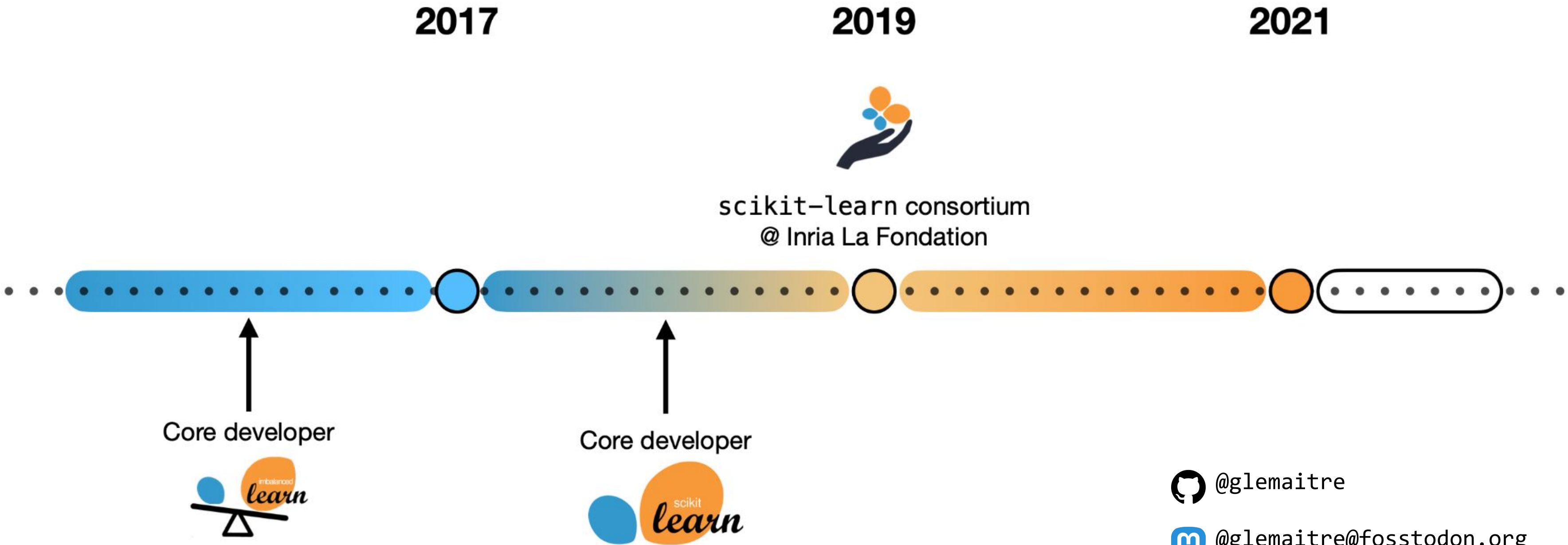
# A case study of time-series forecasting for bike rentals

## Open Source Experience 2022

Guillaume Lemaitre - November 9, 2022

# About me

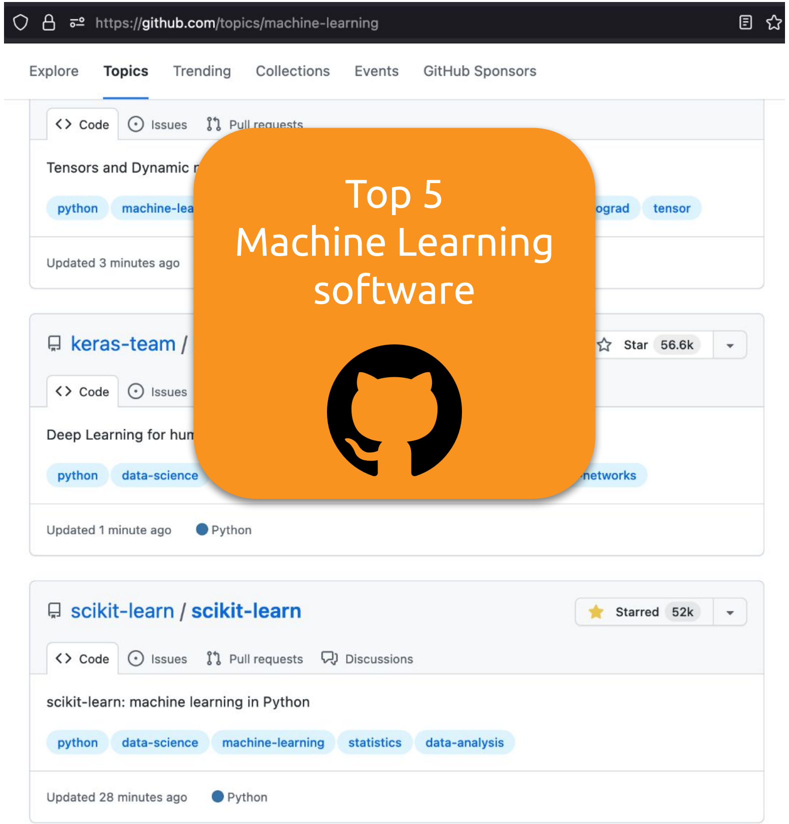
## Guillaume Lemaitre - Software Engineer




