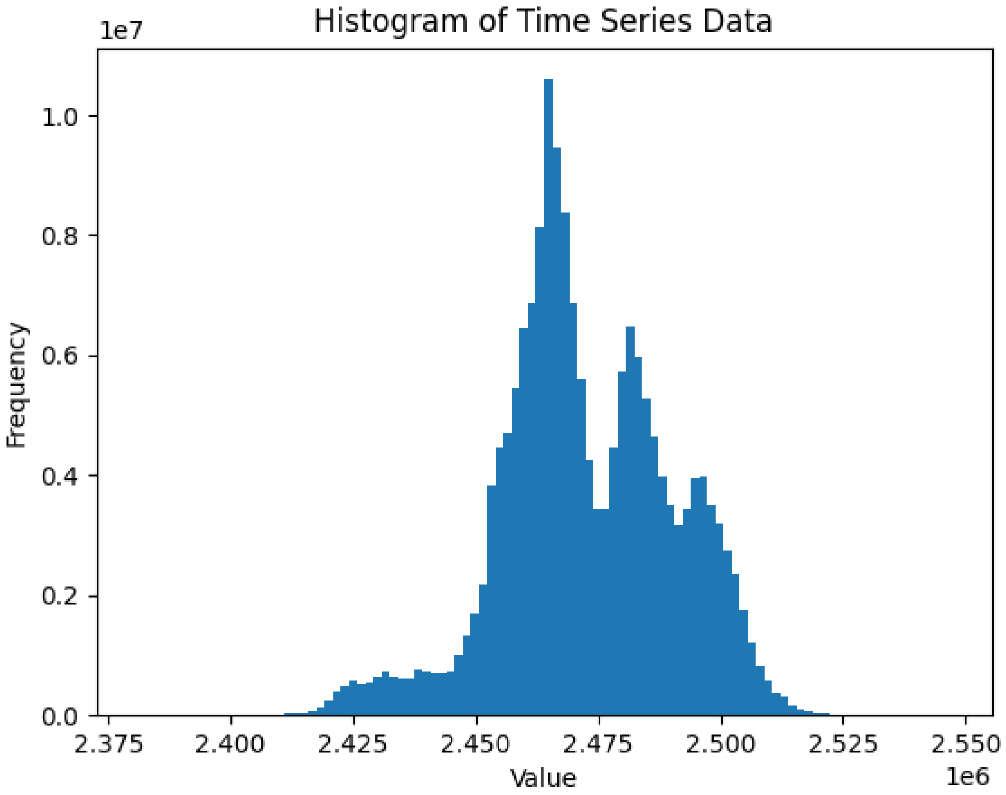
Photoplethysmography Challenge

Juan Montesinos

DATA ANALYSIS

Distribution of the dataset

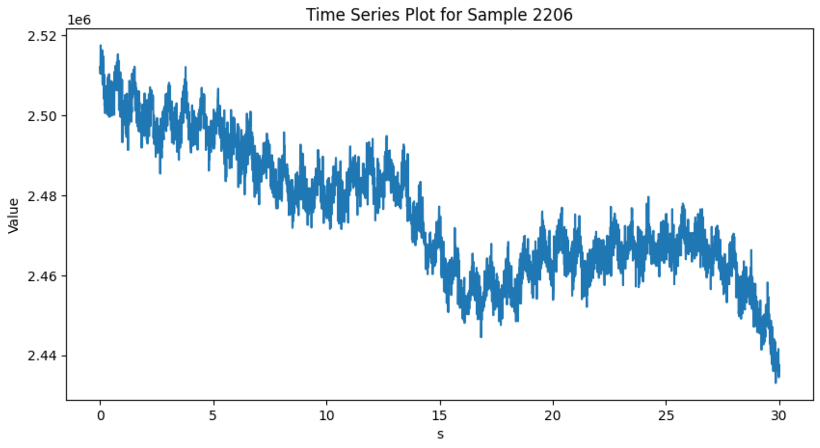


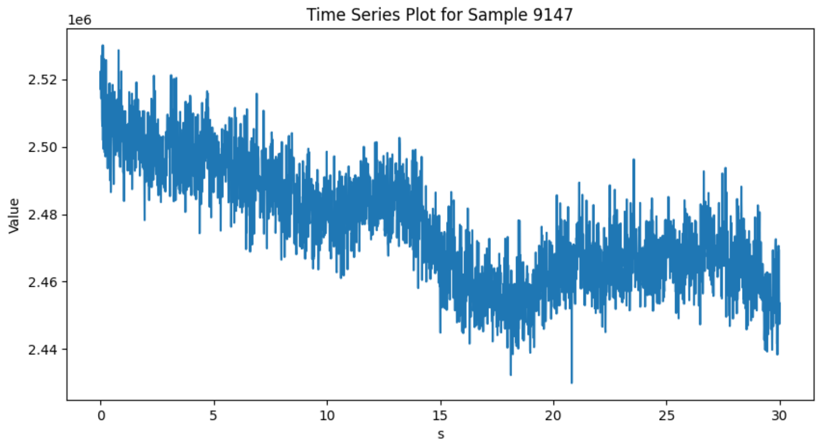
Several populations with certain overlap which could be related to:

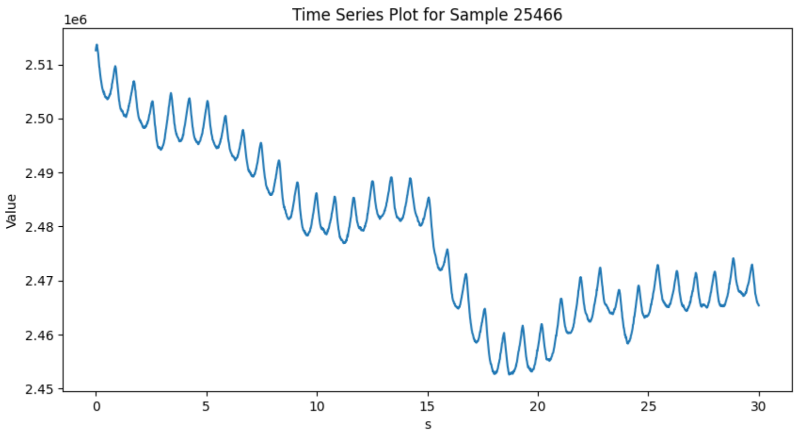
- Data artifacts
- Human modes (running/rest, age)

