MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Time-delayed features and auto-regressive models

Chris Holdgraf
Fellow, Berkeley Institute for Data Science

The past is useful

- Timeseries data almost always have information that is *shared between timepoints*
- Information in the past can help predict what happens in the future
- Often the features best-suited to predict a timeseries are previous values of the same timeseries.

A note on smoothness and auto-correlation

- A common question to ask of a timeseries: how smooth is the data.
- AKA, how correlated is a timepoint with its neighboring timepoints (called **autocorrelation**).
- The amount of auto-correlation in data will impact your models.

Creating time-lagged features

- Let's see how we could build a model that uses values in the past as input features.
- We can use this to assess how auto-correlated our signal is (and lots of other stuff too)

Time-shifting data with Pandas

```
print(df)

         df
    0   0.0
    1   1.0
    2   2.0
    3   3.0
    4   4.0

# Shift a DataFrame/Series by 3 index values towards the past
print(df.shift(3))

         df
    0   NaN
    1   NaN
    2   NaN
    3   0.0
    4   1.0
```

Creating a time-shifted DataFrame

```
# data is a pandas Series containing time series data
data = pd.Series(...)

# Shifts
shifts = [0, 1, 2, 3, 4, 5, 6, 7]

# Create a dictionary of time-shifted data
many_shifts = {'lag_{}'.format(ii): data.shift(ii) for ii in shifts}

# Convert them into a dataframe
many_shifts = pd.DataFrame(many_shifts)
```

Fitting a model with time-shifted features

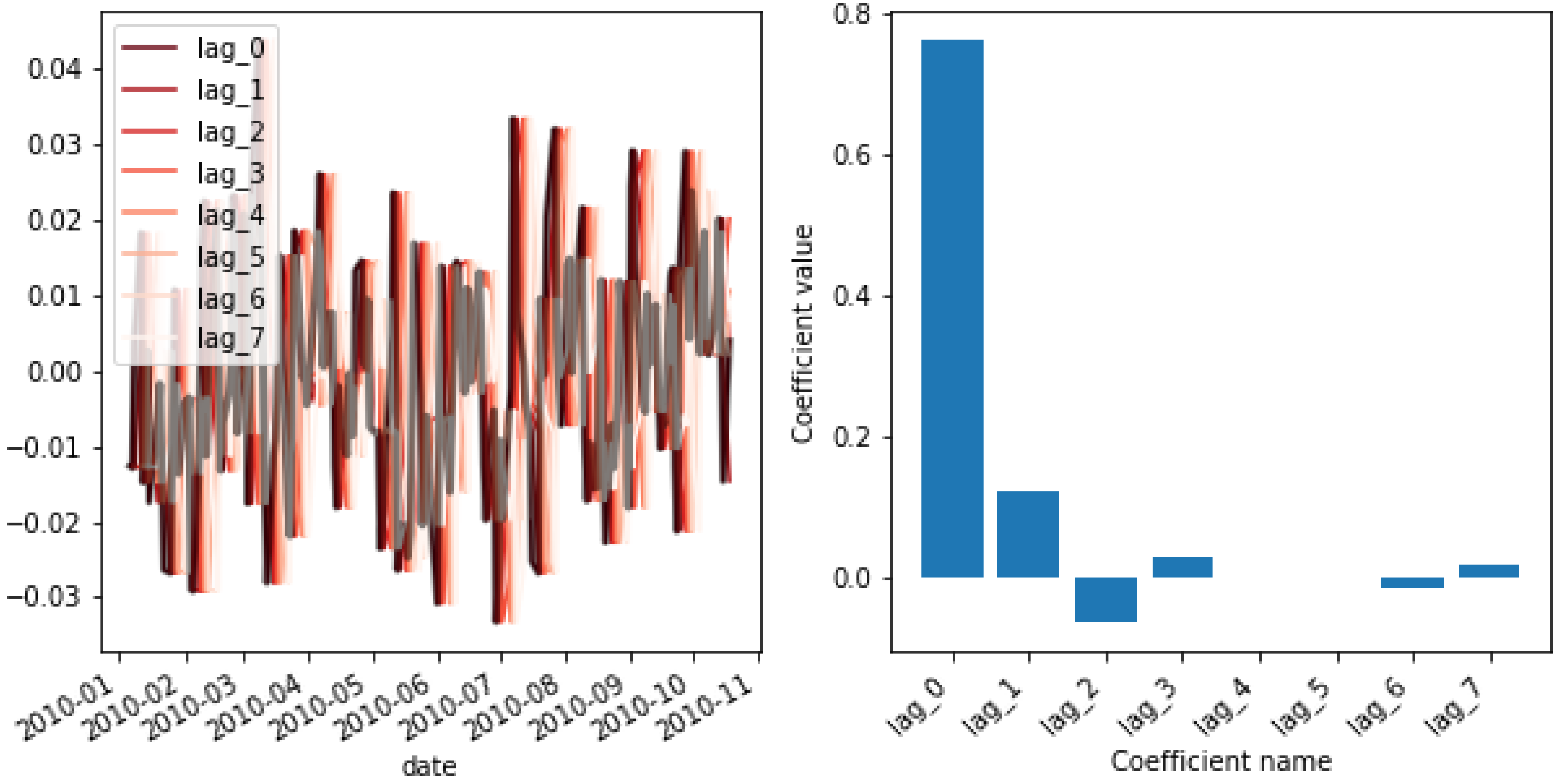
```
# Fit the model using these input features
model = Ridge()
model.fit(many_shifts, data)
```