@glemaitre

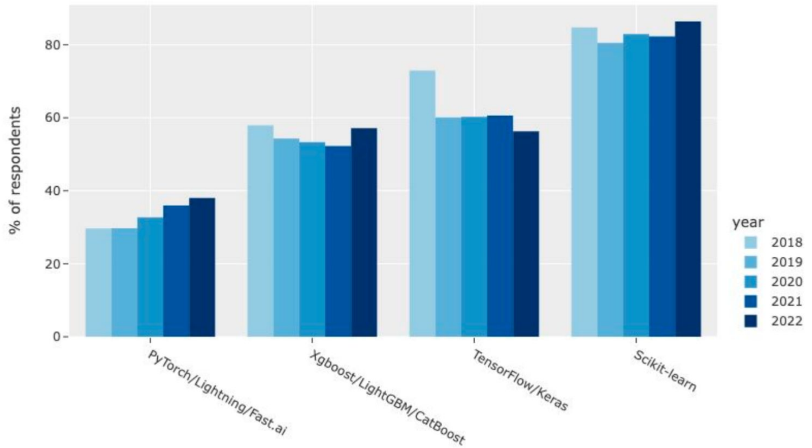
@glemaitre@fosstodon.org

# About scikit-learn




Kaggle DS & ML Survey 2022

## Scikit-learn is the most popular ML framework while PyTorch has been growing steadily year-over-year



# Agenda

Introduction to time-series data

Mind the evaluation

Time-series forecasting

Modeling predictive uncertainty

Beyond scikit-learn

# Tackling time-series as a regression problem

## The bike sharing dataset

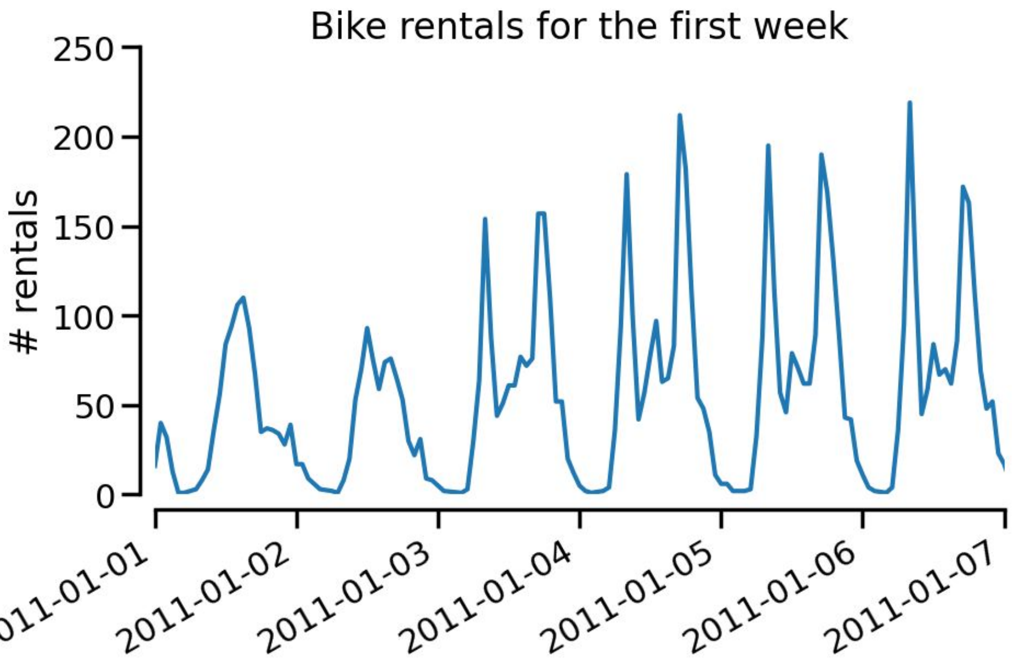
```
from sklearn.datasets import fetch_openml

bike_sharing = fetch_openml(
    "Bike_Sharing_Demand", version=2, as_frame=True, parser="pandas",
)
bike_sharing.frame
```

| dteday | season | year | month | hour | holiday | weekday | workingday | weather | temp | feel_temp | humidity | windspeed | count |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2011-01-01 00:00:00 | spring | 0 | 1 | 0 | False | 6 | False | clear | 9.84 | 14.395 | 0.81 | 0.0000 | 16 |
| 2011-01-01 01:00:00 | spring | 0 | 1 | 1 | False | 6 | False | clear | 9.02 | 13.635 | 0.80 | 0.0000 | 40 |
| 2011-01-01 02:00:00 | spring | 0 | 1 | 2 | False | 6 | False | clear | 9.02 | 13.635 | 0.80 | 0.0000 | 32 |
| 2011-01-01 03:00:00 | spring | 0 | 1 | 3 | False | 6 | False | clear | 9.84 | 14.395 | 0.75 | 0.0000 | 13 |
| 2011-01-01 04:00:00 | spring | 0 | 1 | 4 | False | 6 | False | clear | 9.84 | 14.395 | 0.75 | 0.0000 | 1 |

[1] Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg (https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset)

# Tackling time-series as a regression problem

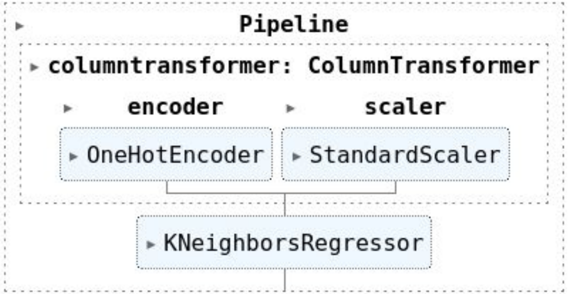
## The bike sharing dataset



# Tackling time-series as a regression problem

## Our first (overfitting) baseline model

```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.neighbors import KNeighborsRegressor

categorical_columns = ["season", "holiday", "workingday", "weather",]
numerical_columns = [
    "year", "month", "hour", "weekday", "temp", "feel_temp", "humidity", "windspeed"
]
preprocessing = ColumnTransformer(transformers=[
    ("encoder", OneHotEncoder(), categorical_columns),
    ("scaler", StandardScaler(), numerical_columns),
])
model = make_pipeline(preprocessing, KNeighborsRegressor(n_neighbors=1))
model
```



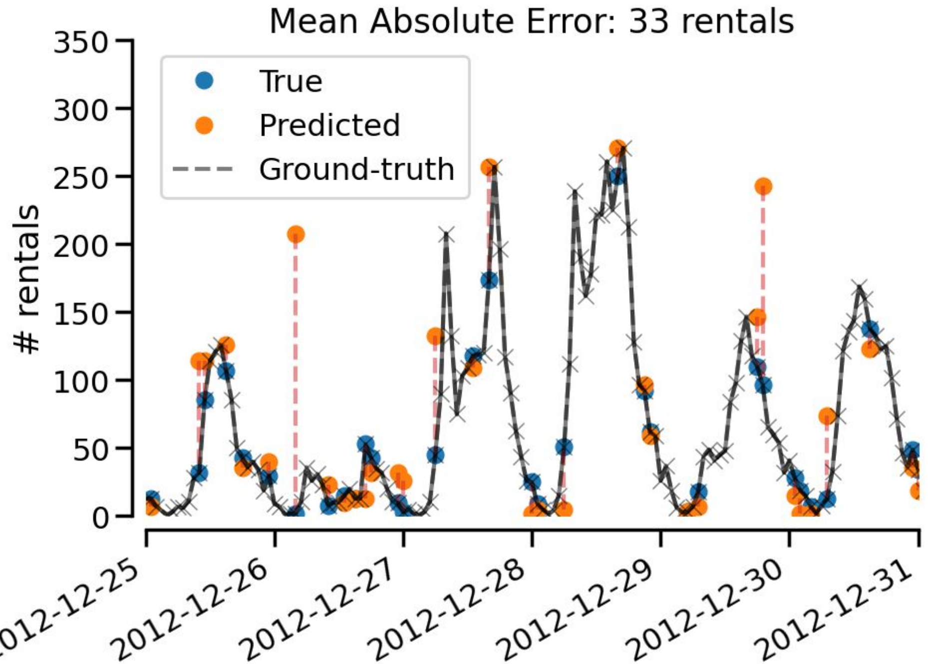
# Tackling time-series as a regression problem

## Evaluation with non-i.i.d data

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, shuffle=True, random_state=42, test_size=0.2
)
```

```python
from sklearn.metrics import mean_absolute_error

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"MAE: {mean_absolute_error(y_test, y_pred):.0f} rentals")
```

```
MAE: 76 rentals
```



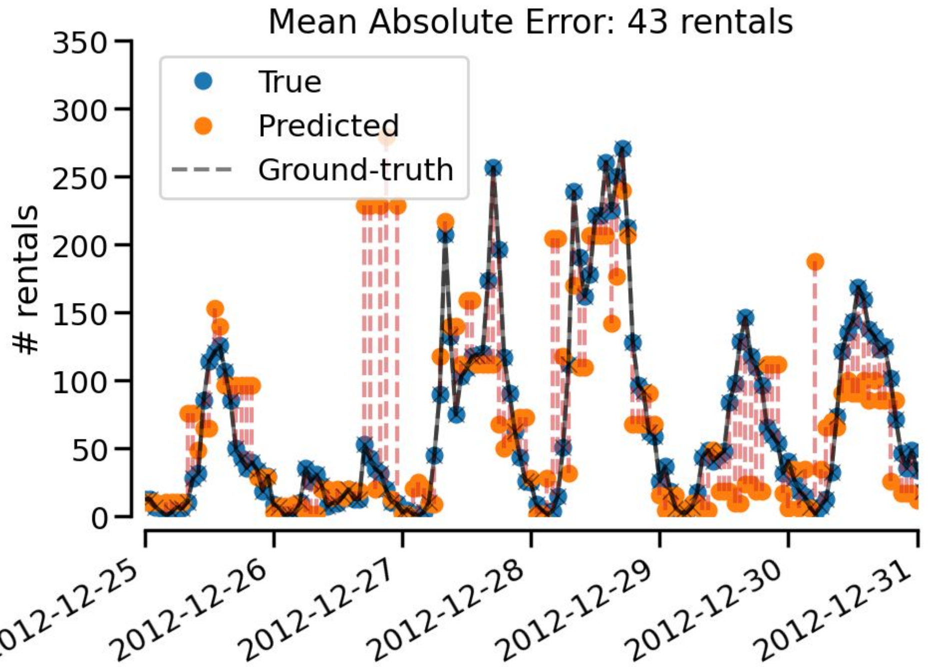
# Tackling time-series as a regression problem

## Evaluation with non-i.i.d data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, shuffle=False, random_state=42, test_size=0.2
)
```