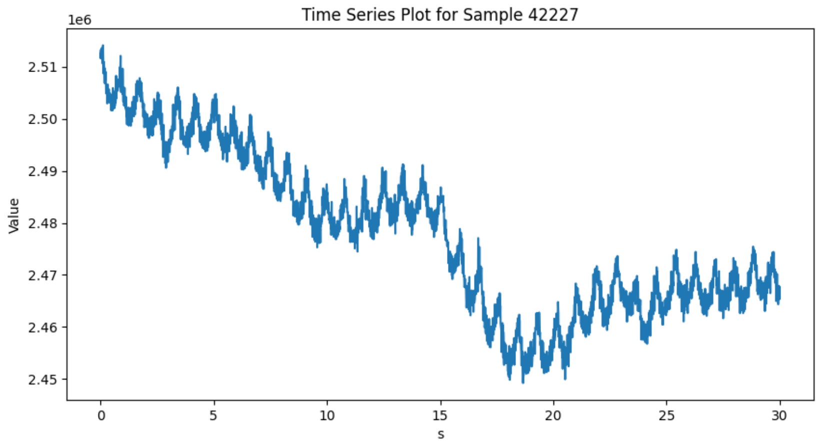
Different levels of onise

Examples of different signals

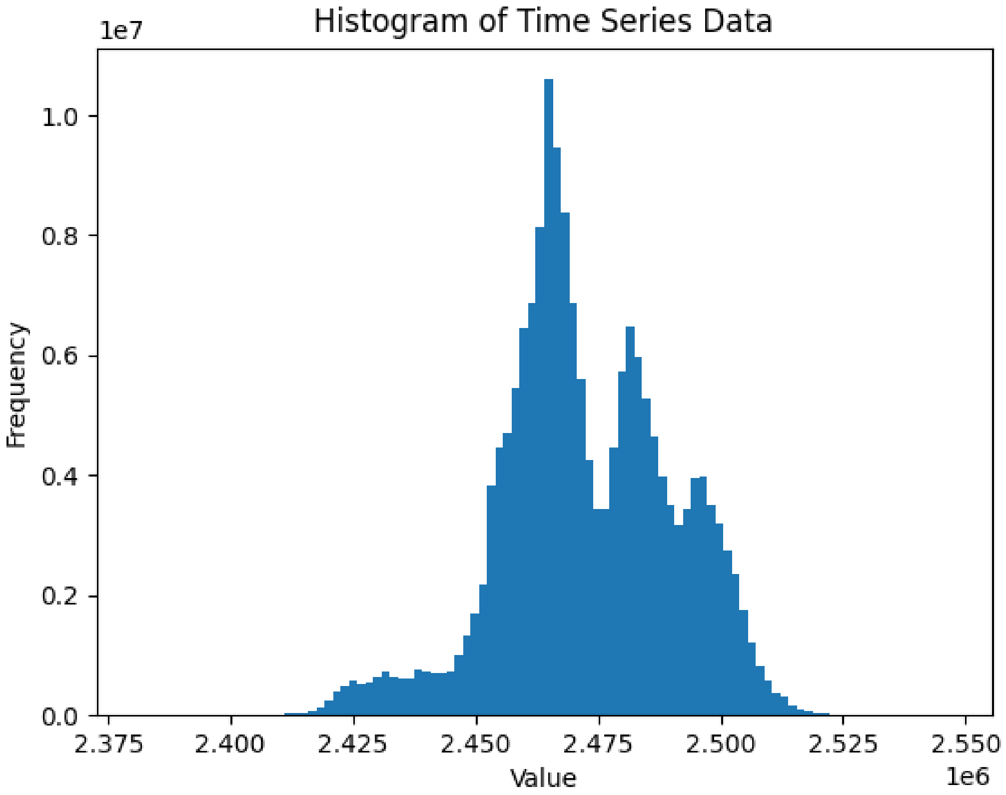








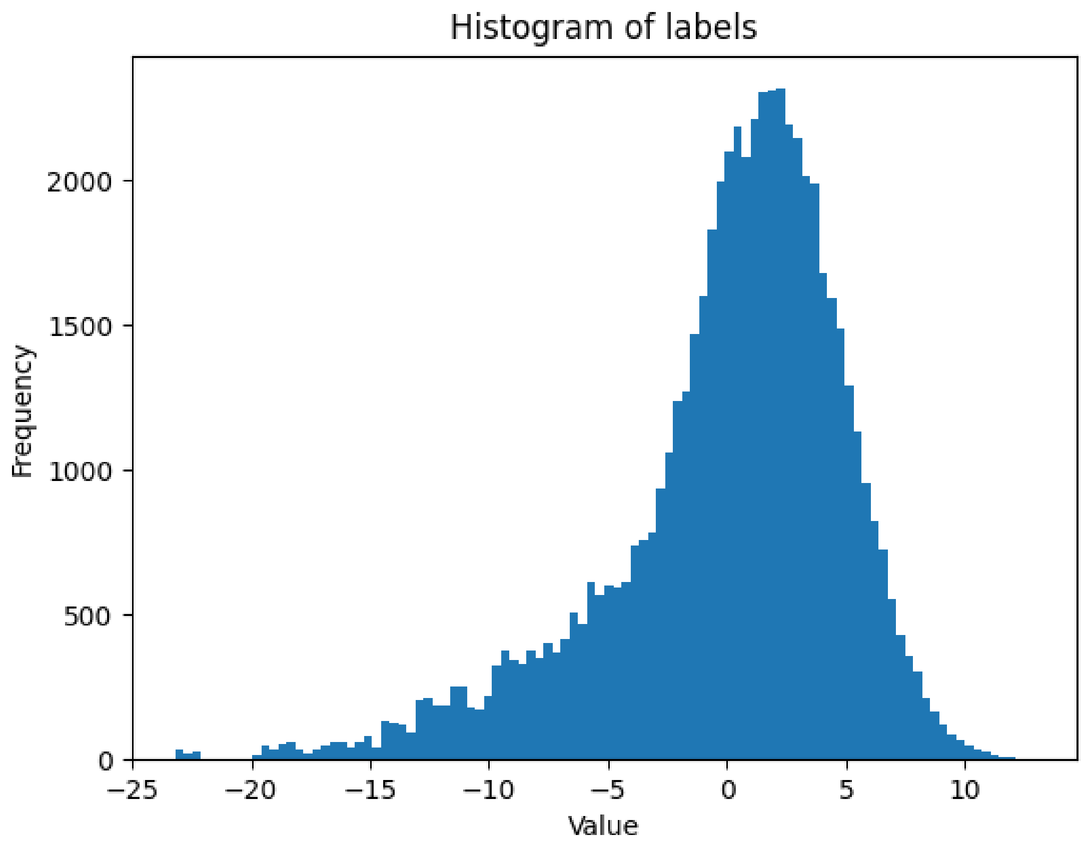
Distribution of the dataset



- At first, we observe several populations with certain overlap which could be related to:
  - Data artifacts
  - Human modes (running/rest, age)

- Different levels of noise:
  - Collection of datasets
  - Different sensors
  - Preprocessed signals
- Signals drifting down.
- Duplicated signals.

