National Research University "Higher School of Economics"

Master of Data Science Online Programme

Master's Thesis

Topic: Electricity Spot Prices Forecasting Using Stochastic Volatility Models

Computations Notebook

Student: Andrei Batyrov (Fall2022)

Date: 09-May-2024

Table of Contents

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import TimeSeriesSplit, cross_validate
from sklearn.cluster import KMeans
from sklearn.metrics import root_mean_squared_error, mean_absolute_error
from statsmodels.tsa.stattools import adfuller, acf, pacf
from scipy.stats import norm, probplot, mannwhitneyu
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from stan_model import StanModel
# This is needed to solve the problem with SSL certificates from www.atsenergo.ru
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

---

1. Description

There are several approaches to modeling and forecasting time series as applied to prices of services, commodities, and derivative instruments. One of the approaches is to model the price as a heteroscedastic process with changing volatility (variance of price) over time.

In our scenario we will consider a typical Stochastic Volatility model that treats the volatility as a latent stochastic process in discrete time (Kim, Shephard, and Chib 1998) Kim, Sangjoon, Neil Shephard, and Siddhartha Chib. 1998. "Stochastic Volatility: Likelihood Inference and

Comparison with ARCH Models." Review of Economic Studies 65: 361–93.

Back

---

2. SV Baseline Model

First, we will examine an **SV Baseline** model. This model can be described as a set of regression-like equations, with the following 4 parameters [Kim, Stan user's guide]:

- μ, mean log volatility
- ϕ, persistence of volatility
- σ, white noise shock scale
- h_t, latent log volatility at time t

The variable ϵ_t represents the white-noise shock (i.e., multiplicative error) on the price at time t, whereas δ_t represents the shock on volatility at time t:

$\epsilon_t \sim \mathcal{N}(0,1)$; $\delta_t \sim \mathcal{N}(0,1)$.

$y_t = e^{\frac{h_t}{2}}\epsilon_t$, where

$h_t = \mu + \phi(h_{t-1} - \mu) + \delta_t\sigma$;

$$h_1 \sim \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{1-\phi^2}}\right).$$

To learn these 4 parameters from the price data, we will use the **Stan** platform for statistical modeling and high-performance statistical computation. It uses its own proprietary probabilistic language and can do Bayesian statistical inference, as well as maximum likelihood estimation (MLE) with derivative-based optimization (Newton, BFGS, etc.). https://mc-stan.org/

Other notable probabilistic programming library is PyMC https://www.pymc.io/

Rearranging the equations above yields the following equations to be used in the Stan code for the SV Baseline model:

$y_t \sim \mathcal{N}(0, e^{\frac{h_t}{2}})$, where

$$h_1 \sim \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{1-\phi^2}}\right);$$

$$h_t \sim \mathcal{N}(\mu + \phi(h_{t-1} - \mu), \sigma),$$

2.1. Price Data

There are two price zones: 1 (European) and 2 (Siberian) in the day-ahead spot electricity markets.

1. For building and examining our models, we will load the hourly data for the price zone 1 for the period 01.05.2023 -- 30.04.2024 (one year back from now) for the peak hour.
2. For model cross-validation, we will load and examine both price zones for both peak and off-peak hours with cross-validation over a sliding window for years 23.06.2014 -- 30.04.2024.
3. For forecasting, we will generate predictions for both price zone for both peak and off-peak hours for the period 01.05.2024 -- 07.05.2024 (one week ahead).

The data is available on the AO "ATC" (Администратор Торговой Системы) platform https://www.atsenergo.ru/results/rsv/index, starting from Aug 8th, 2013 to present. The unit of prices is RUB/MWh (руб./МВт·ч).

In [ ]:
```
# # We need to explicitly specify the column names to correctly parse the xml
# price_data = pd.read_xml(f'https://www.atsenergo.ru/market/stats.xml?period=0&date1=20230501&date2=20240507&type=graph',
#                          names=['ROW_ID',
#                                 'DAT',
#                                 'PRICE_ZONE_CODE',
#                                 'CONSUMER_VOLUME',
#                                 'CONSUMER_PRICE',
#                                 'CONSUMER_RD_VOLUME',
#                                 'CONSUMER_SPOT_VOLUME',
#                                 'CONSUMER_PROVIDE_RD',
#                                 'CONSUMER_MAX_PRICE',
#                                 'CONSUMER_MIN_PRICE',
#                                 'SUPPLIER_VOLUME',
#                                 'SUPPLIER_PRICE',
#                                 'SUPPLIER_RD_VOLUME',
```

```
#                             'SUPPLIER_SPOT_VOLUME',
#                             'SUPPLIER_PROVIDE_RD',
#                             'SUPPLIER_MAX_PRICE',
#                             'SUPPLIER_MIN_PRICE',
#                             'HOUR'],
#                     xpath='//row',
#                     parse_dates=['DAT'])
# # Make datetime
# price_data = price_data.set_index(pd.to_datetime(price_data['DAT'].astype(str) + 'T' + price_data['HOUR'].astype(str) + ':00
# price_data.index.name ='Datetime'
# # We can now drop all unnecessary columns to reduce the dataframe
# price_data = price_data.drop(columns=['ROW_ID',
#                           'DAT',
#                           'CONSUMER_VOLUME',
#                           'CONSUMER_RD_VOLUME',
#                           'CONSUMER_SPOT_VOLUME',
#                           'CONSUMER_PROVIDE_RD',
#                           'CONSUMER_MAX_PRICE',
#                           'CONSUMER_MIN_PRICE',
#                           'SUPPLIER_VOLUME',
#                           'SUPPLIER_PRICE',
#                           'SUPPLIER_RD_VOLUME',
#                           'SUPPLIER_SPOT_VOLUME',
#                           'SUPPLIER_PROVIDE_RD',
#                           'SUPPLIER_MAX_PRICE',
#                           'SUPPLIER_MIN_PRICE']).dropna()
# price_data = price_data.loc[~price_data.index.isna()].sort_index()
# price_data['Weekday'] = price_data.index.day_of_week
# price_data.to_csv('./data/price_data_20230501_20240507.csv')
```

In [ ]:
```
price_data = pd.read_csv('./data/price_data_20230501_20240507.zip', index_col='Datetime', parse_dates=['Datetime'])
price_data = price_data.loc[:'30.04.2024'] # 01.05.2024 -- 07.05.2024 is used for forecasting
```

In [ ]:
```
price_data.shape
```

Out[ ]:
```
(17568, 3)
```

In [ ]:
```
price_data.head(4)
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR
2023-05-01 00:00:00	1	1517.92	0
2023-05-01 00:00:00	2	1517.39	0
2023-05-01 01:00:00	2	1512.35	1
2023-05-01 01:00:00	1	1413.99	1

In [ ]:
```
price_data.tail(4)
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR
2024-04-30 22:00:00	2	1465.38	22
2024-04-30 22:00:00	1	1442.53	22
2024-04-30 23:00:00	2	1439.88	23
2024-04-30 23:00:00	1	1327.94	23

In [ ]:
```
price_data.describe()
```

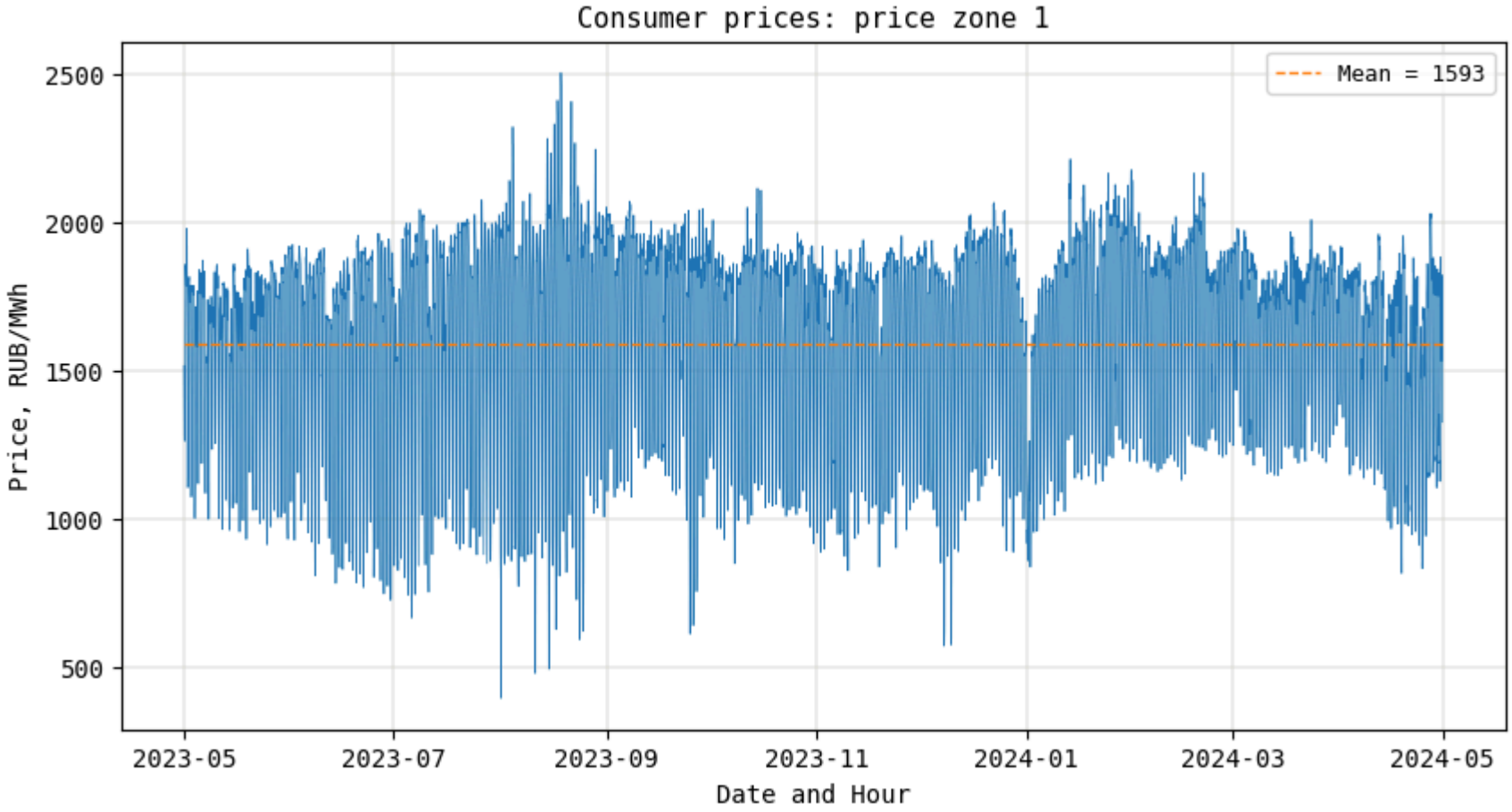
Out[ ]:

	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR
count	17568.000000	17568.000000	17568.000000
mean	1.500000	1424.566452	11.500000
std	0.500014	331.979989	6.922384
min	1.000000	398.480000	0.000000
25%	1.000000	1173.405000	5.750000
50%	1.500000	1394.690000	11.500000
75%	2.000000	1726.827500	17.250000
max	2.000000	2504.960000	23.000000

In [ ]:
```
price_zone = 1
```

In [ ]:
```
# Price zone 1 (European)
price_data_eur = price_data[price_data['PRICE_ZONE_CODE'] == price_zone]
```

In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(price_data_eur.index, price_data_eur['CONSUMER_PRICE'], color='C0', lw=0.5)
plt.hlines(price_data_eur['CONSUMER_PRICE'].mean(), xmin=price_data_eur.index.min(), xmax=price_data_eur.index.max(), color='C
plt.xlabel('Date and Hour', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'Consumer prices: price zone {price_zone}', size=11, family='monospace')
plt.legend(prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```

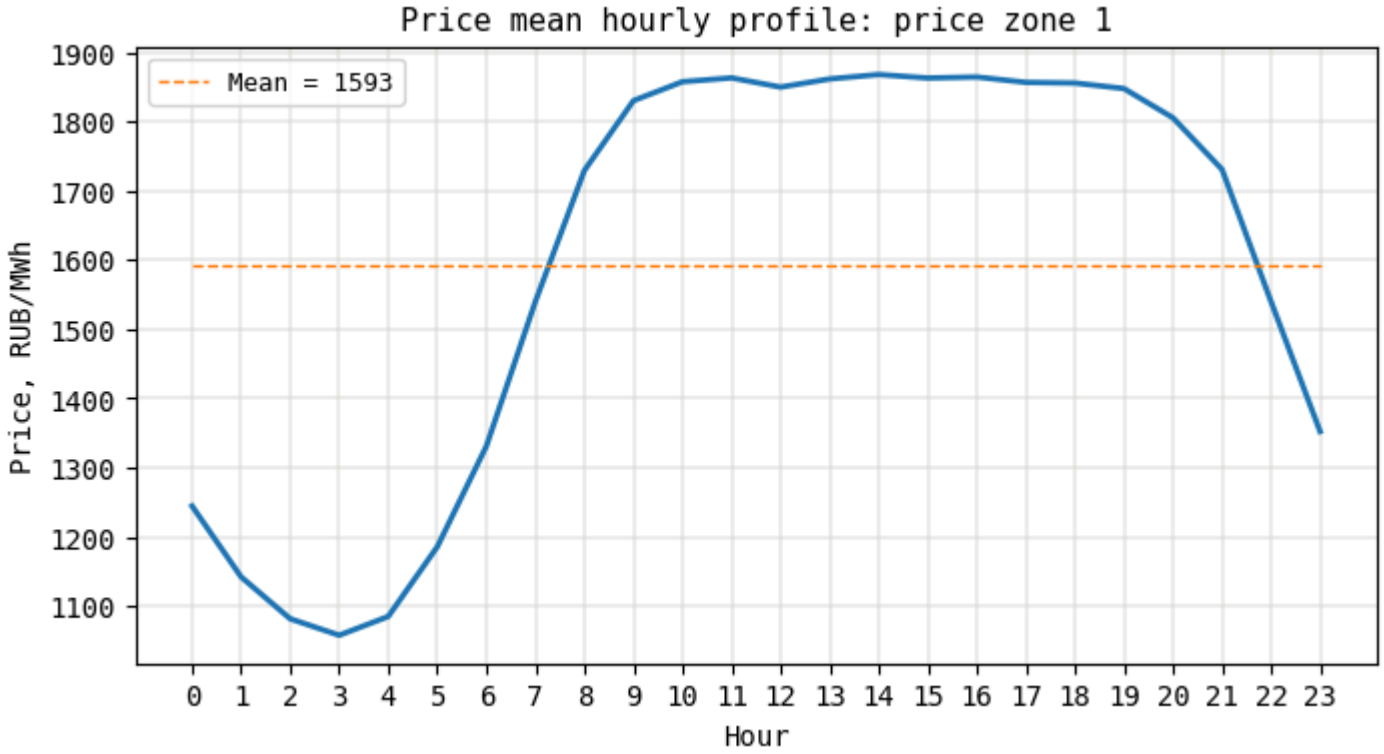


Next, we'll check the hour distribution -- consumer price mean hourly profile.

In [ ]:
```
price_data_daily_agg_eur = price_data_eur.groupby('HOUR')['CONSUMER_PRICE'].mean()
price_data_daily_agg_eur
```

Out[ ]:
```
HOUR
0     1244.706885
1     1142.310683
2     1081.862240
3     1058.018962
4     1085.111284
5     1185.857131
6     1331.013224
7     1540.080082
8     1729.485519
9     1830.166311
10    1857.536230
11    1863.038005
12    1849.733060
13    1861.542568
14    1867.866858
15    1862.852432
16    1864.292022
17    1856.555902
18    1855.611311
19    1847.650683
20    1805.406421
21    1730.848115
22    1539.855820
23    1352.347623
Name: CONSUMER_PRICE, dtype: float64
```

In [ ]:
```
plt.figure(figsize=(8, 4))
plt.plot(price_data_daily_agg_eur.index, price_data_daily_agg_eur, color='C0', lw=2)
plt.hlines(price_data_daily_agg_eur.mean(), xmin=price_data_daily_agg_eur.index.min(), xmax=price_data_daily_agg_eur.index.max
plt.xlabel('Hour', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(price_data_daily_agg_eur.index, size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'Price mean hourly profile: price zone {price_zone}', size=11, family='monospace')
plt.legend(prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```



Let's examine the peak and off-peak hours.

In [ ]:
```
hour_max_eur = price_data_daily_agg_eur.idxmax()
hour_min_eur = price_data_daily_agg_eur.idxmin()
hour_max_eur, hour_min_eur
```

Out[ ]: (14, 3)

In [ ]:
```
price_data_peak_eur = price_data_eur[price_data_eur['HOUR'] == hour_max_eur]
price_data_peak_eur
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR
2023-05-01 14:00:00	1	1803.61	14
2023-05-02 14:00:00	1	1793.39	14
2023-05-03 14:00:00	1	1770.27	14
2023-05-04 14:00:00	1	1674.88	14
2023-05-05 14:00:00	1	1810.13	14
...	...	...	...
2024-04-26 14:00:00	1	1862.04	14
2024-04-27 14:00:00	1	2025.49	14
2024-04-28 14:00:00	1	1810.51	14
2024-04-29 14:00:00	1	1779.80	14
2024-04-30 14:00:00	1	1553.93	14

366 rows × 3 columns

In [ ]:
```
price_data_peak_eur['CONSUMER_PRICE'].describe()
```

Out[ ]:
```
count     366.000000
mean     1867.866858
std       137.479673
min      1132.110000
25%      1793.745000
50%      1869.885000
75%      1944.390000
max      2504.960000
Name: CONSUMER_PRICE, dtype: float64
```

In [ ]:
```
price_data_offpeak_eur = price_data_eur[price_data_eur['HOUR'] == hour_min_eur]
```

```
price_data_offpeak_eur
```

Out[ ]:

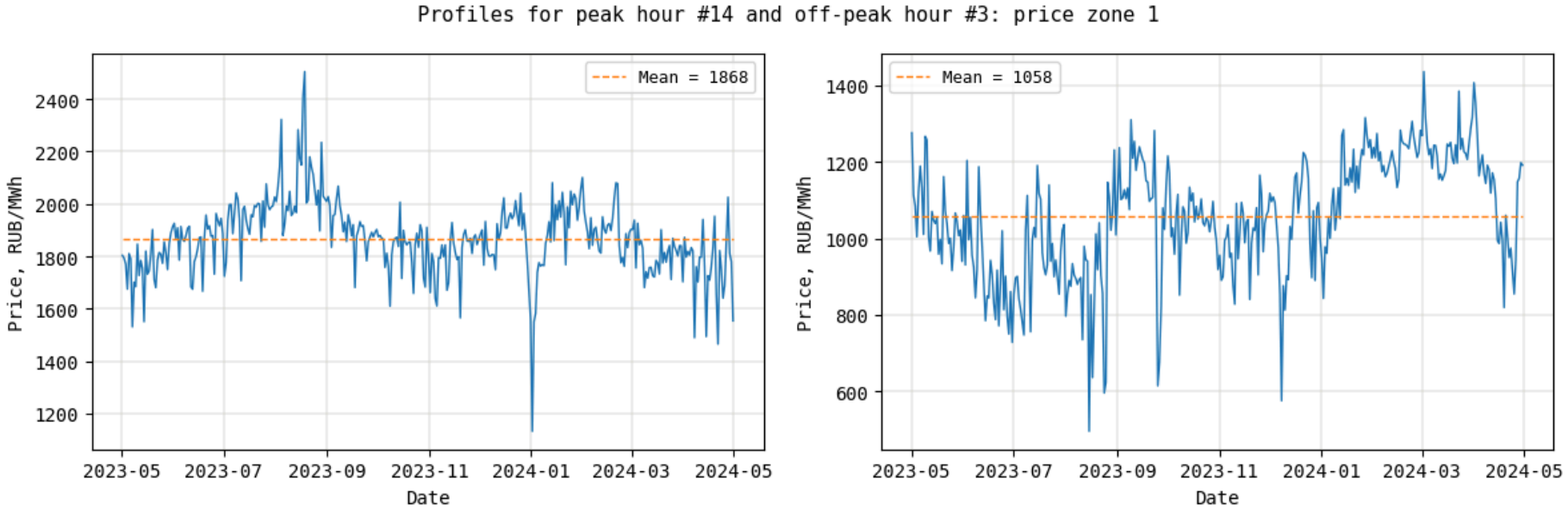
Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR
2023-05-01 03:00:00	1	1275.81	3
2023-05-02 03:00:00	1	1113.04	3
2023-05-03 03:00:00	1	1085.05	3
2023-05-04 03:00:00	1	1003.91	3
2023-05-05 03:00:00	1	1121.00	3
...	...	...	...
2024-04-26 03:00:00	1	944.52	3
2024-04-27 03:00:00	1	1147.59	3
2024-04-28 03:00:00	1	1157.11	3
2024-04-29 03:00:00	1	1197.78	3
2024-04-30 03:00:00	1	1191.04	3

366 rows × 3 columns

In [ ]:
```
price_data_offpeak_eur['CONSUMER_PRICE'].describe()
```

Out[ ]:
```
count     366.000000
mean     1058.018962
std       153.703503
min       495.830000
25%       956.015000
50%      1066.625000
75%      1182.727500
max      1435.760000
Name: CONSUMER_PRICE, dtype: float64
```

In [ ]:
```
plt.figure(figsize=(12, 4))
for i, hour_profile in zip(range(1, 3), [price_data_peak_eur, price_data_offpeak_eur]):
    plt.subplot(1, 2, i)
    plt.plot(hour_profile.index, hour_profile['CONSUMER_PRICE'], color='C0', lw=1)
    plt.hlines(hour_profile['CONSUMER_PRICE'].mean(), xmin=hour_profile.index.min(), xmax=hour_profile.index.max(), color='C1'
    plt.xlabel('Date', size=10, family='monospace')
    plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
    plt.xticks(size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
    plt.legend(prop={'size': 9, 'family': 'monospace'})
    plt.grid(lw=0.25, color='xkcd:cement');
    plt.gca().set_axisbelow(True)
plt.suptitle(f'Profiles for peak hour #{hour_max_eur} and off-peak hour #{hour_min_eur}: price zone {price_zone}', size=11, fa
plt.tight_layout();
```



For further modeling we will consider only the peak hour. During cross-validation we will consider both peak and off-peak hours.

2.2. Test For Stationarity

Apply augmented Dickey-Fuller test for stationarity of price zone 1 (European)

$\mathcal{H}_0$: non-stationary

$\mathcal{H}_1$: stationary

In [ ]:
```
alpha = 0.05
print(f'Peak hour #{hour_max_eur}:', end='    ')
adf_stats, adf_pval, _, _, _, _ = adfuller(price_data_peak_eur['CONSUMER_PRICE'])
print(f'{adf_stats = :.2f}, {adf_pval = :.2f}', end=', ')
print('Stationary') if adf_pval < alpha else print('Non-stationary')
print(f'Off-peak hour #{hour_min_eur}:', end=' ')
adf_stats, adf_pval, _, _, _, _ = adfuller(price_data_offpeak_eur['CONSUMER_PRICE'])
print(f'{adf_stats = :.2f}, {adf_pval = :.2f}', end=', ')
print('Stationary') if adf_pval < alpha else print('Non-stationary')
```

```
Peak hour #14:    adf_stats = -2.56, adf_pval = 0.10, Non-stationary
Off-peak hour #3: adf_stats = -2.71, adf_pval = 0.07, Non-stationary
```

2.3. Modeling

The mean of y_t is modeled as 0 in the original model, which means that we have to either centralize the data to make it oscillate around 0 or change this parameter to reflect our data's real mean, which yields the following final SV Baseline model:

$y_t \sim \mathcal{N}(\bar{y}, e^{\frac{h_t}{2}})$.

We have created a custom wrapper class inheriting from the scikit-learn's `BaseEstimator` and `RegressorMixin` classes to work with our Stan models using the scikit-learn's interfaces: `fit()`, `predict()`, `score()`, `cross_validate()` etc.

In [ ]:
```
# SV Baseline model
with open('./models/sv_base_fit.stan', 'r') as fh:
  sv_base_code_fit = fh.read()
with open('./models/sv_base_predict.stan', 'r') as fh:
  sv_base_code_predict = fh.read()
#print(sv_base_code_fit)
#print(sv_base_code_predict)
num_samples = 1_000
model_sv_base_peak_eur = StanModel(kind='sv_base',
                                   name='SV Baseline',
                                   stan_code_fit=sv_base_code_fit,
```

```
                                    stan_code_predict=sv_base_code_predict,
                                    num_samples=num_samples)
```

In [ ]:
```
%%capture
# Learn parameters
model_sv_base_peak_eur.fit(X=price_data_peak_eur['CONSUMER_PRICE'], y=price_data_peak_eur['CONSUMER_PRICE'])
```

In [ ]:
```
model_sv_base_peak_eur
```

Out[ ]:

StanModel

```
StanModel(kind='sv_base', name='SV Baseline',
          stan_code_fit='// SV Baseline model\n'
                        '// Expected value: mean of y (constant)\n'
                        '// Volatility: stochastic\n'
                        '\n'
                        'data\n'
                        '{\n'
                        '  int<lower=1> N; // Number of train time points '
                        '(equally spaced)\n'
```

In [ ]:
```
fit_sv_base_peak_eur_df = model_sv_base_peak_eur.fit_result_df_
fit_sv_base_peak_eur_df
```

Out[ ]:

parameters draws	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__	mu	phi	sigma	...
0	-2064.475983	0.627137	0.156116	5.0	31.0	0.0	2270.533581	9.073178	0.838394	0.644606	...
1	-2089.379291	0.790054	0.156116	5.0	31.0	0.0	2272.048390	9.284743	0.856929	0.395702	...
2	-2065.733356	0.916532	0.156116	5.0	31.0	0.0	2267.592460	9.482998	0.888277	0.498436	...
3	-2055.273206	1.000000	0.156116	5.0	31.0	0.0	2238.480078	9.277159	0.831822	0.540493	...
4	-2077.763789	0.832592	0.156116	5.0	31.0	0.0	2235.341095	9.075719	0.863893	0.476529	...
...	...	...	...	...	...	...	...	...	...	...	...
995	-2068.737688	1.000000	0.156116	5.0	31.0	0.0	2230.214407	9.062865	0.814645	0.552549	...
996	-2076.354493	0.999004	0.156116	5.0	31.0	0.0	2256.319385	8.883047	0.863724	0.709085	...
997	-2048.724829	0.980658	0.156116	5.0	31.0	0.0	2251.765829	8.890620	0.917324	0.412934	...
998	-2110.670744	0.716123	0.156116	5.0	31.0	0.0	2264.698081	8.937723	0.828389	0.508291	...
999	-2077.668532	0.981206	0.156116	5.0	31.0	0.0	2289.764979	8.864828	0.913017	0.398243	...