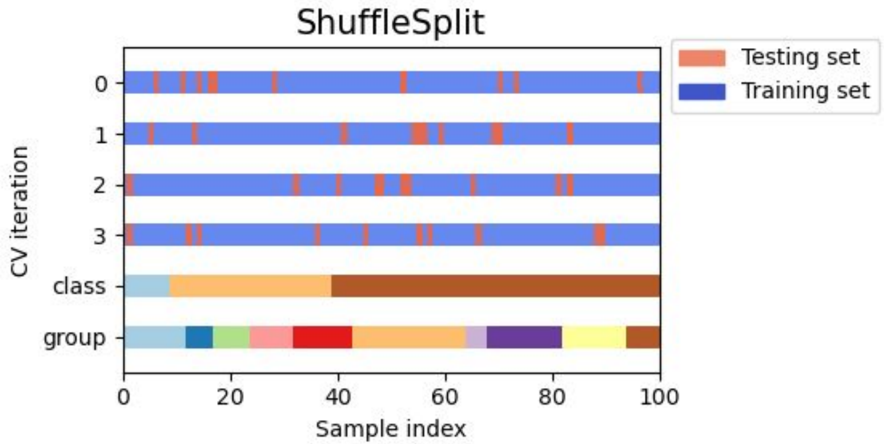
```
from sklearn.metrics import mean_absolute_error

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"MAE: {mean_absolute_error(y_test, y_pred):.0f} rentals")
```

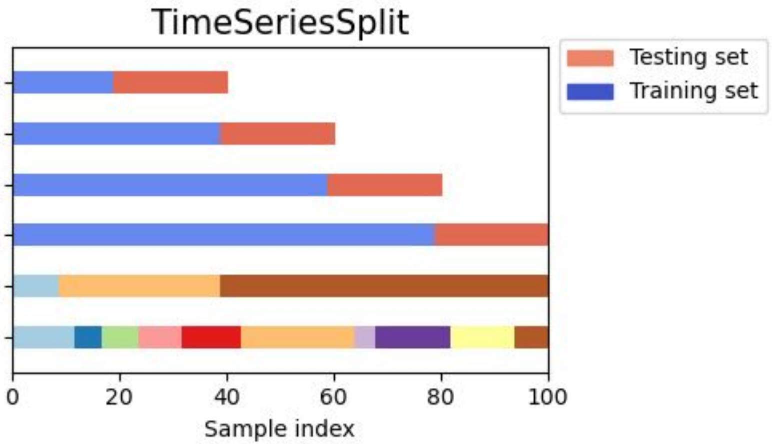
```
MAE: 119 rentals
```



# Tackling time-series as a regression problem

## Evaluation with non-i.i.d data




```python
from sklearn.model_selection import cross_validate, ShuffleSplit

cv = ShuffleSplit()
results = cross_validate(model, X, y, cv=cv, scoring="neg_mean_absolute_error")
print(f"MAE: {-results['test_score'].mean():.0f} rentals")
```

```
MAE: 75 rentals
```

```python
from sklearn.model_selection import cross_validate, TimeSeriesSplit

cv = TimeSeriesSplit()
results = cross_validate(model, X, y, cv=cv, scoring="neg_mean_absolute_error")
print(f"MAE: {-results['test_score'].mean():.0f} rentals")
```

```
MAE: 116 rentals
```

# Tackling time-series as a regression problem

## Evaluation with non-i.i.d data

```
from sklearn.model_selection import cross_validate, ShuffleSplit

cv = ShuffleSplit()
results = cross_validate(model, X, y, cv=cv)
results["test_score"]
```

```
array([0.51609683, 0.5774131 , 0.53495167, 0.55402703, 0.55465608,
       0.55297954, 0.591685  , 0.52655122, 0.53399944, 0.5687863 ])
```

```
from sklearn.model_selection import cross_validate, TimeSeriesSplit

cv = TimeSeriesSplit()
results = cross_validate(model, X, y, cv=cv)
results["test_score"]
```

```
array([-0.06406429,  0.01166642,  0.03483939,  0.12561052,  0.3277383 ])
```

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

# Tackling time-series as a regression problem

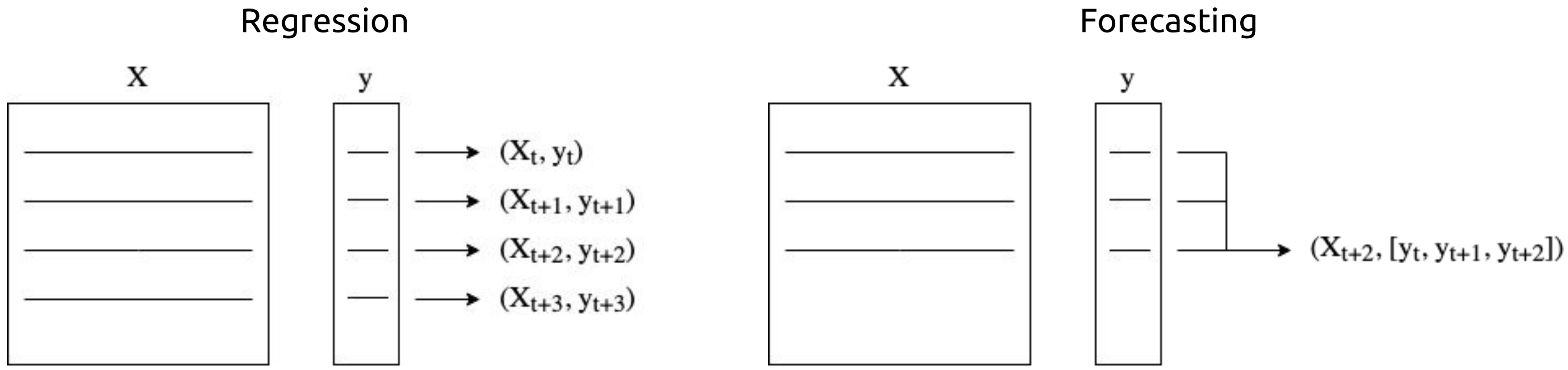
## Evaluation with non-i.i.d data

```python
import numpy as np
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_pinball_loss
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_error


def evaluate(model, X, y, cv):
    def score_func(estimator, X, y):
        y_pred = estimator.predict(X)
        return {
            "mean_absolute_error": mean_absolute_error(y, y_pred),
            "mean_pinball_05_loss": mean_pinball_loss(y, y_pred, alpha=0.05),
            "mean_pinball_50_loss": mean_pinball_loss(y, y_pred, alpha=0.50),
            "mean_pinball_95_loss": mean_pinball_loss(y, y_pred, alpha=0.95),
        }
    cv_results = cross_validate(model, X, y, cv=cv, scoring=score_func)
    for key, value in cv_results.items():
        if key.startswith("test_"):
            print(f"{key[5:]}: {value.mean():.3f} ± {value.std():.3f}")
```

# Forecasting vs. regression

## Modeling time dependence

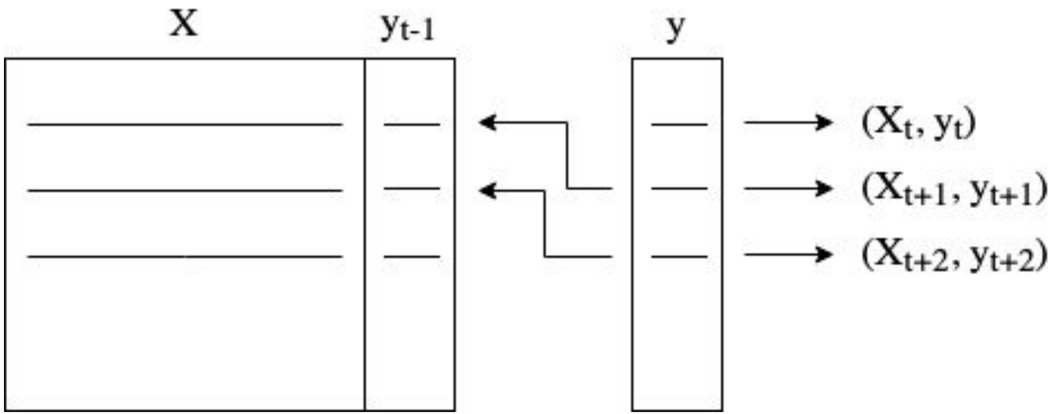
**Forecasting** is the process of making predictions based on **past** and **present** data [1].