Distribution of the labels



Labels follow a skew normal distribution, which is good for our purpose

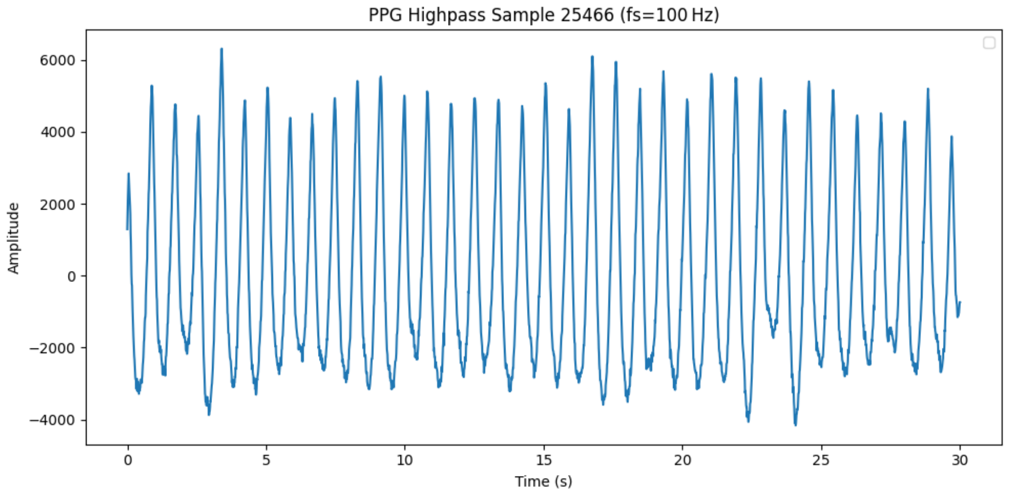
Likewise, the additional features are normally distributed as well:

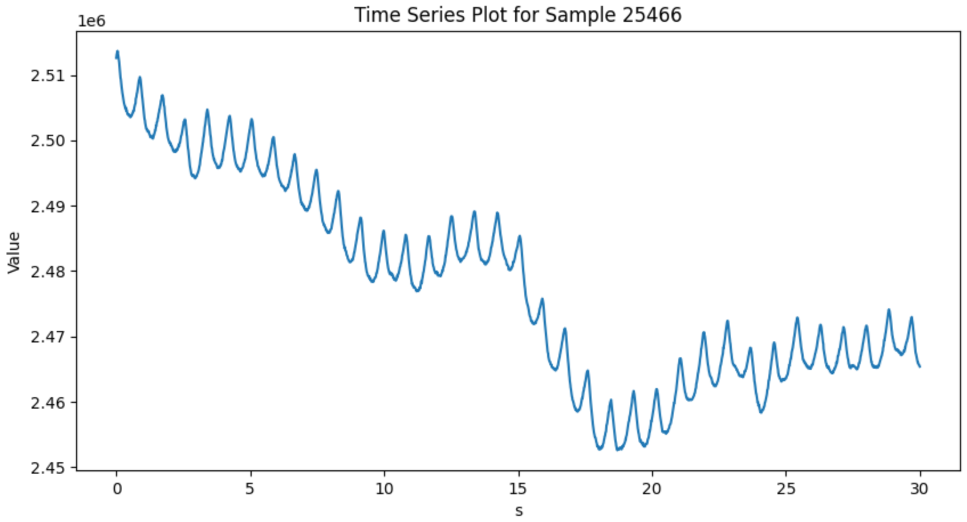
- Data sourced from real world without balancing of any kind.

Why labels follow a normal distribution but not the data?