1000 rows × 743 columns

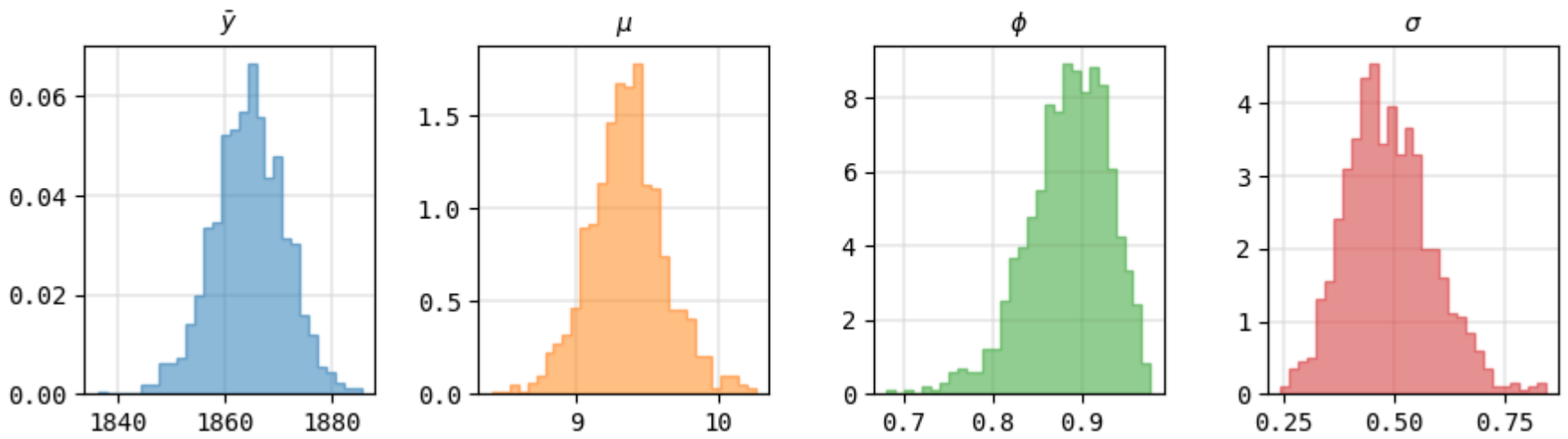
In [ ]:
```
# Learned parameters
fit_sv_base_peak_eur_df[['y_mean', 'mu', 'phi', 'sigma']].describe()
```

Out[ ]:

parameters	y_mean	mu	phi	sigma
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	1864.385372	9.355585	0.883679	0.487833
std	6.673941	0.270878	0.045556	0.097786
min	1836.074242	8.404077	0.681349	0.243562
25%	1860.013375	9.188929	0.857151	0.419125
50%	1864.400904	9.353287	0.887945	0.479947
75%	1869.048063	9.513891	0.917030	0.549931
max	1885.505842	10.269356	0.976029	0.842756

In [ ]:

```
plt.figure(figsize=(9, 3))
for i, param in zip(range(1, 5), fit_sv_base_peak_eur_df[['y_mean', 'mu', 'phi', 'sigma']].describe().columns):
    plt.subplot(1, 4, i)
    plt.hist(fit_sv_base_peak_eur_df[param], bins=30, density=True, histtype='stepfilled', color=f'C{i - 1}', edgecolor=f'C{i
    plt.xticks(size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
    if param == 'y_mean':
        plt.title('$\\bar{y}$', size=10, family='monospace')
    else:
        plt.title(f'$\{param}$', size=10, family='monospace')
    plt.grid(lw=0.25, color='xkcd:cement')
    plt.gca().set_axisbelow(True)
plt.suptitle(f'Learned parameters for peak hour #{hour_max_eur}: {model_sv_base_peak_eur.name}', size=11, family='monospace')
plt.tight_layout();
```



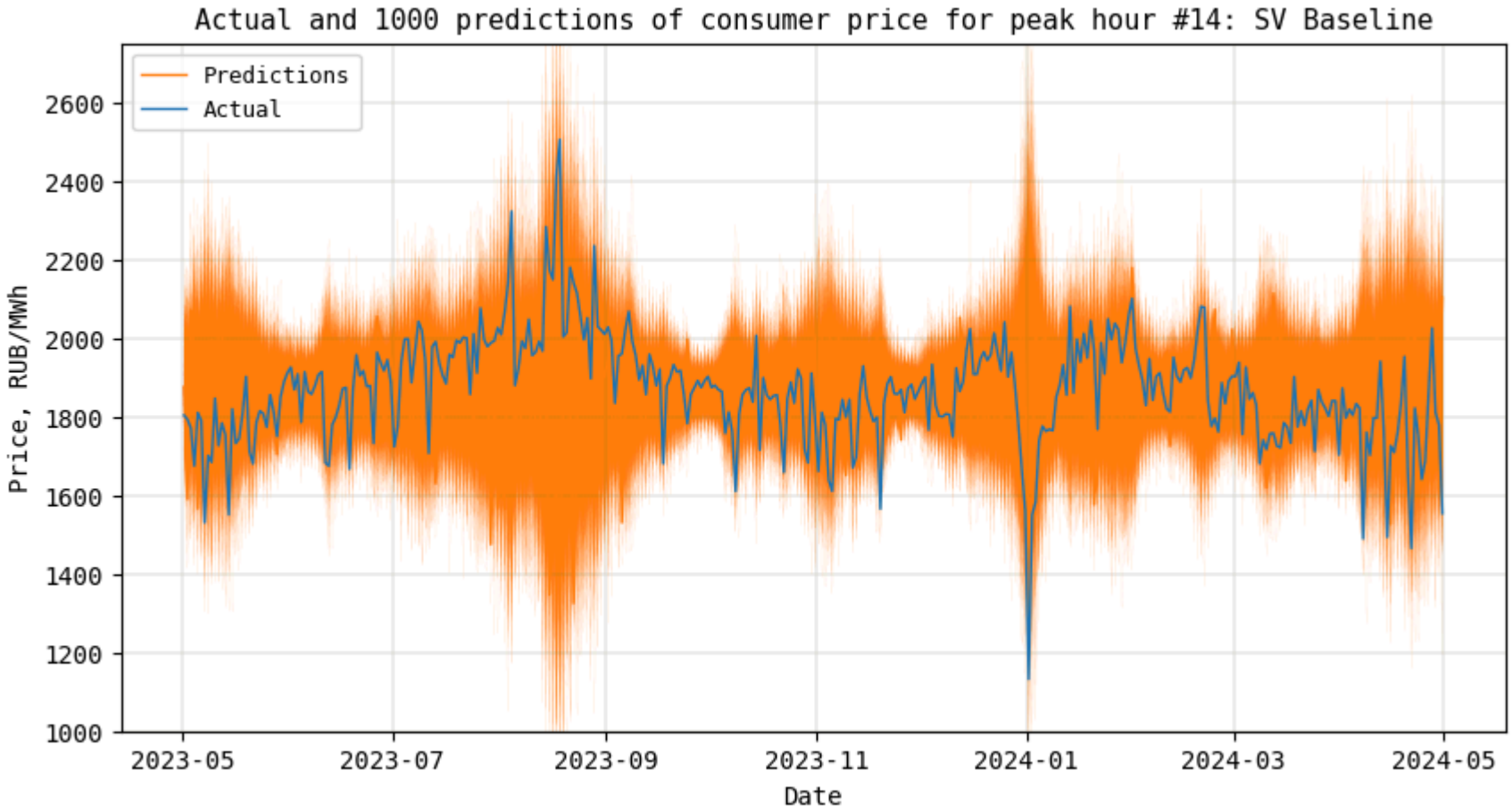
In [ ]:
```
%%capture
# Predict
predict_sv_base_peak_eur_mean = model_sv_base_peak_eur.predict(price_data_peak_eur['CONSUMER_PRICE'])
predict_sv_base_peak_eur_mean.shape
```

In [ ]:
```
%%capture
predict_sv_base_peak_eur_many = model_sv_base_peak_eur.predict_many(price_data_peak_eur['CONSUMER_PRICE'])
predict_sv_base_peak_eur_many.shape
```

In [ ]:
```
%%capture
predict_sv_base_peak_eur_ci = model_sv_base_peak_eur.predict_ci(price_data_peak_eur['CONSUMER_PRICE'], [2.5, 97.5])
len(predict_sv_base_peak_eur_ci)
```

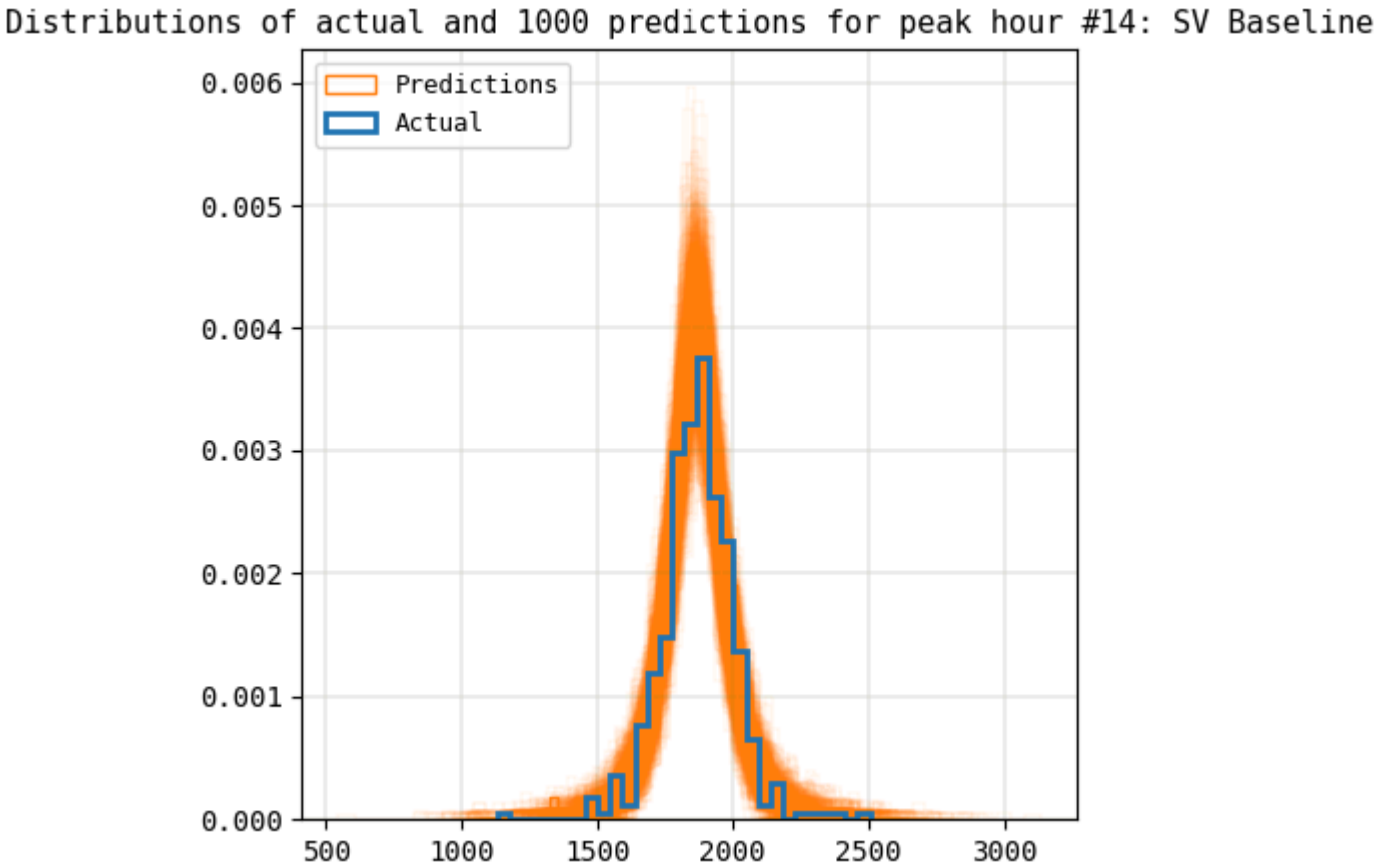
In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(price_data_peak_eur.index, predict_sv_base_peak_eur_many.iloc[0], color='C1', lw=1, label='Predictions')
for i in range(1, predict_sv_base_peak_eur_many.shape[0]):
    plt.plot(price_data_peak_eur.index, predict_sv_base_peak_eur_many.iloc[i], color='C1', lw=0.025)
plt.plot(price_data_peak_eur.index, price_data_peak_eur['CONSUMER_PRICE'], color='C0', lw=1, label='Actual')
plt.xlabel('Date', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
```

```
plt.yticks(size=10, family='monospace')
plt.ylim([1000, 2750])
plt.title(f'Actual and {num_samples} predictions of consumer price for peak hour #{hour_max_eur}: {model_sv_base_peak_eur.name
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```



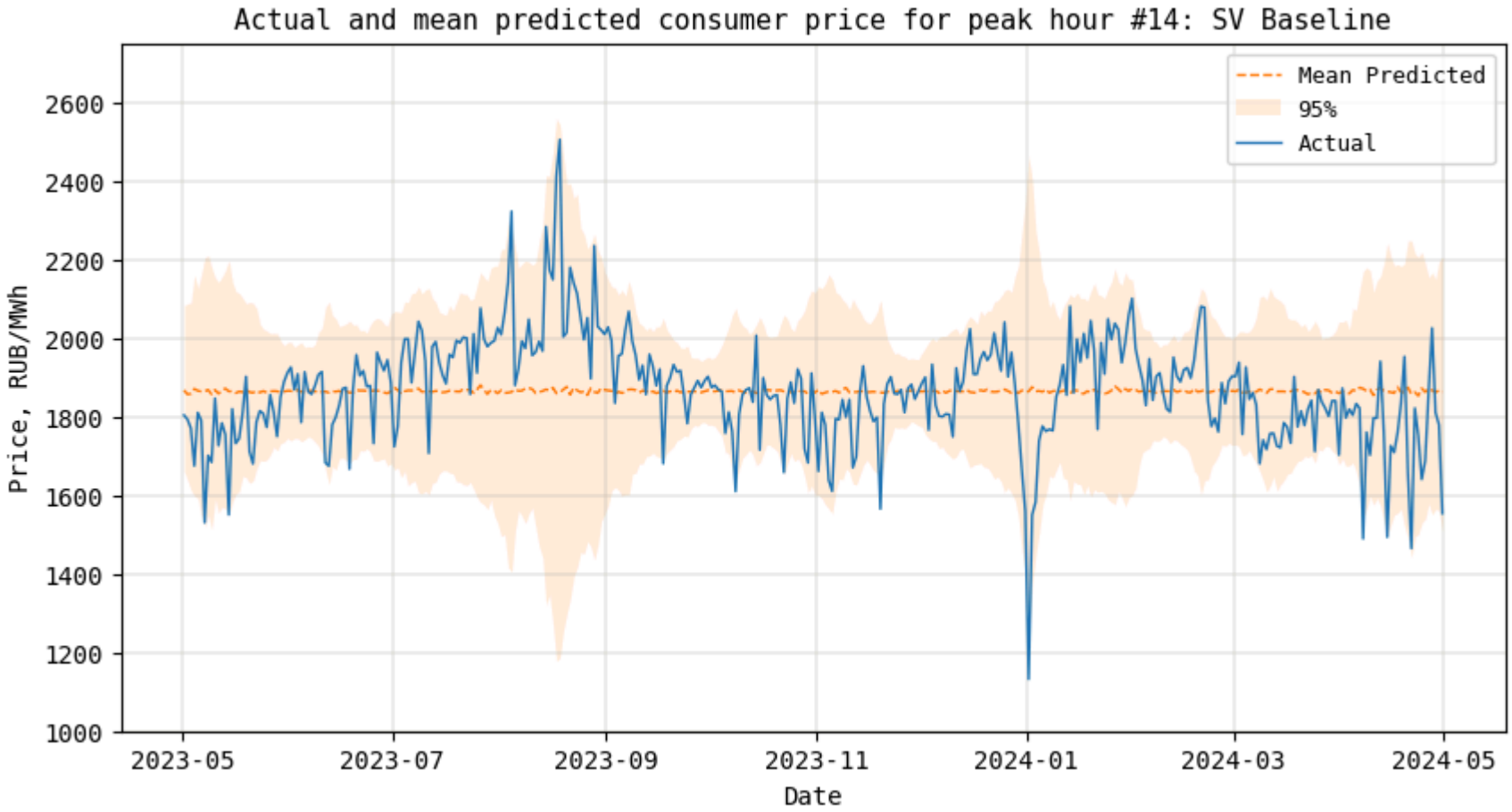
In [ ]:
```
plt.figure(figsize=(5, 5))
plt.hist(predict_sv_base_peak_eur_many.iloc[0], bins=30, density=True, histtype='step', color='white', edgecolor='C1', lw=1, l
for i in range(1, predict_sv_base_peak_eur_many.shape[0]):
    plt.hist(predict_sv_base_peak_eur_many.iloc[i], bins=30, density=True, histtype='step', color='white', edgecolor='C1', lw=
plt.hist(price_data_peak_eur['CONSUMER_PRICE'], bins=30, density=True, histtype='step', color='white', edgecolor='C0', lw=2, l
plt.xticks(size=10, family='monospace')
```

```
plt.yticks(size=10, family='monospace')
plt.grid(lw=0.25, color='xkcd:cement')
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
plt.gca().set_axisbelow(True)
plt.title(f'Distributions of actual and {num_samples} predictions for peak hour #{hour_max_eur}: {model_sv_base_peak_eur.name}
```



In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(price_data_peak_eur.index, predict_sv_base_peak_eur_mean, color='C1', lw=1, ls='--', label='Mean Predicted')
plt.fill_between(price_data_peak_eur.index, predict_sv_base_peak_eur_ci[0], predict_sv_base_peak_eur_ci[1], color='C1', lw=0,
plt.plot(price_data_peak_eur.index, price_data_peak_eur['CONSUMER_PRICE'], color='C0', lw=1, label='Actual')
plt.xlabel('Date', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
```

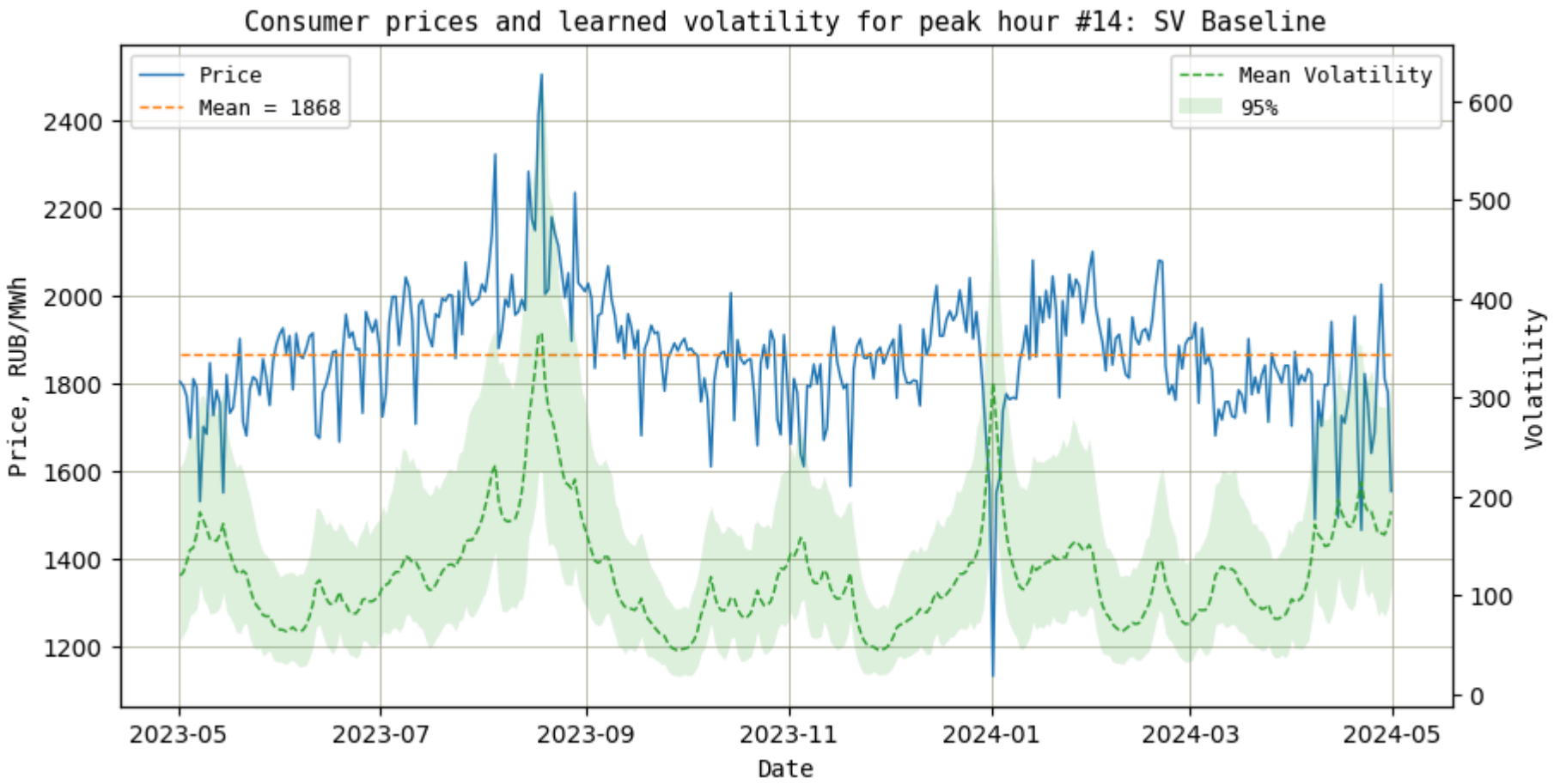
```
plt.yticks(size=10, family='monospace')
plt.ylim([1000, 2750])
plt.title(f'Actual and mean predicted consumer price for peak hour #{hour_max_eur}: {model_sv_base_peak_eur.name}', size=11, f
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'});
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
# Volatility at time t over all predictions
vol_sv_base_peak_eur = model_sv_base_peak_eur.get_volatility()
vol_sv_base_peak_eur.shape
```

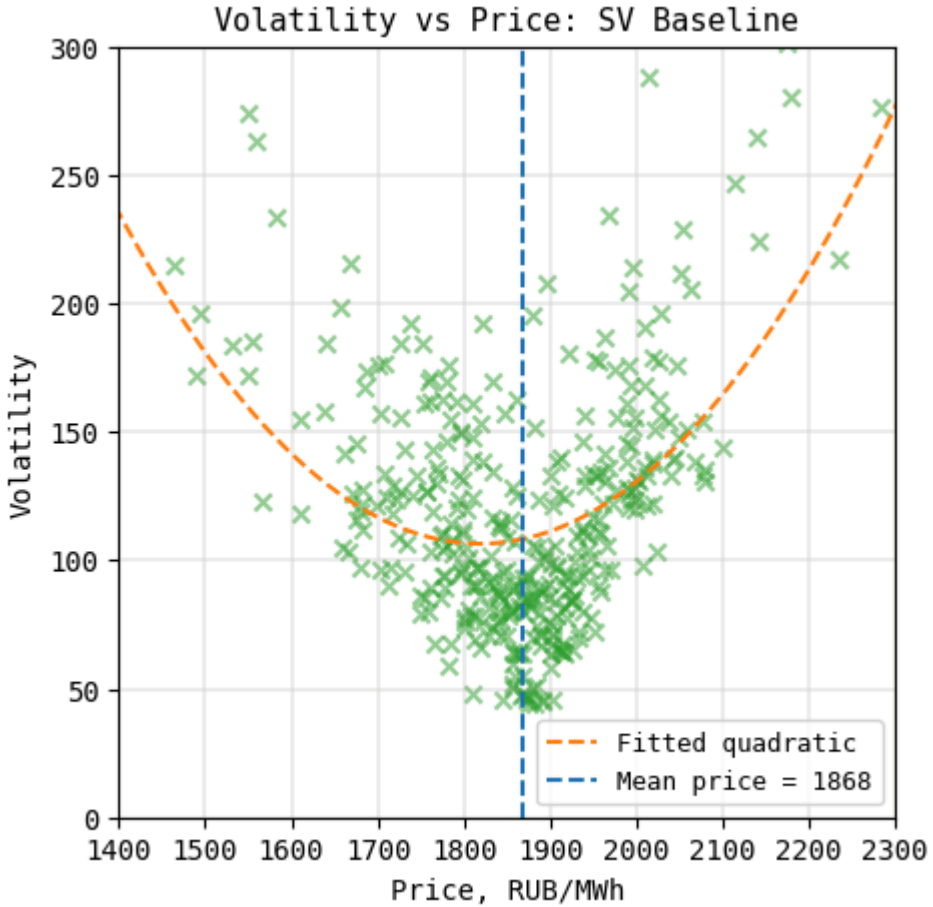
Out[ ]: (1000, 366)

In [ ]:
```
_, ax = plt.subplots(figsize=(10, 5))
ax.plot(price_data_peak_eur.index, price_data_peak_eur['CONSUMER_PRICE'], color='C0', lw=1, label='Price')
ax.hlines(price_data_peak_eur['CONSUMER_PRICE'].mean(), xmin=price_data_peak_eur.index.min(), xmax=price_data_peak_eur.index.m
ax.set_xlabel('Date', size=10, family='monospace')
ax.set_ylabel('Price, RUB/MWh', size=10, family='monospace')
ax.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
ax.grid(lw=0.5, color='xkcd:cement')
ax2 = ax.twinx()
ax2.plot(price_data_peak_eur.index, vol_sv_base_peak_eur.mean(axis=0), color='C2', lw=1, ls='--', label='Mean Volatility')
ax2.fill_between(price_data_peak_eur.index, np.percentile(vol_sv_base_peak_eur, [2.5, 97.5], axis=0)[0], np.percentile(vol_sv_
ax2.set_ylabel('Volatility', size=10, family='monospace')
ax2.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
ax.set_title(f'Consumer prices and learned volatility for peak hour #{hour_max_eur}: {model_sv_base_peak_eur.name}', size=11,
```



In [ ]:
```
plt.figure(figsize=(5, 5))
fitted_quadratic = np.poly1d(np.polyfit(price_data_peak_eur['CONSUMER_PRICE'], vol_sv_base_peak_eur.mean(axis=0), deg=2))
x = np.linspace(price_data_peak_eur['CONSUMER_PRICE'].min(), price_data_peak_eur['CONSUMER_PRICE'].max(), price_data_peak_eur[
plt.scatter(price_data_peak_eur['CONSUMER_PRICE'], vol_sv_base_peak_eur.mean(axis=0), color='C2', marker='x', alpha=0.5)
plt.plot(x, fitted_quadratic(x), color='C1', ls='--', label='Fitted quadratic')
plt.vlines(price_data_peak_eur['CONSUMER_PRICE'].mean(), ymin=0, ymax=350, color='C0', ls='--', label=f"Mean price = {price_da
plt.xlabel('Price, RUB/MWh', size=10, family='monospace')
plt.ylabel('Volatility', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xlim([1400, 2300])
plt.ylim([0, 300])
```

```
plt.title(f'Volatility vs Price: {model_sv_base_peak_eur.name}', size=11, family='monospace')
plt.legend(loc='lower right', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
alpha = 0.05
adf_stats, adf_pval, _, _, _, _ = adfuller(vol_sv_base_peak_eur.mean(axis=0))
print(f'{adf_stats = :.2f}, {adf_pval = :.2f}', end=', ')
print('Stationary') if adf_pval < alpha else print('Non-stationary')
```

```
adf_stats = -2.64, adf_pval = 0.08, Non-stationary
```

We can clearly see that volatility *does* depend on consumer price, and our **SV Baseline** model discovered this relation.

We can see a rather V-shaped dependency: the volatility tends to increase for the prices higher and lower than the mean price, while it's minimal around the mean price.

2.4. Goodness-of-fit

Consider the following metrics for evaluating the quality of the model.

1. Mean Absolute Error: $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, where

   n is the number of target points, y_i are the true target values, $\hat{y}_i$ are the model's predictions.

2. Rooted Mean Squared Error: $RMSE = \sqrt{MSE}$, where

   Mean Squared Error: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

In [ ]:
```
# MAE for mean predictions over all predictions
mae_sv_base = mean_absolute_error(price_data_peak_eur['CONSUMER_PRICE'], predict_sv_base_peak_eur_mean)
mae_sv_base
```

Out[ ]:
```
99.29057658842531
```

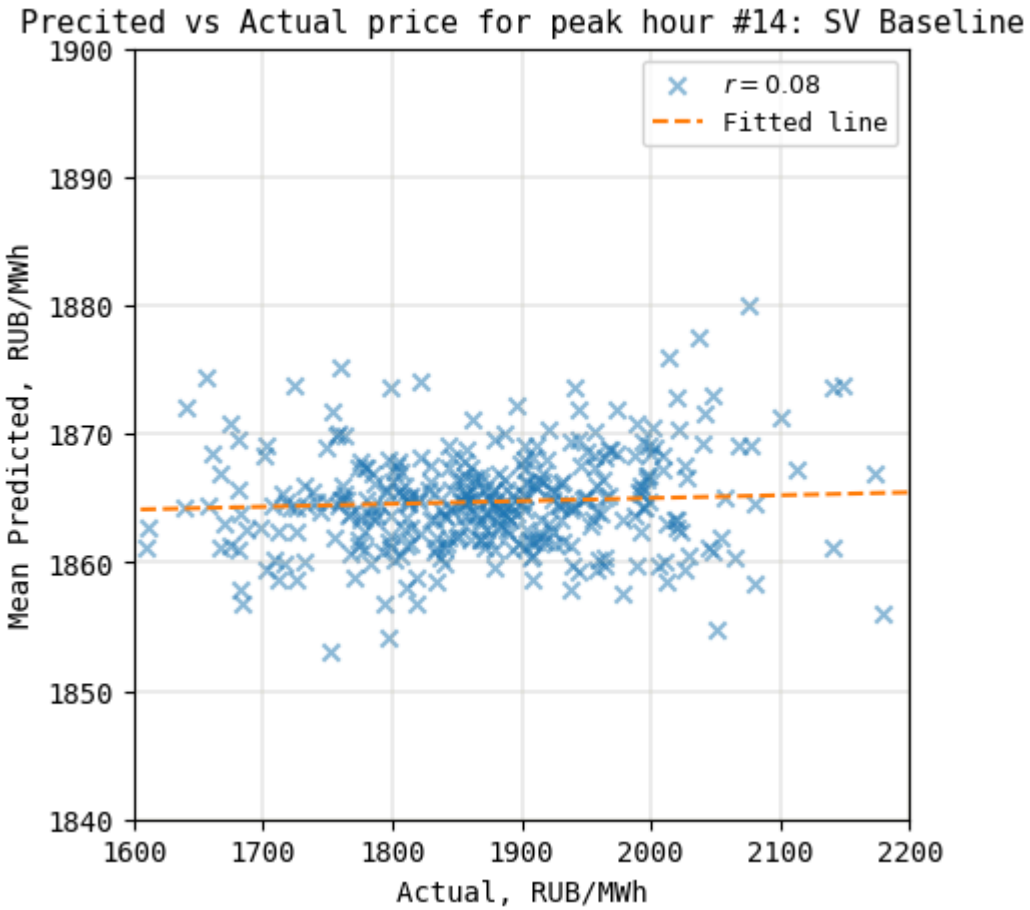
In [ ]:
```
# RMSE for mean predictions over all predictions
rmse_sv_base = root_mean_squared_error(price_data_peak_eur['CONSUMER_PRICE'], predict_sv_base_peak_eur_mean)
rmse_sv_base
```

Out[ ]:
```
137.07475651396808
```

Let's check the correlation between mean predicted and actual prices.

In [ ]:
```
r = np.corrcoef(price_data_peak_eur['CONSUMER_PRICE'], predict_sv_base_peak_eur_mean)[0, 1]
fitted_line = np.poly1d(np.polyfit(price_data_peak_eur['CONSUMER_PRICE'], predict_sv_base_peak_eur_mean, deg=1))
x = np.linspace(price_data_peak_eur['CONSUMER_PRICE'].min(), price_data_peak_eur['CONSUMER_PRICE'].max(), price_data_peak_eur[
plt.figure(figsize=(5, 5))
plt.scatter(price_data_peak_eur['CONSUMER_PRICE'], predict_sv_base_peak_eur_mean, color='C0', marker='x', alpha=0.5, label=f'$
```

```
plt.plot(x, fitted_line(x), color='C1', ls='--', label='Fitted line')
plt.xlabel('Actual, RUB/MWh', size=10, family='monospace')
plt.ylabel('Mean Predicted, RUB/MWh', size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.xlim([1600, 2200])
plt.ylim([1840, 1900])
plt.title(f'Precited vs Actual price for peak hour #{hour_max_eur}: {model_sv_base_peak_eur.name}', size=11, family='monospace
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```

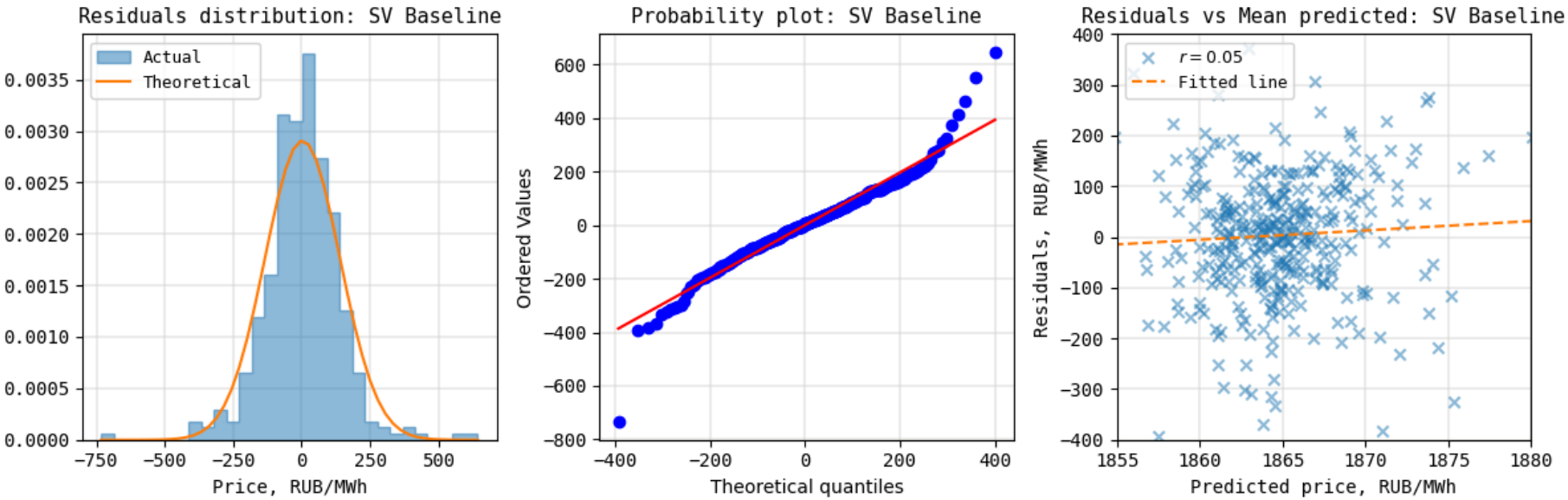


Though the model was able to learn heteroscedasity of the volatility, the correlation between the mean predictions and actual prices is rather weak.