Interpreting the auto-regressive model coefficients

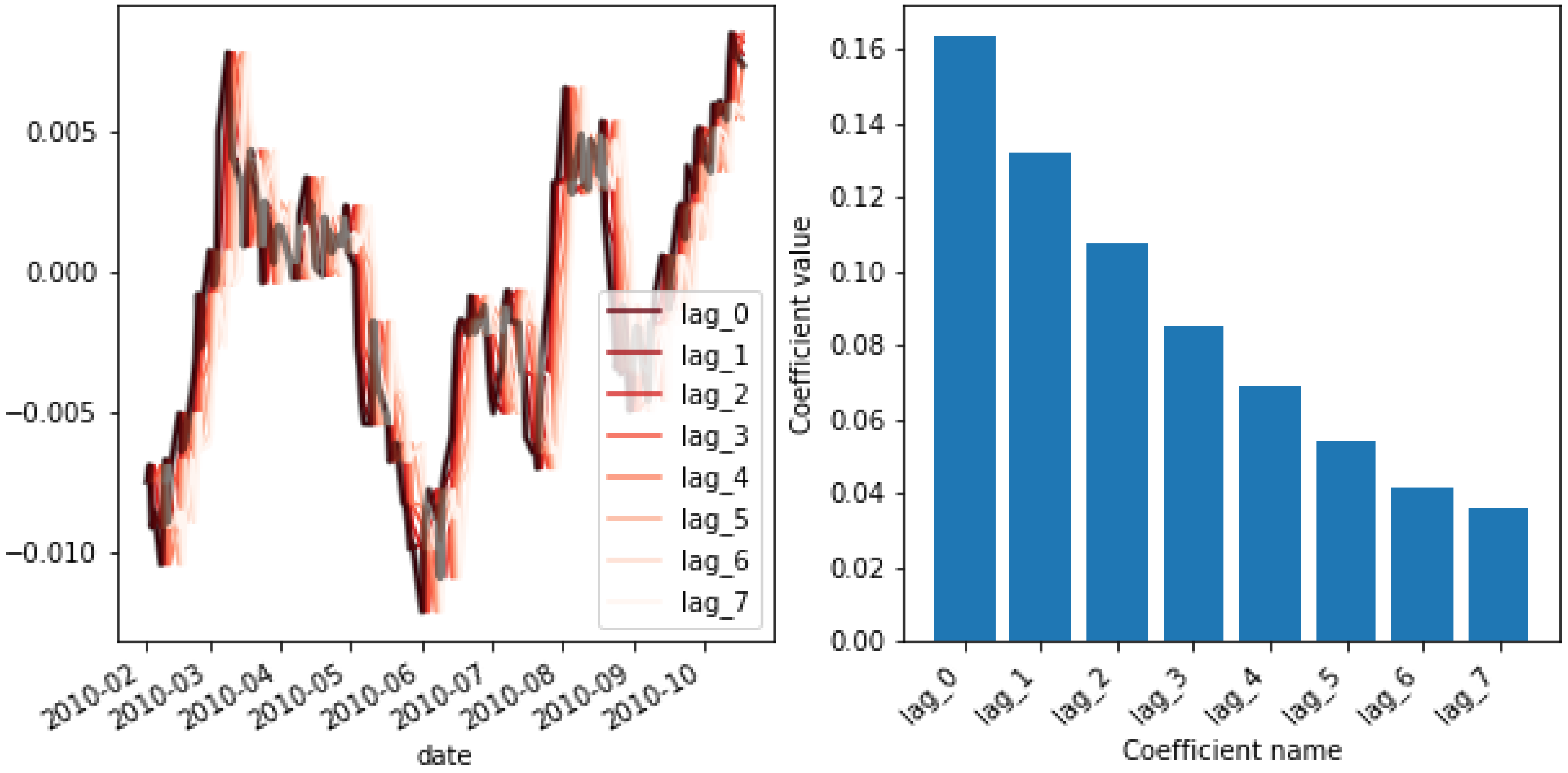
```
# Visualize the fit model coefficients
fig, ax = plt.subplots()
ax.bar(many_shifts.columns, model.coef_)
ax.set(xlabel='Coefficient name', ylabel='Coefficient value')

# Set formatting so it looks nice
plt.setp(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
```

Visualizing coefficients for a rough signal



Visualizing coefficients for a smooth signal



MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Let's practice!

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Cross-validating timeseries data

Chris Holdgraf
Fellow, Berkeley Institute for Data Science

Cross validation with scikit-learn

```
# Iterating over the "split" method yields train/test indices
for tr, tt in cv.split(X, y):
  model.fit(X[tr], y[tr])
  model.score(X[tt], y[tt])
```

Cross validation types: KFold

- `KFold` cross-validation splits your data into multiple "folds" of equal size

- It is one of the most common cross-validation routines

```
from sklearn.model_selection import KFold
cv = KFold(n_splits=5)
for tr, tt in cv.split(X, y):
    ...
```

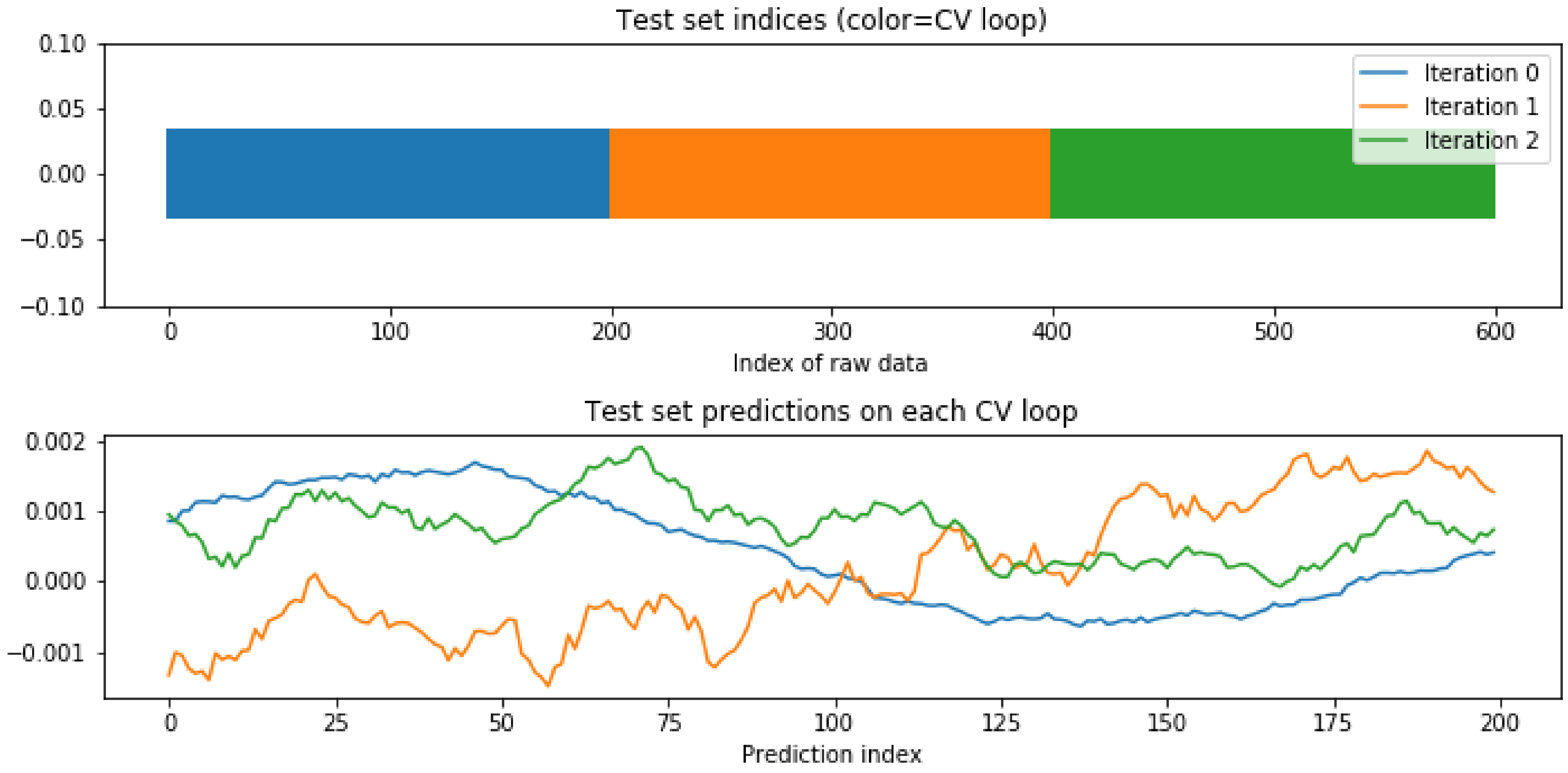
Visualizing model predictions

```
fig, axs = plt.subplots(2, 1)

# Plot the indices chosen for validation on each loop
axs[0].scatter(tt, [0] * len(tt), marker='_', s=2, lw=40)
axs[0].set(ylim=[-.1, .1], title='Test set indices (color=CV loop)',
           xlabel='Index of raw data')

# Plot the model predictions on each iteration
axs[1].plot(model.predict(X[tt]))
axs[1].set(title='Test set predictions on each CV loop',
           xlabel='Prediction index')
```

Visualizing KFold CV behavior



A note on shuffling your data

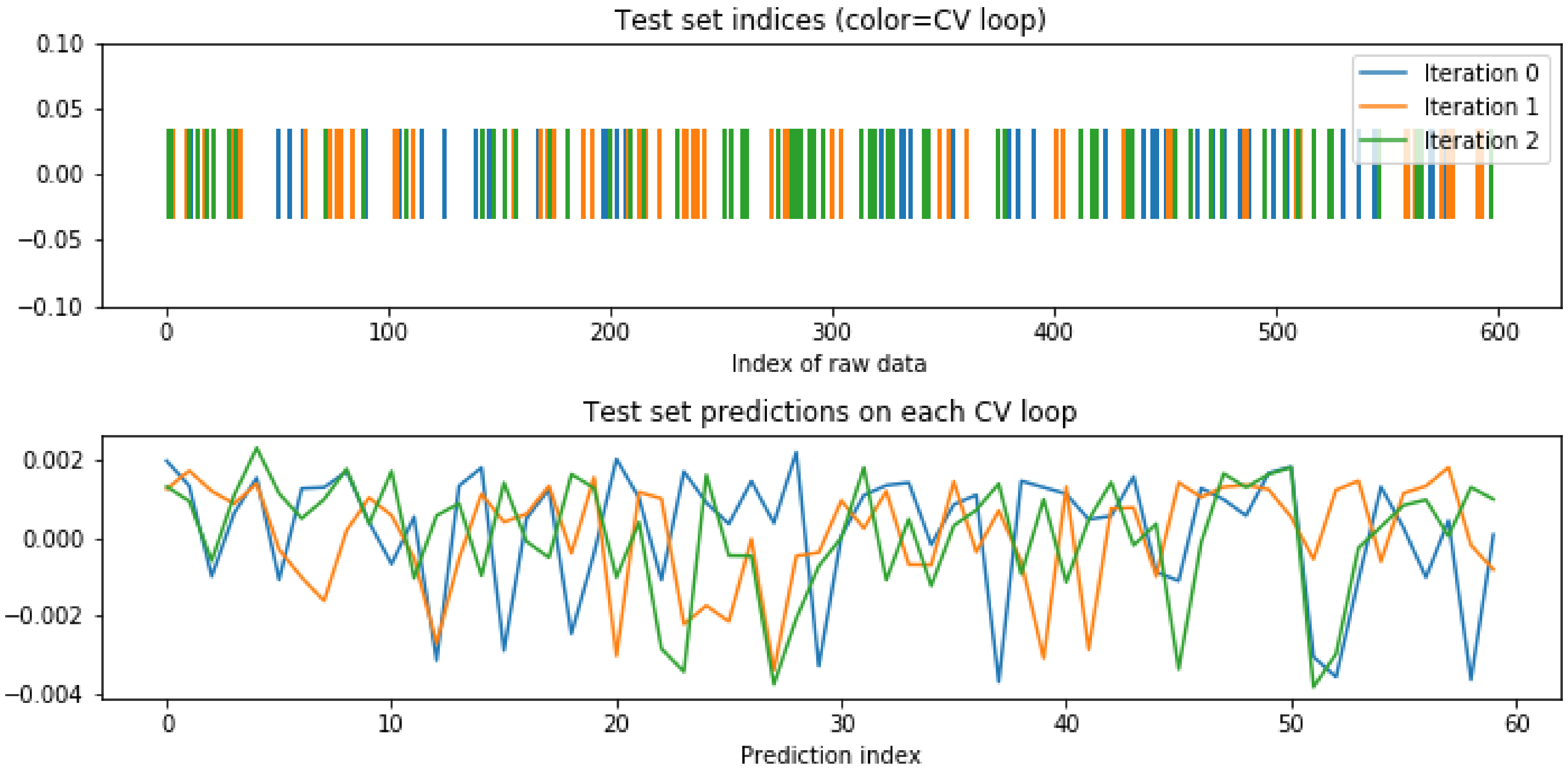
- Many CV iterators let you shuffle data as a part of the cross-validation process.
- This only works if the data is i.i.d., which timeseries usually is **not**.

- You should *not* shuffle your data when making predictions with timeseries.

```
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=3)
for tr, tt in cv.split(X, y):
    ...
```

Visualizing shuffled CV behavior



Using the time series CV iterator

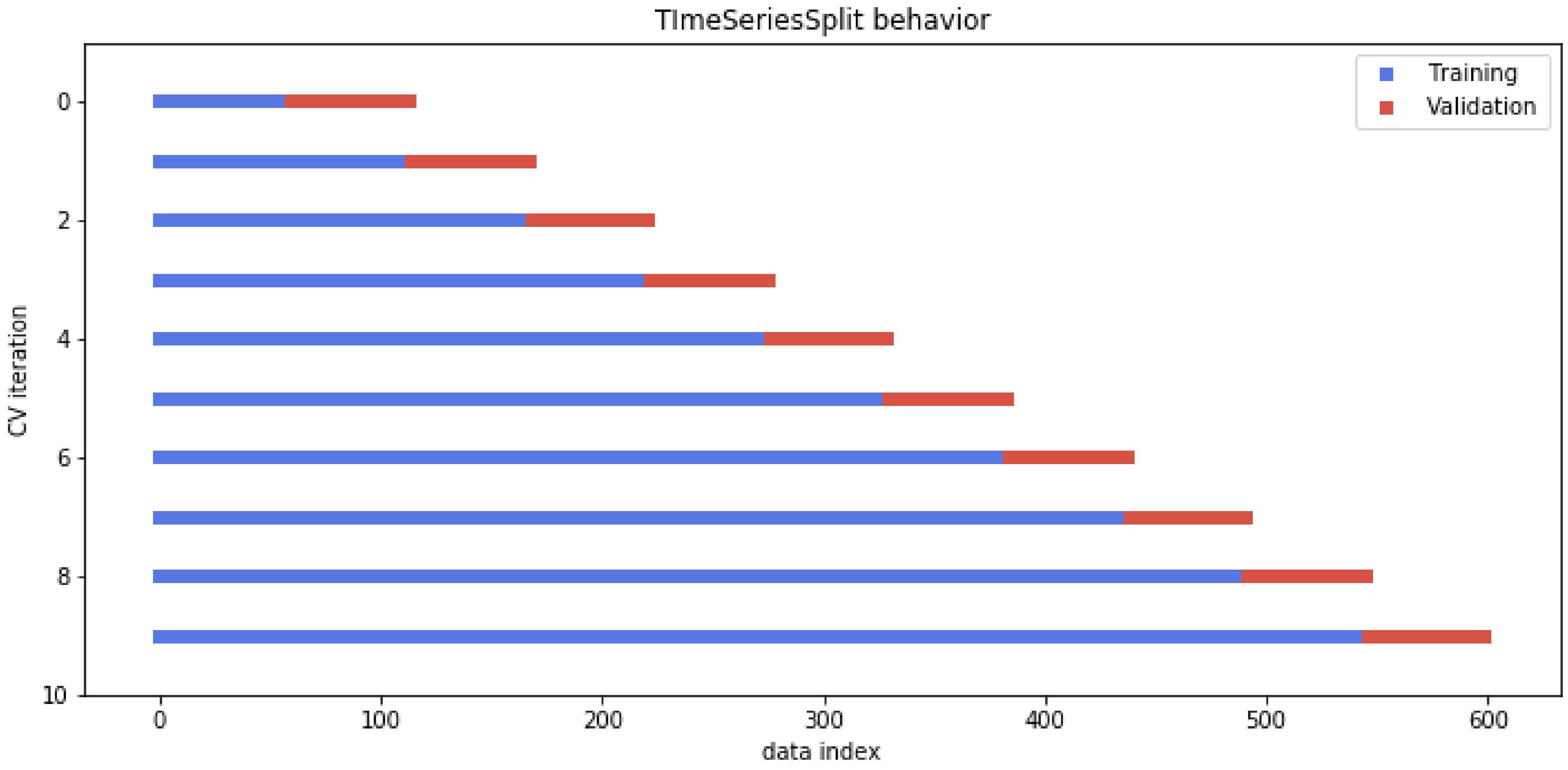
- Thus far, we've broken the linear passage of time in the cross validation
- Depending on your problem, you can't use datapoints in the future to predict data in the past
- One approach: Always use training data from the **past** to predict the **future**

Visualizing time series cross validation iterators

```python
# Import and initialize the cross-validation iterator
from sklearn.model_selection import TimeSeriesSplit
cv = TimeSeriesSplit(n_splits=10)

fig, ax = plt.subplots(figsize=(10, 5))
for ii, (tr, tt) in enumerate(cv.split(X, y)):
    # Plot training and test indices
    l1 = ax.scatter(tr, [ii] * len(tr), c=[plt.cm.coolwarm(.1)],
                    marker='_', lw=6)
    l2 = ax.scatter(tt, [ii] * len(tt), c=[plt.cm.coolwarm(.9)],
                    marker='_', lw=6)
    ax.set(ylim=[10, -1], title='TimeSeriesSplit behavior',
           xlabel='data index', ylabel='CV iteration')
    ax.legend([l1, l2], ['Training', 'Validation'])
```

Visualizing the TimeSeriesSplit cross validation iterator



Custom scoring functions in scikit-learn

```
def myfunction(estimator, X, y):
    y_pred = estimator.predict(X)
    my_custom_score = my_custom_function(y_pred, y)
    return my_custom_score
```

A custom correlation function for scikit-learn

```
def my_pearsonr(est, X, y):
    # Generate predictions and convert to a vector
    y_pred = est.predict(X).squeeze()

    # Use the numpy "corrcoef" function to calculate a correlation matrix
    my_corrcoef_matrix = np.corrcoef(y_pred, y.squeeze())

    # Return a single correlation value from the matrix
    my_corrcoef = my_corrcoef[1, 0]
    return my_corrcoef
```

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Let's practice!

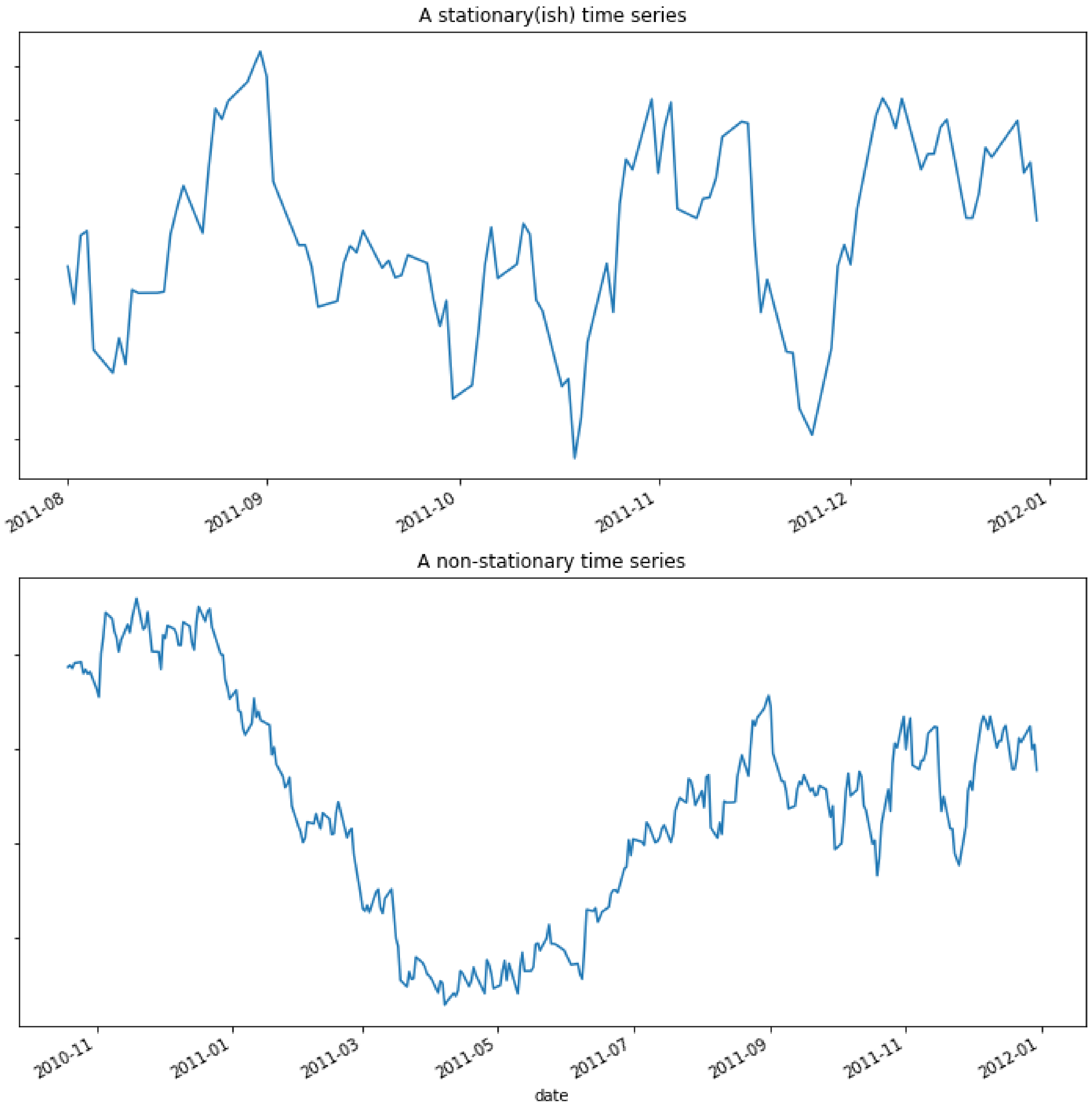
MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Stationarity and stability

Chris Holdgraf
Fellow, Berkeley Institute for Data Science

Stationarity

- Stationary time series do not change their statistical properties over time
- E.g., mean, standard deviation, trends
- Most time series are non-stationary to some extent


A stationary(ish) time series
2011-08
2011-09
2011-10
2011-11
2011-12
2012-01
A non-stationary time series
2010-11
2011-01
2011-03
2011-05
2011-07
2011-09
2011-11
2012-01
date

Model stability

- Non-stationary data results in variability in our model
- The statistical properties the model finds may change with the data
- In addition, we will be less certain about the correct values of model parameters
- How can we quantify this?

Cross validation to quantify parameter stability

- One approach: use cross-validation
- Calculate model parameters on each iteration
- Assess parameter stability across all CV splits

Bootstrapping the mean

- Bootstrapping is a common way to assess variability
- The bootstrap:
    1. Take a random sample of data **with replacement**
    2. Calculate the mean of the sample
    3. Repeat this process many times (1000s)
    4. Calculate the percentiles of the result (usually 2.5, 97.5)

The result is a *95% confidence interval* of the mean of each coefficient.

Bootstrapping the mean

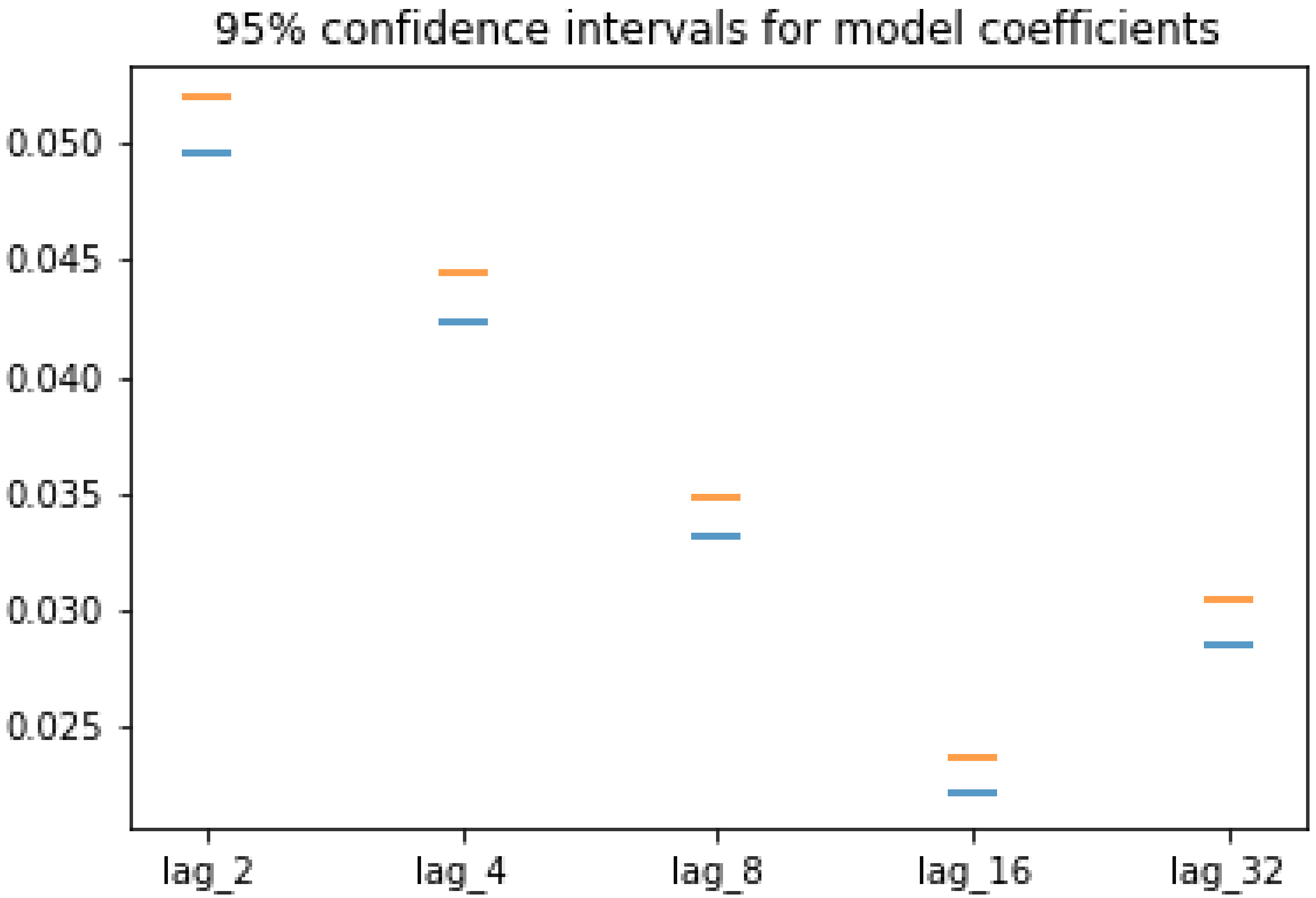
```
from sklearn.utils import resample

# cv_coefficients has shape (n_cv_folds, n_coefficients)
n_boots = 100
bootstrap_means = np.zeros(n_boots, n_coefficients)
for ii in range(n_boots):
    # Generate random indices for our data *with* replacement,
    # then take the sample mean
    random_sample = resample(cv_coefficients)
    bootstrap_means[ii] = random_sample.mean(axis=0)

# Compute the percentiles of choice for the bootstrapped means
percentiles = np.percentile(bootstrap_means, (2.5, 97.5), axis=0)
```

Plotting the bootstrapped coefficients

```
fig, ax = plt.subplots()
ax.scatter(many_shifts.columns, percentiles[0], marker='_', s=200)
ax.scatter(many_shifts.columns, percentiles[1], marker='_', s=200)
```



Assessing model performance stability

- If using the TimeSeriesSplit, can *plot* the model's score over time.
- This is useful in finding certain regions of time that hurt the score
- Also useful to find non-stationary signals

Model performance over time

```
def my_corrcoef(est, X, y):
    """Return the correlation coefficient
    between model predictions and a validation set."""
    return np.corrcoef(y, est.predict(X))[1, 0]

# Grab the date of the first index of each validation set
first_indices = [data.index[tt[0]] for tr, tt in cv.split(X, y)]

# Calculate the CV scores and convert to a Pandas Series
cv_scores = cross_val_score(model, X, y, cv=cv, scoring=my_corrcoef)
cv_scores = pd.Series(cv_scores, index=first_indices)
```

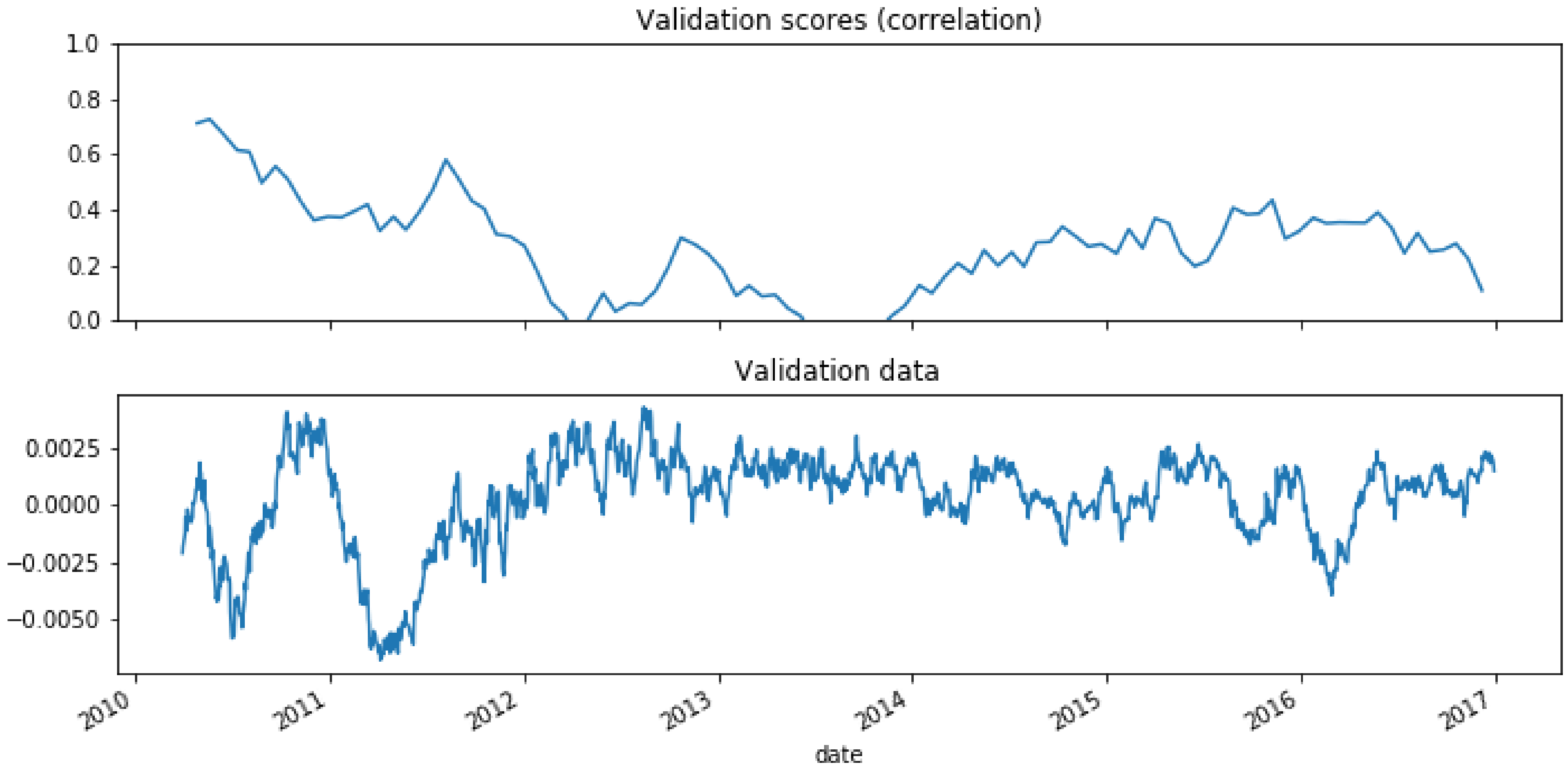
Visualizing model scores as a timeseries

```
fig, axs = plt.subplots(2, 1, figsize=(10, 5), sharex=True)

# Calculate a rolling mean of scores over time
cv_scores_mean = cv_scores.rolling(10, min_periods=1).mean()
cv_scores.plot(ax=axs[0])
axs[0].set(title='Validation scores (correlation)', ylim=[0, 1])

# Plot the raw data
data.plot(ax=axs[1])
axs[1].set(title='Validation data')
```

Visualizing model scores

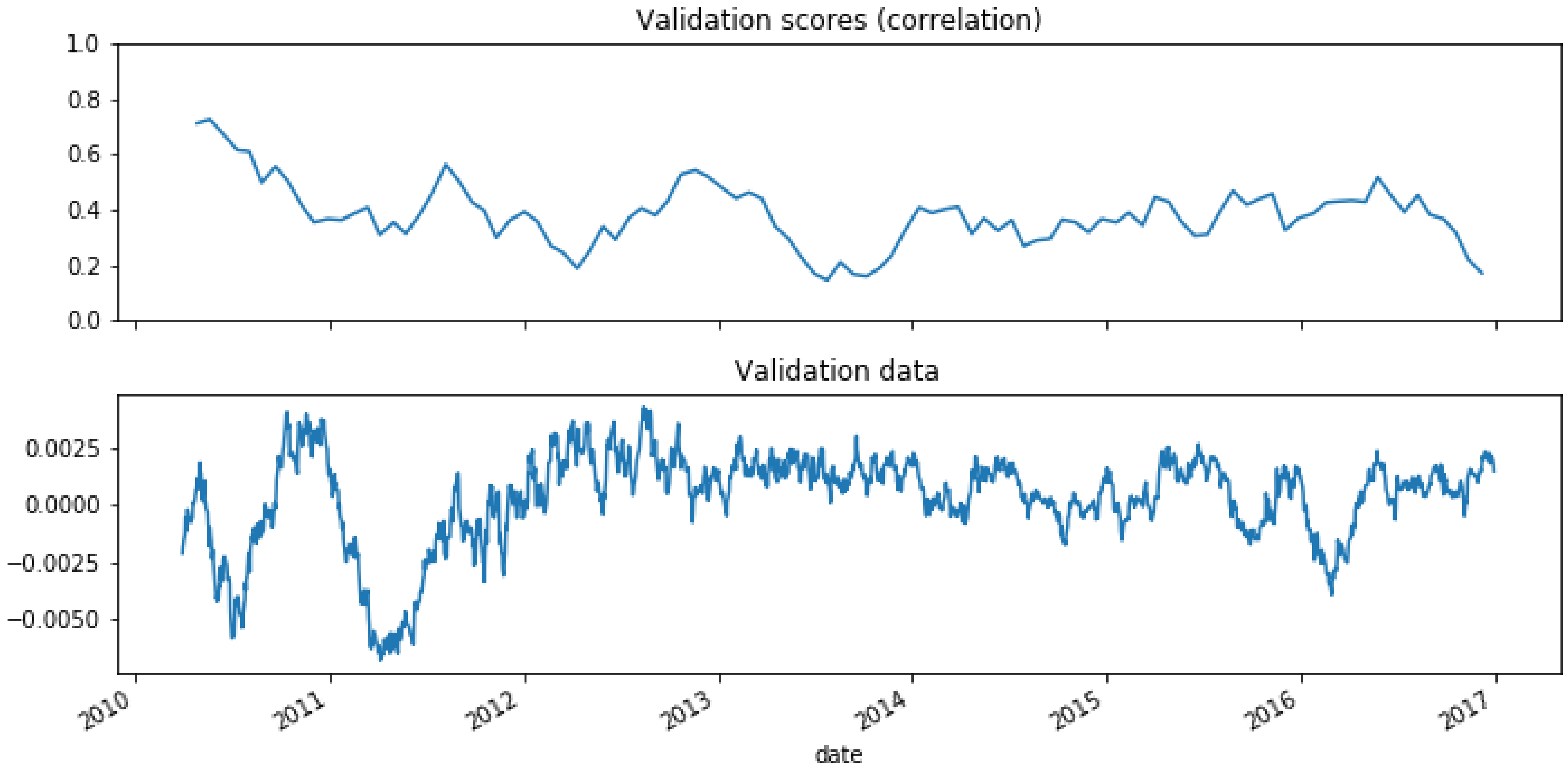


Fixed windows with time series cross-validation

```
# Only keep the last 100 datapoints in the training data
window = 100

# Initialize the CV with this window size
cv = TimeSeriesSplit(n_splits=10, max_train_size=window)
```

Non-stationary signals



MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Let's practice!

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Wrapping-up

Chris Holdgraf
Fellow, Berkeley Institute for Data Science

Timeseries and machine learning

- The many applications of time series + machine learning
- Always visualize your data first
- The scikit-learn API standardizes this process

Feature extraction and classification

- Summary statistics for time series classification
- Combining multiple features into a single input matrix
- Feature extraction for time series data

Model fitting and improving data quality

- Time series features for regression
- Generating predictions over time
- Cleaning and improving time series data

Validating and assessing our model performance

- Cross-validation with time series data (don't shuffle the data!)
- Time series stationarity
- Assessing model coefficient and score stability

Advanced concepts in time series

- Advanced window functions
- Signal processing and filtering details
- Spectral analysis

Advanced machine learning

- Advanced time series feature extraction (e.g., `tsfresh`)
- More complex model architectures for regression and classification
- Production-ready pipelines for time series analysis

Ways to practice

- There are a lot of opportunities to practice your skills with time series data.
- Kaggle has a number of time series predictions challenges
- Quantopian is also useful for learning and using predictive models others have built.