[1] Wikipedia contributors. (2022, August 6). Forecasting. In *Wikipedia, The Free Encyclopedia*. Retrieved 09:53, November 8, 2022, from https://en.wikipedia.org/w/index.php?title=Forecasting&oldid=1102751973

# Time-series forecasting via tabularization

## Injecting delayed target information in X



# Time-series forecasting via tabularization

## Delaying target with Pandas

```
lagged_target = pd.concat(
    [
        y,
        y.shift(1).rename("lagged_count_1h"),
        y.shift(2).rename("lagged_count_2h"),
        y.shift(3).rename("lagged_count_3h"),
        y.shift(24).rename("lagged_count_1d"),
        y.shift(24 + 1).rename("lagged_count_1d_1h"),
        y.shift(7 * 24).rename("lagged_count_7d"),
        y.shift(7 * 24 + 1).rename("lagged_count_7d_1h"),
        y.shift(1).rolling(24).mean().rename("lagged_mean_24h"),
        y.shift(1).rolling(24).max().rename("lagged_max_24h"),
        y.shift(1).rolling(24).min().rename("lagged_min_24h"),
        y.shift(1).rolling(7 * 24).mean().rename("lagged_mean_7d"),
        y.shift(1).rolling(7 * 24).max().rename("lagged_max_7d"),
        y.shift(1).rolling(7 * 24).min().rename("lagged_min_7d"),
    ],
    axis="columns",
)
```

`lagged_target.head()`

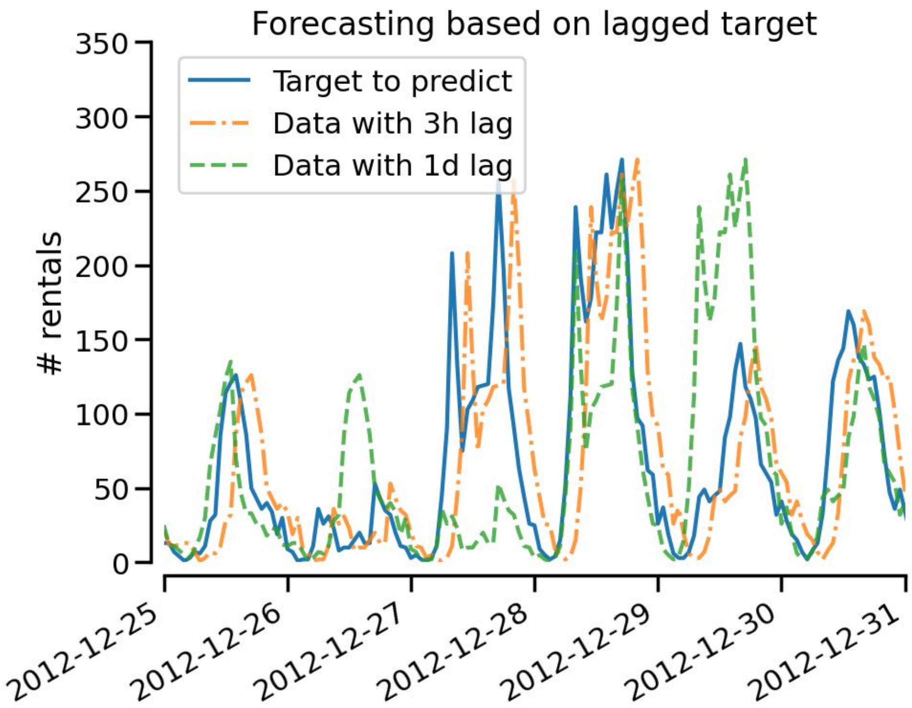
| dteday | count | lagged_count_1h | lagged_count_2h | lagged_count_3h | lagged_count_1d | lagged_count_1d_1h | lagged_count_7d |
|---|---|---|---|---|---|---|---|
| 2011-01-01 00:00:00 | 16 | NaN | NaN | NaN | NaN | NaN | NaN |
| 2011-01-01 01:00:00 | 40 | 16.0 | NaN | NaN | NaN | NaN | NaN |
| 2011-01-01 02:00:00 | 32 | 40.0 | 16.0 | NaN | NaN | NaN | NaN |
| 2011-01-01 03:00:00 | 13 | 32.0 | 40.0 | 16.0 | NaN | NaN | NaN |
| 2011-01-01 04:00:00 | 1 | 13.0 | 32.0 | 40.0 | NaN | NaN | NaN |

`lagged_target.tail()`

| dteday | count | lagged_count_1h | lagged_count_2h | lagged_count_3h | lagged_count_1d | lagged_count_1d_1h | lagged_count_7d |
|---|---|---|---|---|---|---|---|
| 2012-12-31 19:00:00 | 119 | 122.0 | 164.0 | 214.0 | 102.0 | 125.0 | 26.0 |
| 2012-12-31 20:00:00 | 89 | 119.0 | 122.0 | 164.0 | 72.0 | 102.0 | 18.0 |
| 2012-12-31 21:00:00 | 90 | 89.0 | 119.0 | 122.0 | 47.0 | 72.0 | 23.0 |
| 2012-12-31 22:00:00 | 61 | 90.0 | 89.0 | 119.0 | 36.0 | 47.0 | 22.0 |
| 2012-12-31 23:00:00 | 49 | 61.0 | 90.0 | 89.0 | 49.0 | 36.0 | 12.0 |

# Time-series forecasting via tabularization

## Delaying target with Pandas



# Time-series forecasting via tabularization

## Linear models

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge

model = make_pipeline(StandardScaler(), Ridge(alpha=1e-4))
evaluate(model, X, y)
```

```
mean_absolute_error: 43.204 ± 3.057
mean_pinball_05_loss: 21.638 ± 2.077
mean_pinball_50_loss: 21.602 ± 1.529
mean_pinball_95_loss: 21.566 ± 1.393
```

```
from sklearn.linear_model import PoissonRegressor

model = make_pipeline(StandardScaler(), PoissonRegressor(alpha=1e-4))
evaluate(model, X, y)
```

```
mean_absolute_error: 101.267 ± 12.199
mean_pinball_05_loss: 65.596 ± 11.262
mean_pinball_50_loss: 50.633 ± 6.099
mean_pinball_95_loss: 35.670 ± 5.864
```

# Time-series forecasting via tabularization

## Gradient boosting regression trees

```python
from sklearn.ensemble import HistGradientBoostingRegressor


gbrt_mse = HistGradientBoostingRegressor(loss="squared_error")
evaluate(gbrt_mse, X, y)
```

```
mean_absolute_error: 39.088 ± 2.268
mean_pinball_05_loss: 17.700 ± 1.275
mean_pinball_50_loss: 19.544 ± 1.134
mean_pinball_95_loss: 21.388 ± 2.363
```

```python
gbrt_poisson = HistGradientBoostingRegressor(loss="poisson")
evaluate(gbrt_poisson, X, y)
```

