

# Timeseries kinds and applications

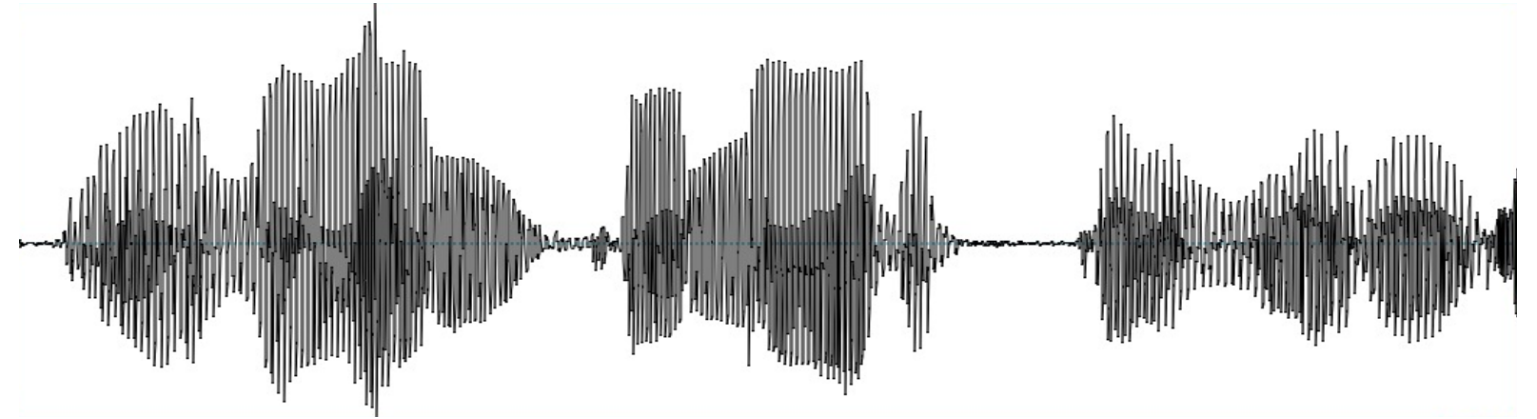
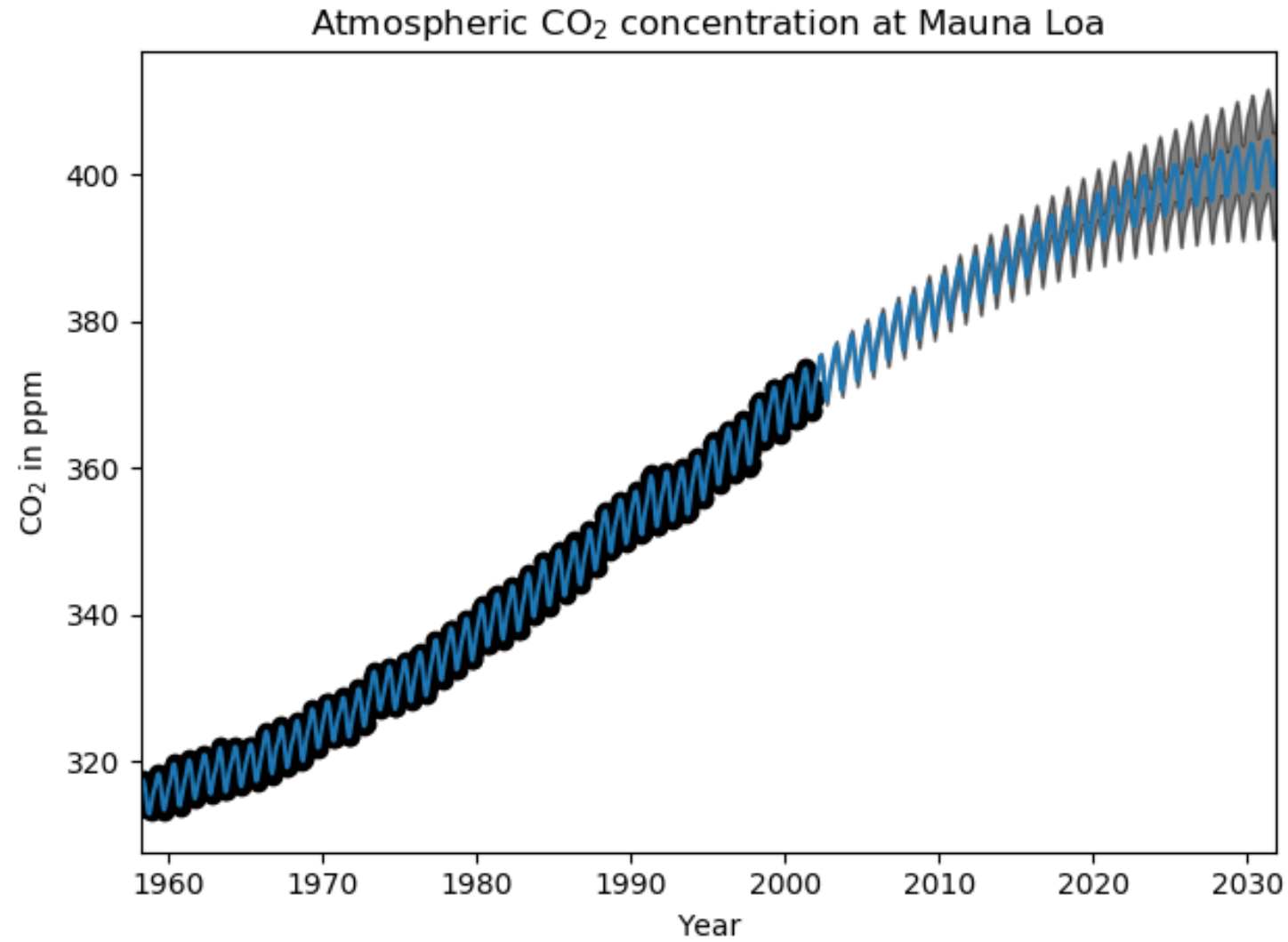
MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON



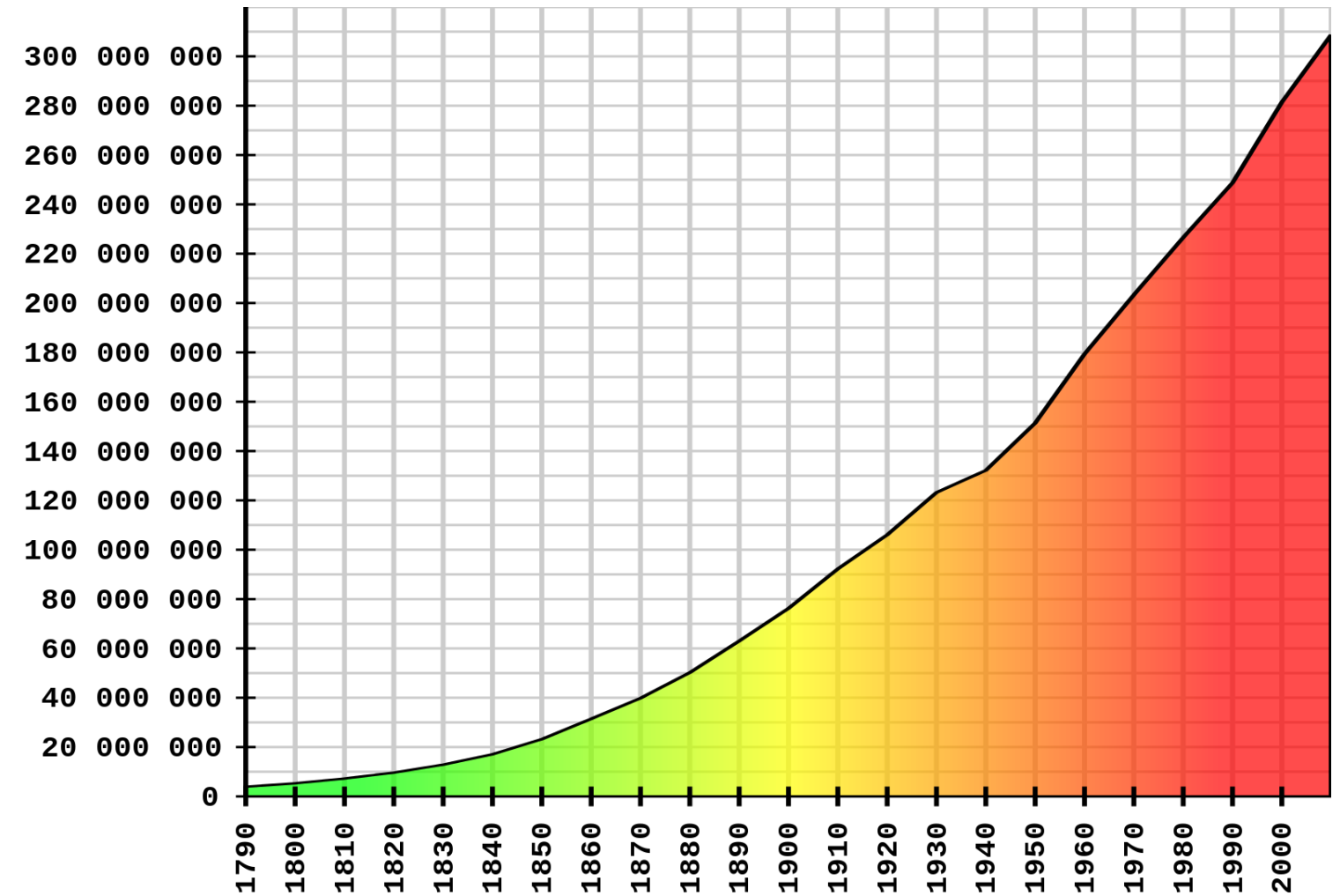