We should also check the distribution of residuals, probability plot (similar to Q-Q plot), and residuals vs mean predicted prices.

In [ ]:
```
residuals_sv_base = price_data_peak_eur['CONSUMER_PRICE'] - predict_sv_base_peak_eur_mean.values
x = np.linspace(residuals_sv_base.min(), residuals_sv_base.max())
residuals_sv_base_theor = norm(residuals_sv_base.mean(), residuals_sv_base.std(ddof=1))
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.hist(residuals_sv_base, bins=30, density=True, histtype='stepfilled', color='C0', edgecolor='C0', alpha=0.5, label='Actual
plt.plot(x, residuals_sv_base_theor.pdf(x), color='C1', label='Theoretical')
plt.xlabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'Residuals distribution: {model_sv_base_peak_eur.name}', size=11, family='monospace')
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement')
plt.gca().set_axisbelow(True)
plt.subplot(1, 3, 2)
probplot(residuals_sv_base, dist=residuals_sv_base_theor, plot=plt)
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'Probability plot: {model_sv_base_peak_eur.name}', size=11, family='monospace')
plt.grid(lw=0.25, color='xkcd:cement')
plt.gca().set_axisbelow(True)
plt.subplot(1, 3, 3)
r = np.corrcoef(residuals_sv_base, predict_sv_base_peak_eur_mean)[0, 1]
fitted_line = np.poly1d(np.polyfit(predict_sv_base_peak_eur_mean, residuals_sv_base, deg=1))
x = np.linspace(predict_sv_base_peak_eur_mean.min(), predict_sv_base_peak_eur_mean.max(), predict_sv_base_peak_eur_mean.shape[
plt.scatter(predict_sv_base_peak_eur_mean, residuals_sv_base, color='C0', marker='x', alpha=0.5, label=f'$r = {r:.2f}$')
plt.plot(x, fitted_line(x), color='C1', ls='--', label='Fitted line')
plt.xlabel('Predicted price, RUB/MWh', size=10, family='monospace')
plt.ylabel('Residuals, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xlim([1855, 1880])
plt.ylim([-400, 400])
plt.title(f'Residuals vs Mean predicted: {model_sv_base_peak_eur.name}', size=11, family='monospace')
```

```
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement')
plt.gca().set_axisbelow(True)
plt.tight_layout();
```



Residuals look to be distributed more or less normally. Also they look rather homoscedastic w.r.t predictions, which means that the model is rather acceptable.

Back

---

3. SV Exogenous Model

Next, we will consider an extension of the SV Baseline model. The idea of adding exogenous regressor(s) is inspired by the paper "Probabilistic electricity price forecasting with Bayesian stochastic volatility models" by Maciej Kostrzewski, Jadwiga Kostrzewska, 2019.

The authors propose a stochastic volatility model with a double exponential distribution of jumps, a leverage effect and exogenous variables (in short, the SVDEJX model). This model introduces one exogenous factor -- the logarithm of the hourly air temperature at time t. The model also introduces indicator variables for 3 days of the week: Sat, Sun, Mon.

First, we'll examine the first possible exogenous regressor -- air temperature.

3.1. Temperature Data

In [ ]:
```
# Price zone 1: Moscow
temp_data_eur = pd.read_csv(f'./data/UUEE.01.05.2023.07.05.2024.1.0.0.ru.utf8.00000000.zip',
                        sep=';',
                        encoding='utf-8',
                        skiprows=7,
                        index_col=0,
                        dayfirst=True,
                        usecols=[0, 1],
                        names=['Datetime', 'Temperature'],
                        parse_dates=True).dropna().sort_index()
```

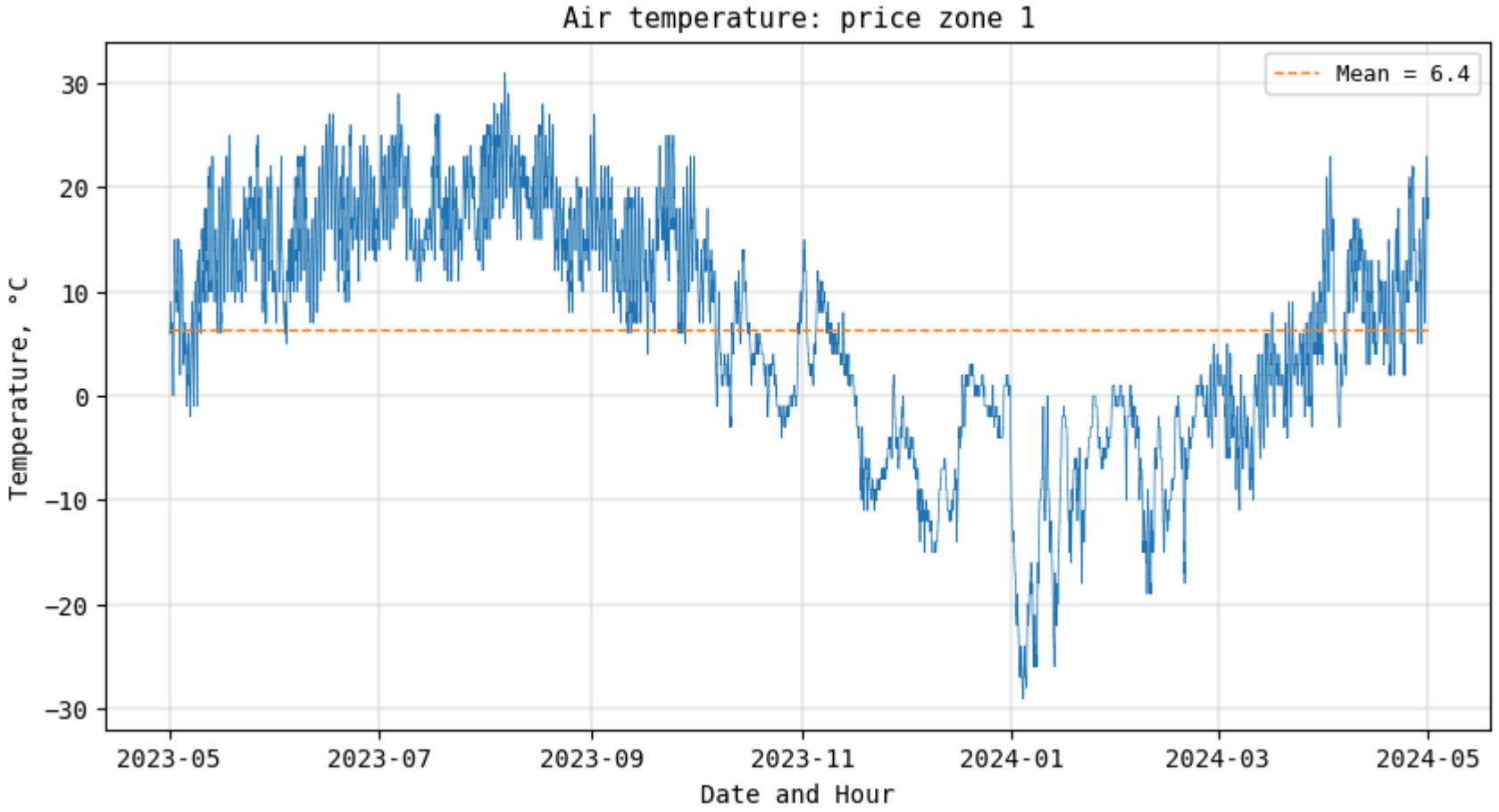
In [ ]:
```
temp_data_eur = temp_data_eur.loc[:'30.04.2024'] # 01.05.2024 -- 07.05.2024 is used for forecasting
temp_data_eur
```

Out[ ]:

Datetime	Temperature
2023-05-01 00:00:00	6.0
2023-05-01 00:30:00	6.0
2023-05-01 01:00:00	6.0
2023-05-01 01:30:00	6.0
2023-05-01 02:00:00	6.0
...	...
2024-04-30 21:30:00	18.0
2024-04-30 22:00:00	19.0
2024-04-30 22:30:00	19.0
2024-04-30 23:00:00	18.0
2024-04-30 23:30:00	17.0

17519 rows × 1 columns

In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(temp_data_eur.index, temp_data_eur['Temperature'], color='C0', lw=0.5)
plt.hlines(temp_data_eur['Temperature'].mean(), xmin=temp_data_eur.index.min(), xmax=temp_data_eur.index.max(), color='C1', lw
plt.xlabel('Date and Hour', size=10, family='monospace')
plt.ylabel('Temperature, °C', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'Air temperature: price zone {price_zone}', size=11, family='monospace')
plt.legend(prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
# Join consumer price and air temperature data
price_temp_data = pd.merge(price_data, temp_data_eur, on='Datetime')
price_temp_data.shape
```

Out[ ]: (8760, 4)

In [ ]:
```
price_temp_data.head(4)
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Temperature
2023-05-01 00:00:00	1	1517.92	0	6.0
2023-05-01 01:00:00	1	1413.99	1	6.0
2023-05-01 02:00:00	1	1345.22	2	6.0
2023-05-01 03:00:00	1	1275.81	3	6.0

In [ ]:
```
price_temp_data.tail(4)
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Temperature
2024-04-30 20:00:00	1	1822.91	20	18.0
2024-04-30 21:00:00	1	1742.75	21	17.0
2024-04-30 22:00:00	1	1442.53	22	19.0
2024-04-30 23:00:00	1	1327.94	23	18.0

In [ ]:
```
price_temp_data_eur = price_temp_data[price_temp_data['PRICE_ZONE_CODE'] == 1]
```

In [ ]:
```
price_temp_data_peak_eur = price_temp_data_eur[price_temp_data_eur['HOUR'] == hour_max_eur]
price_temp_data_peak_eur
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Temperature
2023-05-01 14:00:00	1	1803.61	14	7.0
2023-05-02 14:00:00	1	1793.39	14	14.0
2023-05-03 14:00:00	1	1770.27	14	12.0
2023-05-04 14:00:00	1	1674.88	14	13.0
2023-05-05 14:00:00	1	1810.13	14	4.0
...	...	...	...	...
2024-04-26 14:00:00	1	1862.04	14	22.0
2024-04-27 14:00:00	1	2025.49	14	10.0
2024-04-28 14:00:00	1	1810.51	14	15.0
2024-04-29 14:00:00	1	1779.80	14	18.0
2024-04-30 14:00:00	1	1553.93	14	22.0

366 rows × 4 columns

In [ ]:
```
price_temp_data_offpeak_eur = price_temp_data_eur[price_temp_data_eur['HOUR'] == hour_min_eur]
price_temp_data_offpeak_eur
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Temperature
2023-05-01 03:00:00	1	1275.81	3	6.0
2023-05-02 03:00:00	1	1113.04	3	2.0
2023-05-03 03:00:00	1	1085.05	3	9.0
2023-05-04 03:00:00	1	1003.91	3	4.0
2023-05-05 03:00:00	1	1121.00	3	5.0
...	...	...	...	...
2024-04-26 03:00:00	1	944.52	3	11.0
2024-04-27 03:00:00	1	1147.59	3	15.0
2024-04-28 03:00:00	1	1157.11	3	6.0
2024-04-29 03:00:00	1	1197.78	3	7.0
2024-04-30 03:00:00	1	1191.04	3	8.0

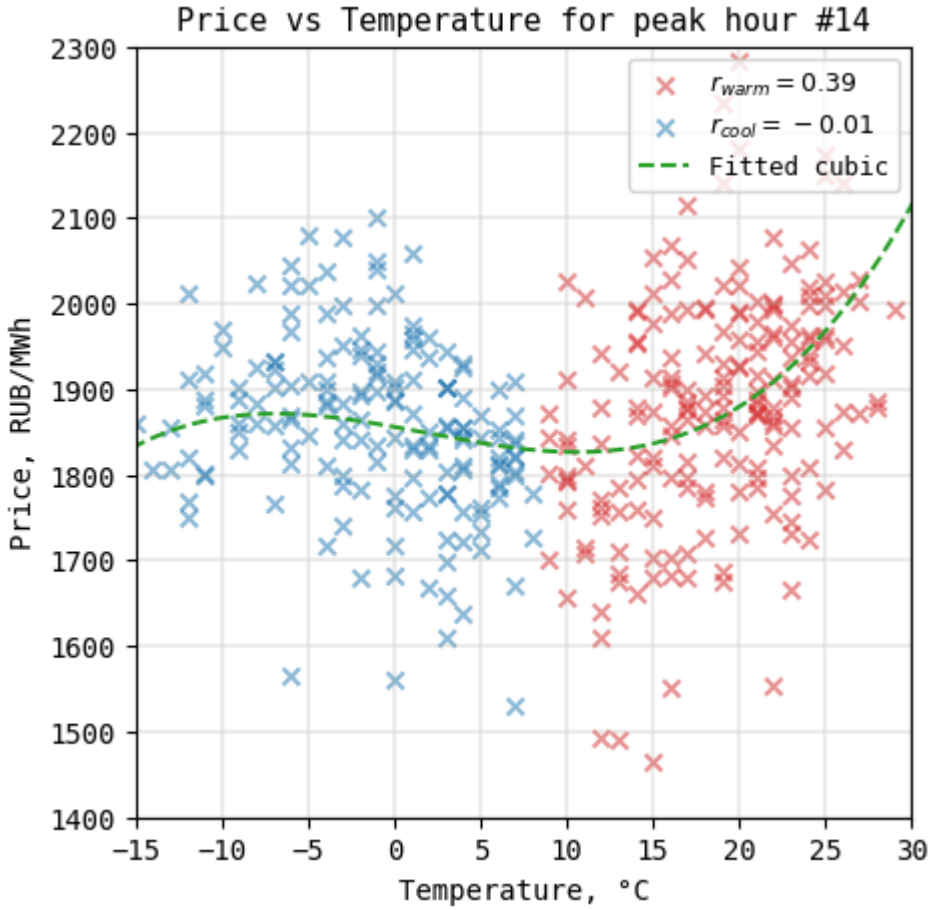
366 rows × 4 columns

Let's check the correlation between the Price and Temperature. We are expecting a rather V-shaped dependency, since both high and low air temperatures should increase electricity demand for cooling and heating and thus the price.

In [ ]:
```
price_temp_data_peak_eur.loc[:, 'Cluster'] = KMeans(n_clusters=2, random_state=0).fit_predict(price_temp_data_peak_eur[['Tempe
r_warm = price_temp_data_peak_eur[price_temp_data_peak_eur['Cluster'] == price_temp_data_peak_eur['Cluster'].max()]['CONSUMER_
r_cool = price_temp_data_peak_eur[price_temp_data_peak_eur['Cluster'] == price_temp_data_peak_eur['Cluster'].min()]['CONSUMER_
fitted_cubic = np.poly1d(np.polyfit(price_temp_data_peak_eur['Temperature'], price_temp_data_peak_eur['CONSUMER_PRICE'], deg=3
x = np.linspace(price_temp_data_peak_eur['Temperature'].min(), price_temp_data_peak_eur['Temperature'].max(), price_temp_data_
plt.figure(figsize=(5, 5))
plt.scatter(price_temp_data_peak_eur[price_temp_data_peak_eur['Cluster'] == price_temp_data_peak_eur['Cluster'].max()]['Temper
plt.scatter(price_temp_data_peak_eur[price_temp_data_peak_eur['Cluster'] == price_temp_data_peak_eur['Cluster'].min()]['Temper
plt.plot(x, fitted_cubic(x), color='C2', ls='--', label='Fitted cubic')
plt.xlabel('Temperature, °C', size=10, family='monospace')
```

```
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.xlim([-15, 30])
plt.ylim([1400, 2300])
plt.title(f'Price vs Temperature for peak hour #{hour_max_eur}', size=11, family='monospace')
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```

```
/tmp/ipykernel_1828/4233981735.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  price_temp_data_peak_eur.loc[:, 'Cluster'] = KMeans(n_clusters=2, random_state=0).fit_predict(price_temp_data_peak_eur[['Temperature']])
```



We can confirm that the price *does* depend on the air temperature:

- coefficient of correlation (Pearson) for high air temperatures is 0.39,
- coefficient of correlation (Pearson) for low air temperatures is -0.01.

The price tends to be higher for higher and lower air temperatures, which looks reasonable: electricity demand is higher during the hot and cold season of the year (both cooling and heating are required), while the demand is minimal during semi-seasons when minimal heating and cooling are required. At the same time, the price responds more to the higher air temperatures. This might be due to the fact that in the studied scenario electricity is used more for cooling than for heating.

Next, let's examine the second possible exogenous regressor -- the day of week.

3.2. Weekday

In [ ]:
```
price_temp_data_peak_eur['day_name'] = price_temp_data_peak_eur.index.day_name()
price_temp_data_peak_eur['Weekday'] = price_temp_data_peak_eur.index.day_of_week
price_temp_data_peak_eur
```

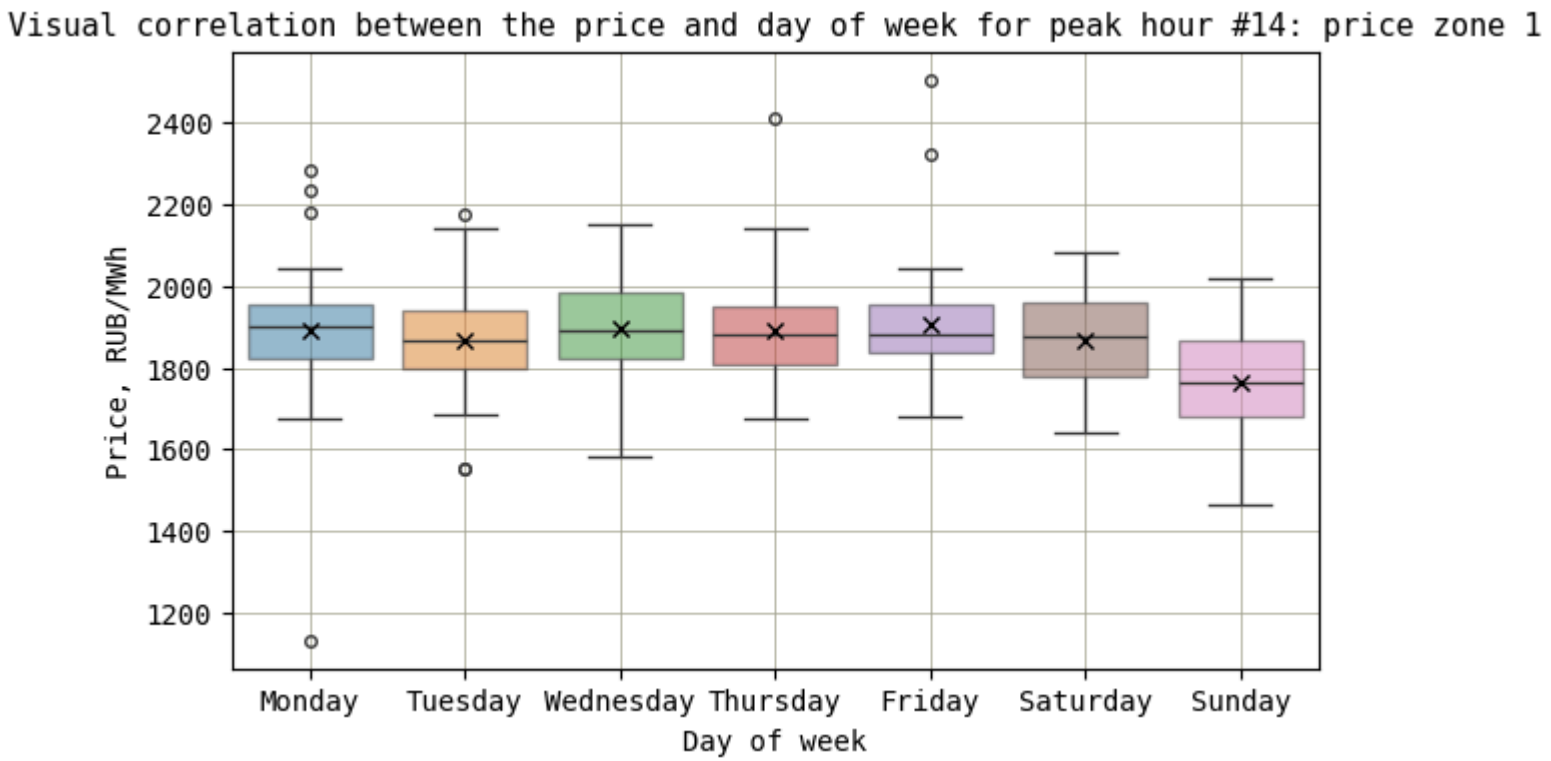
```
/tmp/ipykernel_1828/672662706.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  price_temp_data_peak_eur['day_name'] = price_temp_data_peak_eur.index.day_name()
/tmp/ipykernel_1828/672662706.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  price_temp_data_peak_eur['Weekday'] = price_temp_data_peak_eur.index.day_of_week
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Temperature	Cluster	day_name	Weekday
2023-05-01 14:00:00	1	1803.61	14	7.0	0	Monday	0
2023-05-02 14:00:00	1	1793.39	14	14.0	1	Tuesday	1
2023-05-03 14:00:00	1	1770.27	14	12.0	1	Wednesday	2
2023-05-04 14:00:00	1	1674.88	14	13.0	1	Thursday	3
2023-05-05 14:00:00	1	1810.13	14	4.0	0	Friday	4
...	...	...	...	...	...	...	...
2024-04-26 14:00:00	1	1862.04	14	22.0	1	Friday	4
2024-04-27 14:00:00	1	2025.49	14	10.0	1	Saturday	5
2024-04-28 14:00:00	1	1810.51	14	15.0	1	Sunday	6
2024-04-29 14:00:00	1	1779.80	14	18.0	1	Monday	0
2024-04-30 14:00:00	1	1553.93	14	22.0	1	Tuesday	1

366 rows × 7 columns

In [ ]:
```
plt.figure(figsize=(7, 4))
sns.boxplot(data=price_temp_data_peak_eur[['CONSUMER_PRICE', 'Weekday', 'day_name']].sort_values(by='Weekday'), x='day_name',
plt.xlabel('Day of week', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.grid(lw=0.5, color='xkcd:cement')
plt.title(f'Visual correlation between the price and day of week for peak hour #{hour_max_eur}: price zone 1', size=11, family
```



Though the mean values look equal, at least on Sundays the price tends to be lower. Let's introduce the second new regressor -- day of week.

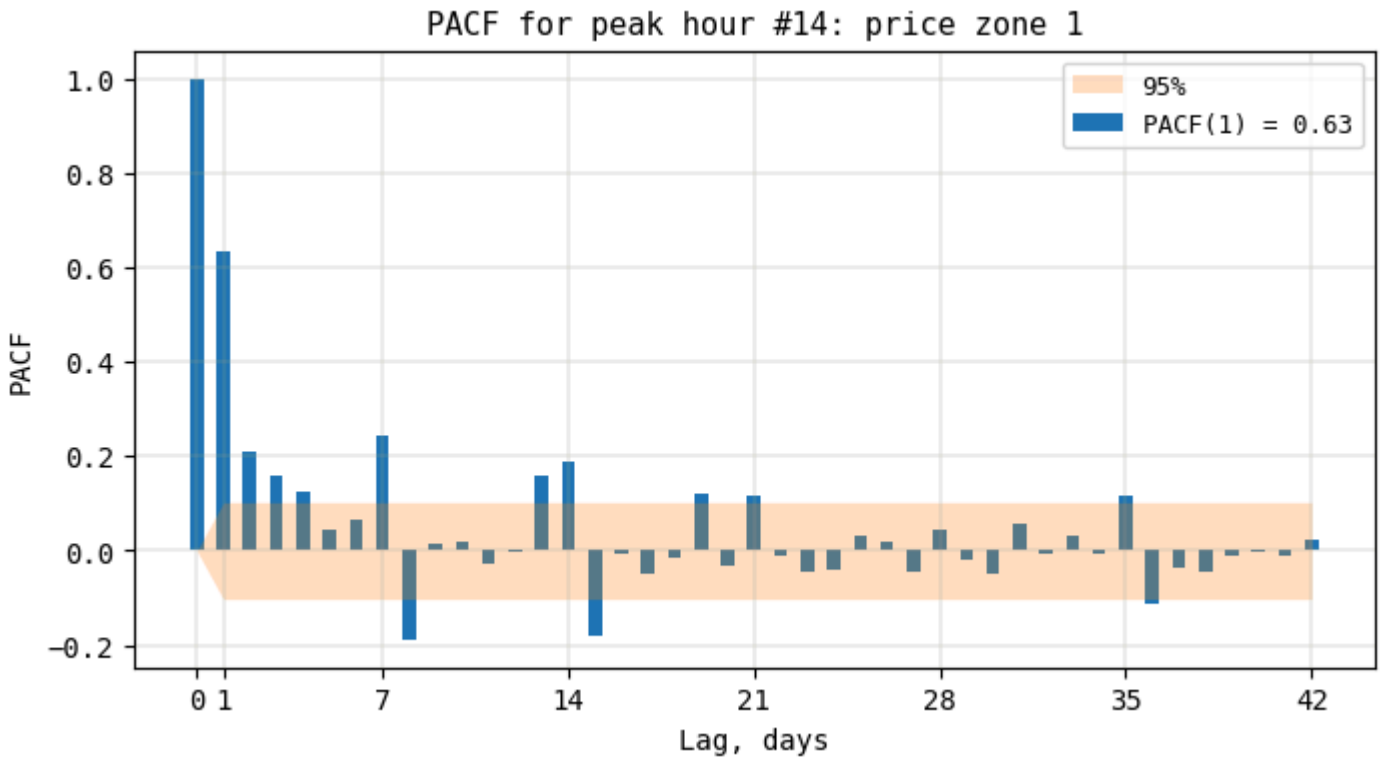
3.3. Autoregressive Component

Kostrzewski et al. extend their model with the minimum of the previous day's 24 hourly log prices [p. 615]. Based om this idea, we will check the correlation between the prices at time t (today) and $t-1$ (yesterday). For that, we'll compute the partial autocorrelation function which is the correlation between two observations that the shorter lags between those observations do not explain, i.e. the partial correlation for each lag is the unique correlation between those two observations after removing out the intervening correlations.

In [ ]:
```
# PACF for Peak hour + Price zone 1 with CI at 5% significance level for 42 day lags
pacf_corr_price_temp_data_peak_eur, pacf_ci_price_temp_data_peak_eur = pacf(price_temp_data_peak_eur['CONSUMER_PRICE'], nlags=
```

In [ ]:
```
plt.figure(figsize=(8, 4))
plt.bar(x=range(0, 43), height=pacf_corr_price_temp_data_peak_eur, width=0.5, align='center', label=f'PACF(1) = {pacf_corr_pri
```

```
plt.fill_between(range(0, 43), pacf_corr_price_temp_data_peak_eur - pacf_ci_price_temp_data_peak_eur[:, 0], pacf_corr_price_te
plt.xlabel('Lag, days', size=10, family='monospace')
plt.ylabel('PACF', size=10, family='monospace')
plt.xticks([0, 1, 7, 14, 21, 28, 35, 42], size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'PACF for peak hour #{hour_max_eur}: price zone 1', size=11, family='monospace')
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```



Since our price is not stationary [2.2], the PACF decays rather slowly during the period of 42 days (6 weeks). We have shown above [3.2] that the price *is* correlated with the day of week. Observe that the PACF has spikes exactly every 7 days (1 week) which further supports the correctness of inclusion of the day of week as an exogenous regressor.

PACF suggests that using at least 7 or more lags (new features) might be possible. We already decided to include the day of week to explain the seasonality of the price. The strongest correlation is seen for the lag of one (0.63), while other lags are much weaker correlated. Taking into

account computational compexity of inference with Stan, using only the lag of one day looks to be a reasonable compromise. In other words, we will consider an autoregressive model with lag of one AR(1) as the autoregressive component.

A first-order autoregressive model AR(1) is the following:

$y_t = \mathcal{N}(\alpha + \beta y_{t-1}, \sigma)$, where

α and β are the intercept and slope of autoregression and $\sigma \sim \mathcal{N}(0, 1)$ (constant normal volatility). Since we model volatility as a stochastic process (not simply constant), we will use only the expected value of this model.

We have shown that the consumer price is correlated with both the air temperature [3.1], day of week [3.2], and with itself with at least lag of one day [3.3]. Let's propose a new model **SV X**, which extends the ideas of our regression-like SV Baseline model and [Kostrzewski] with three exogenous regressors -- air temperature, day of week, and autoregressive component:

$y_t \sim \mathcal{N}(\overline{y} + \alpha y_{t-1} + \beta_3 X_{t-1}^3 + \beta_2 X_{t-1}^2 + \beta_1 X_{t-1} + \gamma D_t + \xi, e^{\frac{h_t}{2}})$, where

X_{t-1} is the hourly air temperature at time $t-1$. To prevent target leakage, the temperature readings are lagged one day behind ($t-1$);

D_t is the day of week at time t;

$h_t \sim \mathcal{N}(\mu + \phi(h_{t-1} - \mu), \sigma)$;

$h_1 \sim \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{1-\phi^2}}\right)$.

Thus, we are introducing:

- 1 new parameter: α for the autoregressive component;
- 3 new parameters: $\beta_{i=1...3}$ of the air temperature regressor. The unit of air temperature is °C;
- 1 new parameter: γ for the day of week regressor. The weekdays are numbered from 0 (Monday) to 6 (Sunday);
- 1 new parameter: ξ for the constant term (intercept) for all exogenous regressors.