```
mean_absolute_error: 39.307 ± 2.808
mean_pinball_05_loss: 16.669 ± 1.541
mean_pinball_50_loss: 19.653 ± 1.404
mean_pinball_95_loss: 22.638 ± 2.983
```

# Time-series forecasting via tabularization

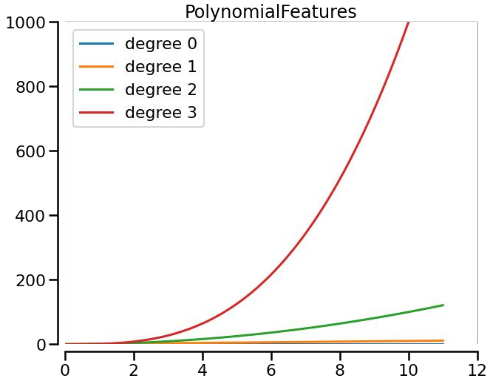
## Linear models with feature engineering

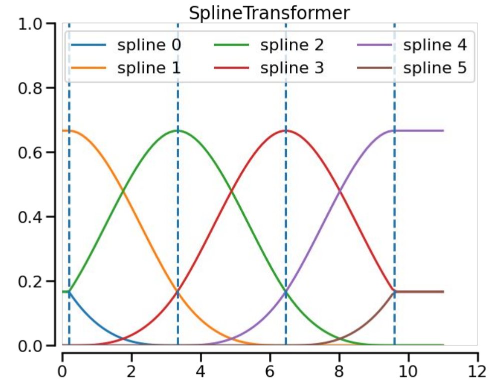
```python
X.shape
```

```
(17210, 13)
```

```python
from sklearn.preprocessing import SplineTransformer

SplineTransformer(n_knots=10, degree=3).fit_transform(X).shape
```

```
(17210, 156)
```



# Time-series forecasting via tabularization

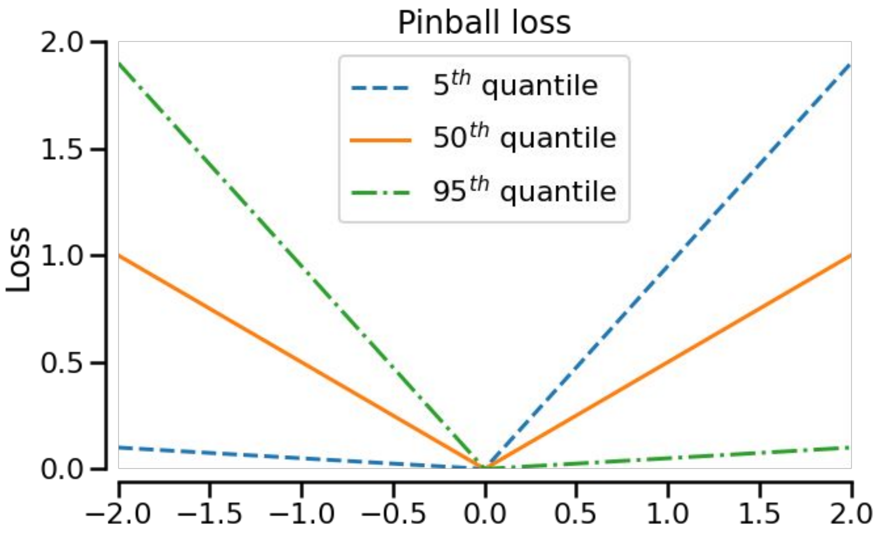
## Linear models with feature engineering

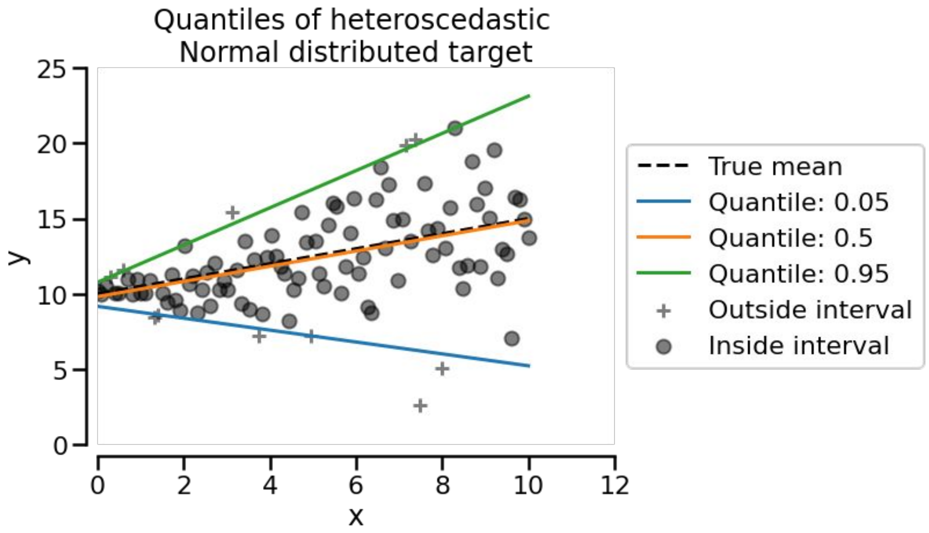
```python
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.pipeline import make_pipeline


pipeline = make_pipeline(
    SplineTransformer(n_knots=12, degree=3),
    PoissonRegressor(alpha=1e-6, max_iter=300),
)

with warnings.catch_warnings():
    warnings.simplefilter("ignore", ConvergenceWarning)
    evaluate(pipeline, X, y)
```

```
mean_absolute_error: 54.486 ± 17.952
mean_pinball_05_loss: 16.356 ± 2.416
mean_pinball_50_loss: 27.243 ± 8.976
mean_pinball_95_loss: 38.130 ± 19.391
```

# Modeling predictive uncertainty

## Estimating quantiles



# Modeling predictive uncertainty

## Gradient boosting regression trees with quantile loss

```
gbrt_percentile_05 = HistGradientBoostingRegressor(loss="quantile", quantile=0.05)
evaluate(gbrt_percentile_05, X, y)
```

```
mean_absolute_error: 92.476 ± 16.235
mean_pinball_05_loss: 5.874 ± 0.925
mean_pinball_50_loss: 46.238 ± 8.117
mean_pinball_95_loss: 86.602 ± 15.310
```

```
gbrt_median = HistGradientBoostingRegressor(loss="quantile", quantile=0.5)
evaluate(gbrt_median, X, y)
```

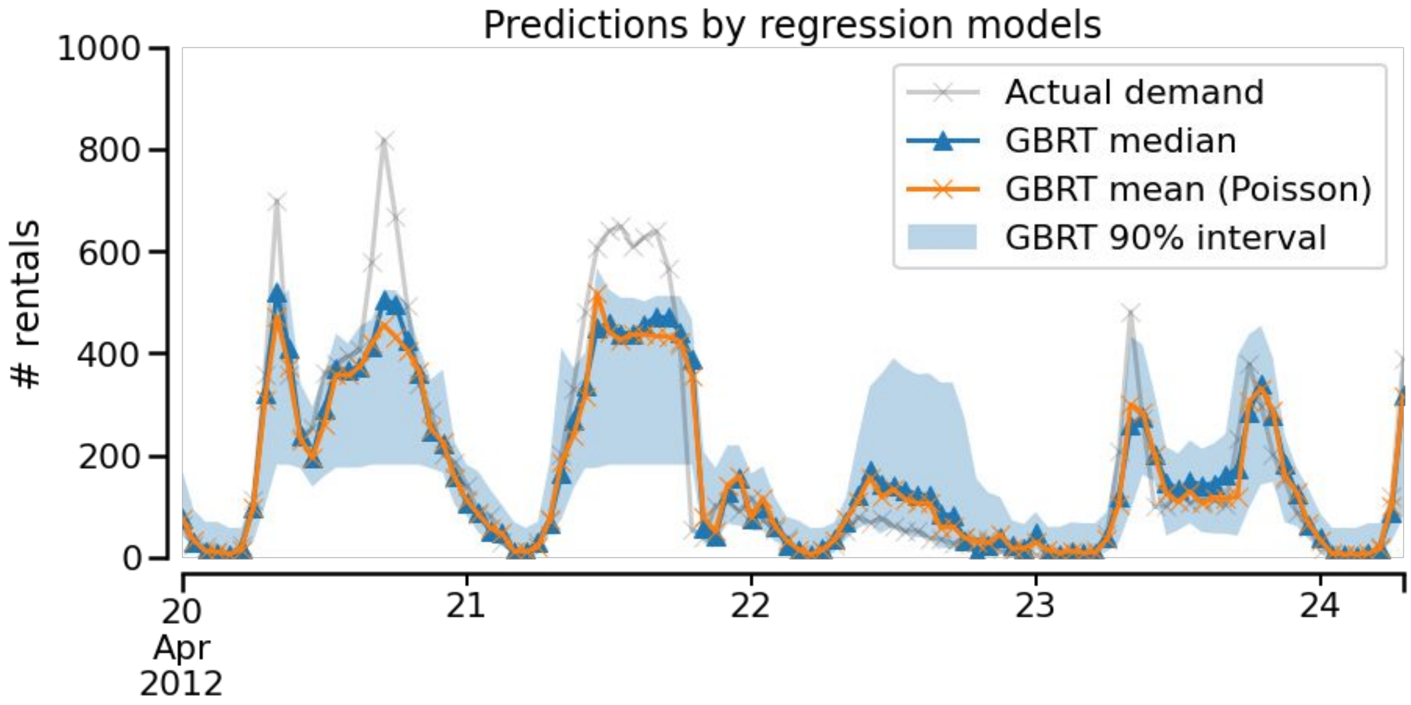
```
mean_absolute_error: 39.854 ± 3.167
mean_pinball_05_loss: 17.147 ± 1.067
mean_pinball_50_loss: 19.927 ± 1.584
mean_pinball_95_loss: 22.706 ± 3.131
```