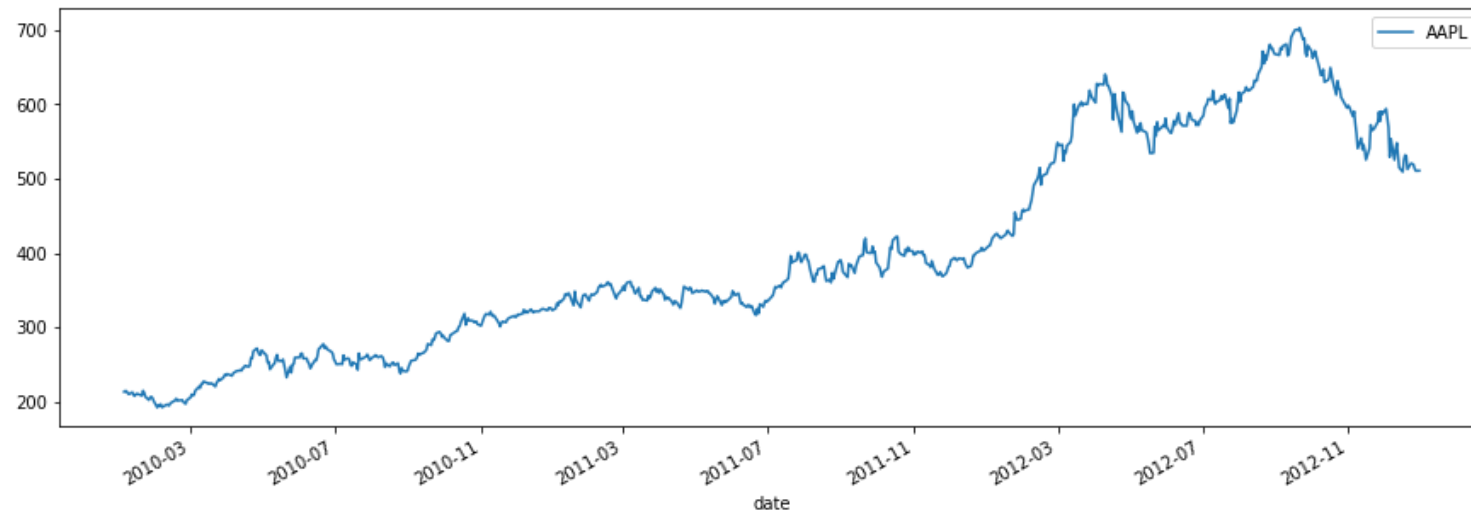
**Chris Holdgraf**

