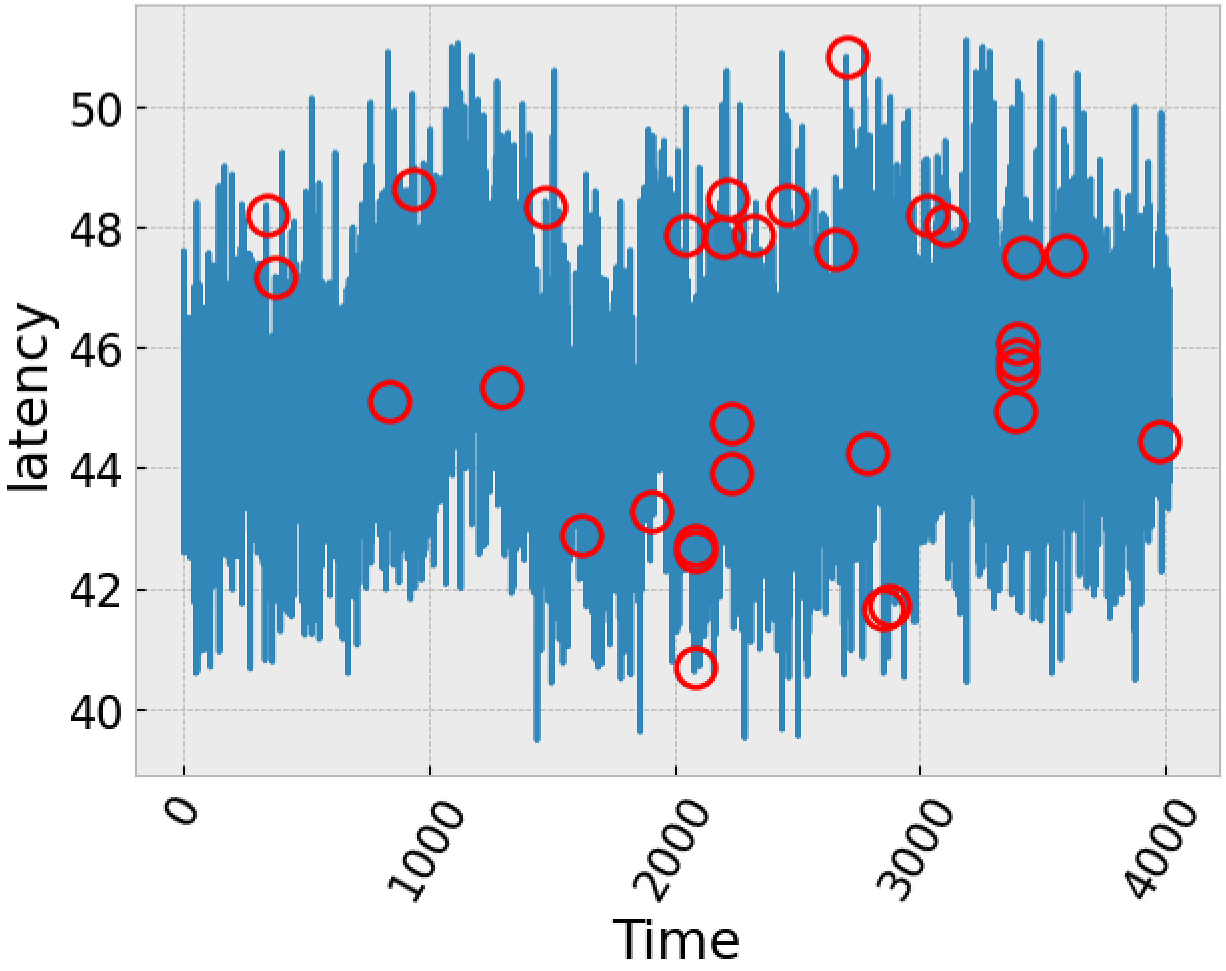
```
df.iloc[outliers] = np.nan  # set outliers to NaN
# Plot
def impute_plot(df, outliers):
    fig = plt.figure(figsize=(7, 5))
    plt.plot(df)
    plt.plot(df.iloc[outliers], 'or', ms=14, mfc="None", mew=2)
    plt.xlabel("Time"), plt.xticks(rotation=60), plt.ylabel("latency")
impute_plot(df, outliers)
```



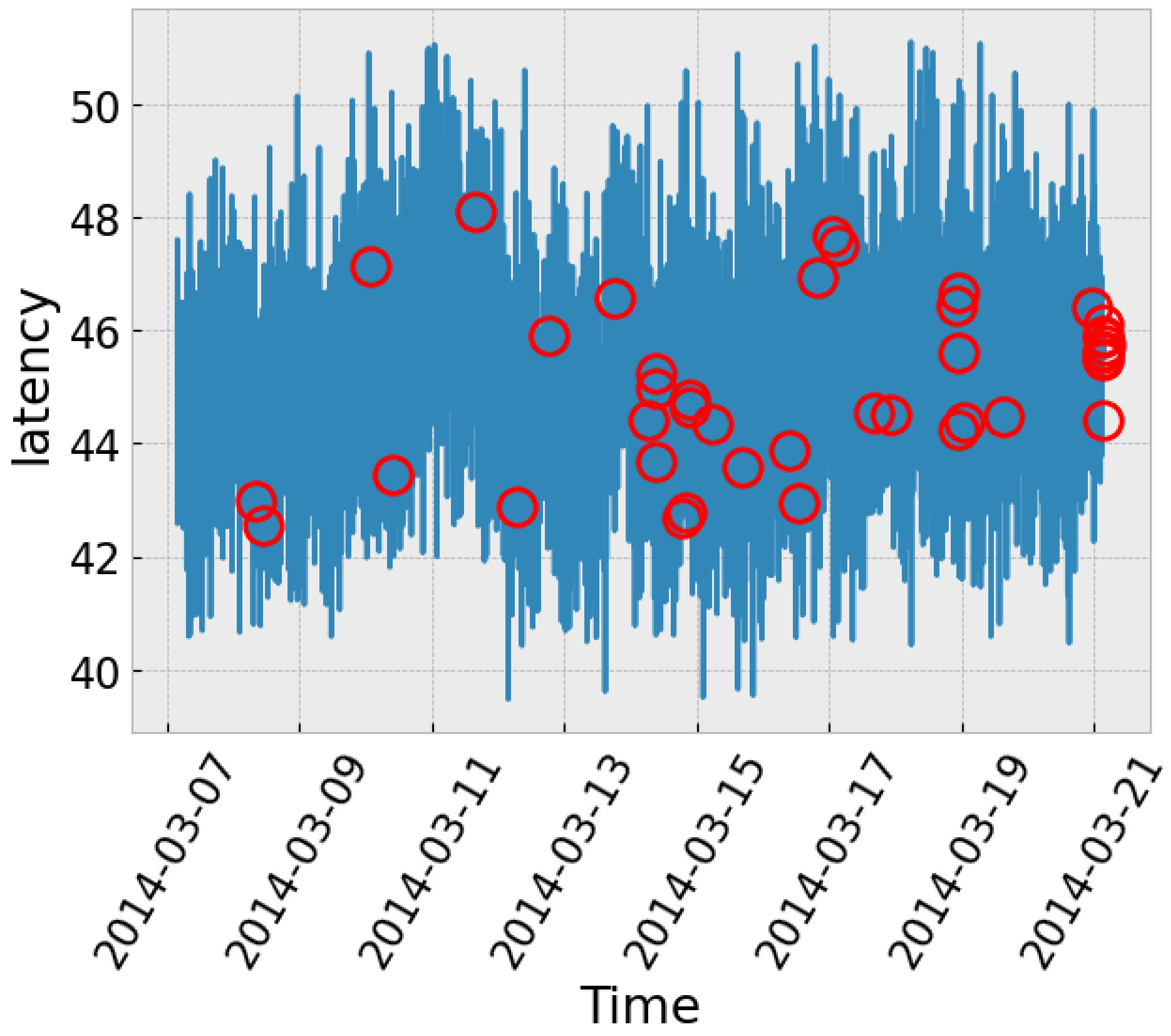
```
# mean imputation
impute_plot(df.fillna(df.mean()), outliers)
```



```
# df[df.duplicated()]
# polynomial interpolation
impute_plot(df.reset_index(drop=True).interpolate(method="polynomial", order=2
```



```
warnings.filterwarnings('ignore')

# arima handles missing values implicitly using the statespace framework
# via a kalman filter: https://stats.stackexchange.com/a/141385
# model.fittedvalues returns the imputed values using only information availab
# model.filter_results.smoothed_forecasts returns the imputed values using inf

model = ARIMA(df["value"], order=(3, 1, 0)).fit()
impute_plot(pd.Series(model.filter_results.smoothed_forecasts.ravel(), index =
```


50
48
46
44
42
40
latency
2014-03-07
2014-03-09
2014-03-11
2014-03-13
2014-03-15
2014-03-17
2014-03-19
2014-03-21
Time