```
gbrt_percentile_95 = HistGradientBoostingRegressor(loss="quantile", quantile=0.95)
evaluate(gbrt_percentile_95, X, y)
```

```
mean_absolute_error: 72.009 ± 6.143
mean_pinball_05_loss: 62.901 ± 7.443
mean_pinball_50_loss: 36.005 ± 3.071
mean_pinball_95_loss: 9.109 ± 1.305
```

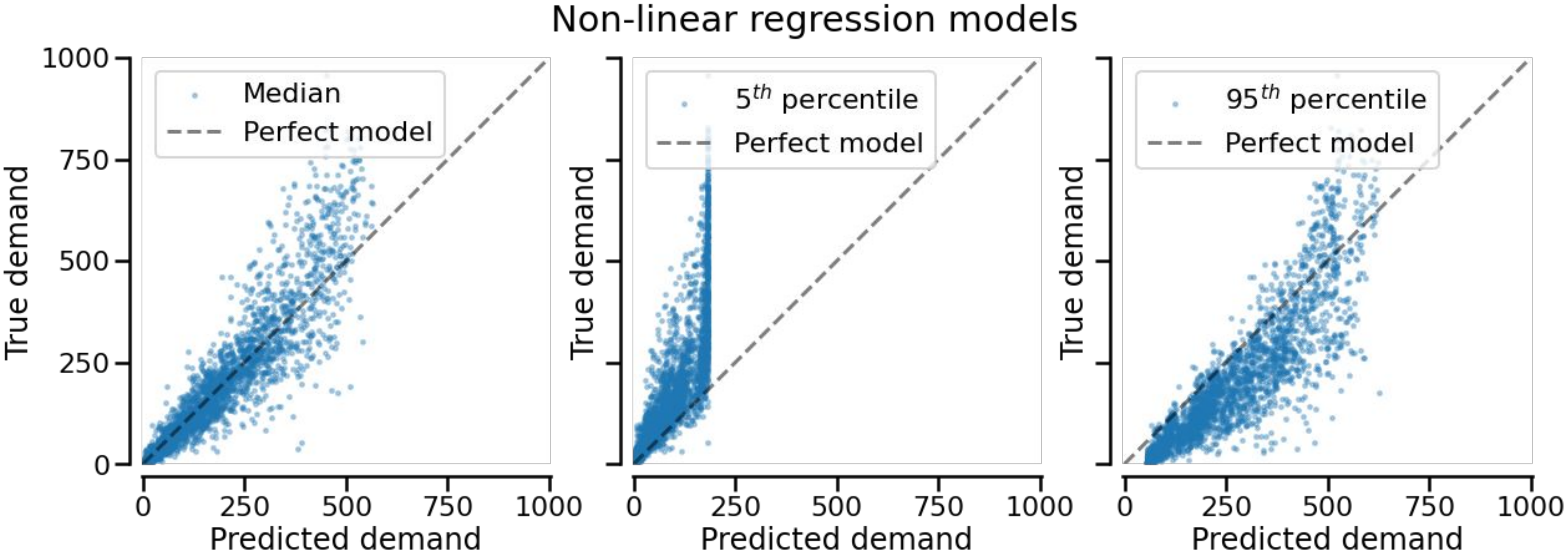
# Modeling predictive uncertainty

## Qualitative evaluation of quantile calibration



# Modeling predictive uncertainty

## Qualitative evaluation of quantile calibration



# Modeling predictive uncertainty

## Quantile calibration through effective coverage

```
(median_predictions > y_test).mean()
```

```
0.5246666666666666
```

```
(percentile_5_predictions > y_test).mean()
```

```
0.081
```

```
(percentile_95_predictions > y_test).mean()
```

```
0.8823333333333333
```

```
np.logical_and(
    percentile_5_predictions < y_test,
    percentile_95_predictions > y_test,
).mean()
```

```
0.8013333333333333
```

# Modeling predictive uncertainty

## Causes of miscalibrated models

### In theory

The pinball loss is guaranteed to be minimized by models that estimate the quantiles perfectly.

### In practice

This is an asymptotic property with access to an infinite number of data points.

With a finite set of samples, models with the same pinball loss can **trade calibration for ranking power**

# Modeling predictive uncertainty

## Can we do better?

```python
from numpy import quantile

fixed_05_quantile, fixed_95_quantile = np.quantile(y_train, [0.05, 0.95])
fixed_05_quantile, fixed_95_quantile
```

```
(4.0, 423.0)
```

```python
np.logical_and(
    fixed_05_quantile < y_test, fixed_95_quantile > y_test,
).mean()
```

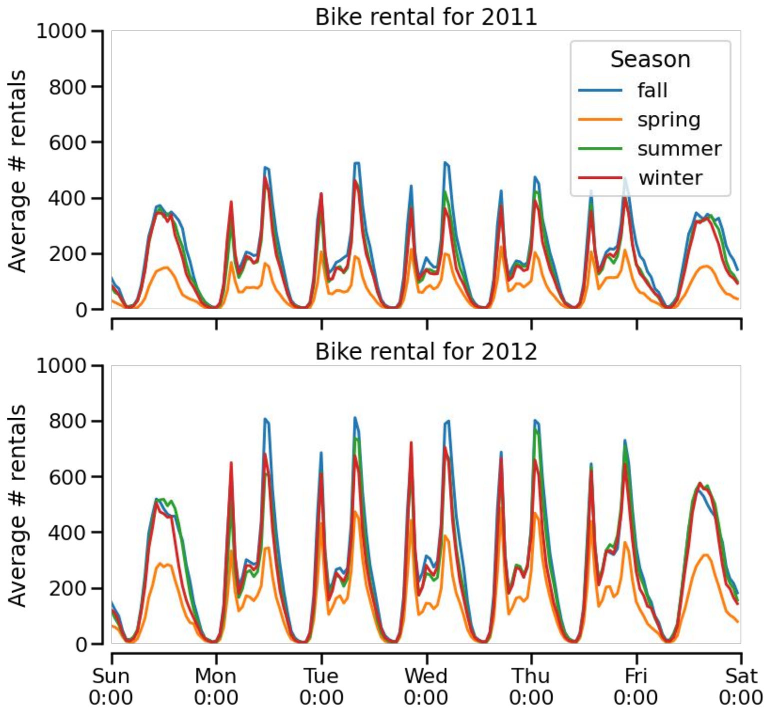
```
0.831
```