Remove drift

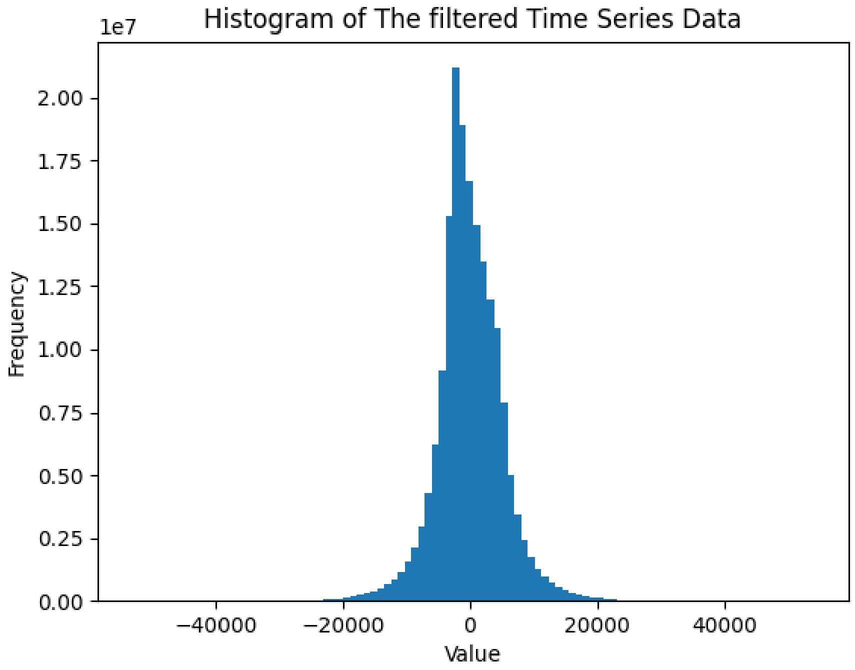
Use a high-pass filter to remove frequencies below 0.5 Hz