Fellow, Berkeley Institute for Data Science

# Time Series



# Time Series



# What makes a time series?

| Datapoint | Datapoint | Datapoint | Datapoint | Datapoint | Datapoint |
|-----------|-----------|-----------|-----------|-----------|-----------|
| 1         | 34        | 12        | 54        | 76        | 40        |

| Timepoint | Timepoint | Timepoint | Timepoint | Timepoint | Timepoint |
|-----------|-----------|-----------|-----------|-----------|-----------|
| 2:00      | 2:01      | 2:02      | 2:03      | 2:04      | 2:05      |

| Timepoint | Timepoint | Timepoint | Timepoint | Timepoint | Timepoint |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Jan       | Feb       | March     | April     | May       | Jun       |

| Timepoint | Timepoint | Timepoint | Timepoint | Timepoint | Timepoint |
|-----------|-----------|-----------|-----------|-----------|-----------|
|-----------|-----------|-----------|-----------|-----------|-----------|

# Reading in a time series with Pandas

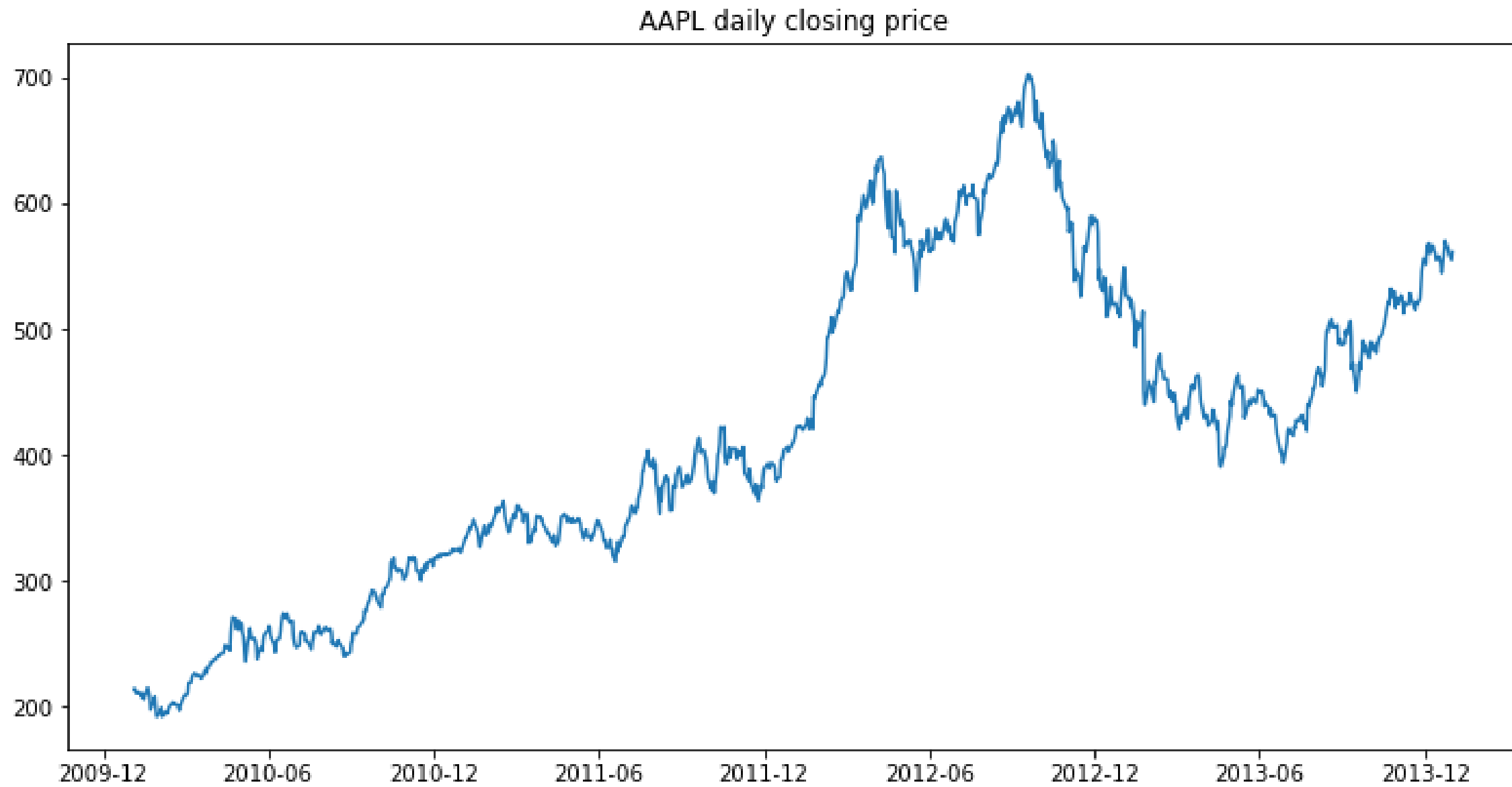
```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv('data.csv')
data.head()
```

|     | date       | symbol | close      | volume      |
|-----|------------|--------|------------|-------------|
| 0   | 2010-01-04 | AAPL   | 214.009998 | 123432400.0 |
| 46  | 2010-01-05 | AAPL   | 214.379993 | 150476200.0 |
| 92  | 2010-01-06 | AAPL   | 210.969995 | 138040000.0 |
| 138 | 2010-01-07 | AAPL   | 210.580000 | 119282800.0 |
| 184 | 2010-01-08 | AAPL   | 211.980005 | 111902700.0 |