```python
mean_pinball_loss(
    y_test, np.full_like(y_test, fill_value=fixed_05_quantile), alpha=0.05
)
```

```
8.624333333333333
```

```python
mean_pinball_loss(
    y_test, np.full_like(y_test, fill_value=fixed_95_quantile), alpha=0.95
)
```

```
26.398333333333344
```



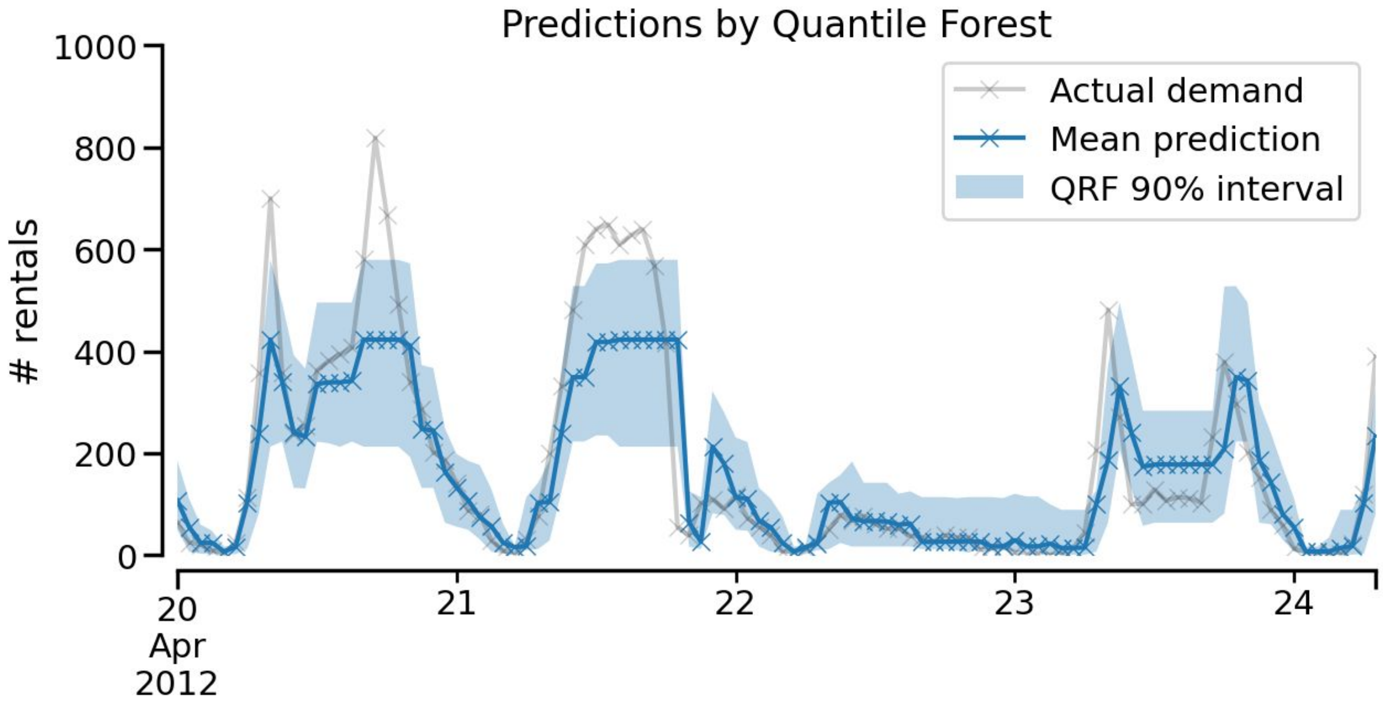
# Modeling predictive uncertainty

## Can we do better?

- Collecting more data-points (in case the model is overfitting)
- Better tuning of the model hyper-parameters
- Engineering more predictive features from the same data
- Integrate features that will reduce the uncertainty
- Try other kinds of quantile regression models

# Modeling predictive uncertainty

## Uncertainty prediction with quantile forest



# Modeling predictive uncertainty

## Uncertainty prediction with quantile forest

```
(percentile_5_predictions > y_test).mean()
```

```
0.052
```

```
(percentile_95_predictions > y_test).mean()
```

```
0.9063333333333333
```

```
np.logical_and(
    percentile_5_predictions < y_test,
    percentile_95_predictions > y_test,
).mean()
```

```
0.8393333333333334
```

# Modeling predictive uncertainty

## Uncertainty prediction MAPIE

Uncertainty prediction based on conformal prediction methods and estimates both **aleatoric** and **epistemic** uncertainty at the same time:

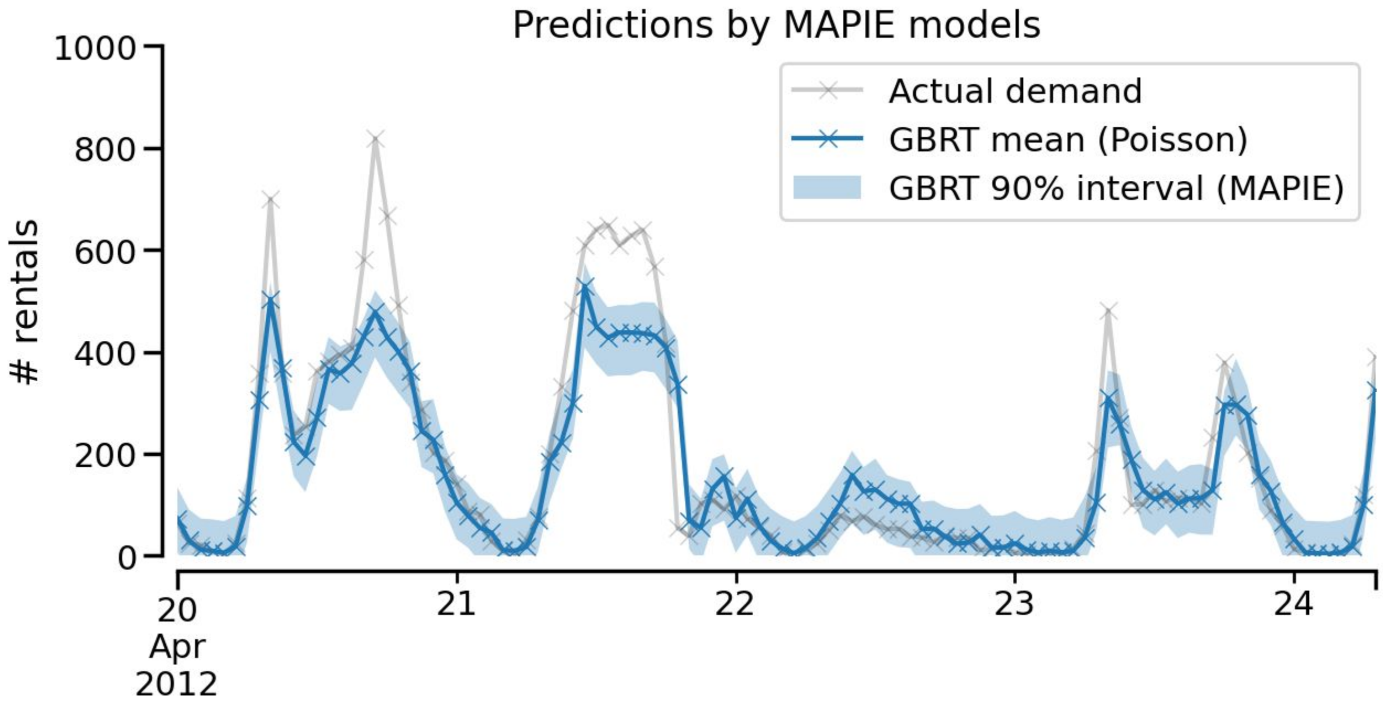
- aleatoric uncertainty due to limited information in the input data;

- epistemic uncertainty due to a limited number of samples.