3.4. Modeling

In [ ]:
```
# SV Exogenous model
with open('./models/sv_x_fit.stan', 'r') as fh:
  sv_x_code_fit = fh.read()
with open('./models/sv_x_predict.stan', 'r') as fh:
  sv_x_code_predict = fh.read()
#print(sv_x_code_fit)
#print(sv_x_code_predict)
num_samples = 1_000
model_sv_x_peak_eur = StanModel(kind='sv_x',
                                name='SV X',
                                stan_code_fit=sv_x_code_fit,
                                stan_code_predict=sv_x_code_predict,
                                num_samples=num_samples)
```

In [ ]:
```
%%capture
# Learn parameters
model_sv_x_peak_eur.fit(X=price_temp_data_peak_eur[['Temperature', 'Weekday']], y=price_temp_data_peak_eur['CONSUMER_PRICE'])
```

In [ ]:
```
model_sv_x_peak_eur
```

Out[ ]:
```
StanModel
StanModel(kind='sv_x', name='SV X',
          stan_code_fit='// SV Exogenous model\n'
                        '// Expected value: mean of y (constant)\n'
                        '// Volatility: stochastic\n'
                        '// Exogenous regressor(s)\n'
                        '\n'
                        'data\n'
                        '{\n'
                        '  int<lower=1> N; // Number of train time points '
```

In [ ]:
```
fit_sv_x_peak_eur_df = model_sv_x_peak_eur.fit_result_df_
fit_sv_x_peak_eur_df
```

Out[ ]:

parameters	lp__	accept_stat__	stepsize__	treedepth__	n_leapfrog__	divergent__	energy__	mu	phi	sigma	...
draws											
0	-1951.218729	0.930593	0.01665	8.0	255.0	0.0	2143.081821	8.739556	0.535499	0.767469	...
1	-1978.266993	0.919487	0.01665	8.0	255.0	0.0	2159.554413	8.608655	0.826828	0.406100	...
2	-1958.635980	0.986432	0.01665	8.0	255.0	0.0	2167.897693	8.783923	0.583520	0.634996	...
3	-1964.652603	0.998746	0.01665	8.0	255.0	0.0	2156.845175	8.882198	0.552389	0.601453	...
4	-1962.632445	0.933863	0.01665	8.0	255.0	0.0	2169.792379	8.681807	0.697040	0.675216	...
...	...	...	...	...	...	...	...	...	...	...	...
995	-1980.358869	0.991700	0.01665	8.0	255.0	0.0	2187.326160	8.782553	0.846919	0.356468	...
996	-1976.538706	0.874052	0.01665	8.0	255.0	0.0	2164.395033	8.623049	0.932263	0.371829	...
997	-1988.850192	0.897802	0.01665	8.0	255.0	0.0	2169.978355	8.554678	0.767010	0.500945	...
998	-1973.825484	0.999437	0.01665	8.0	255.0	0.0	2162.099823	8.937058	0.764436	0.606943	...
999	-1996.648034	0.993331	0.01665	8.0	255.0	0.0	2177.006249	8.670863	0.926523	0.294480	...

1000 rows × 749 columns

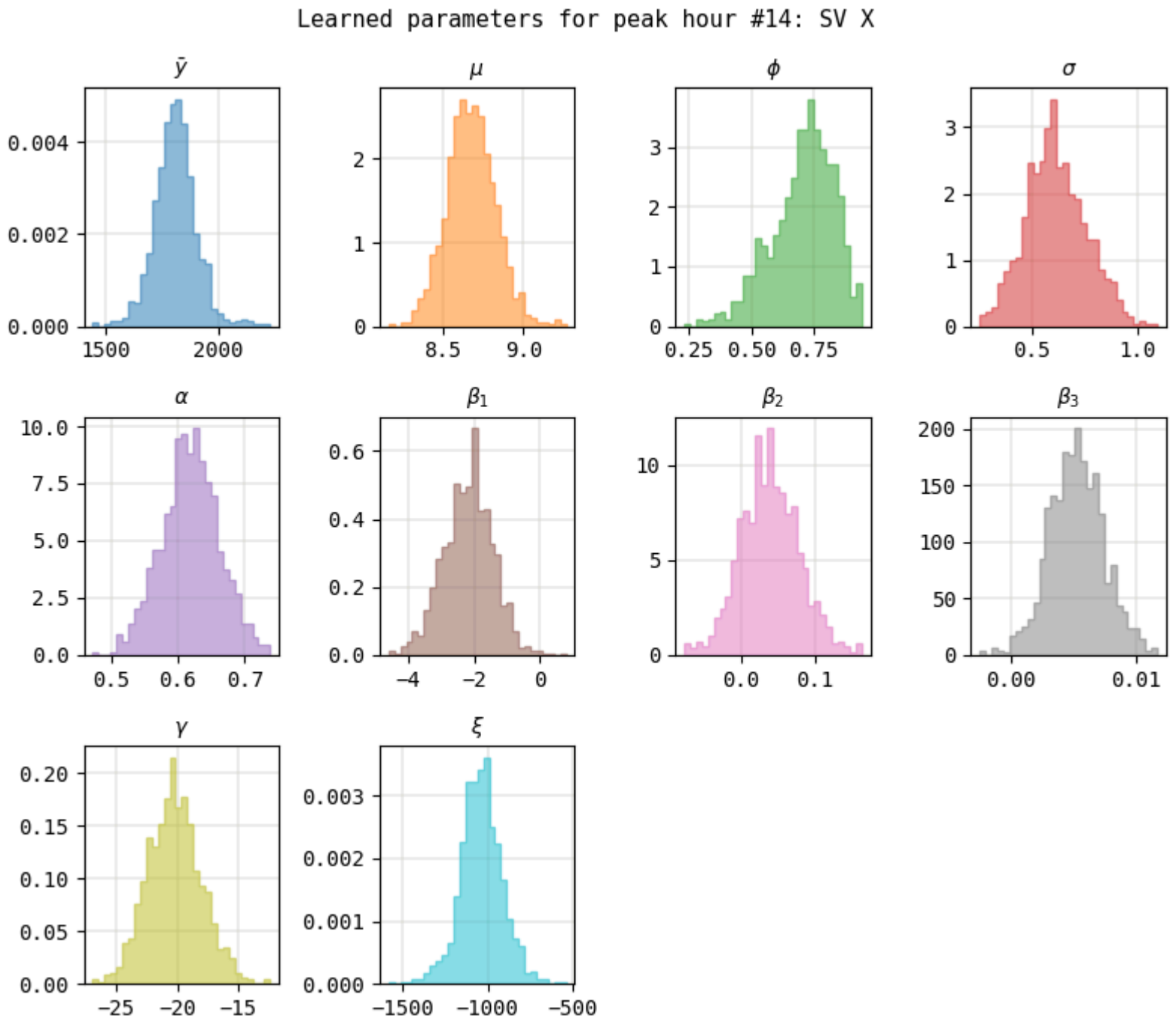
In [ ]:
```
# Learned parameters
fit_sv_x_peak_eur_df[['y_mean', 'mu', 'phi', 'sigma', 'alpha', 'beta_1', 'beta_2', 'beta_3', 'gamma', 'xi']].describe()
```

Out[ ]:

parameters	y_mean	mu	phi	sigma	alpha	beta_1	beta_2	beta_3	gamma	
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.0
mean	1808.293678	8.678890	0.704120	0.612444	0.617629	-2.192741	0.039307	0.005241	-20.238689	-1040.6
std	92.623180	0.151474	0.125148	0.142502	0.043085	0.751987	0.040201	0.002195	2.171578	125.8
min	1445.671463	8.167667	0.231199	0.252915	0.472030	-4.609939	-0.076763	-0.002447	-26.900703	-1586.0
25%	1749.728266	8.579556	0.630905	0.512730	0.589529	-2.692256	0.012404	0.003744	-21.738720	-1120.7
50%	1808.039005	8.674644	0.722635	0.605386	0.617939	-2.148165	0.038409	0.005233	-20.286842	-1041.8
75%	1859.828782	8.775039	0.796641	0.705762	0.646647	-1.703474	0.066478	0.006752	-18.777293	-963.2
max	2226.075159	9.262667	0.941146	1.094405	0.738125	0.784350	0.163865	0.011746	-12.387278	-539.7

In [ ]:

```
plt.figure(figsize=(8, 7))
ncols = 4
nrows = int(np.ceil(10 / ncols))
for i, param in zip(range(1, 11), fit_sv_x_peak_eur_df[['y_mean', 'mu', 'phi', 'sigma', 'alpha', 'beta_1', 'beta_2', 'beta_3',
    plt.subplot(nrows, ncols, i)
    plt.hist(fit_sv_x_peak_eur_df[param], bins=30, density=True, histtype='stepfilled', color=f'C{i - 1}', edgecolor=f'C{i - 1
    plt.xticks(size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
    if param == 'y_mean':
        plt.title('$\\bar{y}$', size=10, family='monospace')
    else:
        plt.title(f'$\{param}$', size=10, family='monospace')
    plt.grid(lw=0.25, color='xkcd:cement')
    plt.gca().set_axisbelow(True)
plt.suptitle(f'Learned parameters for peak hour #{hour_max_eur}: {model_sv_x_peak_eur.name}', size=11, family='monospace')
plt.tight_layout();
```

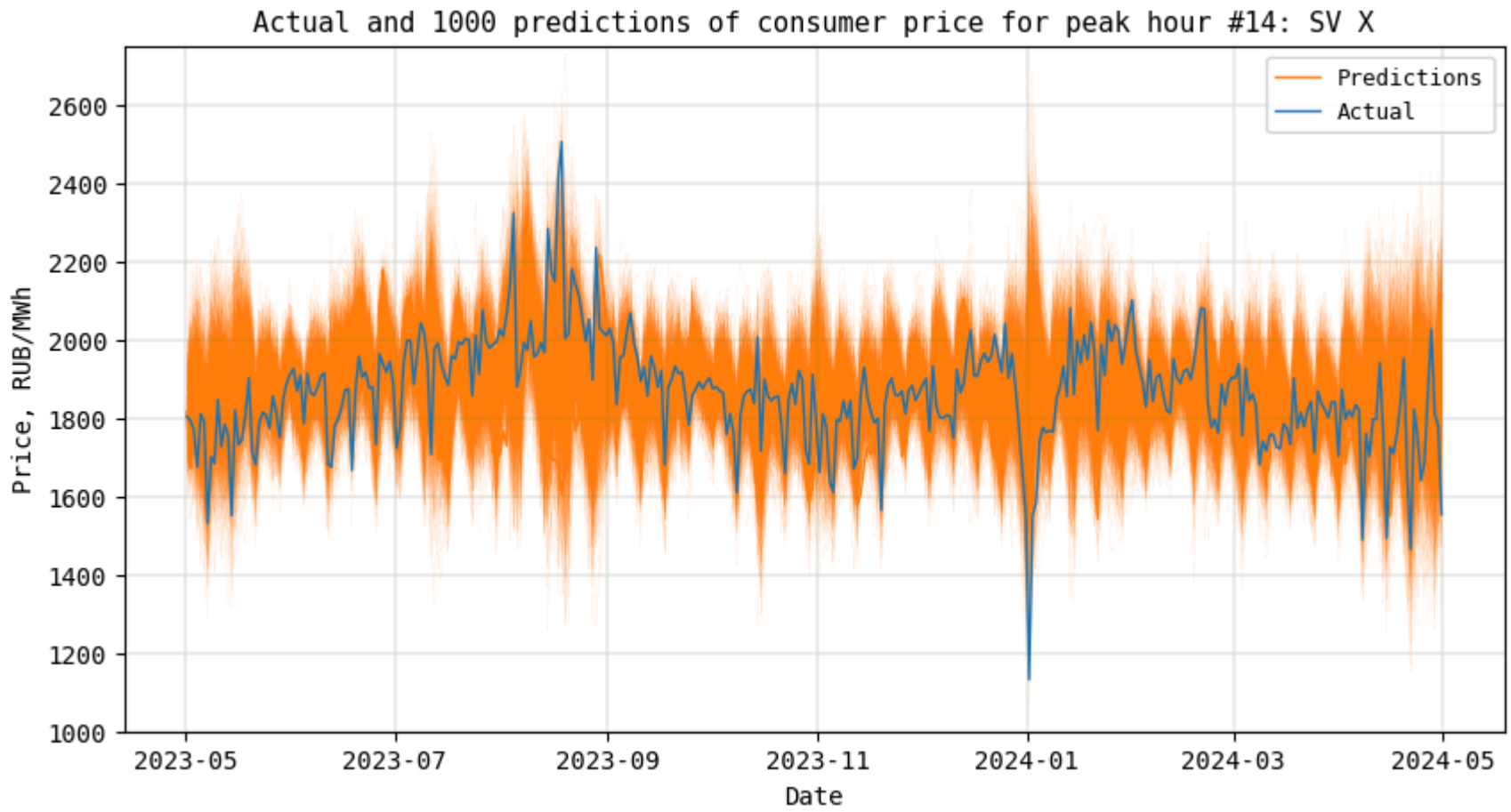

Learned parameters for peak hour #14: SV X
$\bar{y}$
μ
ϕ
σ
α
β_1
β_2
β_3
γ
ξ

In [ ]:
```
%%capture
# Predict
predict_sv_x_peak_eur_mean = model_sv_x_peak_eur.predict(price_temp_data_peak_eur[['Temperature', 'Weekday']])
predict_sv_x_peak_eur_mean.shape
```

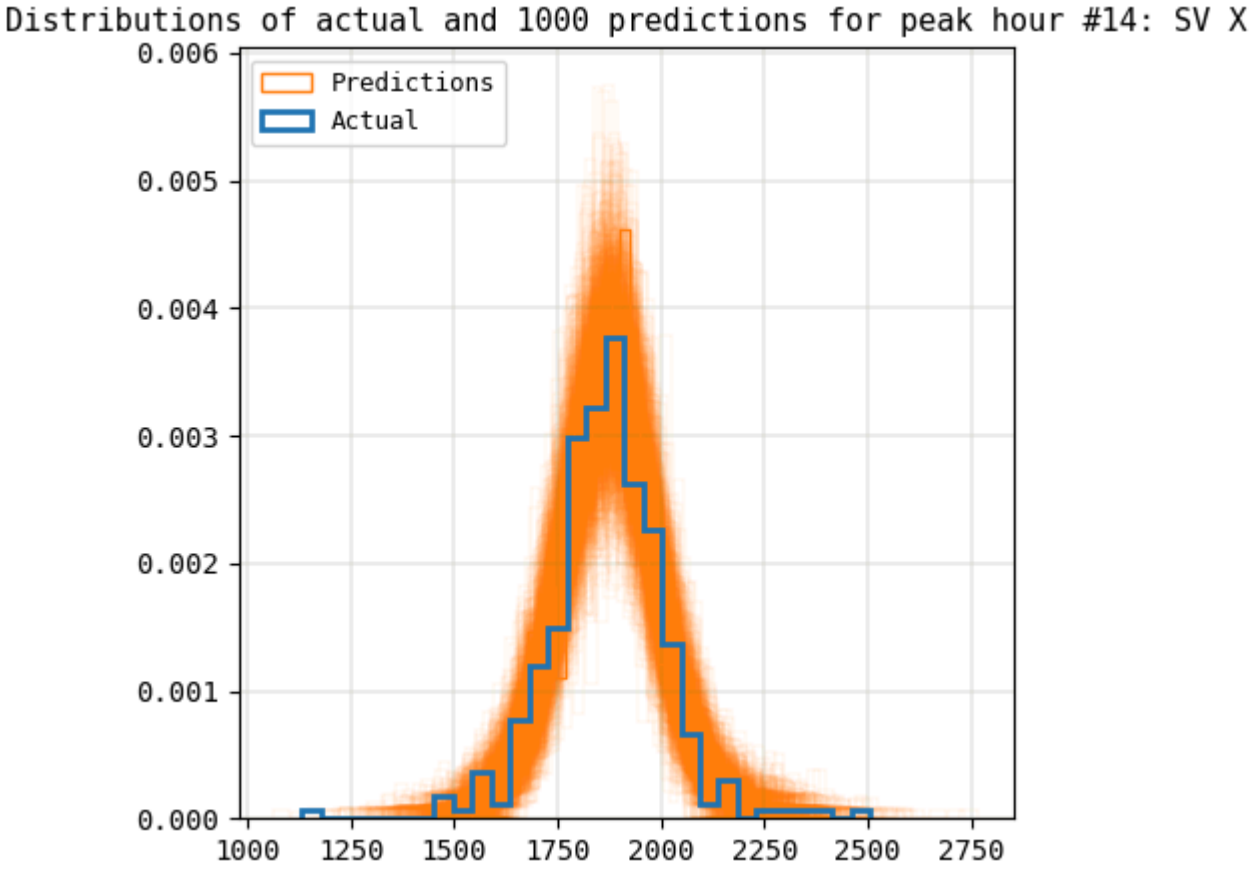
In [ ]:
```
%%capture
predict_sv_x_peak_eur_many = model_sv_x_peak_eur.predict_many(price_temp_data_peak_eur[['Temperature', 'Weekday']])
predict_sv_x_peak_eur_many.shape
```

In [ ]:
```
%%capture
predict_sv_x_peak_eur_ci = model_sv_x_peak_eur.predict_ci(price_temp_data_peak_eur[['Temperature', 'Weekday']], [2.5, 97.5])
len(predict_sv_x_peak_eur_ci)
```

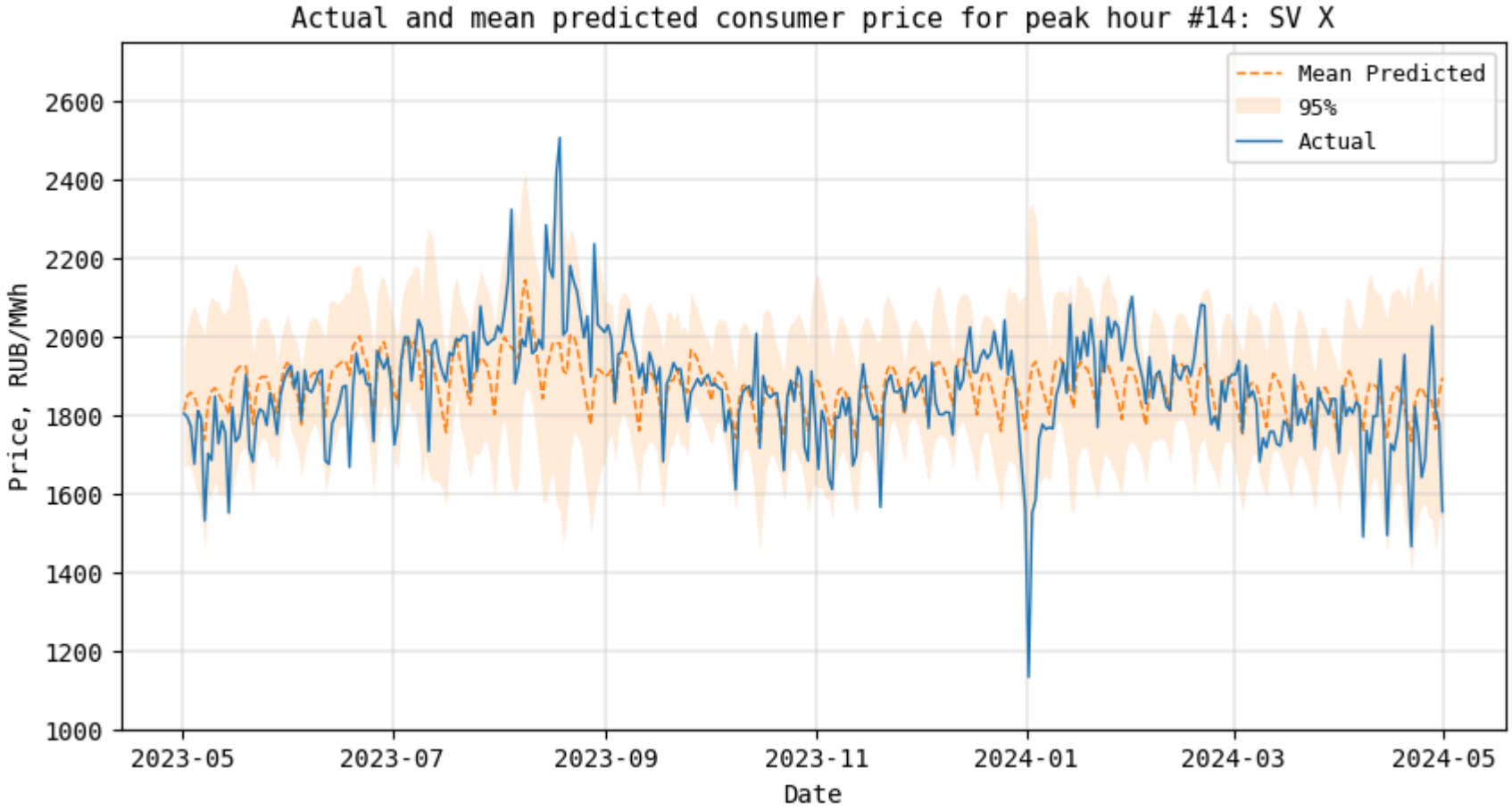
In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(price_temp_data_peak_eur.index, predict_sv_x_peak_eur_many.iloc[0], color='C1', lw=1, label='Predictions')
for i in range(1, predict_sv_x_peak_eur_many.shape[0]):
    plt.plot(price_temp_data_peak_eur.index, predict_sv_x_peak_eur_many.iloc[i], color='C1', lw=0.025)
plt.plot(price_temp_data_peak_eur.index, price_temp_data_peak_eur['CONSUMER_PRICE'], color='C0', lw=1, label='Actual')
plt.xlabel('Date', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.ylim([1000, 2750])
plt.title(f'Actual and {num_samples} predictions of consumer price for peak hour #{hour_max_eur}: {model_sv_x_peak_eur.name}',
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'});
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
plt.figure(figsize=(5, 5))
plt.hist(predict_sv_x_peak_eur_many.iloc[0], bins=30, density=True, histtype='step', color='white', edgecolor='C1', lw=1, labe
for i in range(1, predict_sv_x_peak_eur_many.shape[0]):
    plt.hist(predict_sv_x_peak_eur_many.iloc[i], bins=30, density=True, histtype='step', color='white', edgecolor='C1', lw=0.0
plt.hist(price_temp_data_peak_eur['CONSUMER_PRICE'], bins=30, density=True, histtype='step', color='white', edgecolor='C0', lw
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.grid(lw=0.25, color='xkcd:cement')
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
plt.gca().set_axisbelow(True)
plt.title(f'Distributions of actual and {num_samples} predictions for peak hour #{hour_max_eur}: {model_sv_x_peak_eur.name}',
```



In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(price_temp_data_peak_eur.index, predict_sv_x_peak_eur_mean, color='C1', lw=1, ls='--', label='Mean Predicted')
plt.fill_between(price_temp_data_peak_eur.index, predict_sv_x_peak_eur_ci[0], predict_sv_x_peak_eur_ci[1], color='C1', lw=0, a
plt.plot(price_temp_data_peak_eur.index, price_temp_data_peak_eur['CONSUMER_PRICE'], color='C0', lw=1, label='Actual')
plt.xlabel('Date', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.ylim([1000, 2750])
plt.title(f'Actual and mean predicted consumer price for peak hour #{hour_max_eur}: {model_sv_x_peak_eur.name}', size=11, fami
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'});
plt.grid(lw=0.25, color='xkcd:cement');
```

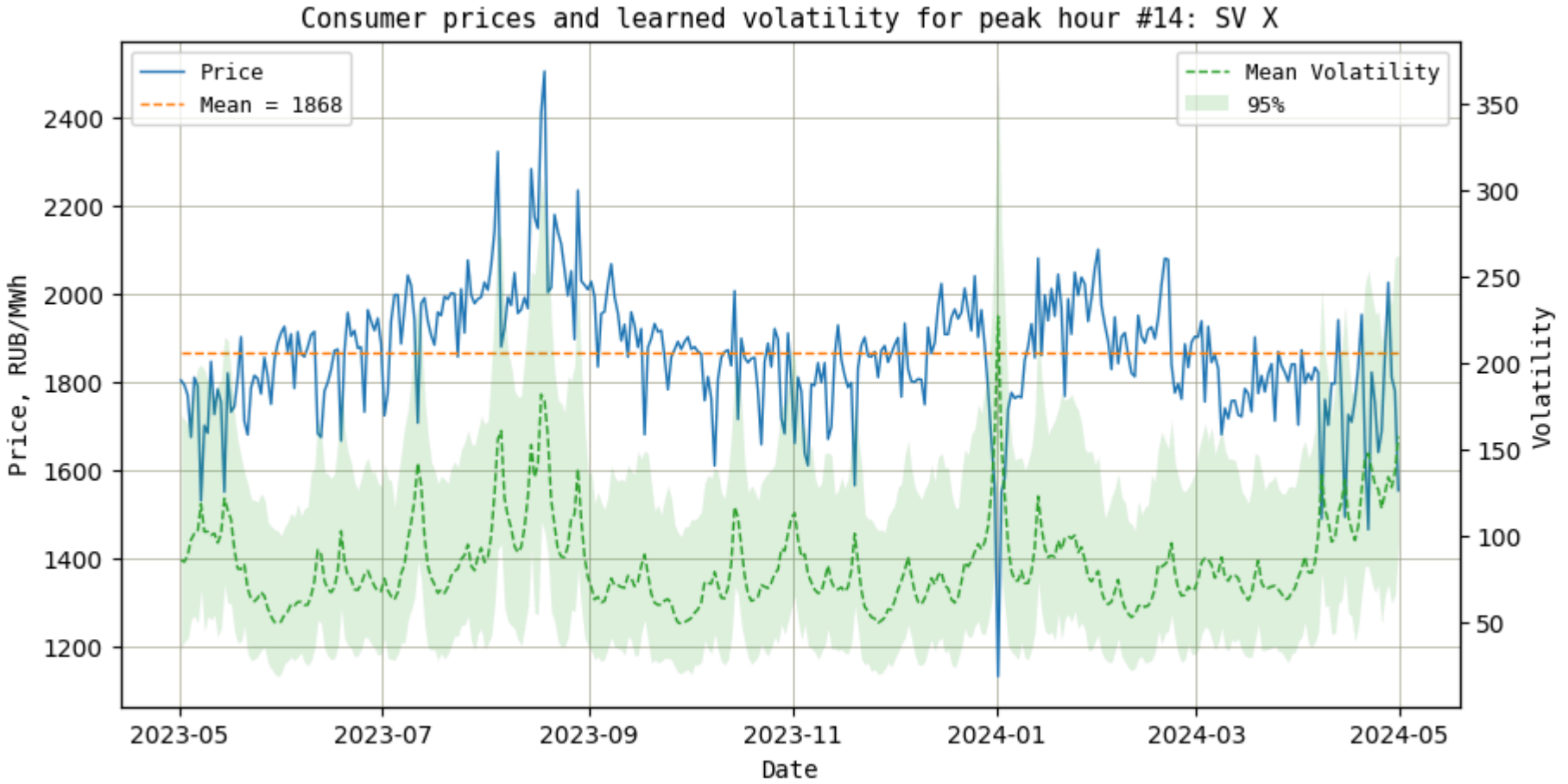


In [ ]:
```
# Volatility at time t over all predictions
vol_sv_x_peak_eur = model_sv_x_peak_eur.get_volatility()
vol_sv_x_peak_eur.shape
```

Out[ ]:
```
(1000, 366)
```

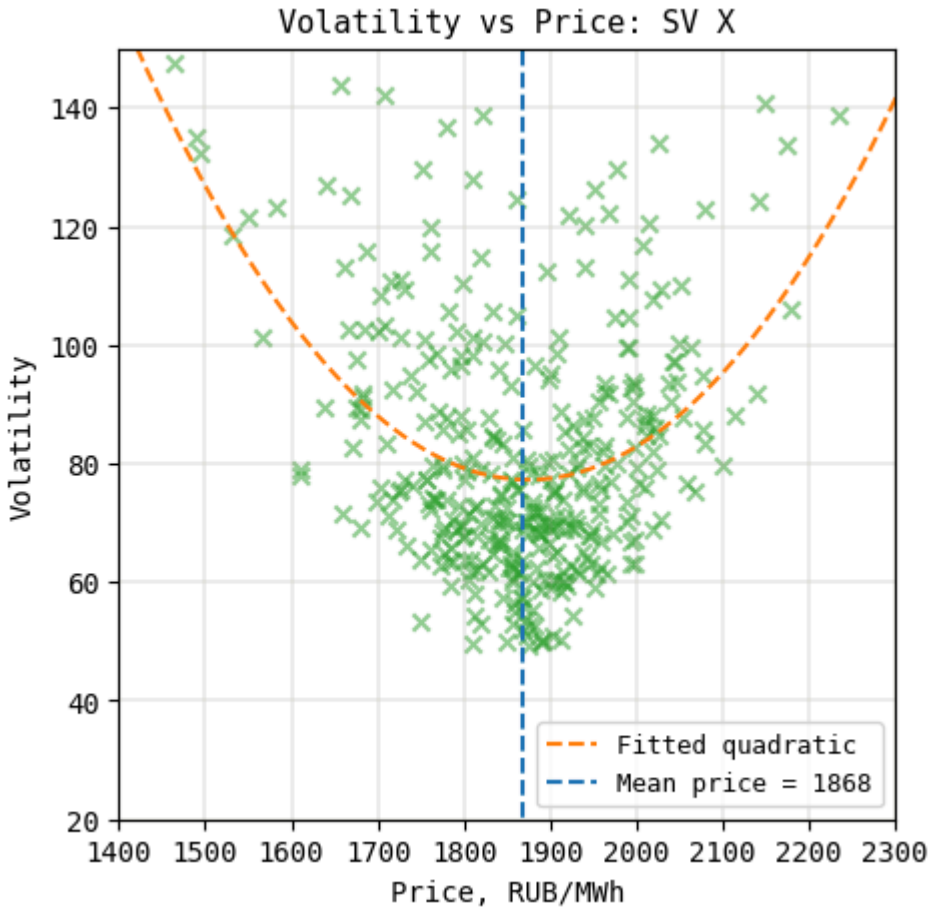
In [ ]:
```
_, ax = plt.subplots(figsize=(10, 5))
ax.plot(price_temp_data_peak_eur.index, price_temp_data_peak_eur['CONSUMER_PRICE'], color='C0', lw=1, label='Price')
ax.hlines(price_temp_data_peak_eur['CONSUMER_PRICE'].mean(), xmin=price_temp_data_peak_eur.index.min(), xmax=price_temp_data_p
ax.set_xlabel('Date', size=10, family='monospace')
ax.set_ylabel('Price, RUB/MWh', size=10, family='monospace')
ax.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
```

```
ax.grid(lw=0.5, color='xkcd:cement')
ax2 = ax.twinx()
ax2.plot(price_temp_data_peak_eur.index, vol_sv_x_peak_eur.mean(axis=0), color='C2', lw=1, ls='--', label='Mean Volatility')
ax2.fill_between(price_temp_data_peak_eur.index, np.percentile(vol_sv_x_peak_eur, [2.5, 97.5], axis=0)[0], np.percentile(vol_s
ax2.set_ylabel('Volatility', size=10, family='monospace')
ax2.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
ax.set_title(f'Consumer prices and learned volatility for peak hour #{hour_max_eur}: {model_sv_x_peak_eur.name}', size=11, fam
```



In [ ]:
```
plt.figure(figsize=(5, 5))
fitted_quadratic = np.poly1d(np.polyfit(price_temp_data_peak_eur['CONSUMER_PRICE'], vol_sv_x_peak_eur.mean(axis=0), deg=2))
x = np.linspace(price_temp_data_peak_eur['CONSUMER_PRICE'].min(), price_temp_data_peak_eur['CONSUMER_PRICE'].max(), price_temp
plt.scatter(price_temp_data_peak_eur['CONSUMER_PRICE'], vol_sv_x_peak_eur.mean(axis=0), color='C2', marker='x', alpha=0.5)
```

```
plt.plot(x, fitted_quadratic(x), color='C1', ls='--', label='Fitted quadratic')
plt.vlines(price_temp_data_peak_eur['CONSUMER_PRICE'].mean(), ymin=0, ymax=350, color='C0', ls='--', label=f"Mean price = {pri
plt.xlabel('Price, RUB/MWh', size=10, family='monospace')
plt.ylabel('Volatility', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xlim([1400, 2300])
plt.ylim([20, 150])
plt.title(f'Volatility vs Price: {model_sv_x_peak_eur.name}', size=11, family='monospace')
plt.legend(loc='lower right', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
alpha = 0.05
adf_stats, adf_pval, _, _, _, _ = adfuller(vol_sv_x_peak_eur.mean(axis=0))
print(f'{adf_stats = :.2f}, {adf_pval = :.2f}', end=', ')
print('Stationary') if adf_pval < alpha else print('Non-stationary')
```

```
adf_stats = -5.58, adf_pval = 0.00, Stationary
```

As with SV Baseline, we can clearly see that volatility *does* depend on consumer price, and our **SV X** model discovered this relation.