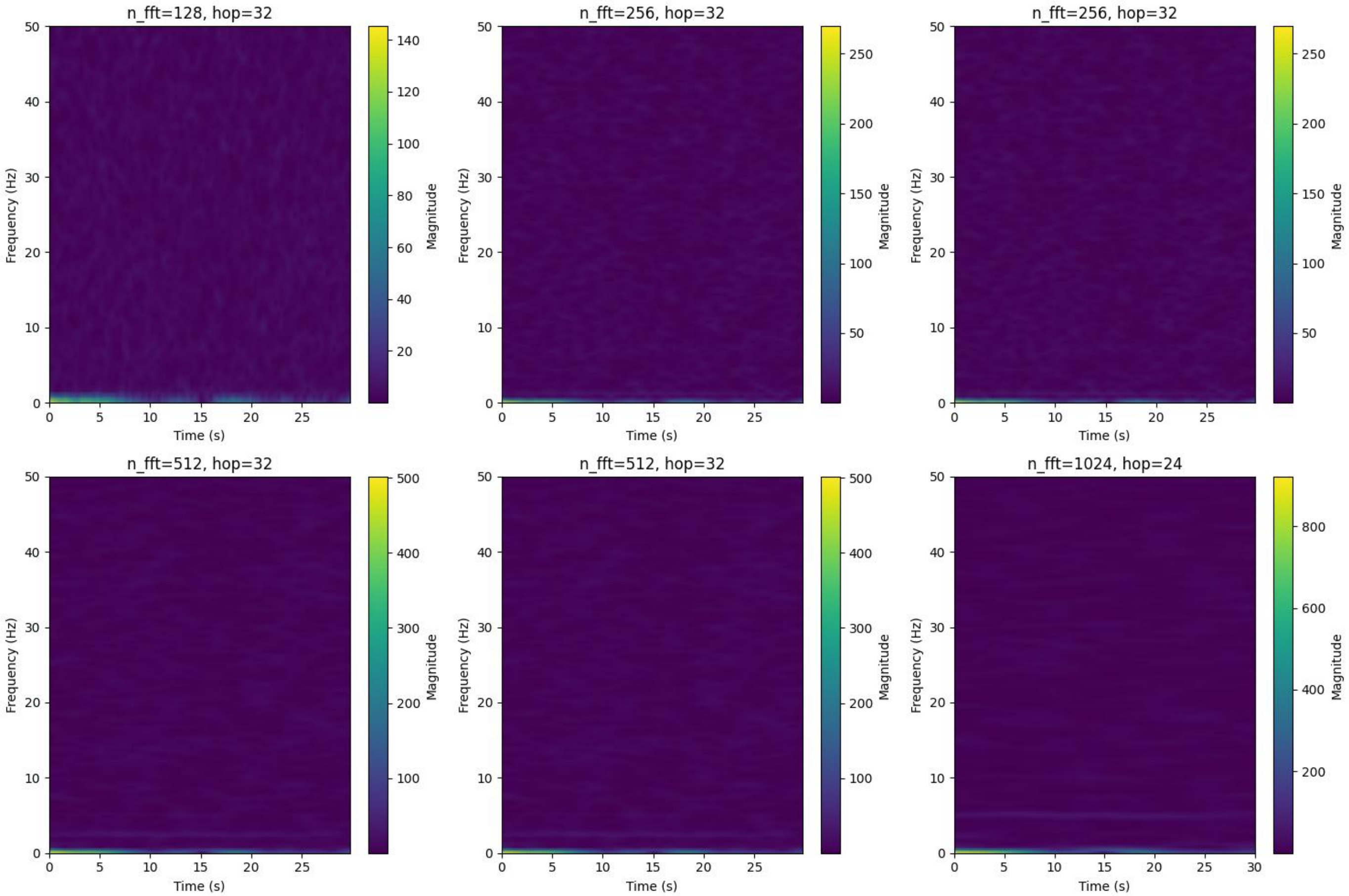
Remove drift

Use a high-pass filter to remove frequencies below 0.5 Hz



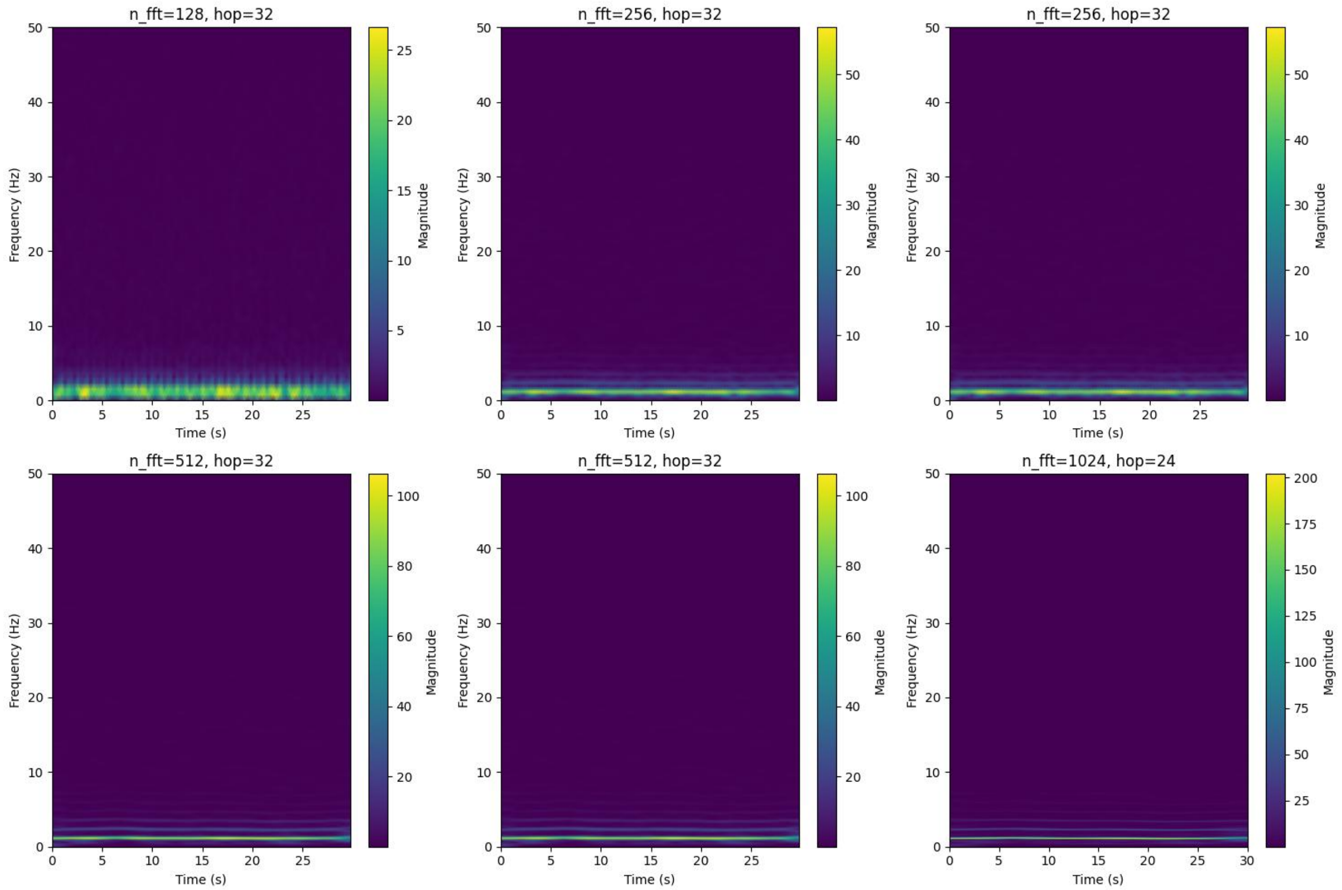
Remove drift

Use a high-pass filter to remove frequencies below 0.5 Hz



Remove drift

Use a high-pass filter to remove frequencies below 0.5 Hz



Model Architecture, Training

```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.n_fft, self.hop = 1024, 24
        window: Tensor = torch.hamming_window(window_length=self.n_fft)
        self.register_buffer(name="window", tensor=window)

        self.block1: Sequential = conv_block(in_channels=1, out_channels=64, kernel_size=7, activation=nn.ReLU(), pooling=nn.MaxPool2d(kernel…2))
        self.block2: Sequential = conv_block(in_channels=64, out_channels=128, kernel_size=7, activation=nn.ReLU(), pooling=nn.MaxPool2d(kernel…2))
        self.block3: Sequential = conv_block(in_channels=128, out_channels=128, kernel_size=3, activation=nn.ReLU(), pooling=nn.MaxPool2d(kernel…2))
        self.block4: Sequential = conv_block(in_channels=128, out_channels=256, kernel_size=3, activation=nn.ReLU(), pooling=nn.MaxPool2d(kernel…2))
        self.block5: Sequential = conv_block(
            in_channels=256,
            out_channels=256,
            kernel_size=3,
            activation=nn.ReLU(),
            pooling=nn.AdaptiveMaxPool2d(output_size=1),
        )

        self.fc_ts = nn.Sequential(nn.Linear(in_features=256, out_features=128), nn.ReLU())

        self.feats_encoder = nn.Sequential(