```
from mapie.regression import MapieRegressor

gbrt_mean_poisson_mapie = MapieRegressor(
    HistGradientBoostingRegressor(loss="poisson"), cv=5
)
gbrt_mean_poisson_mapie.fit(X_train, y_train)
mean_predictions, predictions_90_pi = gbrt_mean_poisson_mapie.predict(X_test, alpha=0.1
```

# Modeling predictive uncertainty

## Uncertainty prediction MAPIE



# Modeling predictive uncertainty

## Uncertainty prediction MAPIE

```
(predictions_90_pi_low > y_test).mean()
```

```
0.052
```

```
(predictions_90_pi_high > y_test).mean()
```

```
0.9063333333333333
```

```
np.logical_and(
    predictions_90_pi_low < y_test,
    predictions_90_pi_high > y_test,
).mean()
```

```
0.842
```

# Beyond scikit-learn

## Limitations of the current approach

- Feature engineering not integrated within the cross-validation
- Predicting a single step or in other words the next hour

# Beyond scikit-learn

## Example of `sktime`

```
from sktime.forecasting.model_selection import temporal_train_test_split


y_train, y_test = temporal_train_test_split(y, test_size=7 * 24)
```

```
from sktime.forecasting.base import ForecastingHorizon

fh = ForecastingHorizon(y_test.index, is_relative=False)
fh
```

```
ForecastingHorizon(['2012-12-25 00:00:00', '2012-12-25 01:00:00',
               '2012-12-25 02:00:00', '2012-12-25 03:00:00',
               '2012-12-25 04:00:00', '2012-12-25 05:00:00',
               '2012-12-25 06:00:00', '2012-12-25 07:00:00',
               '2012-12-25 08:00:00', '2012-12-25 09:00:00',
               ...
               '2012-12-31 14:00:00', '2012-12-31 15:00:00',
               '2012-12-31 16:00:00', '2012-12-31 17:00:00',
               '2012-12-31 18:00:00', '2012-12-31 19:00:00',
               '2012-12-31 20:00:00', '2012-12-31 21:00:00',
               '2012-12-31 22:00:00', '2012-12-31 23:00:00'],
              dtype='datetime64[ns]', name='dteday', length=168, freq='H', is_relative=
```

# Beyond scikit-learn

## Example of `sktime`

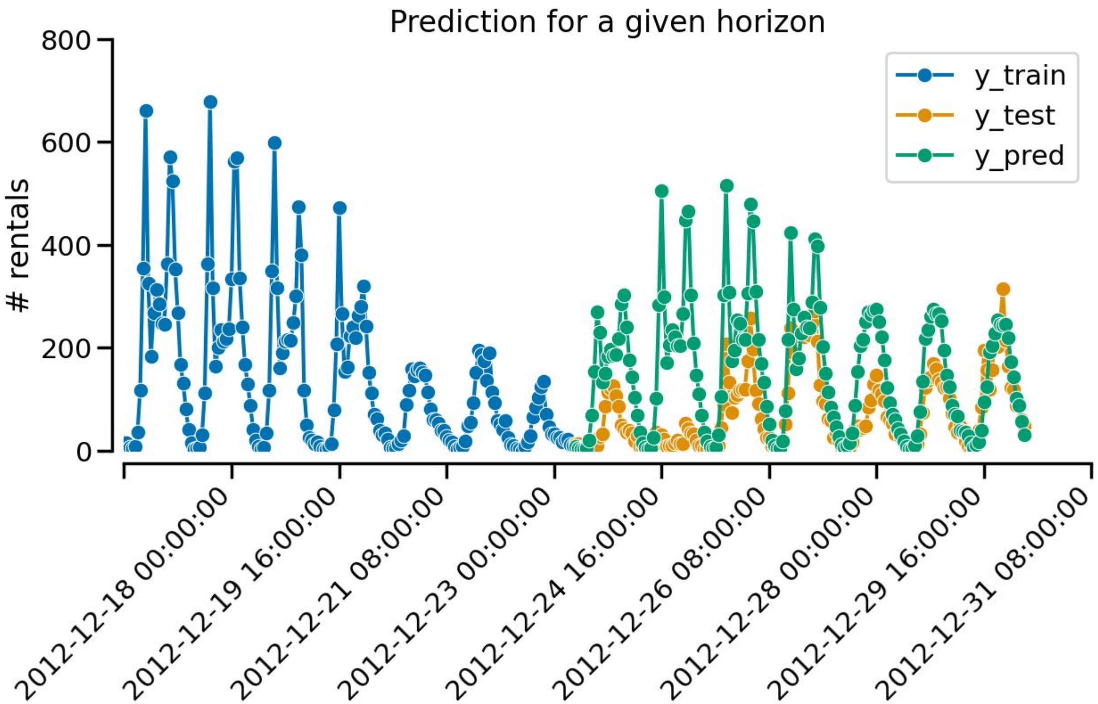
```
from sktime.forecasting.compose import make_reduction

regressor = HistGradientBoostingRegressor(
    loss="poisson", max_leaf_nodes=64, max_iter=300
)
forecaster = make_reduction(regressor, window_length=7 * 24, strategy="recursive")
forecaster.fit(y_train)
```

```
▸ RecursiveTabularRegressionForecaster
▸ estimator: HistGradientBoostingRegressor
    ▸ HistGradientBoostingRegressor
```

```
y_pred = forecaster.predict(fh=fh)
```

# Beyond scikit-learn

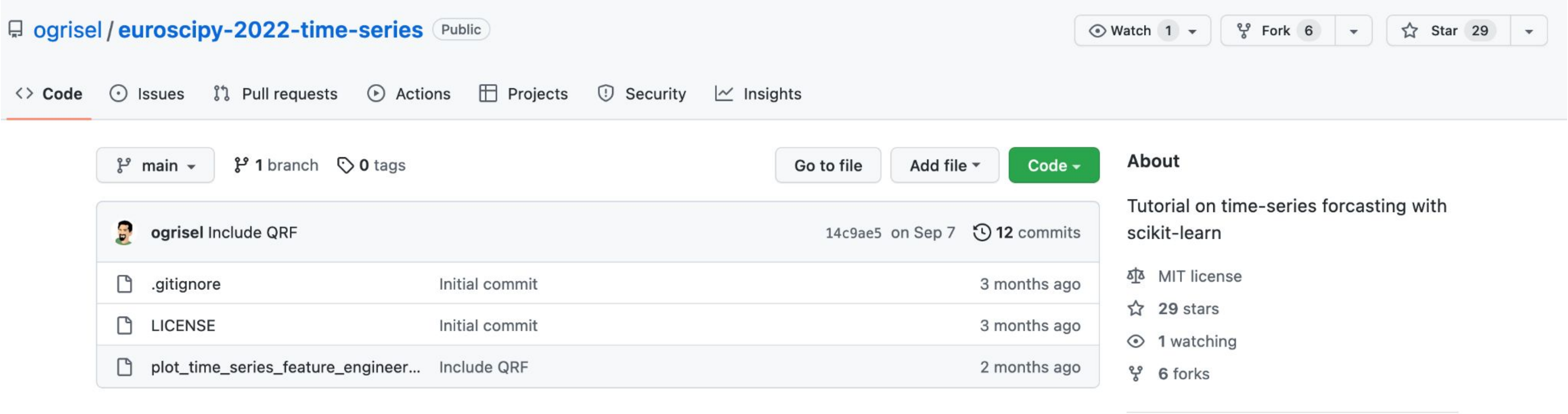
## Example of `sktime`



# scikit-learn @ Inria fondation partners:

# Tutorial material

## GitHub repository




https://github.com/ogrisel/euroscipy-2022-time-series