Again, as with SV Baseline, we can see a rather V-shaped dependency: the volatility tends to increase for the prices higher and lower than the mean price, while it's minimal around the mean price.

3.5. Goodness-of-fit

In [ ]:
```
# MAE for mean predictions over all draws
mae_sv_x = mean_absolute_error(price_temp_data_peak_eur['CONSUMER_PRICE'], predict_sv_x_peak_eur_mean)
mae_sv_x
```

Out[ ]:
```
84.66766554737612
```

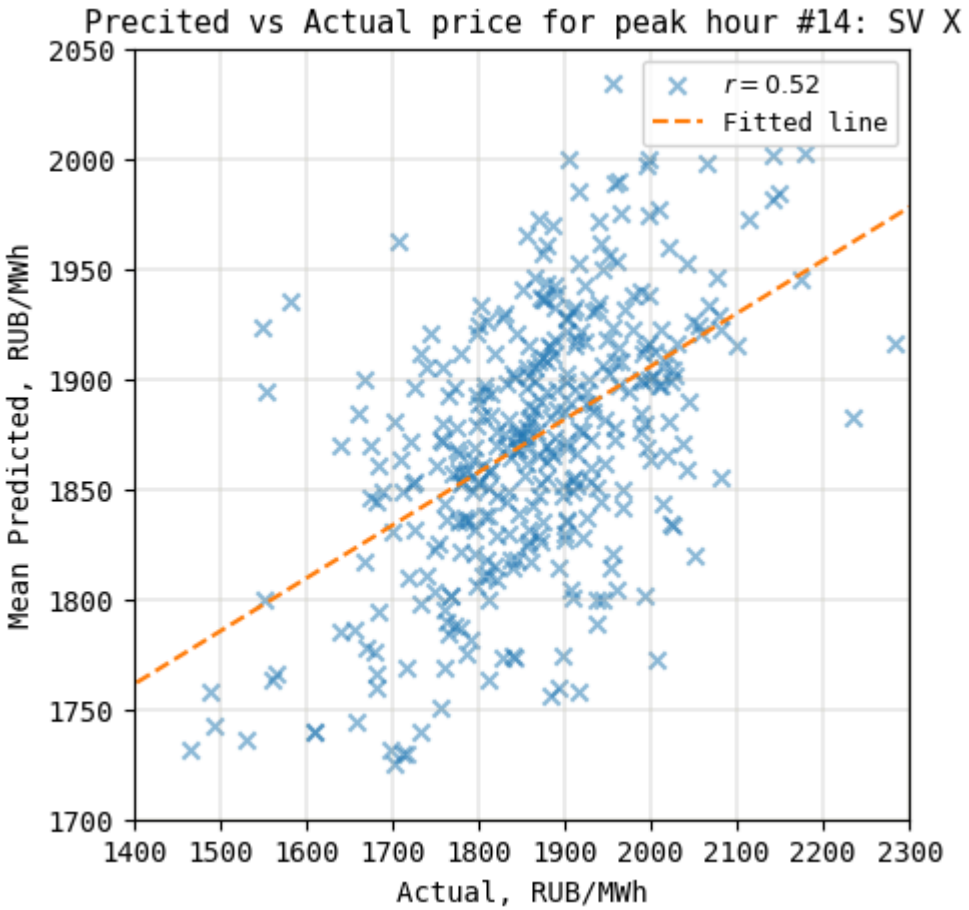
In [ ]:
```
# RMSE for mean predictions over all draws
rmse_sv_x = root_mean_squared_error(price_temp_data_peak_eur['CONSUMER_PRICE'], predict_sv_x_peak_eur_mean)
rmse_sv_x
```

Out[ ]:
```
117.56220534302652
```

Let's check the correlation between mean predicted and actual prices.

In [ ]:
```
r = np.corrcoef(price_temp_data_peak_eur['CONSUMER_PRICE'], predict_sv_x_peak_eur_mean)[0, 1]
fitted_line = np.poly1d(np.polyfit(price_temp_data_peak_eur['CONSUMER_PRICE'], predict_sv_x_peak_eur_mean, deg=1))
x = np.linspace(price_temp_data_peak_eur['CONSUMER_PRICE'].min(), price_temp_data_peak_eur['CONSUMER_PRICE'].max(), price_temp
plt.figure(figsize=(5, 5))
plt.scatter(price_temp_data_peak_eur['CONSUMER_PRICE'], predict_sv_x_peak_eur_mean, color='C0', marker='x', alpha=0.5, label=f
plt.plot(x, fitted_line(x), color='C1', ls='--', label='Fitted line')
plt.xlabel('Actual, RUB/MWh', size=10, family='monospace')
plt.ylabel('Mean Predicted, RUB/MWh', size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xticks(size=10, family='monospace')
```

```
plt.xlim([1400, 2300])
plt.ylim([1700, 2050])
plt.title(f'Precited vs Actual price for peak hour #{hour_max_eur}: {model_sv_x_peak_eur.name}', size=11, family='monospace')
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement');
```

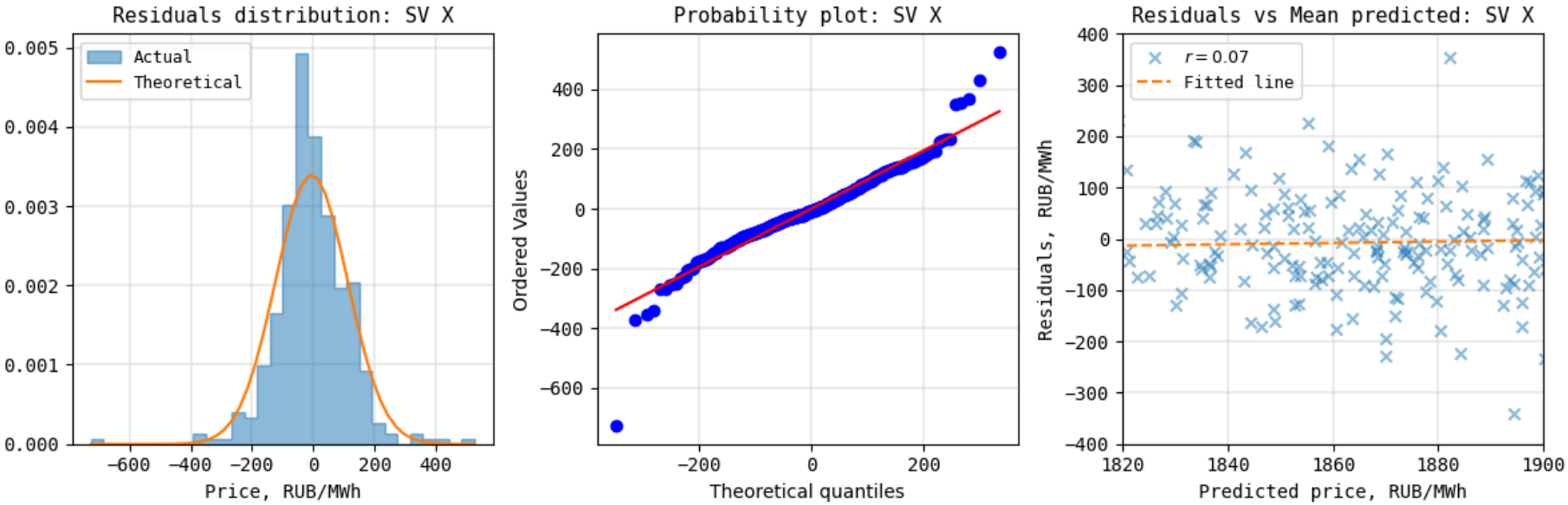


This time the correlation is stronger, as compared to the SV Baseline model.

We should also check the distribution of residuals, probability plot (similar to Q-Q plot), and residuals vs mean predicted prices.

In [ ]:
```
residuals_sv_x = price_data_peak_eur['CONSUMER_PRICE'] - predict_sv_x_peak_eur_mean.values
x = np.linspace(residuals_sv_x.min(), residuals_sv_x.max())
```

```
residuals_sv_x_theor = norm(residuals_sv_x.mean(), residuals_sv_x.std(ddof=1))
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.hist(residuals_sv_x, bins=30, density=True, histtype='stepfilled', color='C0', edgecolor='C0', alpha=0.5, label='Actual')
plt.plot(x, residuals_sv_x_theor.pdf(x), color='C1', label='Theoretical')
plt.xlabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'Residuals distribution: {model_sv_x_peak_eur.name}', size=11, family='monospace')
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement')
plt.gca().set_axisbelow(True)
plt.subplot(1, 3, 2)
probplot(residuals_sv_x, dist=residuals_sv_x_theor, plot=plt)
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.title(f'Probability plot: {model_sv_x_peak_eur.name}', size=11, family='monospace')
plt.grid(lw=0.25, color='xkcd:cement')
plt.gca().set_axisbelow(True)
plt.subplot(1, 3, 3)
r = np.corrcoef(residuals_sv_x, predict_sv_x_peak_eur_mean)[0, 1]
fitted_line = np.poly1d(np.polyfit(predict_sv_x_peak_eur_mean, residuals_sv_x, deg=1))
x = np.linspace(predict_sv_x_peak_eur_mean.min(), predict_sv_x_peak_eur_mean.max(), predict_sv_x_peak_eur_mean.shape[0])
plt.scatter(predict_sv_x_peak_eur_mean, residuals_sv_x, color='C0', marker='x', alpha=0.5, label=f'$r = {r:.2f}$')
plt.plot(x, fitted_line(x), color='C1', ls='--', label='Fitted line')
plt.xlabel('Predicted price, RUB/MWh', size=10, family='monospace')
plt.ylabel('Residuals, RUB/MWh', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(size=10, family='monospace')
plt.xlim([1820, 1900])
plt.ylim([-400, 400])
plt.title(f'Residuals vs Mean predicted: {model_sv_x_peak_eur.name}', size=11, family='monospace')
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
plt.grid(lw=0.25, color='xkcd:cement')
plt.gca().set_axisbelow(True)
plt.tight_layout();
```



Residuals look to be distributed more or less normally. Also they look rather homoscedastic w.r.t. predictions, which means that the model is rather acceptable.

Overall, the SV X model looks to be a better fit for our price data with the air temperature and day of week as exogenous regressors, as compared to the SV Baseline model.

Back

---

4. Cross-validation

We have trained and tested our models on the same fixed year-long time frame. It is a standard practice to train and test models on different portions of the whole data to generate the distribution of metric(s) and ensure model robustness. This process is known as cross-validation: all data is divided into a number of non-intersecting subsets of samples (folds) some of which are used for training and the rest for testing. For data with i.i.d. samples, i.e. when the samples are independent and thus the order of samples is not important, the order of folds is also not important. This is not the case for time series data, where training folds must not be precede the testing folds, otherwise the target leakage will occur and the model cannot be trusted.

Cross-validation setup.

Models:

1. SV Baseline + Peak hour + Price zone 1 (European)
2. SV Baseline + Peak hour + Price zone 2 (Siberian)
3. SV Baseline + Off-peak hour + Price zone 1
4. SV Baseline + Off-peak hour + Price zone 2
5. SV X + Peak hour + Price zone 1
6. SV X + Peak hour + Price zone 2
7. SV X + Off-peak hour + Price zone 1
8. SV X + Off-peak hour + Price zone 2

Time frame: 23.06.2014 -- 30.04.2024 (3600 days), roughly 10 years.

4.1. SV Baseline

Load price data for the whole time frame and both price zones.

In [ ]:
```
# # We need to explicitly specify the column names to correctly parse the xml
# price_data = pd.read_xml(f'https://www.atsenergo.ru/market/stats.xml?period=0&date1=20140623&date2=20240430&type=graph',
#                      names=['ROW_ID',
#                             'DAT',
#                             'PRICE_ZONE_CODE',
#                             'CONSUMER_VOLUME',
#                             'CONSUMER_PRICE',
#                             'CONSUMER_RD_VOLUME',
#                             'CONSUMER_SPOT_VOLUME',
#                             'CONSUMER_PROVIDE_RD',
#                             'CONSUMER_MAX_PRICE',
#                             'CONSUMER_MIN_PRICE',
#                             'SUPPLIER_VOLUME',
#                             'SUPPLIER_PRICE',
#                             'SUPPLIER_RD_VOLUME',
#                             'SUPPLIER_SPOT_VOLUME',
#                             'SUPPLIER_PROVIDE_RD',
```

```
#                             'SUPPLIER_MAX_PRICE',
#                             'SUPPLIER_MIN_PRICE',
#                             'HOUR'],
#                     xpath='//row',
#                     parse_dates=['DAT'])
# # Make datetime
# price_data = price_data.set_index(pd.to_datetime(price_data['DAT'].astype(str) + 'T' + price_data['HOUR'].astype(str) + ':00
# price_data.index.name ='Datetime'
# # We can now drop all unnecessary columns to reduce the dataframe
# price_data = price_data.drop(columns=['ROW_ID',
#                         'DAT',
#                         'CONSUMER_VOLUME',
#                         'CONSUMER_RD_VOLUME',
#                         'CONSUMER_SPOT_VOLUME',
#                         'CONSUMER_PROVIDE_RD',
#                         'CONSUMER_MAX_PRICE',
#                         'CONSUMER_MIN_PRICE',
#                         'SUPPLIER_VOLUME',
#                         'SUPPLIER_PRICE',
#                         'SUPPLIER_RD_VOLUME',
#                         'SUPPLIER_SPOT_VOLUME',
#                         'SUPPLIER_PROVIDE_RD',
#                         'SUPPLIER_MAX_PRICE',
#                         'SUPPLIER_MIN_PRICE']).dropna()
# price_data = price_data.loc[~price_data.index.isna()].sort_index()
# price_data['Weekday'] = price_data.index.day_of_week
# price_data.to_csv('./data/price_data_20140623_20240430.csv')
```

In [ ]:
```
price_data = pd.read_csv('./data/price_data_20140623_20240430.zip', index_col='Datetime', parse_dates=['Datetime'])
```

In [ ]:
```
price_data.shape
```

Out[ ]:
```
(172800, 4)
```

In [ ]:
```
price_data.head(4)
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Weekday
2014-06-23 00:00:00	1	1147.08	0	0
2014-06-23 00:00:00	2	685.81	0	0
2014-06-23 01:00:00	2	684.27	1	0
2014-06-23 01:00:00	1	1037.39	1	0

In [ ]:
```
price_data.tail(4)
```

Out[ ]:

Datetime	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Weekday
2024-04-30 22:00:00	2	1465.38	22	1
2024-04-30 22:00:00	1	1442.53	22	1
2024-04-30 23:00:00	2	1439.88	23	1
2024-04-30 23:00:00	1	1327.94	23	1

In [ ]:
```
price_data.describe()
```

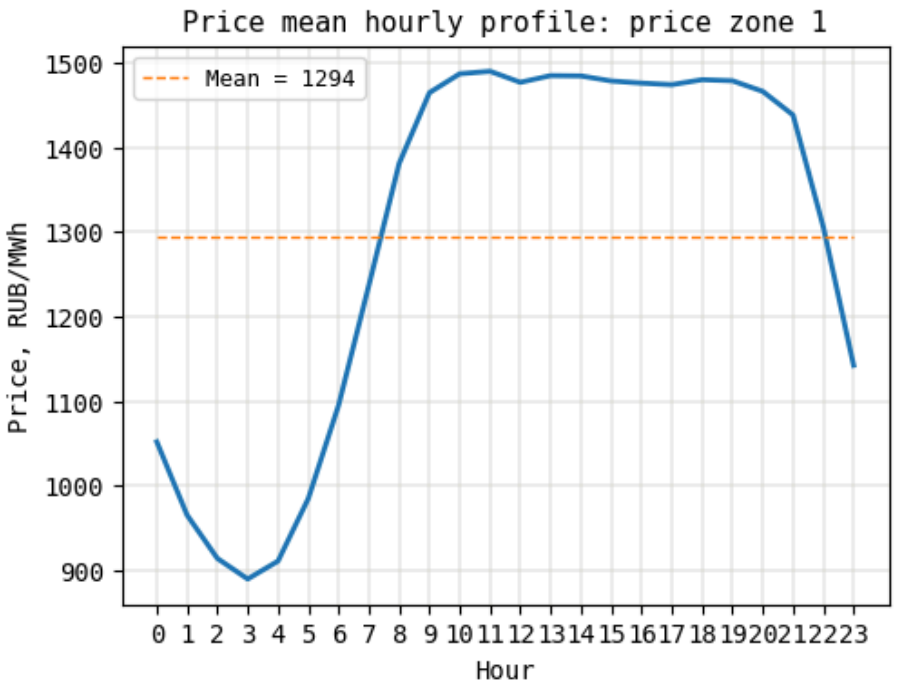
Out[ ]:

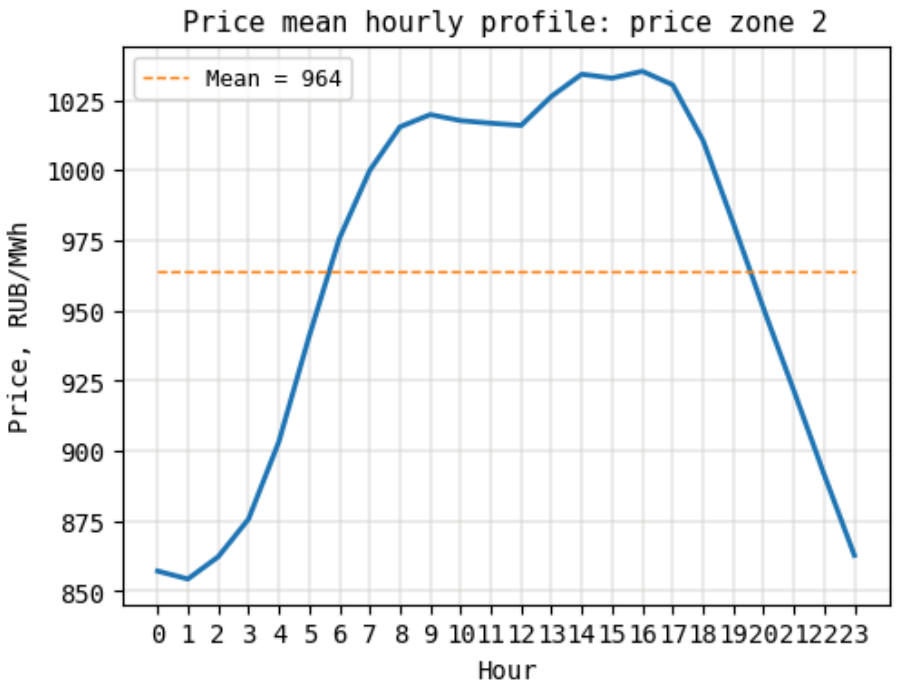
	PRICE_ZONE_CODE	CONSUMER_PRICE	HOUR	Weekday
count	172800.000000	172800.000000	172800.000000	172800.000000
mean	1.500000	1128.835519	11.500000	2.998611
std	0.500001	311.825787	6.922207	2.000352
min	1.000000	0.000000	0.000000	0.000000
25%	1.000000	901.610000	5.750000	1.000000
50%	1.500000	1061.570000	11.500000	3.000000
75%	2.000000	1355.850000	17.250000	5.000000
max	2.000000	2504.960000	23.000000	6.000000

In [ ]:
```
# Price zone 1 (European)
price_data_eur = price_data[price_data['PRICE_ZONE_CODE'] == 1]
# Price zone 2 (Siberian)
price_data_sib = price_data[price_data['PRICE_ZONE_CODE'] == 2]
```

In [ ]:
```
# Hour profiles
price_data_daily_agg_eur = price_data_eur.groupby('HOUR')['CONSUMER_PRICE'].mean()
price_data_daily_agg_sib = price_data_sib.groupby('HOUR')['CONSUMER_PRICE'].mean()
```

In [ ]:
```
plt.figure(figsize=(12, 4))
for i, hour_profile in zip(range(1, 3), [price_data_daily_agg_eur, price_data_daily_agg_sib]):
    plt.subplot(1, 2, i)
    plt.plot(hour_profile.index, hour_profile, color='C0', lw=2)
    plt.hlines(hour_profile.mean(), xmin=hour_profile.index.min(), xmax=hour_profile.index.max(), color='C1', lw=1, ls='--', l
    plt.xlabel('Hour', size=10, family='monospace')
    plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
    plt.xticks(hour_profile.index, size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
    plt.title(f'Price mean hourly profile: price zone {i}', size=11, family='monospace')
    plt.legend(prop={'size': 9, 'family': 'monospace'})
    plt.grid(lw=0.25, color='xkcd:cement');
```





In [ ]:
```
# Peak and off-peak hours
hour_max_eur = price_data_daily_agg_eur.idxmax()
hour_min_eur = price_data_daily_agg_eur.idxmin()
hour_max_sib = price_data_daily_agg_sib.idxmax()
hour_min_sib = price_data_daily_agg_sib.idxmin()
(hour_max_eur, hour_min_eur), (hour_max_sib, hour_min_sib)
```

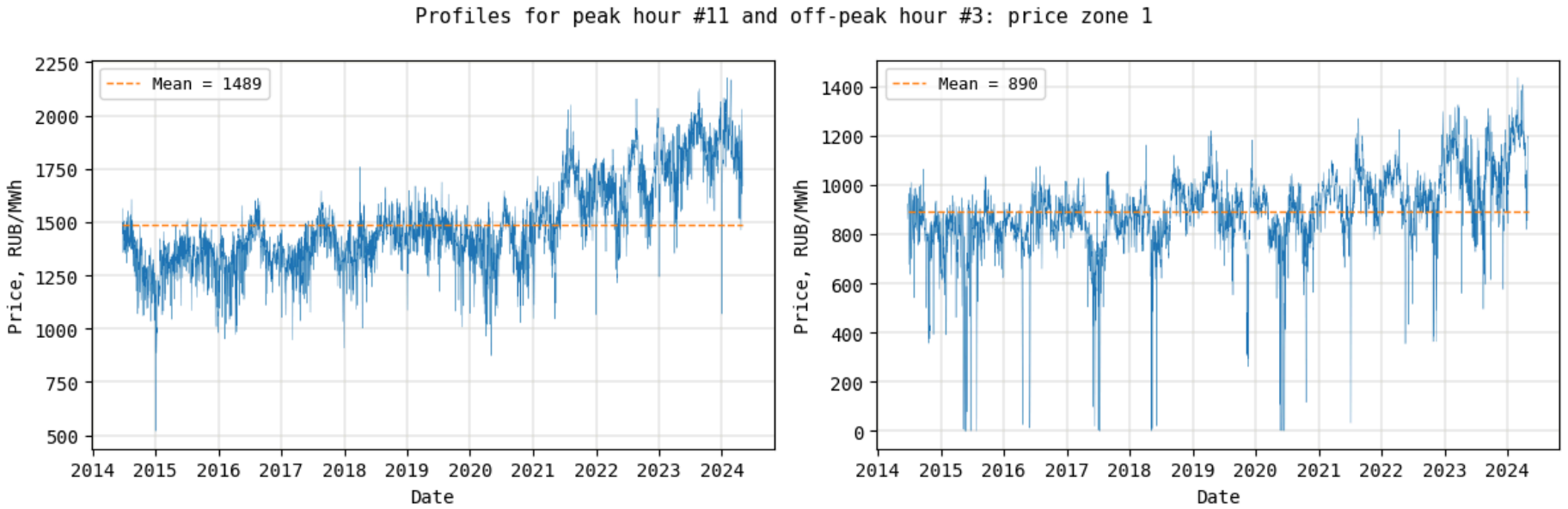
Out[ ]:
```
((11, 3), (16, 1))
```

All cross-validation subsets for SV Baseline model.

In [ ]:
```
price_data_peak_eur = price_data_eur[price_data_eur['HOUR'] == hour_max_eur]
price_data_offpeak_eur = price_data_eur[price_data_eur['HOUR'] == hour_min_eur]
price_data_peak_sib = price_data_sib[price_data_sib['HOUR'] == hour_max_sib]
price_data_offpeak_sib = price_data_sib[price_data_sib['HOUR'] == hour_min_sib]
```

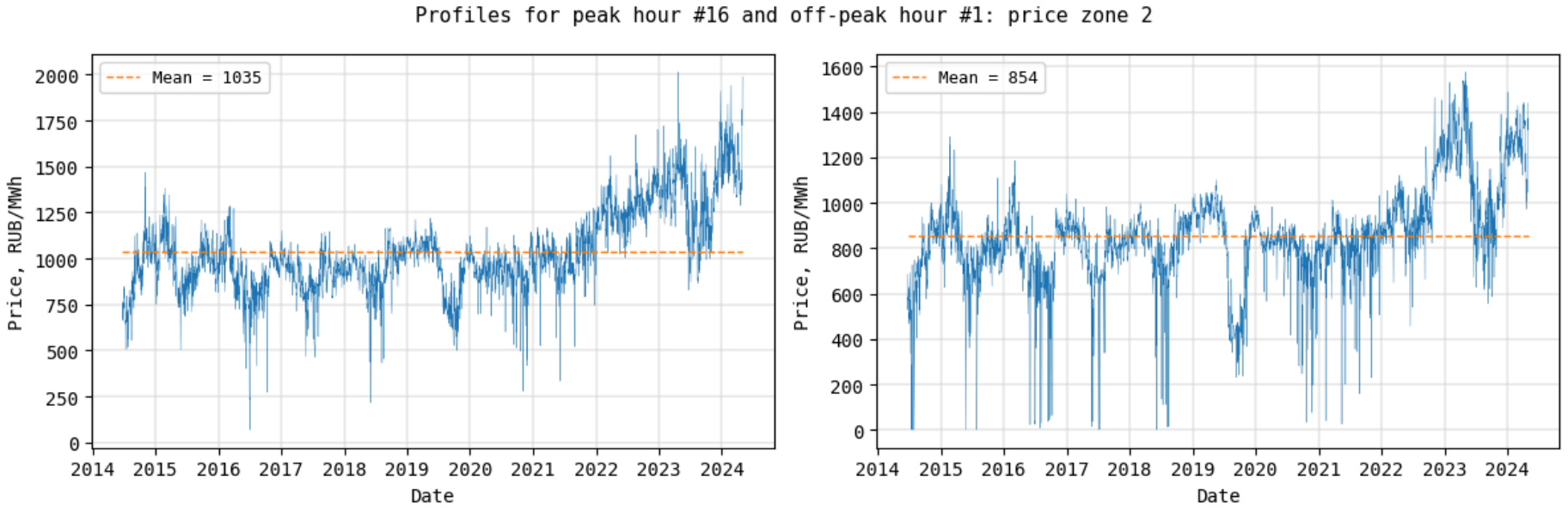
In [ ]:
```
plt.figure(figsize=(12, 4))
for i, hour_profile in zip(range(1, 3), [price_data_peak_eur, price_data_offpeak_eur]):
```

```
    plt.subplot(1, 2, i)
    plt.plot(hour_profile.index, hour_profile['CONSUMER_PRICE'], color='C0', lw=0.25)
    plt.hlines(hour_profile['CONSUMER_PRICE'].mean(), xmin=hour_profile.index.min(), xmax=hour_profile.index.max(), color='C1'
    plt.xlabel('Date', size=10, family='monospace')
    plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
    plt.xticks(size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
    plt.legend(prop={'size': 9, 'family': 'monospace'})
    plt.grid(lw=0.25, color='xkcd:cement');
    plt.gca().set_axisbelow(True)
plt.suptitle(f'Profiles for peak hour #{hour_max_eur} and off-peak hour #{hour_min_eur}: price zone 1', size=11, family='monos
plt.tight_layout();
```



In [ ]:
```
plt.figure(figsize=(12, 4))
for i, hour_profile in zip(range(1, 3), [price_data_peak_sib, price_data_offpeak_sib]):
    plt.subplot(1, 2, i)
    plt.plot(hour_profile.index, hour_profile['CONSUMER_PRICE'], color='C0', lw=0.25)
    plt.hlines(hour_profile['CONSUMER_PRICE'].mean(), xmin=hour_profile.index.min(), xmax=hour_profile.index.max(), color='C1'
    plt.xlabel('Date', size=10, family='monospace')
    plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
    plt.xticks(size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
```

```
    plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'})
    plt.grid(lw=0.25, color='xkcd:cement');
    plt.gca().set_axisbelow(True)
plt.suptitle(f'Profiles for peak hour #{hour_max_sib} and off-peak hour #{hour_min_sib}: price zone 2', size=11, family='monos
plt.tight_layout();
```



All SV Baseline models to cross-validate.

In [ ]:
```
with open('./models/sv_base_fit.stan', 'r') as fh:
  sv_base_code_fit = fh.read()
with open('./models/sv_base_predict.stan', 'r') as fh:
  sv_base_code_predict = fh.read()
num_samples = 1_000
```

In [ ]:
```
# 1) SV Baseline + Peak hour + Price zone 1
model_sv_base_peak_eur = StanModel(kind='sv_base',
                                   name='SV Baseline + Peak hour + Price zone 1',
                                   stan_code_fit=sv_base_code_fit,
                                   stan_code_predict=sv_base_code_predict,
                                   num_samples=num_samples)
```

In [ ]:
```
# 2) SV Baseline + Peak hour + Price zone 2
model_sv_base_peak_sib = StanModel(kind='sv_base',
                                   name='SV Baseline + Peak hour + Price zone 2',
                                   stan_code_fit=sv_base_code_fit,
                                   stan_code_predict=sv_base_code_predict,
                                   num_samples=num_samples)
```

In [ ]:
```
# 3) SV Baseline + Off-peak hour + Price zone 1
model_sv_base_offpeak_eur = StanModel(kind='sv_base',
                                   name='SV Baseline + Off-peak hour + Price zone 1',
                                   stan_code_fit=sv_base_code_fit,
                                   stan_code_predict=sv_base_code_predict,
                                   num_samples=num_samples)
```

In [ ]:
```
# 4) SV Baseline + Off-peak hour + Price zone 2
model_sv_base_offpeak_sib = StanModel(kind='sv_base',
                                   name='SV Baseline + Off-peak hour + Price zone 2',
                                   stan_code_fit=sv_base_code_fit,
                                   stan_code_predict=sv_base_code_predict,
                                   num_samples=num_samples)
```

Cross-validation strategy:

1. Choose some starting date;
2. Train for 360 days (approx. 1 year) from starting date;
3. Predict 90 days (approx. 3 months = 1 quarter) ahead using the most recent log volatility obtained during training;
4. Move the date sliding window repeat from 2.

The number of date sliding windows $N_w = \frac{3600 - 30 \times 12}{30 \times 3} = 36$.

Let's use scikit-learn's Time Series cross-validator to correctly split our consumer price (SV Baseline / SV X) and air temperature (SV X) data into train and test subsets.

In [ ]:
```
n_windows = 36 # Number of sliding windows
ts_cv_sv_base = TimeSeriesSplit(n_splits=n_windows, max_train_size=30*12, test_size=30*3, gap=0)
```

In [ ]:
```
print('Split # | Train fold                    | Test fold')
print('--------|-------------------------------|-------------------------------')
# All 4 subsets have equal size, so get split indices from any of the four
for i, split in enumerate(ts_cv_sv_base.split(price_data_peak_eur)):
    train_start = price_data_peak_eur.iloc[split[0]].index.min().date()
    train_end = price_data_peak_eur.iloc[split[0]].index.max().date()
    test_start = price_data_peak_eur.iloc[split[1]].index.min().date()
    test_end = price_data_peak_eur.iloc[split[1]].index.max().date()
    print(f'{i + 1:>7} | {train_start} -- {train_end} ({(train_end - train_start).days + 1}) | {test_start} -- {test_end} ({(t
```

Split #	Train fold	Test fold
1	2014-06-23 -- 2015-06-17 (360)	2015-06-18 -- 2015-09-15 (90)
2	2014-09-21 -- 2015-09-15 (360)	2015-09-16 -- 2015-12-14 (90)
3	2014-12-20 -- 2015-12-14 (360)	2015-12-15 -- 2016-03-13 (90)
4	2015-03-20 -- 2016-03-13 (360)	2016-03-14 -- 2016-06-11 (90)
5	2015-06-18 -- 2016-06-11 (360)	2016-06-12 -- 2016-09-09 (90)
6	2015-09-16 -- 2016-09-09 (360)	2016-09-10 -- 2016-12-08 (90)
7	2015-12-15 -- 2016-12-08 (360)	2016-12-09 -- 2017-03-08 (90)
8	2016-03-14 -- 2017-03-08 (360)	2017-03-09 -- 2017-06-06 (90)
9	2016-06-12 -- 2017-06-06 (360)	2017-06-07 -- 2017-09-04 (90)
10	2016-09-10 -- 2017-09-04 (360)	2017-09-05 -- 2017-12-03 (90)
11	2016-12-09 -- 2017-12-03 (360)	2017-12-04 -- 2018-03-03 (90)
12	2017-03-09 -- 2018-03-03 (360)	2018-03-04 -- 2018-06-01 (90)
13	2017-06-07 -- 2018-06-01 (360)	2018-06-02 -- 2018-08-30 (90)
14	2017-09-05 -- 2018-08-30 (360)	2018-08-31 -- 2018-11-28 (90)
15	2017-12-04 -- 2018-11-28 (360)	2018-11-29 -- 2019-02-26 (90)
16	2018-03-04 -- 2019-02-26 (360)	2019-02-27 -- 2019-05-27 (90)
17	2018-06-02 -- 2019-05-27 (360)	2019-05-28 -- 2019-08-25 (90)
18	2018-08-31 -- 2019-08-25 (360)	2019-08-26 -- 2019-11-23 (90)
19	2018-11-29 -- 2019-11-23 (360)	2019-11-24 -- 2020-02-21 (90)
20	2019-02-27 -- 2020-02-21 (360)	2020-02-22 -- 2020-05-21 (90)
21	2019-05-28 -- 2020-05-21 (360)	2020-05-22 -- 2020-08-19 (90)
22	2019-08-26 -- 2020-08-19 (360)	2020-08-20 -- 2020-11-17 (90)
23	2019-11-24 -- 2020-11-17 (360)	2020-11-18 -- 2021-02-15 (90)
24	2020-02-22 -- 2021-02-15 (360)	2021-02-16 -- 2021-05-16 (90)
25	2020-05-22 -- 2021-05-16 (360)	2021-05-17 -- 2021-08-14 (90)
26	2020-08-20 -- 2021-08-14 (360)	2021-08-15 -- 2021-11-12 (90)
27	2020-11-18 -- 2021-11-12 (360)	2021-11-13 -- 2022-02-10 (90)
28	2021-02-16 -- 2022-02-10 (360)	2022-02-11 -- 2022-05-11 (90)
29	2021-05-17 -- 2022-05-11 (360)	2022-05-12 -- 2022-08-09 (90)
30	2021-08-15 -- 2022-08-09 (360)	2022-08-10 -- 2022-11-07 (90)
31	2021-11-13 -- 2022-11-07 (360)	2022-11-08 -- 2023-02-05 (90)
32	2022-02-11 -- 2023-02-05 (360)	2023-02-06 -- 2023-05-06 (90)
33	2022-05-12 -- 2023-05-06 (360)	2023-05-07 -- 2023-08-04 (90)
34	2022-08-10 -- 2023-08-04 (360)	2023-08-05 -- 2023-11-02 (90)
35	2022-11-08 -- 2023-11-02 (360)	2023-11-03 -- 2024-01-31 (90)
36	2023-02-06 -- 2024-01-31 (360)	2024-02-01 -- 2024-04-30 (90)

In [ ]:
```
#price_data_peak_eur.iloc[all_splits[0][0]]
```

In [ ]:
```
%%capture
models = [model_sv_base_peak_eur, model_sv_base_peak_sib, model_sv_base_offpeak_eur, model_sv_base_offpeak_sib]
subsets = [price_data_peak_eur, price_data_peak_sib, price_data_offpeak_eur, price_data_offpeak_sib]
cv_results_sv_base_df = pd.DataFrame(columns=['model_kind', 'model_name', 'train_indices', 'test_indices', 'fit_time', 'score_
print('Running cross-validation...')
for model, subset in tqdm(zip(models, subsets)):
    print(f'Model: {model.name}')
    cv_result = cross_validate(model,
                               X=subset['CONSUMER_PRICE'],
                               y=subset['CONSUMER_PRICE'],
                               cv=ts_cv_sv_base,
                               scoring=['neg_mean_absolute_error', 'neg_root_mean_squared_error'],
                               return_indices=True,
                               n_jobs=4)
    cv_result_df = pd.concat([pd.Series(cv_result['indices']['train'], name='train_indices'),
    pd.Series(cv_result['indices']['test'], name='test_indices'),
    pd.Series(cv_result['fit_time'], name='fit_time'),
    pd.Series(cv_result['score_time'], name='score_time'),
    pd.Series(-cv_result['test_neg_mean_absolute_error'], name='test_mae'),
    pd.Series(-cv_result['test_neg_root_mean_squared_error'], name='test_rmse')], axis=1)
    cv_results_sv_base_df = pd.concat([cv_results_sv_base_df, pd.concat([pd.Series([model.kind] * n_windows, name='model_kind'
cv_results_sv_base_df = cv_results_sv_base_df.reset_index(drop=True)
print('Done!')
```

In [ ]:
```
cv_results_sv_base_df
```

Out[ ]:

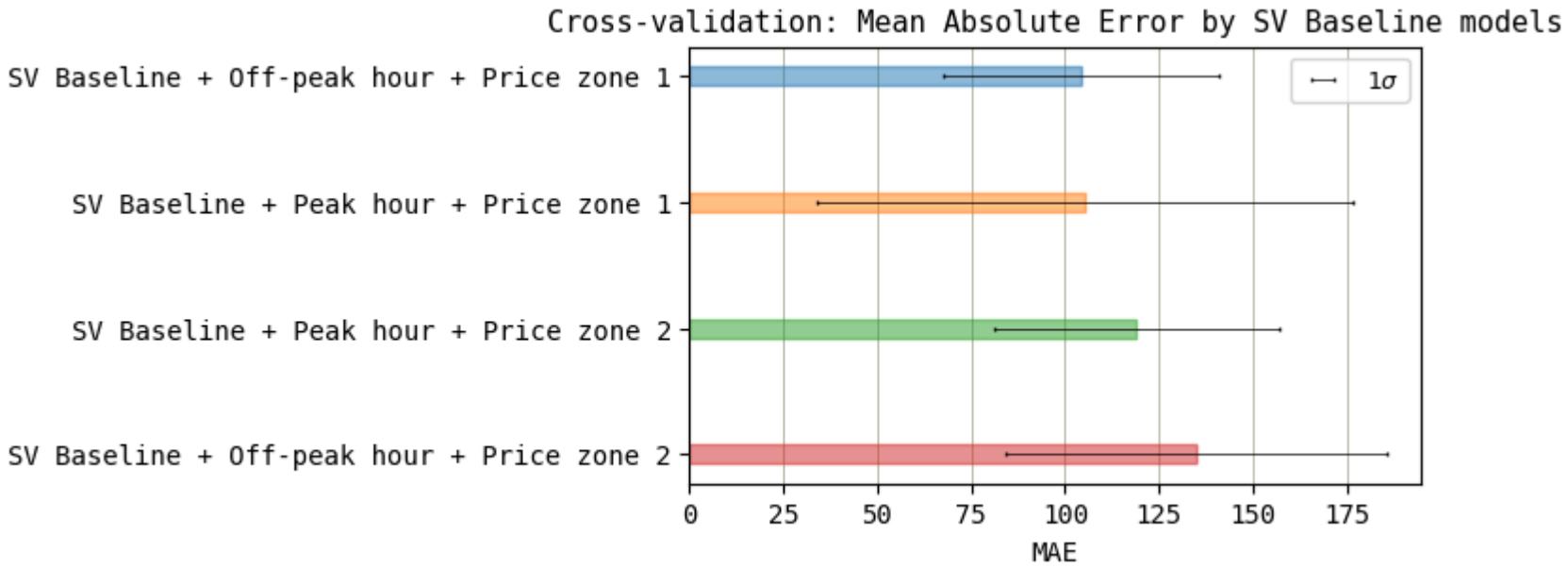
	model_kind	model_name	train_indices	test_indices	fit_time	score_time	test_mae	test_rmse
0	sv_base	SV Baseline + Peak hour + Price zone 1	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[360, 361, 362, 363, 364, 365, 366, 367, 368, ...	0.655952	0.241488	66.932230	82.187065
1	sv_base	SV Baseline + Peak hour + Price zone 1	[90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, ...	[450, 451, 452, 453, 454, 455, 456, 457, 458, ...	6.409832	0.208298	91.498069	102.352921
2	sv_base	SV Baseline + Peak hour + Price zone 1	[180, 181, 182, 183, 184, 185, 186, 187, 188, ...	[540, 541, 542, 543, 544, 545, 546, 547, 548, ...	10.234515	0.201207	98.059416	131.124286
3	sv_base	SV Baseline + Peak hour + Price zone 1	[270, 271, 272, 273, 274, 275, 276, 277, 278, ...	[630, 631, 632, 633, 634, 635, 636, 637, 638, ...	6.432677	0.206495	90.657318	116.004756
4	sv_base	SV Baseline + Peak hour + Price zone 1	[360, 361, 362, 363, 364, 365, 366, 367, 368, ...	[720, 721, 722, 723, 724, 725, 726, 727, 728, ...	0.751844	0.065485	144.874059	158.346917
...	...	...	...	...	...	...	...	...
139	sv_base	SV Baseline + Off-peak hour + Price zone 2	[2790, 2791, 2792, 2793, 2794, 2795, 2796, 279...	[3150, 3151, 3152, 3153, 3154, 3155, 3156, 315...	10.887597	0.193706	303.764088	326.017876
140	sv_base	SV Baseline + Off-peak hour + Price zone 2	[2880, 2881, 2882, 2883, 2884, 2885, 2886, 288...	[3240, 3241, 3242, 3243, 3244, 3245, 3246, 324...	0.702327	0.083814	175.065520	206.851707
141	sv_base	SV Baseline + Off-peak hour + Price zone 2	[2970, 2971, 2972, 2973, 2974, 2975, 2976, 297...	[3330, 3331, 3332, 3333, 3334, 3335, 3336, 333...	9.431044	0.206065	261.340821	287.419361
142	sv_base	SV Baseline + Off-peak hour + Price zone 2	[3060, 3061, 3062, 3063, 3064, 3065, 3066, 306...	[3420, 3421, 3422, 3423, 3424, 3425, 3426, 342...	0.686467	0.051898	121.891326	137.366055
143	sv_base	SV Baseline + Off-peak hour + Price zone 2	[3150, 3151, 3152, 3153, 3154, 3155, 3156, 315...	[3510, 3511, 3512, 3513, 3514, 3515, 3516, 351...	0.636099	0.244597	159.867295	178.763519

144 rows × 8 columns

In [ ]:
```
# Mean MAE by model
mae_sv_base_cross = cv_results_sv_base_df.groupby('model_name')['test_mae']
mae_sv_base_cross.mean().sort_values()
```

Out[ ]:
```
model_name
SV Baseline + Off-peak hour + Price zone 1    104.422811
SV Baseline + Peak hour + Price zone 1        105.385055
SV Baseline + Peak hour + Price zone 2        119.140971
SV Baseline + Off-peak hour + Price zone 2    134.863451
Name: test_mae, dtype: float64
```

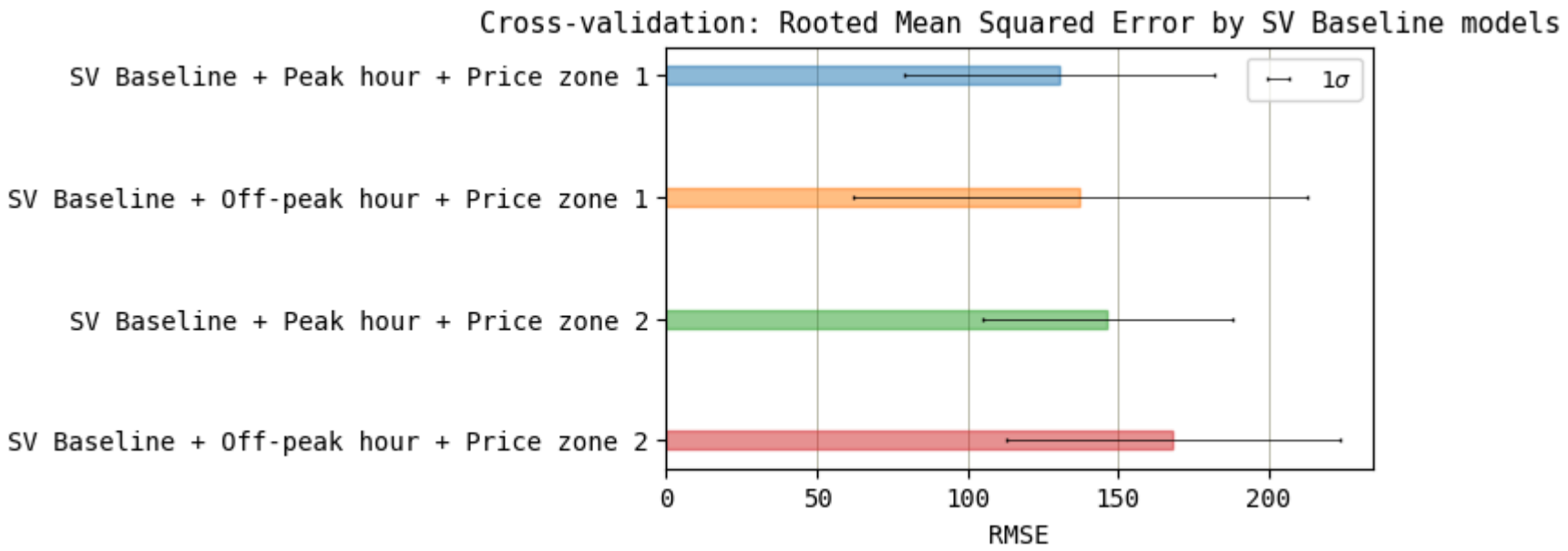
In [ ]:
```
plt.figure(figsize=(5, 3))
bars = plt.barh(range(mae_sv_base_cross.mean().shape[0]), mae_sv_base_cross.mean().sort_values(), xerr=mae_sv_base_cross.std(d
plt.gca().invert_yaxis()
plt.xlabel('MAE', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(range(mae_sv_base_cross.mean().shape[0]), mae_sv_base_cross.mean().sort_values().index, size=10, family='monospace'
plt.grid(axis='x', lw=0.5, color='xkcd:cement')
for i in range(len(bars)):
    bars[i].set_edgecolor(f'C{i}')
    bars[i].set_color(f'C{i}')
plt.gca().set_axisbelow(True)
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.title('Cross-validation: Mean Absolute Error by SV Baseline models', size=11, family='monospace');
```



In [ ]:
```
# Mean RMSE by model
rmse_sv_base_cross = cv_results_sv_base_df.groupby('model_name')['test_rmse']
rmse_sv_base_cross.mean().sort_values()
```

Out[ ]:
```
model_name
SV Baseline + Peak hour + Price zone 1        130.526412
SV Baseline + Off-peak hour + Price zone 1    137.501447
SV Baseline + Peak hour + Price zone 2        146.445320
SV Baseline + Off-peak hour + Price zone 2    168.305299
Name: test_rmse, dtype: float64
```

In [ ]:
```
plt.figure(figsize=(5, 3))
bars = plt.barh(range(rmse_sv_base_cross.mean().shape[0]), rmse_sv_base_cross.mean().sort_values(), xerr=rmse_sv_base_cross.st
plt.gca().invert_yaxis()
plt.xlabel('RMSE', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(range(rmse_sv_base_cross.mean().shape[0]), rmse_sv_base_cross.mean().sort_values().index, size=10, family='monospac
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.grid(axis='x', lw=0.5, color='xkcd:cement')
for i in range(len(bars)):
    bars[i].set_edgecolor(f'C{i}')
    bars[i].set_color(f'C{i}')
plt.gca().set_axisbelow(True)
plt.title('Cross-validation: Rooted Mean Squared Error by SV Baseline models', size=11, family='monospace');
```



Both metrics show that on average:

- SV Baseline models for Price zone 1 (European) have higher quality than for Price zone 2 (Siberian);
- the best model is for Price zone 1 for Peak hour;
- the worst model is for Price zone 2 for Off-peak hour.

4.2. SV X

Load air temperature data for the whole time frame and both price zones.

In [ ]:
```
# Price zone 1: Moscow
temp_data_eur = pd.read_csv(f'./data/UUEE.23.06.2014.30.04.2024.1.0.0.ru.utf8.00000000.zip',
                            sep=';',
                            encoding='utf-8',
                            skiprows=7,
                            index_col=0,
                            dayfirst=True,
                            usecols=[0, 1],
```

```
                        names=['Datetime', 'Temperature'],
                        parse_dates=True).dropna().sort_index()
```

In [ ]:
```
temp_data_eur
```

Out[ ]:

Datetime	Temperature
2014-06-23 00:00:00	8.0
2014-06-23 00:30:00	9.0
2014-06-23 01:00:00	8.0
2014-06-23 01:30:00	10.0
2014-06-23 02:00:00	12.0
...	...
2024-04-30 21:30:00	18.0
2024-04-30 22:00:00	19.0
2024-04-30 22:30:00	19.0
2024-04-30 23:00:00	18.0
2024-04-30 23:30:00	17.0

164068 rows × 1 columns

In [ ]:
```
# Price zone 2: Novosibirsk
temp_data_sib = pd.read_csv(f'./data/UNNT.23.06.2014.30.04.2024.1.0.0.ru.utf8.00000000.zip',
                        sep=';',
                        encoding='utf-8',
                        skiprows=7,
                        index_col=0,
                        dayfirst=True,
                        usecols=[0, 1],
                        names=['Datetime', 'Temperature'],
```

```
                          parse_dates=True).dropna().sort_index()
# Don't forget to convert local Novosibirsk time to Moscow time, since all price data is in Moscow time
temp_data_sib.index = temp_data_sib.index - pd.Timedelta(hours=4)
```

In [ ]:
```
temp_data_sib
```

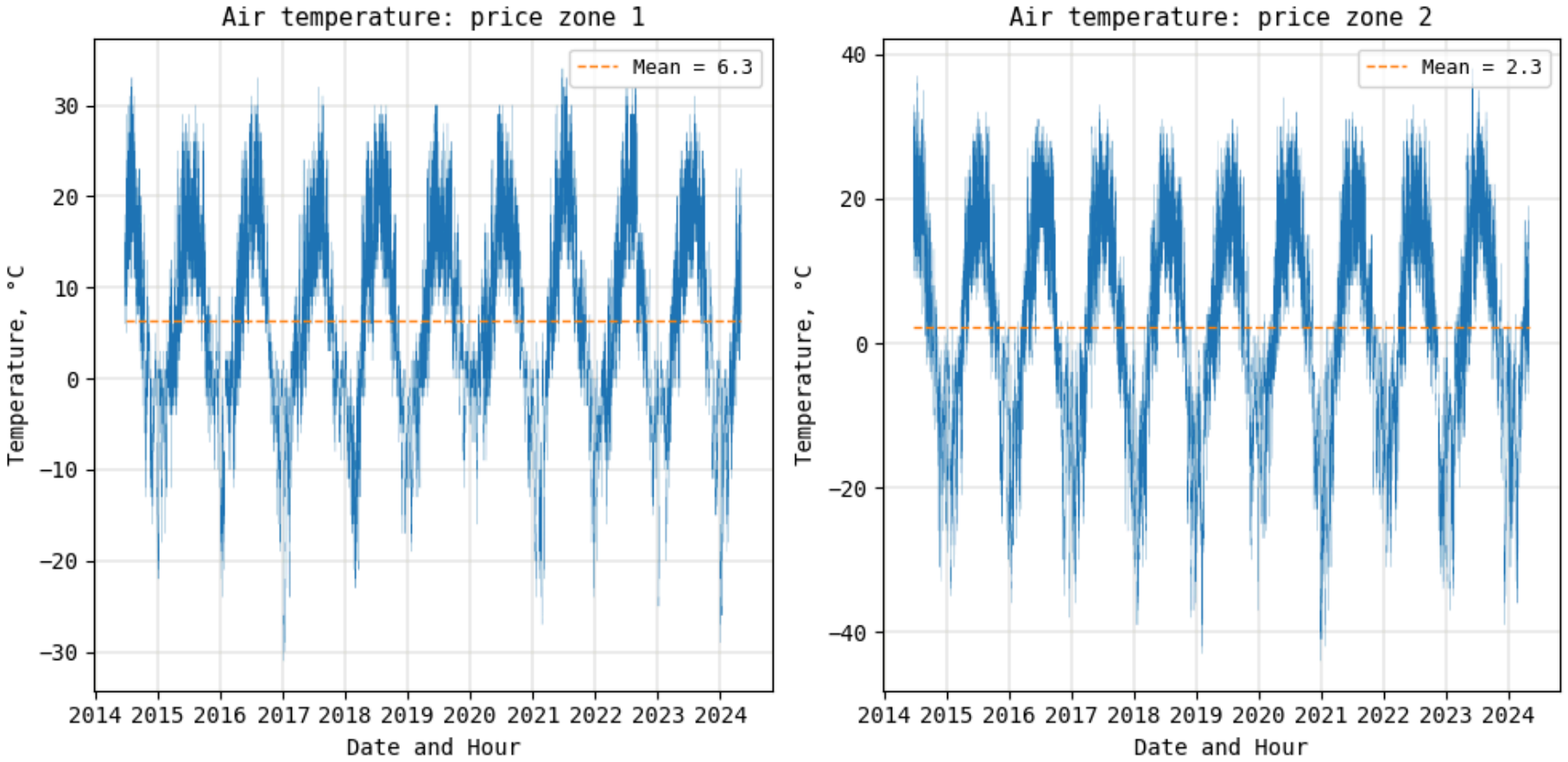
Out[ ]:

Datetime	Temperature
2014-06-22 20:00:00	21.0
2014-06-22 20:30:00	20.0
2014-06-22 21:00:00	19.0
2014-06-22 21:30:00	18.0
2014-06-22 22:00:00	17.0
...	...
2024-04-30 17:30:00	3.0
2024-04-30 18:00:00	2.0
2024-04-30 18:30:00	1.0
2024-04-30 19:00:00	1.0
2024-04-30 19:30:00	-1.0

163741 rows × 1 columns

In [ ]:
```
plt.figure(figsize=(10, 5))
for i, temp_history in zip(range(1, 3), [temp_data_eur, temp_data_sib]):
    plt.subplot(1, 2, i)
    plt.plot(temp_history.index, temp_history['Temperature'], color='C0', lw=0.1)
    plt.hlines(temp_history['Temperature'].mean(), xmin=temp_history.index.min(), xmax=temp_history.index.max(), color='C1', l
    plt.xlabel('Date and Hour', size=10, family='monospace')
    plt.ylabel('Temperature, °C', size=10, family='monospace')
    plt.xticks(size=10, family='monospace')
```

```
    plt.yticks(size=10, family='monospace')
    plt.title(f'Air temperature: price zone {i}', size=11, family='monospace')
    plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
    plt.grid(lw=0.25, color='xkcd:cement')
plt.tight_layout();
```



In [ ]:
```
# Join consumer price and air temperature data
price_temp_data_peak_eur = pd.merge(price_data_peak_eur, temp_data_eur, on='Datetime')
price_temp_data_offpeak_eur = pd.merge(price_data_offpeak_eur, temp_data_eur, on='Datetime')
price_temp_data_peak_sib = pd.merge(price_data_peak_sib, temp_data_sib, on='Datetime')
price_temp_data_offpeak_sib = pd.merge(price_data_offpeak_sib, temp_data_sib, on='Datetime')
```

All SV X models to cross-validate.

In [ ]:
```
with open('./models/sv_x_fit.stan', 'r') as fh:
  sv_x_code_fit = fh.read()
with open('./models/sv_x_predict.stan', 'r') as fh:
  sv_x_code_predict = fh.read()
num_samples = 1_000
```

In [ ]:
```
# 5) SV X + Peak hour + Price zone 1
model_sv_x_peak_eur = StanModel(kind='sv_x',
                                name='SV X + Peak hour + Price zone 1',
                                stan_code_fit=sv_x_code_fit,
                                stan_code_predict=sv_x_code_predict,
                                num_samples=num_samples)
```

In [ ]:
```
# 6) SV X + Peak hour + Price zone 2
model_sv_x_peak_sib = StanModel(kind='sv_x',
                                name='SV X + Peak hour + Price zone 2',
                                stan_code_fit=sv_x_code_fit,
                                stan_code_predict=sv_x_code_predict,
                                num_samples=num_samples)
```

In [ ]:
```
# 7) SV X + Off-peak hour + Price zone 1
model_sv_x_offpeak_eur = StanModel(kind='sv_x',
                                name='SV X + Off-peak hour + Price zone 1',
                                stan_code_fit=sv_x_code_fit,
                                stan_code_predict=sv_x_code_predict,
                                num_samples=num_samples)
```

In [ ]:
```
# 8) SV X + Off-peak hour + Price zone 2
model_sv_x_offpeak_sib = StanModel(kind='sv_x',
                                name='SV X + Off-peak hour + Price zone 2',
                                stan_code_fit=sv_x_code_fit,
                                stan_code_predict=sv_x_code_predict,
                                num_samples=num_samples)
```

In [ ]:
```
n_windows = 36 # Number of sliding windows
ts_cv_sv_x = TimeSeriesSplit(n_splits=n_windows, max_train_size=30*12, test_size=30*3, gap=0)
```

In [ ]:
```
print('Split # | Train fold                    | Test fold')
print('--------|-------------------------------|-------------------------------')
# Due to air temperature having missing data, the 4 subsets are not strictly equal, but anyway have around 3600 days each
# So get split indices from any of the four
for i, split in enumerate(ts_cv_sv_x.split(price_temp_data_peak_eur)):
    train_start = price_temp_data_peak_eur.iloc[split[0]].index.min().date()
    train_end = price_temp_data_peak_eur.iloc[split[0]].index.max().date()
    test_start = price_temp_data_peak_eur.iloc[split[1]].index.min().date()
    test_end = price_temp_data_peak_eur.iloc[split[1]].index.max().date()
    print(f'{i + 1:>7} | {train_start} -- {train_end} ({(train_end - train_start).days + 1}) | {test_start} -- {test_end} ({(t
```

Split #	Train fold	Test fold
1	2014-06-23 -- 2015-06-09 (352)	2015-06-10 -- 2015-09-10 (93)
2	2014-09-13 -- 2015-09-10 (363)	2015-09-11 -- 2015-12-10 (91)
3	2014-12-12 -- 2015-12-10 (364)	2015-12-11 -- 2016-03-09 (90)
4	2015-03-12 -- 2016-03-09 (364)	2016-03-10 -- 2016-06-08 (91)
5	2015-06-10 -- 2016-06-08 (365)	2016-06-09 -- 2016-09-07 (91)
6	2015-09-11 -- 2016-09-07 (363)	2016-09-08 -- 2016-12-06 (90)
7	2015-12-11 -- 2016-12-06 (362)	2016-12-07 -- 2017-03-06 (90)
8	2016-03-10 -- 2017-03-06 (362)	2017-03-07 -- 2017-06-04 (90)
9	2016-06-09 -- 2017-06-04 (361)	2017-06-06 -- 2017-09-03 (90)
10	2016-09-08 -- 2017-09-03 (361)	2017-09-04 -- 2017-12-02 (90)
11	2016-12-07 -- 2017-12-02 (361)	2017-12-03 -- 2018-03-02 (90)
12	2017-03-07 -- 2018-03-02 (361)	2018-03-03 -- 2018-05-31 (90)
13	2017-06-06 -- 2018-05-31 (360)	2018-06-01 -- 2018-08-29 (90)
14	2017-09-04 -- 2018-08-29 (360)	2018-08-30 -- 2018-11-27 (90)
15	2017-12-03 -- 2018-11-27 (360)	2018-11-28 -- 2019-02-25 (90)
16	2018-03-03 -- 2019-02-25 (360)	2019-02-26 -- 2019-05-26 (90)
17	2018-06-01 -- 2019-05-26 (360)	2019-05-27 -- 2019-08-24 (90)
18	2018-08-30 -- 2019-08-24 (360)	2019-08-25 -- 2019-11-22 (90)
19	2018-11-28 -- 2019-11-22 (360)	2019-11-23 -- 2020-02-20 (90)
20	2019-02-26 -- 2020-02-20 (360)	2020-02-21 -- 2020-05-20 (90)
21	2019-05-27 -- 2020-05-20 (360)	2020-05-21 -- 2020-08-18 (90)
22	2019-08-25 -- 2020-08-18 (360)	2020-08-19 -- 2020-11-16 (90)
23	2019-11-23 -- 2020-11-16 (360)	2020-11-17 -- 2021-02-14 (90)
24	2020-02-21 -- 2021-02-14 (360)	2021-02-15 -- 2021-05-15 (90)
25	2020-05-21 -- 2021-05-15 (360)	2021-05-16 -- 2021-08-13 (90)
26	2020-08-19 -- 2021-08-13 (360)	2021-08-14 -- 2021-11-11 (90)
27	2020-11-17 -- 2021-11-11 (360)	2021-11-12 -- 2022-02-09 (90)
28	2021-02-15 -- 2022-02-09 (360)	2022-02-10 -- 2022-05-10 (90)
29	2021-05-16 -- 2022-05-10 (360)	2022-05-11 -- 2022-08-08 (90)
30	2021-08-14 -- 2022-08-08 (360)	2022-08-09 -- 2022-11-06 (90)
31	2021-11-12 -- 2022-11-06 (360)	2022-11-07 -- 2023-02-04 (90)
32	2022-02-10 -- 2023-02-04 (360)	2023-02-05 -- 2023-05-05 (90)
33	2022-05-11 -- 2023-05-05 (360)	2023-05-06 -- 2023-08-04 (91)
34	2022-08-09 -- 2023-08-04 (361)	2023-08-05 -- 2023-11-02 (90)
35	2022-11-07 -- 2023-11-02 (361)	2023-11-03 -- 2024-01-31 (90)
36	2023-02-05 -- 2024-01-31 (361)	2024-02-01 -- 2024-04-30 (90)

In [ ]:
```
%%capture
models = [model_sv_x_peak_eur, model_sv_x_peak_sib, model_sv_x_offpeak_eur, model_sv_x_offpeak_sib]
```

```
subsets = [price_temp_data_peak_eur, price_temp_data_peak_sib, price_temp_data_offpeak_eur, price_temp_data_offpeak_sib]
cv_results_sv_x_df = pd.DataFrame(columns=['model_kind', 'model_name', 'train_indices', 'test_indices', 'fit_time', 'score_tim
print('Running cross-validation...')
for model, subset in tqdm(zip(models, subsets)):
    print(f'Model: {model.name}')
    cv_result = cross_validate(model,
                               X=subset[['Temperature', 'Weekday']],
                               y=subset['CONSUMER_PRICE'],
                               cv=ts_cv_sv_x,
                               scoring=['neg_mean_absolute_error', 'neg_root_mean_squared_error'],
                               return_indices=True,
                               n_jobs=4)
    cv_result_df = pd.concat([pd.Series(cv_result['indices']['train'], name='train_indices'),
    pd.Series(cv_result['indices']['test'], name='test_indices'),
    pd.Series(cv_result['fit_time'], name='fit_time'),
    pd.Series(cv_result['score_time'], name='score_time'),
    pd.Series(-cv_result['test_neg_mean_absolute_error'], name='test_mae'),
    pd.Series(-cv_result['test_neg_root_mean_squared_error'], name='test_rmse')], axis=1)
    cv_results_sv_x_df = pd.concat([cv_results_sv_x_df, pd.concat([pd.Series([model.kind] * n_windows, name='model_kind'), pd.
cv_results_sv_x_df = cv_results_sv_x_df.reset_index(drop=True)
print('Done!')
```

In [ ]:
```
cv_results_sv_x_df
```

Out[ ]:

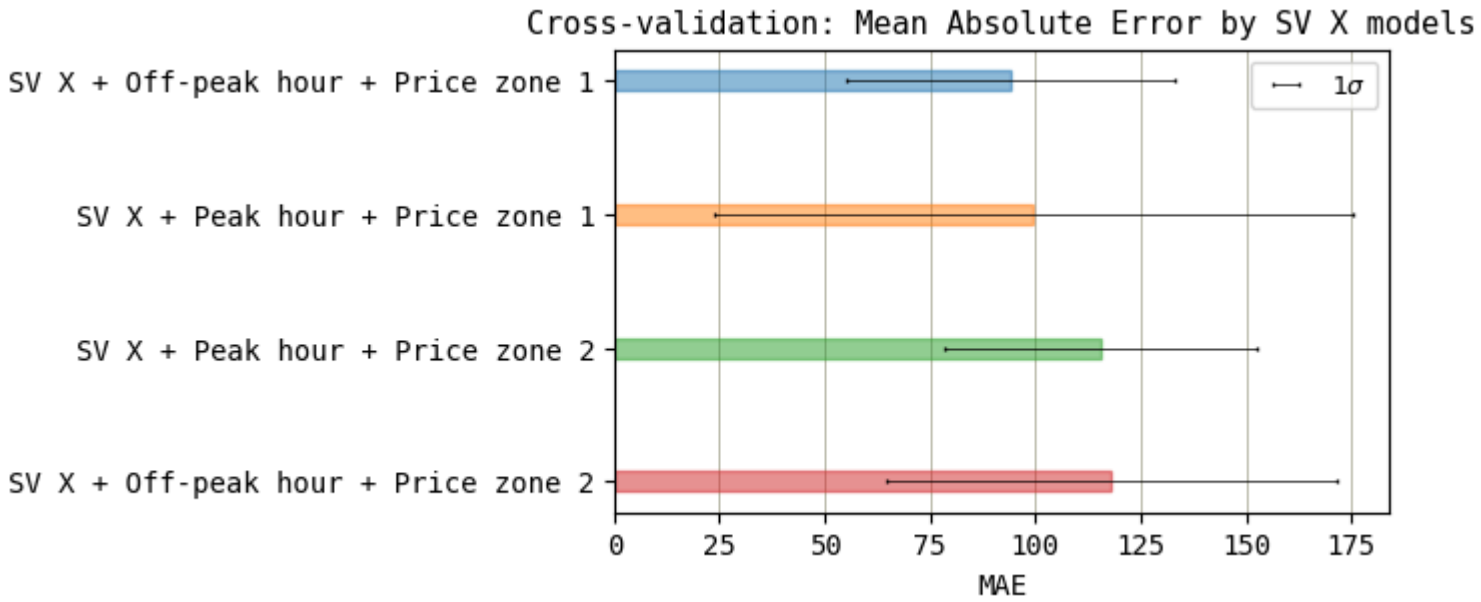
	model_kind	model_name	train_indices	test_indices	fit_time	score_time	test_mae	test_rmse
0	sv_x	SV X + Peak hour + Price zone 1	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[352, 353, 354, 355, 356, 357, 358, 359, 360, ...	77.184585	0.213684	59.273340	78.164893
1	sv_x	SV X + Peak hour + Price zone 1	[82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 9...	[442, 443, 444, 445, 446, 447, 448, 449, 450, ...	81.478046	0.221729	78.322507	90.978897
2	sv_x	SV X + Peak hour + Price zone 1	[172, 173, 174, 175, 176, 177, 178, 179, 180, ...	[532, 533, 534, 535, 536, 537, 538, 539, 540, ...	68.001533	0.235052	103.507513	133.680507
3	sv_x	SV X + Peak hour + Price zone 1	[262, 263, 264, 265, 266, 267, 268, 269, 270, ...	[622, 623, 624, 625, 626, 627, 628, 629, 630, ...	69.512063	0.232054	77.432192	107.638945
4	sv_x	SV X + Peak hour + Price zone 1	[352, 353, 354, 355, 356, 357, 358, 359, 360, ...	[712, 713, 714, 715, 716, 717, 718, 719, 720, ...	57.932176	0.207648	107.541759	126.230519
...	...	...	...	...	...	...	...	...
139	sv_x	SV X + Off-peak hour + Price zone 2	[2760, 2761, 2762, 2763, 2764, 2765, 2766, 276...	[3120, 3121, 3122, 3123, 3124, 3125, 3126, 312...	46.890198	0.231709	246.928214	280.323125
140	sv_x	SV X + Off-peak hour + Price zone 2	[2850, 2851, 2852, 2853, 2854, 2855, 2856, 285...	[3210, 3211, 3212, 3213, 3214, 3215, 3216, 321...	72.183671	0.218006	171.616812	204.608783
141	sv_x	SV X + Off-peak hour + Price zone 2	[2940, 2941, 2942, 2943, 2944, 2945, 2946, 294...	[3300, 3301, 3302, 3303, 3304, 3305, 3306, 330...	54.560874	0.208171	183.400849	216.105417
142	sv_x	SV X + Off-peak hour + Price zone 2	[3030, 3031, 3032, 3033, 3034, 3035, 3036, 303...	[3390, 3391, 3392, 3393, 3394, 3395, 3396, 339...	61.695394	0.203312	96.976854	119.808967
143	sv_x	SV X + Off-peak hour + Price zone 2	[3120, 3121, 3122, 3123, 3124, 3125, 3126, 312...	[3480, 3481, 3482, 3483, 3484, 3485, 3486, 348...	46.028104	0.190644	101.983238	137.364621

144 rows × 8 columns

In [ ]:
```
# Mean MAE by model
mae_sv_x_cross = cv_results_sv_x_df.groupby('model_name')['test_mae']
mae_sv_x_cross.mean().sort_values()
```

Out[ ]:
```
model_name
SV X + Off-peak hour + Price zone 1     93.923140
SV X + Peak hour + Price zone 1         99.539011
SV X + Peak hour + Price zone 2        115.566551
SV X + Off-peak hour + Price zone 2    118.135477
Name: test_mae, dtype: float64
```

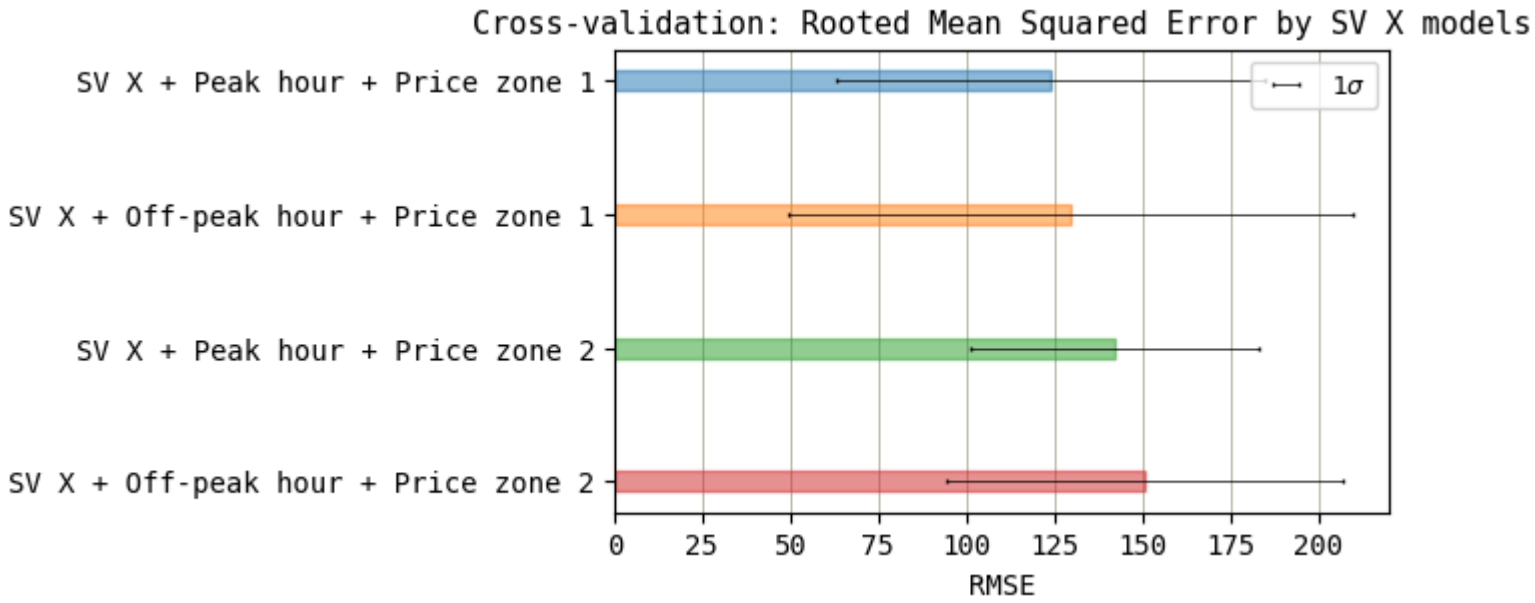
In [ ]:
```
plt.figure(figsize=(5, 3))
bars = plt.barh(range(mae_sv_x_cross.mean().shape[0]), mae_sv_x_cross.mean().sort_values(), xerr=mae_sv_x_cross.std(ddof=1), h
plt.gca().invert_yaxis()
plt.xlabel('MAE', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(range(mae_sv_x_cross.mean().shape[0]), mae_sv_x_cross.mean().sort_values().index, size=10, family='monospace')
plt.grid(axis='x', lw=0.5, color='xkcd:cement')
for i in range(len(bars)):
    bars[i].set_edgecolor(f'C{i}')
    bars[i].set_color(f'C{i}')
plt.gca().set_axisbelow(True)
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.title('Cross-validation: Mean Absolute Error by SV X models', size=11, family='monospace');
```



In [ ]:
```
# Mean RMSE by model
rmse_sv_x_cross = cv_results_sv_x_df.groupby('model_name')['test_rmse']
rmse_sv_x_cross.mean().sort_values()
```

Out[ ]:
```
model_name
SV X + Peak hour + Price zone 1        123.668770
SV X + Off-peak hour + Price zone 1    129.595937
SV X + Peak hour + Price zone 2        142.114654
SV X + Off-peak hour + Price zone 2    150.690504
Name: test_rmse, dtype: float64
```

In [ ]:
```
plt.figure(figsize=(5, 3))
bars = plt.barh(range(rmse_sv_x_cross.mean().shape[0]), rmse_sv_x_cross.mean().sort_values(), xerr=rmse_sv_x_cross.std(ddof=1)
plt.gca().invert_yaxis()
plt.xlabel('RMSE', size=10, family='monospace')
plt.xticks(size=10, family='monospace')
plt.yticks(range(rmse_sv_x_cross.mean().shape[0]), rmse_sv_x_cross.mean().sort_values().index, size=10, family='monospace')
plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
plt.grid(axis='x', lw=0.5, color='xkcd:cement')
for i in range(len(bars)):
    bars[i].set_edgecolor(f'C{i}')
    bars[i].set_color(f'C{i}')
plt.gca().set_axisbelow(True)
plt.title('Cross-validation: Rooted Mean Squared Error by SV X models', size=11, family='monospace');
```



As with SV Baseline model, both metrics for SV X model show that on average:

- models for Price zone 1 (European) have higher quality than for Price zone 2 (Siberian);
- the best model is for Price zone 1 for Peak hour;
- the worst model is for Price zone 2 for Off-peak hour.

4.3. Model Comparison

Let's compare the distributions of MAE and RMSE metrics obtained during cross-validation.

In [ ]:
```
cv_results_sv_all_df = pd.concat([cv_results_sv_base_df, cv_results_sv_x_df], axis=0)
cv_results_sv_all_df
```

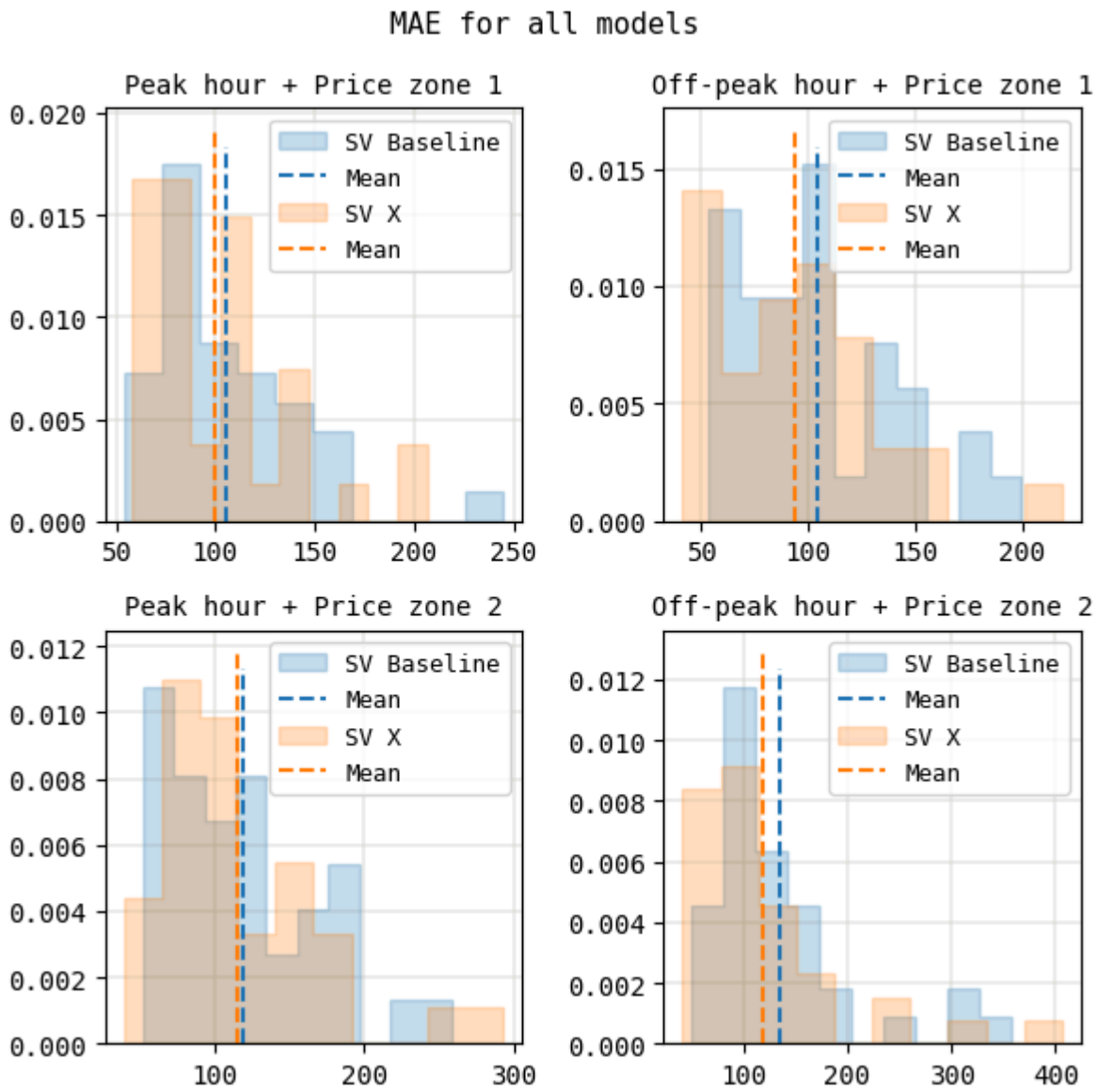
Out[ ]:

	model_kind	model_name	train_indices	test_indices	fit_time	score_time	test_mae	test_rmse
0	sv_base	SV Baseline + Peak hour + Price zone 1	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[360, 361, 362, 363, 364, 365, 366, 367, 368, ...	0.655952	0.241488	66.932230	82.187065
1	sv_base	SV Baseline + Peak hour + Price zone 1	[90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, ...	[450, 451, 452, 453, 454, 455, 456, 457, 458, ...	6.409832	0.208298	91.498069	102.352921
2	sv_base	SV Baseline + Peak hour + Price zone 1	[180, 181, 182, 183, 184, 185, 186, 187, 188, ...	[540, 541, 542, 543, 544, 545, 546, 547, 548, ...	10.234515	0.201207	98.059416	131.124286
3	sv_base	SV Baseline + Peak hour + Price zone 1	[270, 271, 272, 273, 274, 275, 276, 277, 278, ...	[630, 631, 632, 633, 634, 635, 636, 637, 638, ...	6.432677	0.206495	90.657318	116.004756
4	sv_base	SV Baseline + Peak hour + Price zone 1	[360, 361, 362, 363, 364, 365, 366, 367, 368, ...	[720, 721, 722, 723, 724, 725, 726, 727, 728, ...	0.751844	0.065485	144.874059	158.346917
...	...	...	...	...	...	...	...	...
139	sv_x	SV X + Off-peak hour + Price zone 2	[2760, 2761, 2762, 2763, 2764, 2765, 2766, 276...	[3120, 3121, 3122, 3123, 3124, 3125, 3126, 312...	46.890198	0.231709	246.928214	280.323125
140	sv_x	SV X + Off-peak hour + Price zone 2	[2850, 2851, 2852, 2853, 2854, 2855, 2856, 285...	[3210, 3211, 3212, 3213, 3214, 3215, 3216, 321...	72.183671	0.218006	171.616812	204.608783
141	sv_x	SV X + Off-peak hour + Price zone 2	[2940, 2941, 2942, 2943, 2944, 2945, 2946, 294...	[3300, 3301, 3302, 3303, 3304, 3305, 3306, 330...	54.560874	0.208171	183.400849	216.105417
142	sv_x	SV X + Off-peak hour + Price zone 2	[3030, 3031, 3032, 3033, 3034, 3035, 3036, 303...	[3390, 3391, 3392, 3393, 3394, 3395, 3396, 339...	61.695394	0.203312	96.976854	119.808967
143	sv_x	SV X + Off-peak hour + Price zone 2	[3120, 3121, 3122, 3123, 3124, 3125, 3126, 312...	[3480, 3481, 3482, 3483, 3484, 3485, 3486, 348...	46.028104	0.190644	101.983238	137.364621

288 rows × 8 columns

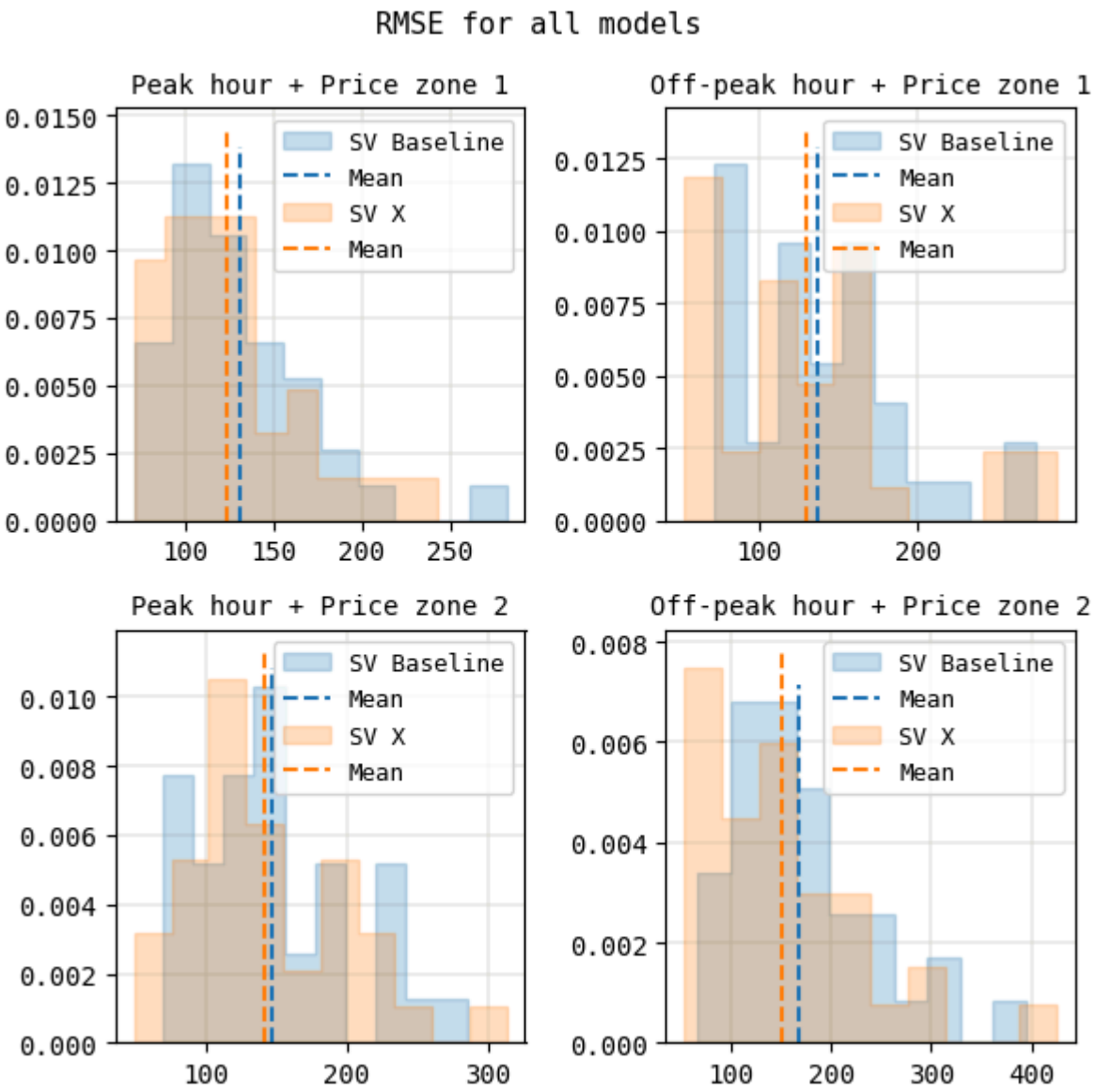
In [ ]:
```
cv_results_peak_eur = cv_results_sv_all_df[cv_results_sv_all_df['model_name'].str.contains('\+ Peak hour \+ Price zone 1')]
cv_results_offpeak_eur = cv_results_sv_all_df[cv_results_sv_all_df['model_name'].str.contains('\+ Off-peak hour \+ Price zone
cv_results_peak_sib = cv_results_sv_all_df[cv_results_sv_all_df['model_name'].str.contains('\+ Peak hour \+ Price zone 2')]
cv_results_offpeak_sib = cv_results_sv_all_df[cv_results_sv_all_df['model_name'].str.contains('\+ Off-peak hour \+ Price zone
```

In [ ]:
```
plt.figure(figsize=(6, 6))
for i, subset in zip(range(1, 5), [cv_results_peak_eur, cv_results_offpeak_eur, cv_results_peak_sib, cv_results_offpeak_sib]):
    plt.subplot(2, 2, i)
    plt.hist(subset[subset['model_kind'] == 'sv_base']['test_mae'], bins=10, density=True, histtype='stepfilled', color='C0',
    plt.vlines(subset[subset['model_kind'] == 'sv_base']['test_mae'].mean(), ymin=0, ymax=plt.ylim()[1], color='C0', ls='--',
    plt.hist(subset[subset['model_kind'] == 'sv_x']['test_mae'], bins=10, density=True, histtype='stepfilled', color='C1', edg
    plt.vlines(subset[subset['model_kind'] == 'sv_x']['test_mae'].mean(), ymin=0, ymax=plt.ylim()[1], color='C1', ls='--', lab
    plt.xticks(size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
    plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
    plt.title(f"{' + '.join(subset['model_name'].unique()[0].split(' + ')[1:])}", size=10, family='monospace')
    plt.grid(lw=0.25, color='xkcd:cement')
    plt.gca().set_axisbelow(True)
plt.suptitle(f'MAE for all models', size=11, family='monospace')
plt.tight_layout();
```



In [ ]:
```
plt.figure(figsize=(6, 6))
for i, subset in zip(range(1, 5), [cv_results_peak_eur, cv_results_offpeak_eur, cv_results_peak_sib, cv_results_offpeak_sib]):
    plt.subplot(2, 2, i)
    plt.hist(subset[subset['model_kind'] == 'sv_base']['test_rmse'], bins=10, density=True, histtype='stepfilled', color='C0',
    plt.vlines(subset[subset['model_kind'] == 'sv_base']['test_rmse'].mean(), ymin=0, ymax=plt.ylim()[1], color='C0', ls='--',
```

```
    plt.hist(subset[subset['model_kind'] == 'sv_x']['test_rmse'], bins=10, density=True, histtype='stepfilled', color='C1', ed
    plt.vlines(subset[subset['model_kind'] == 'sv_x']['test_rmse'].mean(), ymin=0, ymax=plt.ylim()[1], color='C1', ls='--', la
    plt.xticks(size=10, family='monospace')
    plt.yticks(size=10, family='monospace')
    plt.legend(loc='upper right', prop={'size': 9, 'family': 'monospace'})
    plt.title(f"{' + '.join(subset['model_name'].unique()[0].split(' + ')[1:])}", size=10, family='monospace')
    plt.grid(lw=0.25, color='xkcd:cement')
    plt.gca().set_axisbelow(True)
plt.suptitle(f'RMSE for all models', size=11, family='monospace')
plt.tight_layout();
```



We should also check the statistical significance of the difference in the metrics. We will use a non-parametric one-tailed Mann-Whitney U test for two independent samples to see if the distributions of metrics for SV Baseline and SV X models are identical or have a shift between each other.

$\mathcal{H}_0 : F_{X_1}(x) = F_{X_2}(x)$ (distributed identically)

$\mathcal{H}_1 : F_{X_1}(x) = F_{X_2}(x + \Delta x)$ (distributed with some shift Δx), where

$F_X(x)$ is the distribution of a given metric for a given model.

Statistic: Mann-Whitney's U

Null distribution: no assumption; generated empirically by permutation test; for large samples it can be approximated by $\mathcal{N}(\mu = \frac{n_1 n_2}{2}, \sigma^2 = \frac{n_1 n_2 (n_1 + n_2 + 1)}{12})$.

In [ ]:
```
# Two independent samples one-tailed Mann-Whitney U test for MAE
alpha = 0.05
mwu_stats, mwu_pval = mannwhitneyu(cv_results_sv_base_df['test_mae'], cv_results_sv_x_df['test_mae'], alternative='greater')
print(f'{mwu_stats = :.0f}, {mwu_pval = :.4f}', end=', ')
print("SV Baseline's MAE is greater than SV X's MAE (shifted to the right)") if mwu_pval < alpha else print('SV Baseline and S
```

```
mwu_stats = 11785, mwu_pval = 0.0225, SV Baseline's MAE is greater than SV X's MAE (shifted to the right)
```

In [ ]:
```
# Two independent samples one-tailed Mann-Whitney U test for RMSE
alpha = 0.05
mwu_stats, mwu_pval = mannwhitneyu(cv_results_sv_base_df['test_rmse'], cv_results_sv_x_df['test_rmse'], alternative='greater')
print(f'{mwu_stats = :.0f}, {mwu_pval = :.4f}', end=', ')
print("SV Baseline's RMSE is greater than SV X's RMSE (shifted to the right)") if mwu_pval < alpha else print('SV Baseline and
```

```
mwu_stats = 11566, mwu_pval = 0.0451, SV Baseline's RMSE is greater than SV X's RMSE (shifted to the right)
```

The statistical significance of difference in both metrics, obtained during cross-validation, between SV Baseline and SV X models *can* be confirmed by hypothesis testing with one-tailed Mann-Whitney U test fro two independent samples.

Metrics for both models have heavy right tails, which signals that the models were not good for all splits. This might be due to market regime change or other factors.

In [ ]:
```
print(' Train / test fold with best RMSE:', price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rm
'--', price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rmse'] == cv_results_sv_base_df['test_rm

print(price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rmse'] == cv_results_sv_base_df['test_rm
'--', price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rmse'] == cv_results_sv_base_df['test_rm
```

```
print('Train / test fold with worst RMSE:', price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rm
'--', price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rmse'] == cv_results_sv_base_df['test_rm

print(price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rmse'] == cv_results_sv_base_df['test_rm
'--', price_temp_data_peak_eur.iloc[cv_results_sv_base_df[cv_results_sv_base_df['test_rmse'] == cv_results_sv_base_df['test_rm
```

```
Train / test fold with best RMSE: 2016-09-16 -- 2017-09-11 / 2017-09-12 -- 2017-12-10
Train / test fold with worst RMSE: 2018-09-07 -- 2019-09-01 / 2019-09-02 -- 2019-11-30
```

In [ ]:
```
cv_results_peak_eur.groupby('model_kind')[['test_mae', 'test_rmse']].mean().T.style.highlight_min(color='palegreen', axis=1).s
```

Peak hour + Price zone 1

model_kind	sv_base	sv_x
test_mae	105.385055	99.539011
test_rmse	130.526412	123.668770

In [ ]:
```
cv_results_peak_sib.groupby('model_kind')[['test_mae', 'test_rmse']].mean().T.style.highlight_min(color='palegreen', axis=1).s
```

Peak hour + Price zone 2

model_kind	sv_base	sv_x
test_mae	119.140971	115.566551
test_rmse	146.445320	142.114654

In [ ]:
```
cv_results_offpeak_eur.groupby('model_kind')[['test_mae', 'test_rmse']].mean().T.style.highlight_min(color='palegreen', axis=1
```

Off-peak hour + Price zone 1

model_kind	sv_base	sv_x
test_mae	104.422811	93.923140
test_rmse	137.501447	129.595937

In [ ]:
```
cv_results_offpeak_sib.groupby('model_kind')[['test_mae', 'test_rmse']].mean().T.style.highlight_min(color='palegreen', axis=1
```

Off-peak hour + Price zone 2

model_kind	sv_base	sv_x
test_mae	134.863451	118.135477
test_rmse	168.305299	150.690504

In [ ]:
```
# MAE % improvement
print(f"{cv_results_sv_base_df['test_mae'].mean() / cv_results_sv_x_df['test_mae'].mean() - 1:.2%}")
```

```
8.58%
```

In [ ]:
```
# RMSE % improvement
print(f"{cv_results_sv_base_df['test_rmse'].mean() / cv_results_sv_x_df['test_rmse'].mean() - 1:.2%}")
```

```
6.72%
```

On average, all four SV X models have better MAE (-8.58%) and RMSE (-6.72%) metrics as compared with SV Baseline models obtained during cross-validation. The statistical significance of this difference *can* be confirmed by hypothesis testing with one-tailed Mann-Whitney U test for two independent samples: $\mathcal{H}_0$ can be rejected at $\alpha = 5\%$.

Back

---

5. Forecasting

Finally, we would like to generate forecasts for the nearest day(s) ahead with our models. Just like with cross-validation, we will generate 8 forecasts:

1. SV Baseline + Peak hour + Price zone 1 (European);
2. SV Baseline + Peak hour + Price zone 2 (Siberian);
3. SV Baseline + Off-peak hour + Price zone 1;
4. SV Baseline + Off-peak hour + Price zone 2;
5. SV X + Peak hour + Price zone 1;

6. SV X + Peak hour + Price zone 2;
7. SV X + Off-peak hour + Price zone 1;
8. SV X + Off-peak hour + Price zone 2;

Forecast strategy:

1. Fit the model on a time frame right before the first forecasted day;
2. Generate one day-ahead prediction (forecast);
3. Move the time frame forward by one day;
4. Repeat from 1 for as many days as required.

First train time frame: 01.05.2023 -- 30.04.2024 (same as used for modeling in [2] and [3]).

Forecast time frame: 01.05.2024 -- 07.05.2024 (1 week).

In [ ]:
```
price_data = pd.read_csv('./data/price_data_20230501_20240507.zip', index_col='Datetime', parse_dates=['Datetime'])
```

In [ ]:
```
# Price zone 1: Moscow
temp_data_eur = pd.read_csv(f'./data/UUEE.01.05.2023.07.05.2024.1.0.0.ru.utf8.00000000.zip',
                            sep=';',
                            encoding='utf-8',
                            skiprows=7,
                            index_col=0,
                            dayfirst=True,
                            usecols=[0, 1],
                            names=['Datetime', 'Temperature'],
                            parse_dates=True).dropna().sort_index()
```

In [ ]:
```
# Price zone 2: Novosibirsk
temp_data_sib = pd.read_csv(f'./data/UNNT.01.05.2023.07.05.2024.1.0.0.ru.utf8.00000000.zip',
                            sep=';',
                            encoding='utf-8',
                            skiprows=7,
                            index_col=0,
                            dayfirst=True,
                            usecols=[0, 1],
                            names=['Datetime', 'Temperature'],
```

```
                          parse_dates=True).dropna().sort_index()
# Don't forget to convert local Novosibirsk time to Moscow time, since all price data is in Moscow time
temp_data_sib.index = temp_data_sib.index - pd.Timedelta(hours=4)
```

In [ ]:
```
price_data_eur = price_data[price_data['PRICE_ZONE_CODE'] == 1]
price_data_sib = price_data[price_data['PRICE_ZONE_CODE'] == 2]
```

In [ ]:
```
price_data_daily_agg_eur = price_data_eur.groupby('HOUR')['CONSUMER_PRICE'].mean()
price_data_daily_agg_sib = price_data_sib.groupby('HOUR')['CONSUMER_PRICE'].mean()
hour_max_eur = price_data_daily_agg_eur.idxmax()
hour_min_eur = price_data_daily_agg_eur.idxmin()
hour_max_sib = price_data_daily_agg_sib.idxmax()
hour_min_sib = price_data_daily_agg_sib.idxmin()
(hour_max_eur, hour_min_eur), (hour_max_sib, hour_min_sib)
```

Out[ ]:
```
((14, 3), (17, 1))
```

In [ ]:
```
price_data_peak_eur = price_data_eur[price_data_eur['HOUR'] == hour_max_eur]
price_data_offpeak_eur = price_data_eur[price_data_eur['HOUR'] == hour_min_eur]
price_data_peak_sib = price_data_sib[price_data_sib['HOUR'] == hour_max_sib]
price_data_offpeak_sib = price_data_sib[price_data_sib['HOUR'] == hour_min_sib]
```

In [ ]:
```
price_temp_data_peak_eur = pd.merge(price_data_peak_eur, temp_data_eur, on='Datetime')
price_temp_data_offpeak_eur = pd.merge(price_data_offpeak_eur, temp_data_eur, on='Datetime')
price_temp_data_peak_sib = pd.merge(price_data_peak_sib, temp_data_sib, on='Datetime')
price_temp_data_offpeak_sib = pd.merge(price_data_offpeak_sib, temp_data_sib, on='Datetime')
```

In [ ]:
```
# Peak hour + Price zone 1
%%capture
forecast_sv_base_peak_eur_many = model_sv_base_peak_eur.forecast_many(X=price_temp_data_peak_eur['CONSUMER_PRICE'], y=price_te
```

In [ ]:
```
%%capture
forecast_sv_x_peak_eur_many = model_sv_x_peak_eur.forecast_many(X=price_temp_data_peak_eur[['Temperature', 'Weekday']], y=pric
```

In [ ]:
```
pd.DataFrame({'sv_base': [
    mean_absolute_error(price_temp_data_peak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_peak_eur_many.mean(axis=1)
    root_mean_squared_error(price_temp_data_peak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_peak_eur_many.mean(axi
    'sv_x': [mean_absolute_error(price_temp_data_peak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_peak_eur_many.mean(a
```

```
    root_mean_squared_error(price_temp_data_peak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_peak_eur_many.mean(axis=1
    index=['mae', 'rmse']).style.highlight_min(color='palegreen', axis=1).set_caption(f'<h4>Peak hour + Price zone 1</h4>')
```

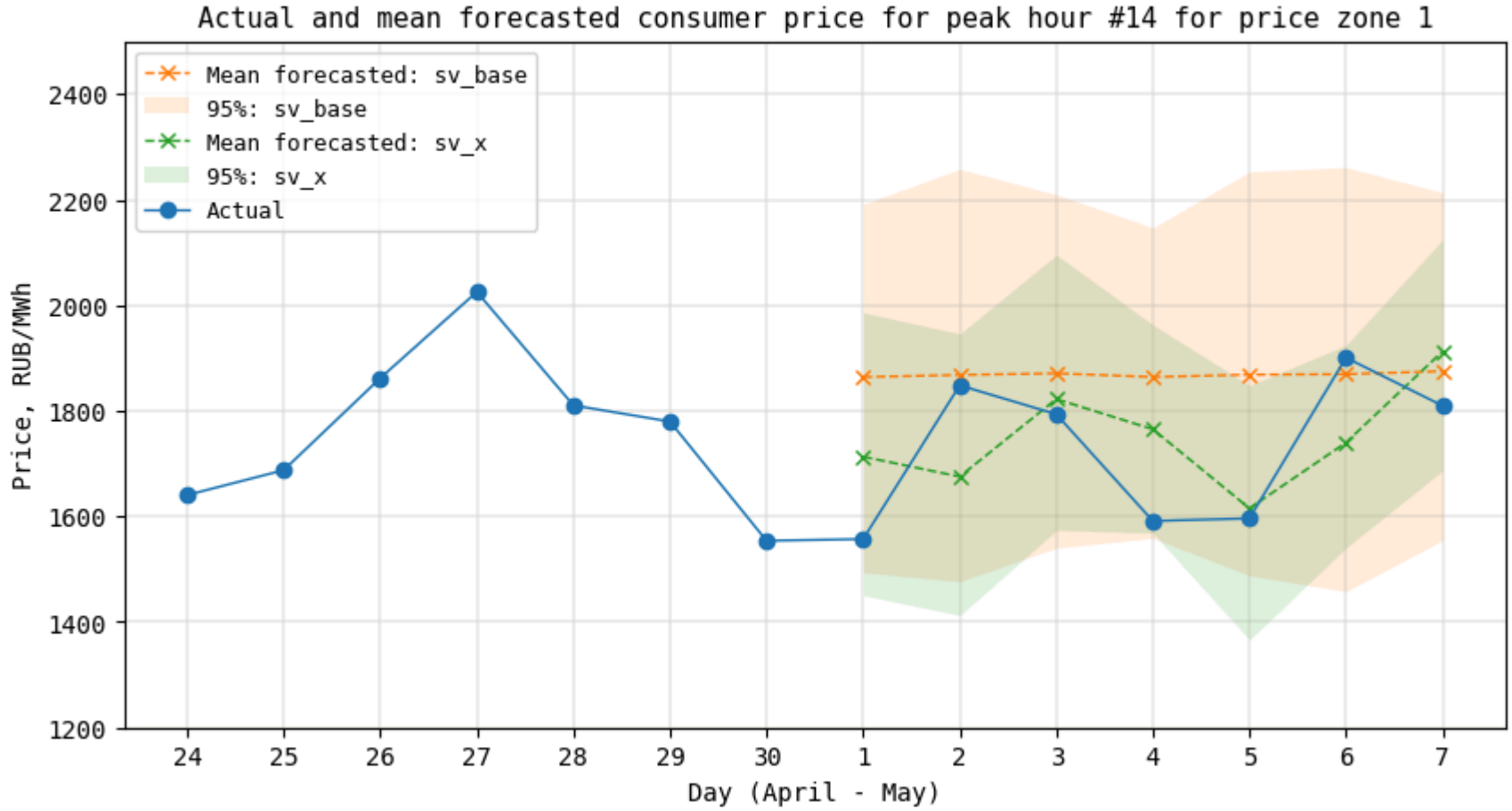
Out[ ]:

Peak hour + Price zone 1

	sv_base	sv_x
mae	149.414014	116.313316
rmse	190.641880	132.060972

In [ ]:

```
plt.figure(figsize=(10, 5))
for i, forecast, model in zip([1, 2], [forecast_sv_base_peak_eur_many, forecast_sv_x_peak_eur_many], [model_sv_base_peak_eur,
    plt.plot(price_temp_data_peak_eur.loc['2024-05-01':]['CONSUMER_PRICE'].index, forecast.mean(axis=1), color=f'C{i}', marker
    plt.fill_between(price_temp_data_peak_eur.loc['2024-05-01':]['CONSUMER_PRICE'].index, np.percentile(forecast, 2.5, axis=1)
plt.plot(price_temp_data_peak_eur.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_peak_eur.loc['2024-04-24':]['CON
plt.xlabel('Day (April - May)', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(price_temp_data_peak_eur.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_peak_eur.loc['2024-04-24':]['C
plt.yticks(size=10, family='monospace')
plt.ylim([1200, 2500])
plt.title(f'Actual and mean forecasted consumer price for peak hour #{hour_max_eur} for price zone 1', size=11, family='monosp
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'});
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
# Peak hour + Price zone 2
%%capture
forecast_sv_base_peak_sib_many = model_sv_base_peak_sib.forecast_many(X=price_temp_data_peak_sib['CONSUMER_PRICE'], y=price_te
```

In [ ]:
```
%%capture
forecast_sv_x_peak_sib_many = model_sv_x_peak_sib.forecast_many(X=price_temp_data_peak_sib[['Temperature', 'Weekday']], y=pric
```

In [ ]:
```
pd.DataFrame({'sv_base': [
    mean_absolute_error(price_temp_data_peak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_peak_sib_many.mean(axis=1)
    root_mean_squared_error(price_temp_data_peak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_peak_sib_many.mean(axi
    'sv_x': [mean_absolute_error(price_temp_data_peak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_peak_sib_many.mean(a
```

```
    root_mean_squared_error(price_temp_data_peak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_peak_sib_many.mean(axis=1
    index=['mae', 'rmse']).style.highlight_min(color='palegreen', axis=1).set_caption(f'<h4>Peak hour + Price zone 2</h4>')
```

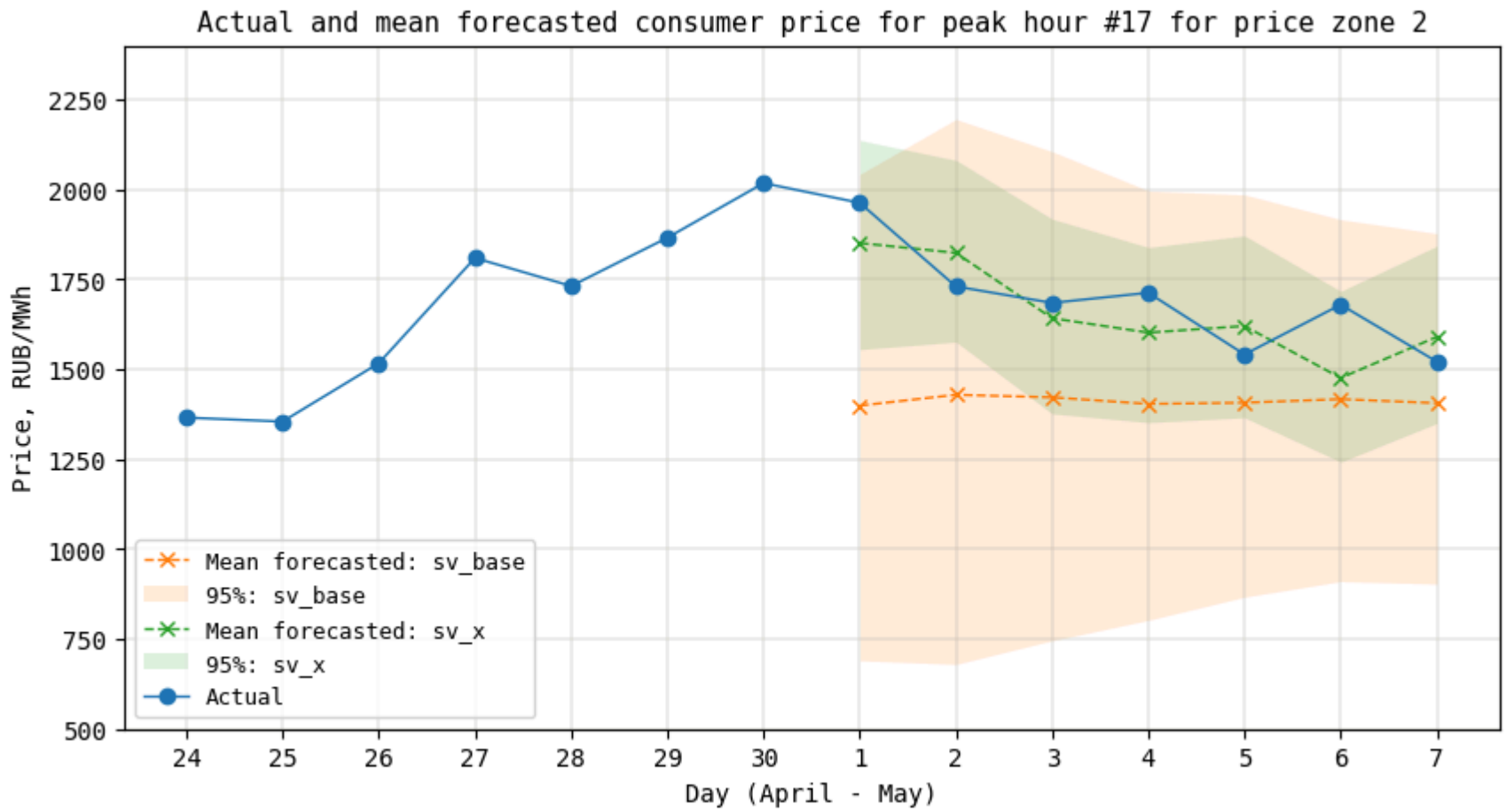
Out[ ]:

Peak hour + Price zone 2

	sv_base	sv_x
mae	278.445365	101.494472
rmse	310.207295	112.009927

In [ ]:

```
plt.figure(figsize=(10, 5))
for i, forecast, model in zip([1, 2], [forecast_sv_base_peak_sib_many, forecast_sv_x_peak_sib_many], [model_sv_base_peak_sib,
    plt.plot(price_temp_data_peak_sib.loc['2024-05-01':]['CONSUMER_PRICE'].index, forecast.mean(axis=1), color=f'C{i}', marker
    plt.fill_between(price_temp_data_peak_sib.loc['2024-05-01':]['CONSUMER_PRICE'].index, np.percentile(forecast, 2.5, axis=1)
plt.plot(price_temp_data_peak_sib.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_peak_sib.loc['2024-04-24':]['CON
plt.xlabel('Day (April - May)', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(price_temp_data_peak_sib.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_peak_sib.loc['2024-04-24':]['C
plt.yticks(size=10, family='monospace')
plt.ylim([500, 2400])
plt.title(f'Actual and mean forecasted consumer price for peak hour #{hour_max_sib} for price zone 2', size=11, family='monosp
plt.legend(loc='lower left', prop={'size': 9, 'family': 'monospace'});
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
# Off-peak hour + Price zone 1
%%capture
forecast_sv_base_offpeak_eur_many = model_sv_base_offpeak_eur.forecast_many(X=price_temp_data_offpeak_eur['CONSUMER_PRICE'], y
```

In [ ]:
```
%%capture
forecast_sv_x_offpeak_eur_many = model_sv_x_offpeak_eur.forecast_many(X=price_temp_data_offpeak_eur[['Temperature', 'Weekday']
```

In [ ]:
```
pd.DataFrame({'sv_base': [
    mean_absolute_error(price_temp_data_offpeak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_offpeak_eur_many.mean(a
    root_mean_squared_error(price_temp_data_offpeak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_offpeak_eur_many.me
    'sv_x': [mean_absolute_error(price_temp_data_offpeak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_offpeak_eur_many.
```

```
    root_mean_squared_error(price_temp_data_offpeak_eur['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_offpeak_eur_many.mean(
    index=['mae', 'rmse']).style.highlight_min(color='palegreen', axis=1).set_caption(f'<h4>Off-peak hour + Price zone 1</h4>'
```

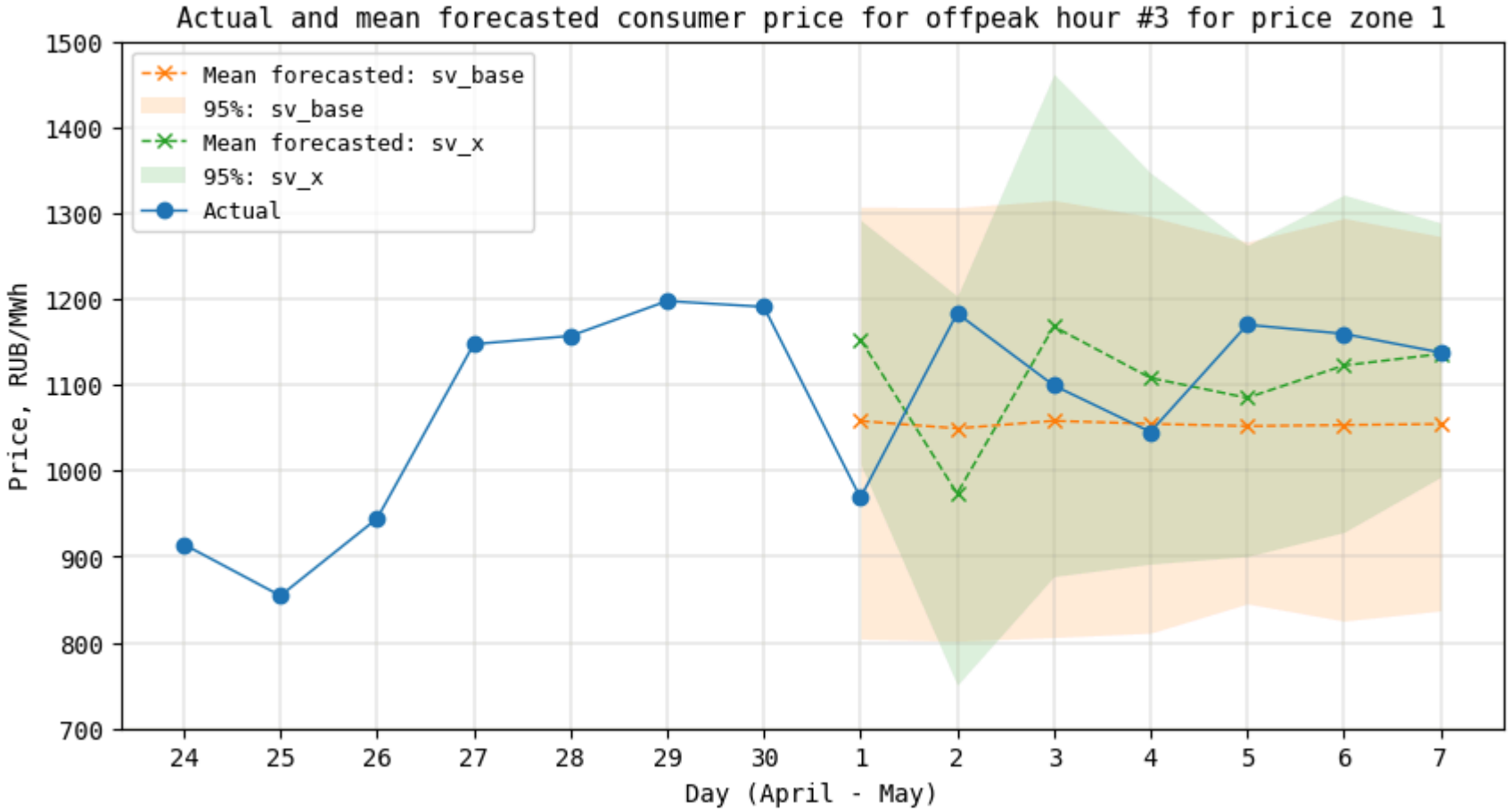
Out[ ]:

Off-peak hour + Price zone 1

	sv_base	sv_x
mae	83.070656	92.747658
rmse	92.487845	116.641078

In [ ]:

```
plt.figure(figsize=(10, 5))
for i, forecast, model in zip([1, 2], [forecast_sv_base_offpeak_eur_many, forecast_sv_x_offpeak_eur_many], [model_sv_base_offp
    plt.plot(price_temp_data_offpeak_eur.loc['2024-05-01':]['CONSUMER_PRICE'].index, forecast.mean(axis=1), color=f'C{i}', mar
    plt.fill_between(price_temp_data_offpeak_eur.loc['2024-05-01':]['CONSUMER_PRICE'].index, np.percentile(forecast, 2.5, axis
plt.plot(price_temp_data_offpeak_eur.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_offpeak_eur.loc['2024-04-24':
plt.xlabel('Day (April - May)', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(price_temp_data_offpeak_eur.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_offpeak_eur.loc['2024-04-24
plt.yticks(size=10, family='monospace')
plt.ylim([700, 1500])
plt.title(f'Actual and mean forecasted consumer price for offpeak hour #{hour_min_eur} for price zone 1', size=11, family='mon
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'});
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:
```
# Off-peak hour + Price zone 2
%%capture
forecast_sv_base_offpeak_sib_many = model_sv_base_offpeak_sib.forecast_many(X=price_temp_data_offpeak_sib['CONSUMER_PRICE'], y
```

In [ ]:
```
%%capture
forecast_sv_x_offpeak_sib_many = model_sv_x_offpeak_sib.forecast_many(X=price_temp_data_offpeak_sib[['Temperature', 'Weekday']
```

In [ ]:
```
pd.DataFrame({'sv_base': [
    mean_absolute_error(price_temp_data_offpeak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_offpeak_sib_many.mean(a
    root_mean_squared_error(price_temp_data_offpeak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_base_offpeak_sib_many.me
    'sv_x': [mean_absolute_error(price_temp_data_offpeak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_offpeak_sib_many.
```

```
    root_mean_squared_error(price_temp_data_offpeak_sib['CONSUMER_PRICE']['2024-05-01':], forecast_sv_x_offpeak_sib_many.mean(
    index=['mae', 'rmse']).style.highlight_min(color='palegreen', axis=1).set_caption(f'<h4>Off-peak hour + Price zone 2</h4>'
```

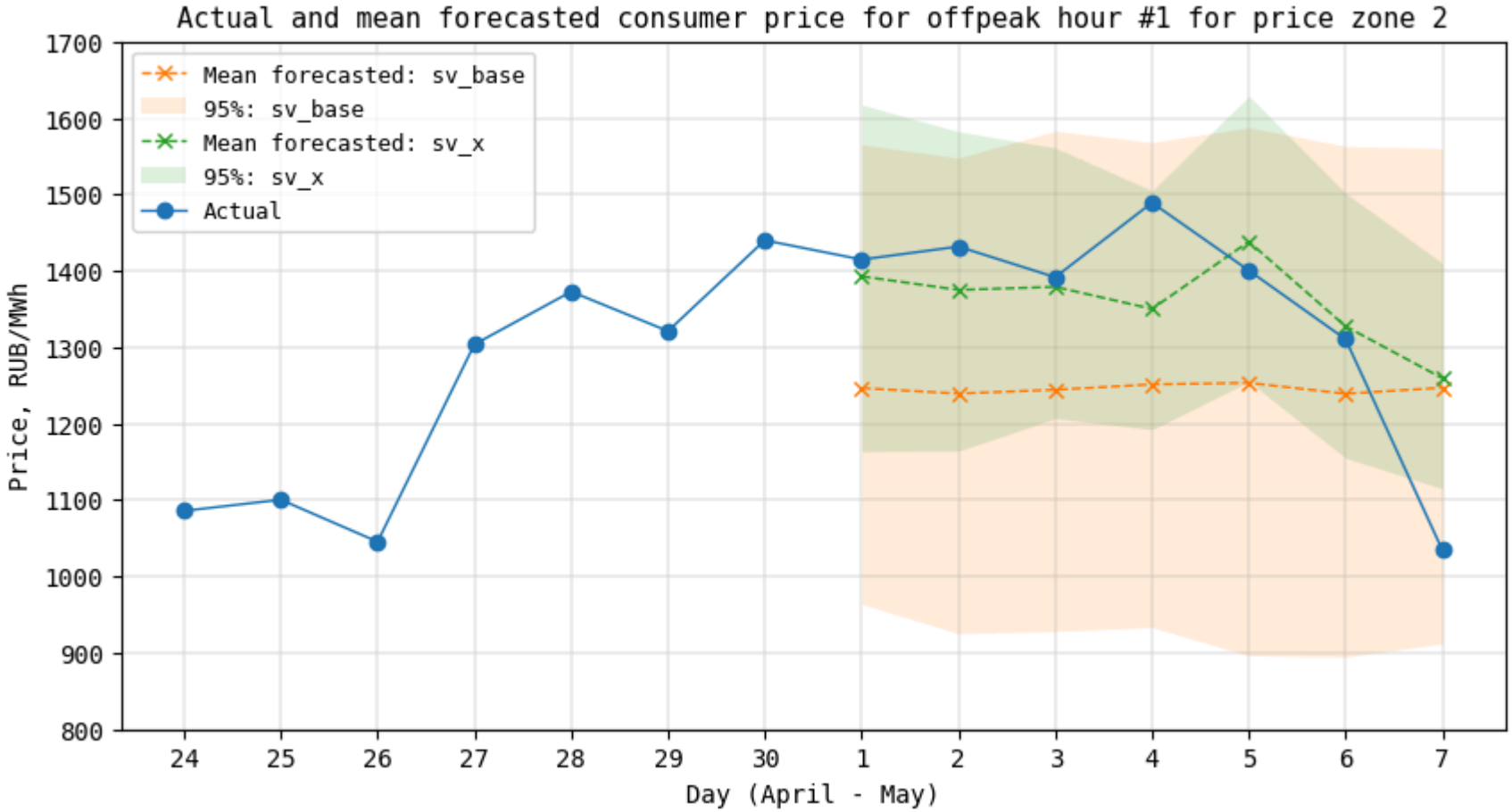
Out[ ]:

Off-peak hour + Price zone 2

	sv_base	sv_x
mae	168.085830	72.837682
rmse	175.435059	103.845358

In [ ]:

```
plt.figure(figsize=(10, 5))
for i, forecast, model in zip([1, 2], [forecast_sv_base_offpeak_sib_many, forecast_sv_x_offpeak_sib_many], [model_sv_base_offp
    plt.plot(price_temp_data_offpeak_sib.loc['2024-05-01':]['CONSUMER_PRICE'].index, forecast.mean(axis=1), color=f'C{i}', mar
    plt.fill_between(price_temp_data_offpeak_sib.loc['2024-05-01':]['CONSUMER_PRICE'].index, np.percentile(forecast, 2.5, axis
plt.plot(price_temp_data_offpeak_sib.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_offpeak_sib.loc['2024-04-24':
plt.xlabel('Day (April - May)', size=10, family='monospace')
plt.ylabel('Price, RUB/MWh', size=10, family='monospace')
plt.xticks(price_temp_data_offpeak_sib.loc['2024-04-24':]['CONSUMER_PRICE'].index, price_temp_data_offpeak_sib.loc['2024-04-24
plt.yticks(size=10, family='monospace')
plt.ylim([800, 1700])
plt.title(f'Actual and mean forecasted consumer price for offpeak hour #{hour_min_sib} for price zone 2', size=11, family='mon
plt.legend(loc='upper left', prop={'size': 9, 'family': 'monospace'});
plt.grid(lw=0.25, color='xkcd:cement');
```



In [ ]:

Back

---

6. Results

In [ ]:
```
# To do
```

Back