# Plotting a pandas timeseries

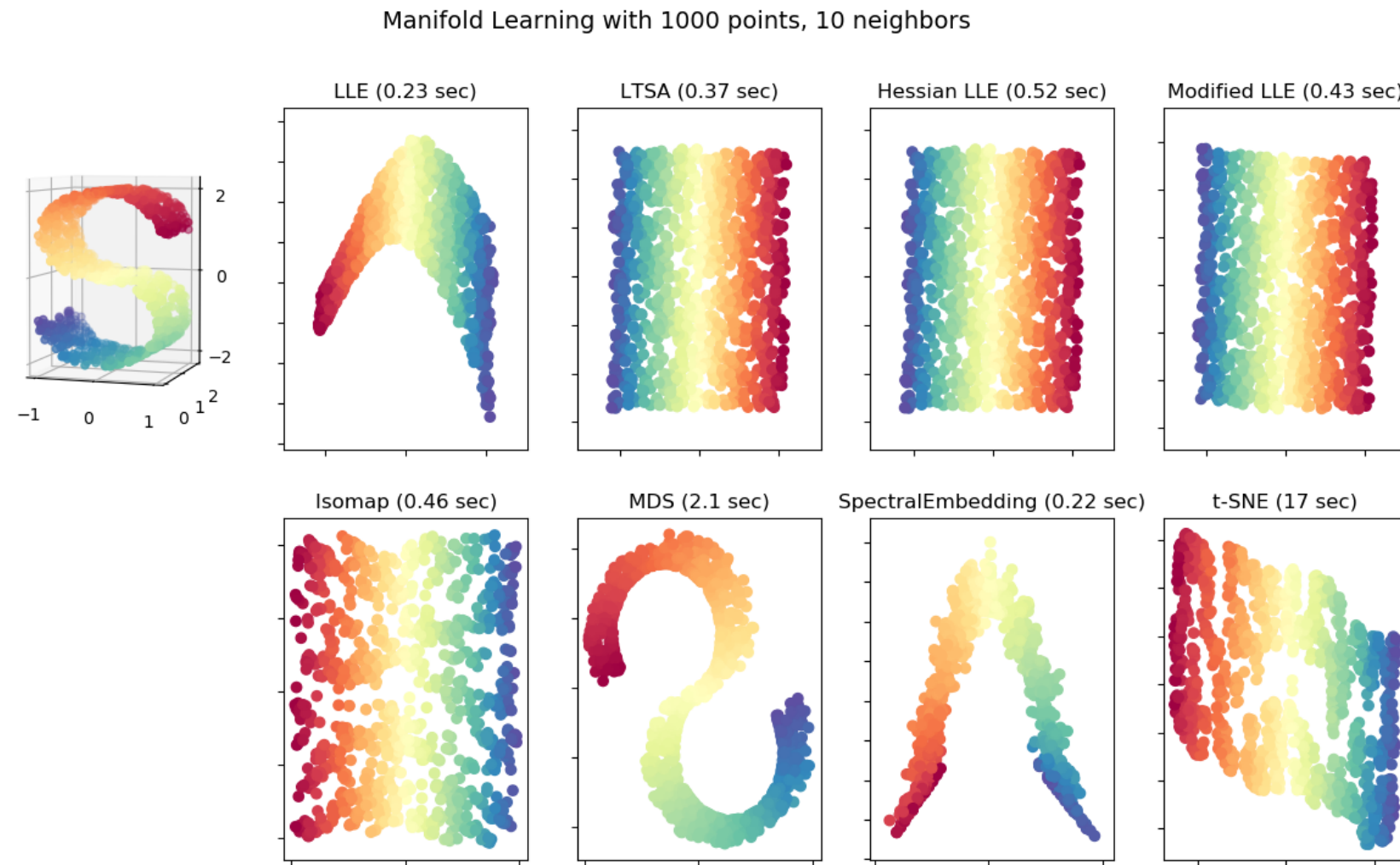
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 6))
data.plot('date', 'close', ax=ax)
ax.set(title="AAPL daily closing price")
```

# A timeseries plot



# Why machine learning?

We can use really big data and really complicated data