            nn.Linear(in_features=5, out_features=64),
            nn.ReLU(),
            nn.Linear(in_features=64, out_features=128),
            nn.ReLU(),
        )

        self.fc1 = nn.Sequential(nn.Linear(in_features=256, out_features=128), nn.ReLU())
        self.fc2 = nn.Sequential(nn.Linear(in_features=128, out_features=64), nn.ReLU())
        self.fc_regression = nn.Linear(in_features=64, out_features=1)

    def forward(self, ts, feats):
        # Process time series STFT -> conv blocks
        sp: Tensor = (      You, 2 days ago • Solution
            torch.stft(
                input=ts,
                n_fft=self.n_fft,
                hop_length=self.hop,
                window=self.window,
                return_complex=True,
            )
            .abs()
            .unsqueeze(dim=1)
        )
        sp: Tensor = sp[:, :, :126]  # Crop frequencies ~0-12 Hz
        x: Any = self.block1(sp)
        x: Any = self.block2(x)
        x: Any = self.block3(x)
        x: Any = self.block4(x)
        x: Any = self.block5(x).squeeze()
        ts_emb: Any = self.fc_ts(x)

        # Process extra features
        feats_emb: Any = self.feats_encoder(feats)

        # Combine and regress
        shared: Tensor = torch.cat(tensors=(ts_emb, feats_emb), dim=1)
        shared: Any = self.fc1(shared)
        shared: Any = self.fc2(shared)
        out: Any = self.fc_regression(shared)
        return out[:, 0]
```

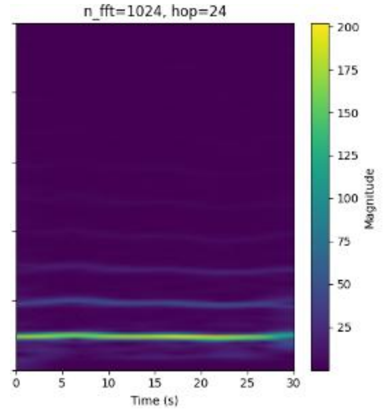
Network inputs:

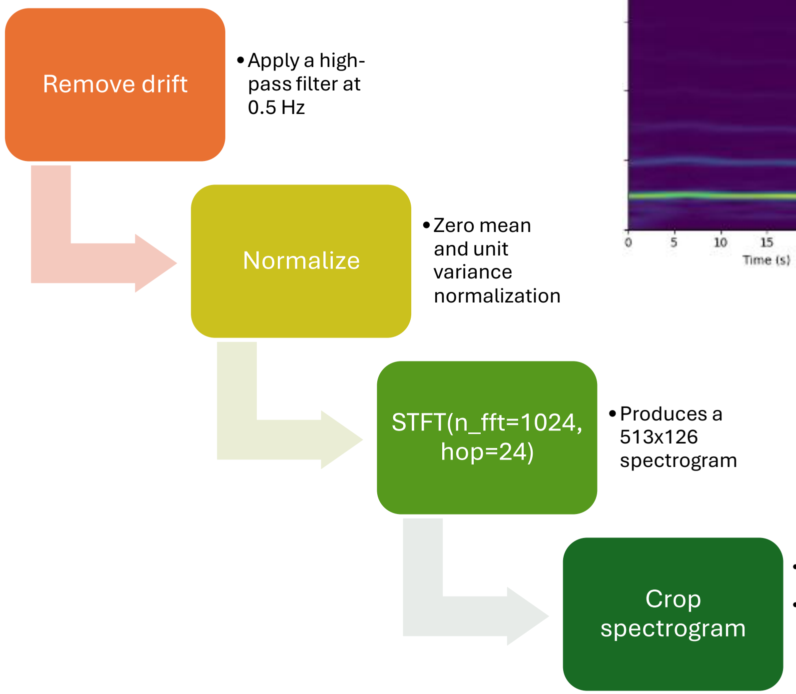
- PPG as spectrograms*
- Features as they are

Predicts:

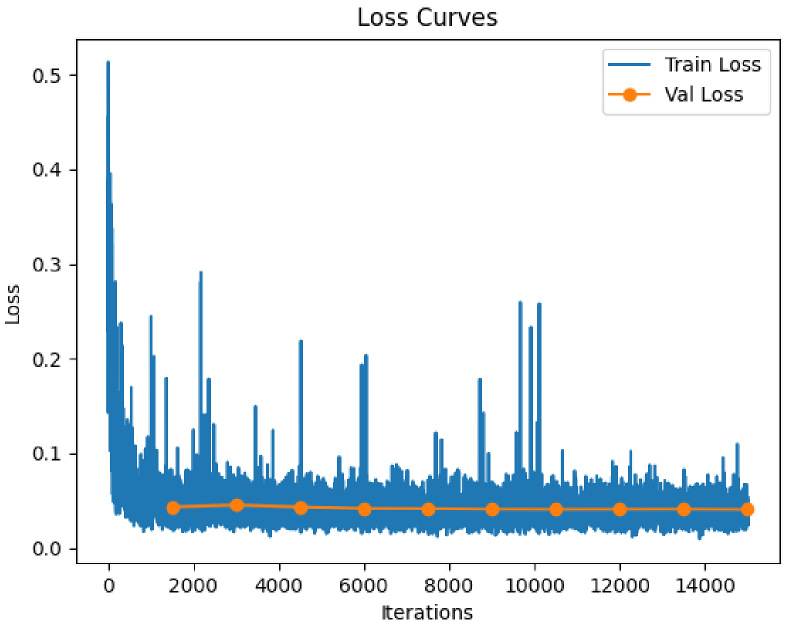
- Regression of z-score "labels" variable

Pre-processing:

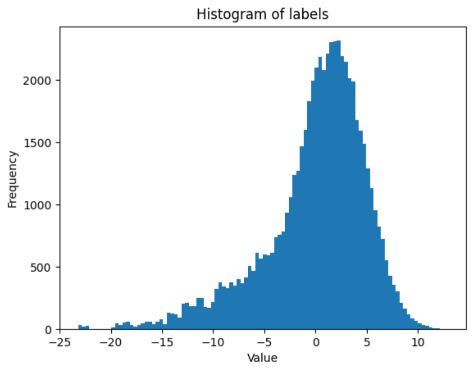




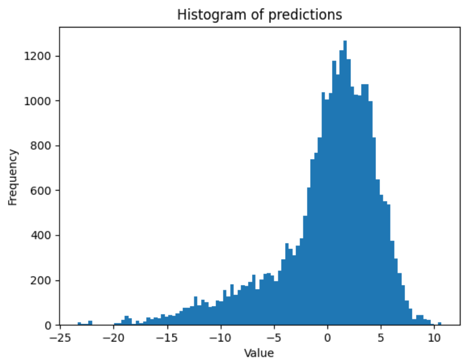
- Optimizer: SGD
- 80/20 train/val split
- 10 epochs
- Smooth L1 loss
- Scheduler: reducing lr 20% ever/epoch



Sanity check: Comparing train label vs test pred distribution



Histogram of training labels



Histogram of predictions for the testset

	Mean Abs.Error
No PPG, only features	2.3
Only PPG, No features	3.05
PPG + Features	1.08

Best mean absolute error for the validation set achieved during training.