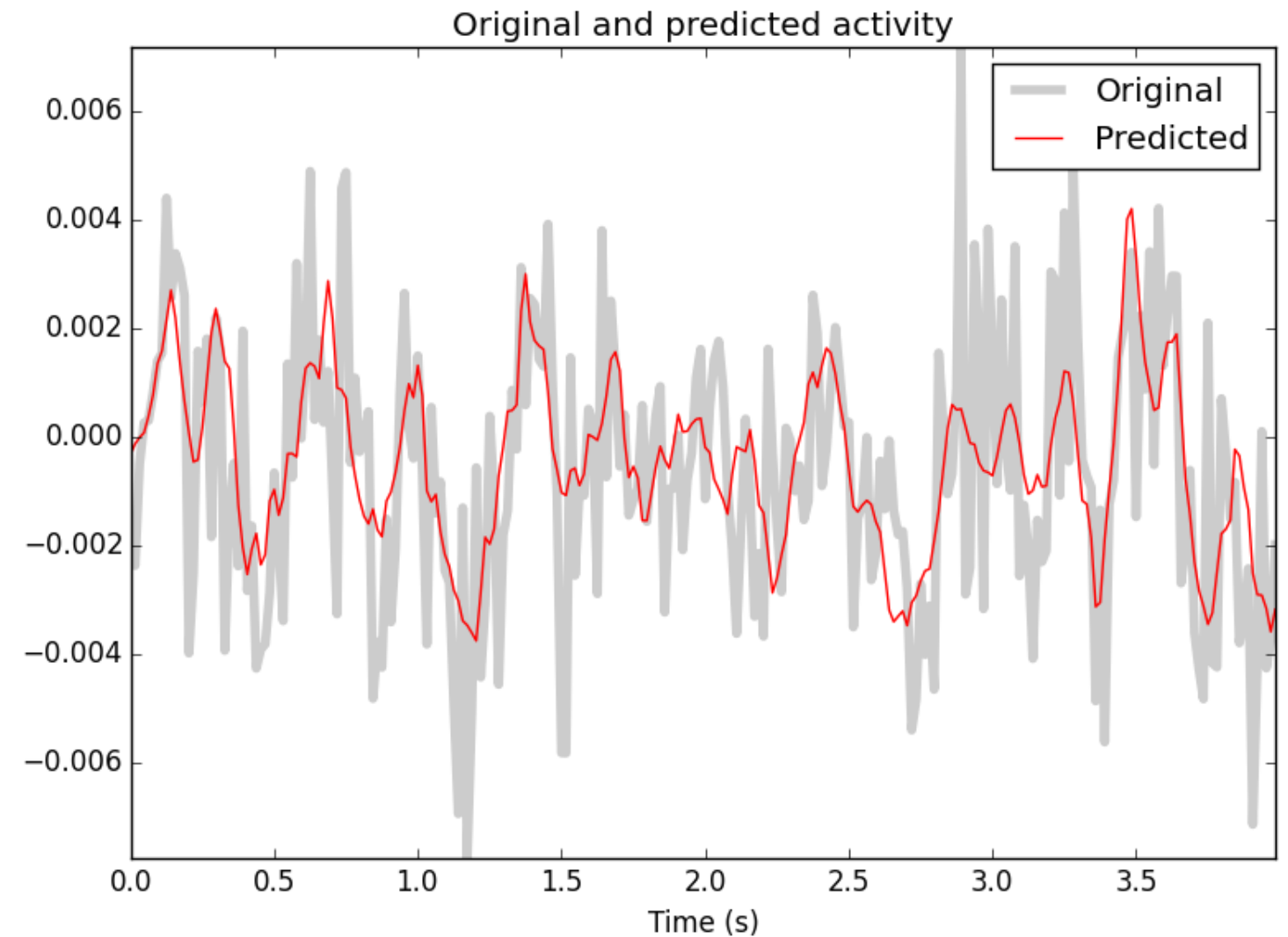




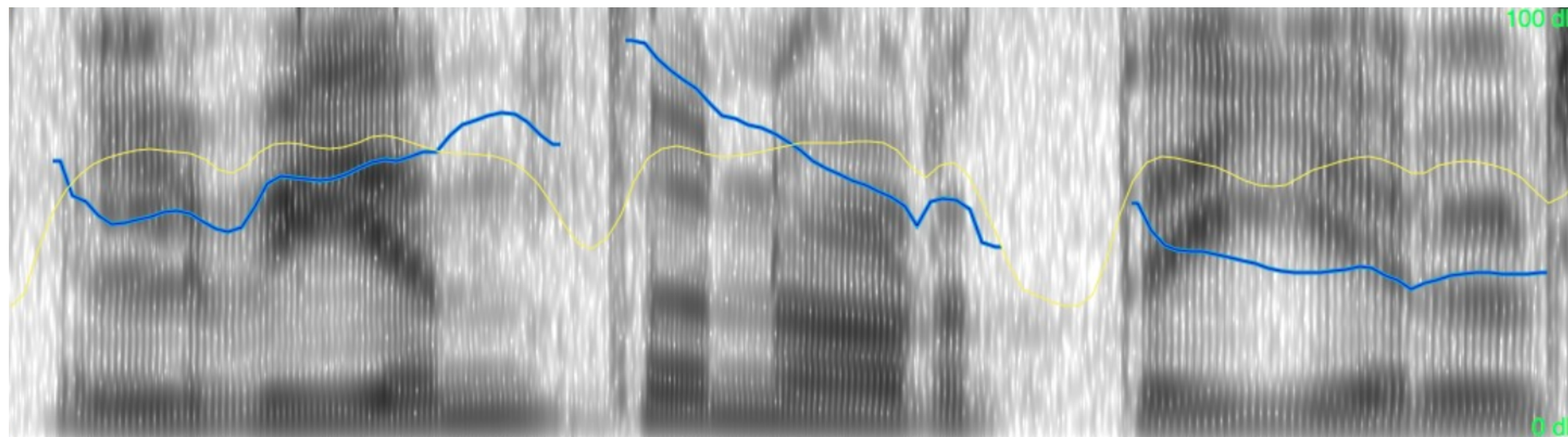
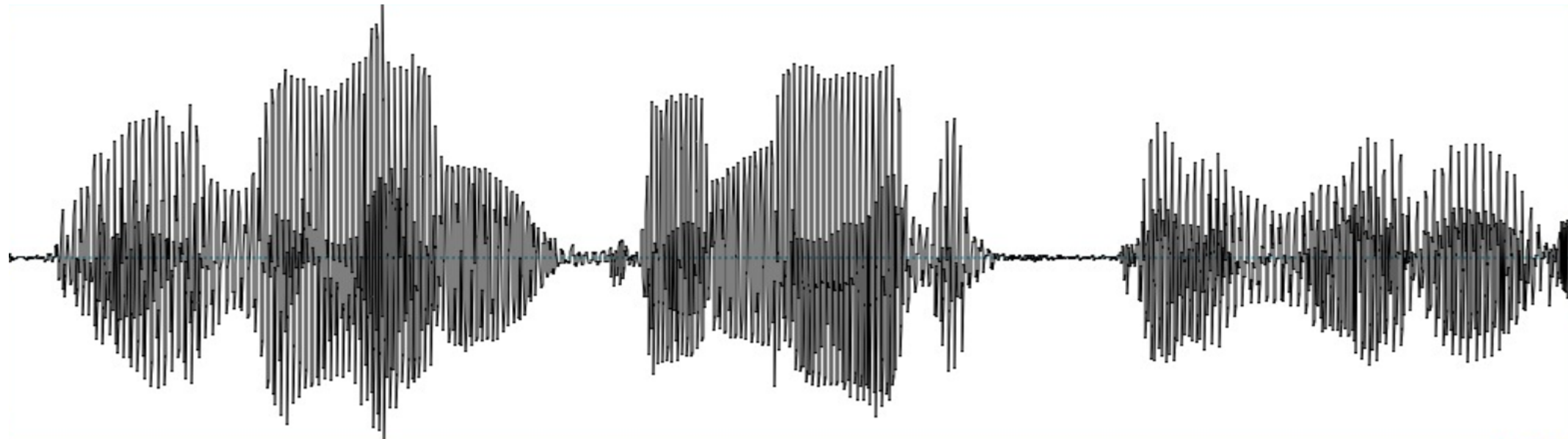
# Why machine learning?

We can...

- Predict the future
- Automate this process



# Why combine these two?



# A machine learning pipeline

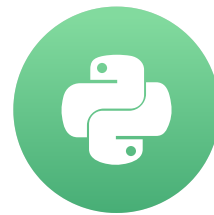
- Feature extraction
- Model fitting
- Prediction and validation

# Let's practice!

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

# Machine learning basics

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON



**Chris Holdgraf**

Fellow, Berkeley Institute for Data Science

# Always begin by looking at your data

```
array.shape
```

```
(10, 5)
```

```
array[:3]
```

```
array([[ 0.735528,  1.00122818, -0.28315978],  
       [-0.94478393,  0.18658748, -0.00241224],  
       [-0.74822942, -1.46636618,  0.69835096]])
```

# Always begin by looking at your data

```
df.head()
```

```
      col1      col2      col3
0  0.735528  1.001228 -0.283160
1 -0.944784  0.186587 -0.002412
2 -0.748229 -1.466366  0.698351
3  1.038589 -0.171248  0.831457
4 -0.161904  0.003972 -0.321933
```

# Always visualize your data

Make sure it looks the way you'd expect.

```
# Using matplotlib
fig, ax = plt.subplots()
ax.plot(...)

# Using pandas
fig, ax = plt.subplots()
df.plot(..., ax=ax)
```



# Scikit-learn

**Scikit-learn** is the most popular machine learning library in Python

```
from sklearn.svm import LinearSVC
```

# Preparing data for scikit-learn

- `scikit-learn` expects a particular structure of data:

**(samples, features)**

- Make sure that your data is *at least two-dimensional*
- Make sure the first dimension is *samples*

# If your data is not shaped properly

- If the axes are swapped:

```
array.T.shape
```

```
(10, 3)
```

# If your data is not shaped properly

- If we're missing an axis, use `.reshape()` :

```
array.shape
```

```
(10,)
```

```
array.reshape([-1, 1]).shape
```

```
(10, 1)
```

- `-1` will automatically fill that axis with remaining values

# Fitting a model with scikit-learn

```
# There is one coefficient per input feature  
model.coef_
```

```
array([[ 0.69417875, -0.5289162 ]])
```

# Investigating the model

```
# There is one coefficient per input feature  
model.coef_
```

```
array([[ 0.69417875, -0.5289162 ]])
```

# Predicting with a fit model

```
# Generate predictions  
predictions = model.predict(X_test)
```

# Let's practice

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

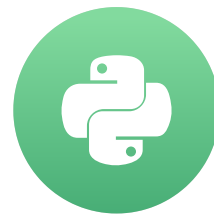


# Combining timeseries data with machine learning

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

**Chris Holdgraf**

Fellow, Berkeley Institute for Data Science



# Getting to know our data

- The datasets that we'll use in this course are all freely-available online
- There are many datasets available to download on the web, the ones we'll use come from Kaggle

# The Heartbeat Acoustic Data

- Many recordings of heart sounds from different patients
- Some had normally-functioning hearts, others had abnormalities
- Data comes in the form of audio files + labels for each file
- Can we find the "abnormal" heart beats?

# Loading auditory data

```
from glob import glob
files = glob('data/heartbeat-sounds/files/*.wav')

print(files)
```

```
['data/heartbeat-sounds/proc/files/murmur__201101051104.wav',
 ...
 'data/heartbeat-sounds/proc/files/murmur__201101051114.wav']
```

# Reading in auditory data

```
import librosa as lr
# `load` accepts a path to an audio file
audio, sfreq = lr.load('data/heartbeat-sounds/proc/files/murmur__201101051104.wav')

print(sfreq)
```

2205

In this case, the sampling frequency is 2205 , meaning there are 2205 samples per second

# Inferring time from samples

- If we know the sampling rate of a timeseries, then we know the timestamp of each datapoint *relative to the first datapoint*
- Note: this assumes the sampling rate is fixed and no data points are lost

# Creating a time array (I)

- Create an array of indices, one for each sample, and divide by the sampling frequency

```
indices = np.arange(0, len(audio))  
time = indices / sfreq
```

# Creating a time array (II)

- Find the time stamp for the  $N-1$ th data point. Then use `linspace()` to interpolate from zero to that time

```
final_time = (len(audio) - 1) / sfreq  
time = np.linspace(0, final_time, sfreq)
```



# The New York Stock Exchange dataset

- This dataset consists of company stock values for 10 years
- Can we detect any patterns in historical records that allow us to predict the value of companies in the future?

# Looking at the data

```
data = pd.read_csv('path/to/data.csv')
```

```
data.columns
```

```
Index(['date', 'symbol', 'close', 'volume'], dtype='object')
```

```
data.head()
```

|   | date       | symbol | close      | volume      |
|---|------------|--------|------------|-------------|
| 0 | 2010-01-04 | AAPL   | 214.009998 | 123432400.0 |
| 1 | 2010-01-04 | ABT    | 54.459951  | 10829000.0  |
| 2 | 2010-01-04 | AIG    | 29.889999  | 7750900.0   |
| 3 | 2010-01-04 | AMAT   | 14.300000  | 18615100.0  |
| 4 | 2010-01-04 | ARNC   | 16.650013  | 11512100.0  |

# Timeseries with Pandas DataFrames

- We can investigate the object type of each column by accessing the `dtypes` attribute

```
df['date'].dtypes
```

```
0    object
1    object
2    object
dtype: object
```

# Converting a column to a time series

- To ensure that a column within a DataFrame is treated as time series, use the `to_datetime()` function

```
df['date'] = pd.to_datetime(df['date'])
```

```
df['date']
```

```
0    2017-01-01
```

```
1    2017-01-02
```

```
2    2017-01-03
```

```
Name: date, dtype: datetime64[ns]
```

# Let's practice!

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON