Welcome to the course!

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

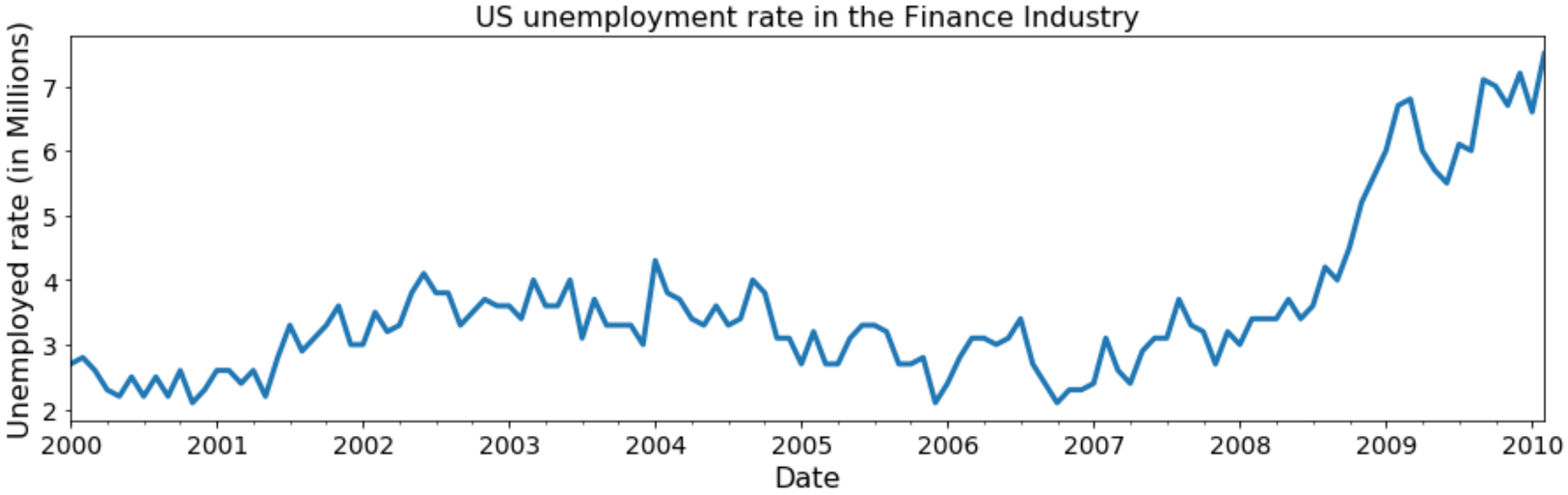
Prerequisites

- Intro to Python for Data Science
- Intermediate Python for Data Science

Time series in the field of Data Science

- Time series are a fundamental way to store and analyze many types of data
- Financial, weather and device data are all best handled as time series

Time series in the field of Data Science



Course overview

- Chapter 1: Getting started and personalizing your first time series plot
- Chapter 2: Summarizing and describing time series data
- Chapter 3: Advanced time series analysis
- Chapter 4: Working with multiple time series
- Chapter 5: Case Study

Reading data with Pandas

```
import pandas as pd
df = pd.read_csv('ch2_co2_levels.csv')
print(df)
```

```
      datestamp    co2
0     1958-03-29  316.1
1     1958-04-05  317.3
2     1958-04-12  317.6
...
...
...
2281  2001-12-15  371.2
2282  2001-12-22  371.3
2283  2001-12-29  371.5
```

Preview data with Pandas

```
print(df.head(n=5))
```

```
    datestamp    co2
0  1958-03-29  316.1
1  1958-04-05  317.3
2  1958-04-12  317.6
3  1958-04-19  317.5
4  1958-04-26  316.4
```

```
print(df.tail(n=5))
```

```
       datestamp    co2
2279  2001-12-01  370.3
2280  2001-12-08  370.8
2281  2001-12-15  371.2
2282  2001-12-22  371.3
2283  2001-12-29  371.5
```

Check data types with Pandas

```
print(df.dtypes)
```

```
datestamp      object
co2           float64
dtype: object
```

Working with dates

To work with time series data in `pandas`, your date columns needs to be of the `datetime64` type.

```
pd.to_datetime(['2009/07/31', 'test'])
```

```
ValueError: Unknown string format
```

```
pd.to_datetime(['2009/07/31', 'test'], errors='coerce')
```

```
DatetimeIndex(['2009-07-31', 'NaT'],
    dtype='datetime64[ns]', freq=None)
```

Let's get started!

VISUALIZING TIME SERIES DATA IN PYTHON

Plot your first time series

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

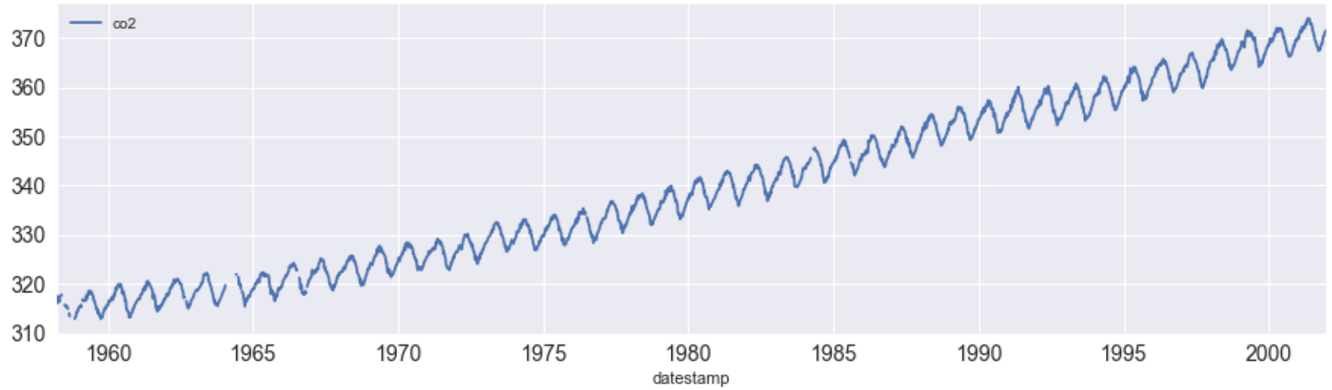
datacamp

The Matplotlib library

- In Python, matplotlib is an extensive package used to plot data
- The pyplot submodule of matplotlib is traditionally imported using the `plt` alias

```
import matplotlib.pyplot as plt
```

Plotting time series data



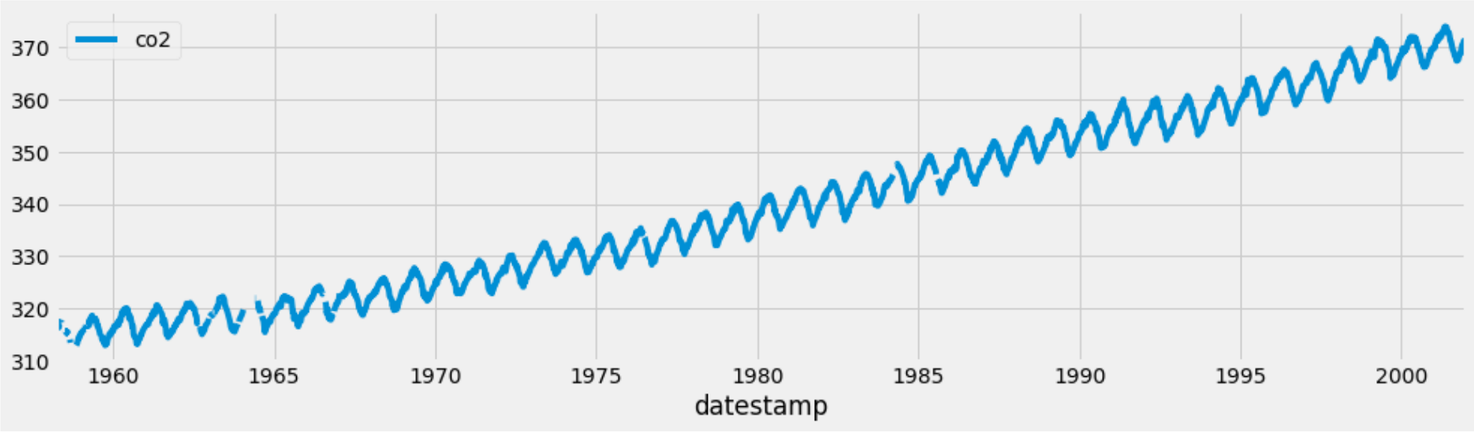
Plotting time series data

```
import matplotlib.pyplot as plt
import pandas as pd


df = df.set_index('date_column')
df.plot()
plt.show()
```

Adding style to your plots

```
plt.style.use('fivethirtyeight')
df.plot()
plt.show()
```

FiveThirtyEight style



Matplotlib style sheets

```
print(plt.style.available)
```

```
['seaborn-dark-palette', 'seaborn-darkgrid',
'seaborn-dark', 'seaborn-notebook',
'seaborn-pastel', 'seaborn-white',
'classic', 'ggplot', 'grayscale',
'dark_background', 'seaborn-poster',
'seaborn-muted', 'seaborn', 'bmh',
'seaborn-paper', 'seaborn-whitegrid',
'seaborn-bright', 'seaborn-talk',
'fivethirtyeight', 'seaborn-colorblind',
'seaborn-deep', 'seaborn-ticks']
```

Describing your graphs with labels

```
ax = df.plot(color='blue')
```

```
ax.set_xlabel('Date')
ax.set_ylabel('The values of my Y axis')
ax.set_title('The title of my plot')
plt.show()
```

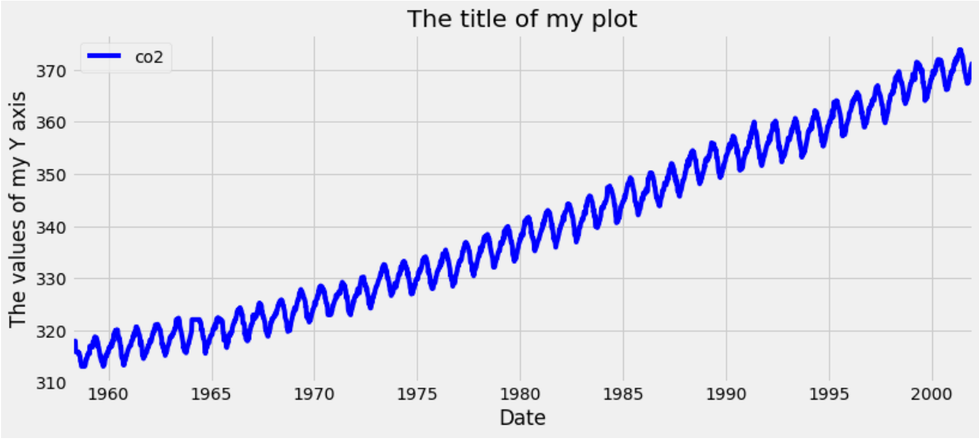
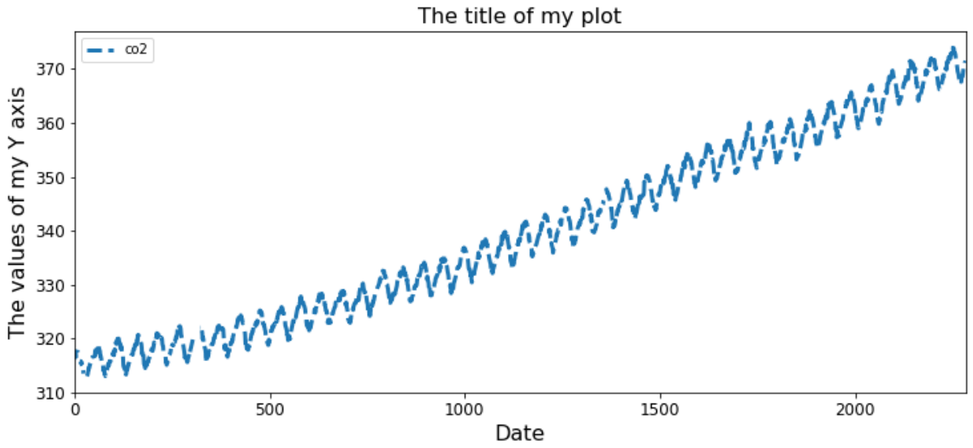


Figure size, linewidth, linestyle and fontsize

```
ax = df.plot(figsize=(12, 5), fontsize=12,
             linewidth=3, linestyle='--')
ax.set_xlabel('Date', fontsize=16)
ax.set_ylabel('The values of my Y axis', fontsize=16)
ax.set_title('The title of my plot', fontsize=16)
plt.show()
```



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Customize your time series plot

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

Slicing time series data

```
discoveries['1960':'1970']
```

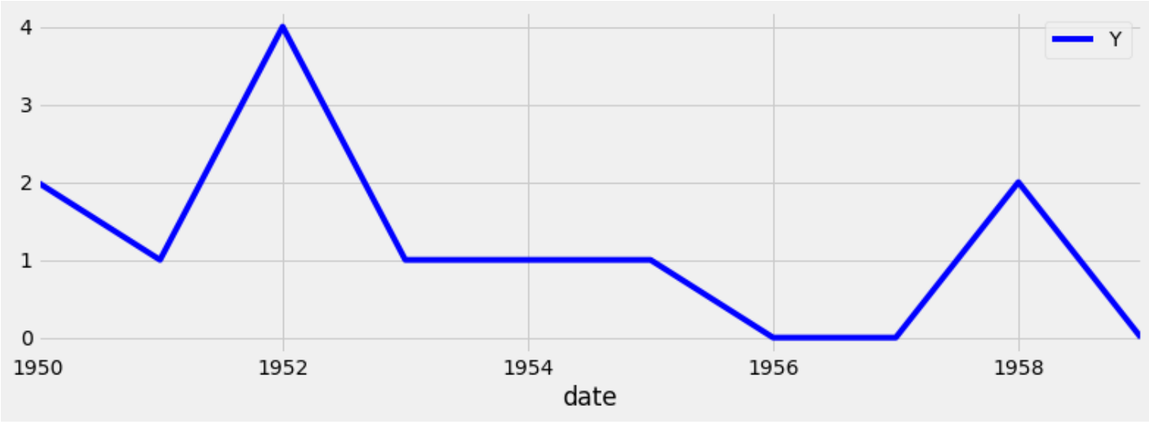
```
discoveries['1950-01':'1950-12']
```

```
discoveries['1960-01-01':'1960-01-15']
```

Plotting subset of your time series data

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
df_subset = discoveries['1960':'1970']
```

```
ax = df_subset.plot(color='blue', fontsize=14)
plt.show()
```



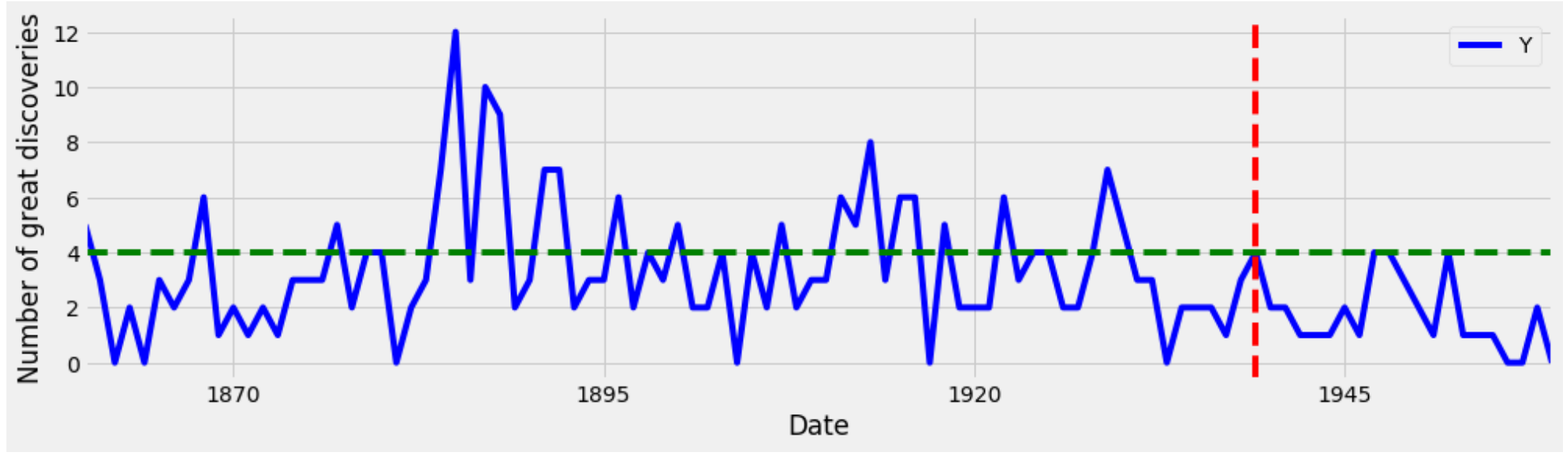
Adding markers

```
ax.axvline(x='1969-01-01',
           color='red',
           linestyle='--')
```

```
ax.axhline(y=100,
           color='green',
           linestyle='--')
```

Using markers: the full code

```
ax = discoveries.plot(color='blue')
ax.set_xlabel('Date')
ax.set_ylabel('Number of great discoveries')
ax.axvline('1969-01-01', color='red', linestyle='--')
ax.axhline(4, color='green', linestyle='--')
```



Highlighting regions of interest

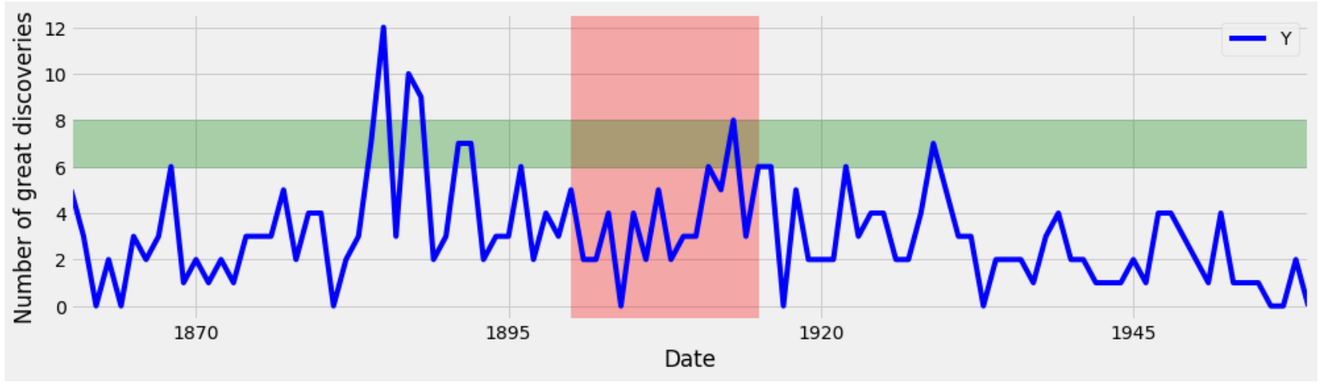
```
ax.axvspan('1964-01-01', '1968-01-01',
           color='red', alpha=0.5)
```

```
ax.axhspan(8, 6, color='green',
           alpha=0.2)
```

Highlighting regions of interest: the full code

```
ax = discoveries.plot(color='blue')
ax.set_xlabel('Date')
ax.set_ylabel('Number of great discoveries')
```

```
ax.axvspan('1964-01-01', '1968-01-01', color='red',
alpha=0.3)
ax.axhspan(8, 6, color='green', alpha=0.3)
```



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Clean your time series data

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

The CO2 level time series

A snippet of the weekly measurements of CO2 levels at the Mauna Loa Observatory, Hawaii.

```
datastamp    co2
1958-03-29  316.1
1958-04-05  317.3
1958-04-12  317.6
...
...
2001-12-15  371.2
2001-12-22  371.3
2001-12-29  371.5
```

Finding missing values in a DataFrame

```
print(df.isnull())
```

```
datestamp   co2
1958-03-29  False
1958-04-05  False
1958-04-12  False
```

```
print(df.notnull())
```

```
datestamp    co2
1958-03-29   True
1958-04-05   True
1958-04-12   True
...
```

Counting missing values in a DataFrame

```
print(df.isnull().sum())
```

```
datestamp     0
co2          59
dtype: int64
```

Replacing missing values in a DataFrame

```
print(df)
```

```
...
5  1958-05-03  316.9
6  1958-05-10    NaN
7  1958-05-17  317.5
...
```

```
df = df.fillna(method='bfill')
print(df)
```

```
...
5  1958-05-03  316.9
6  1958-05-10  317.5
7  1958-05-17  317.5
...
```

Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Plot aggregates of your data

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

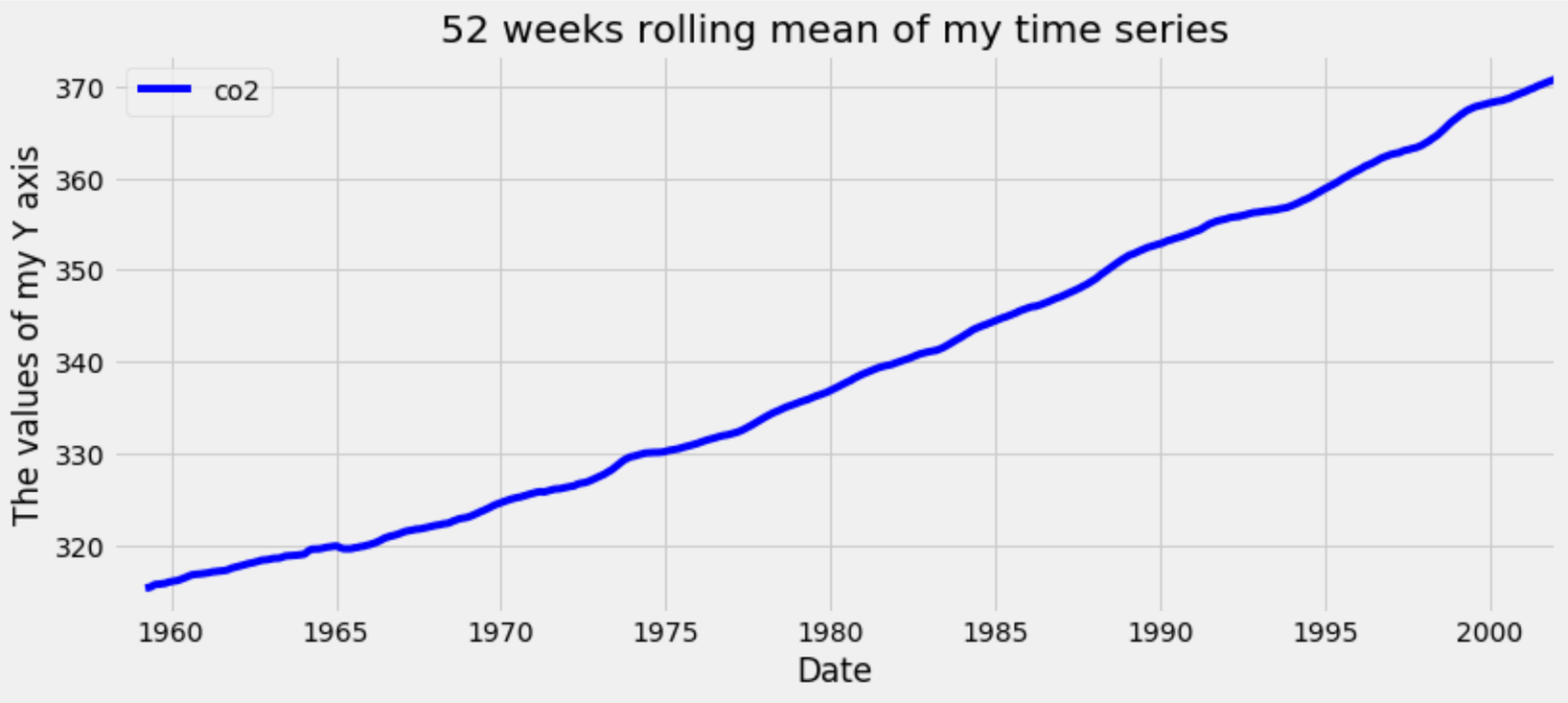
Moving averages

- In the field of time series analysis, a moving average can be used for many different purposes:
  - smoothing out short-term fluctuations
  - removing outliers
  - highlighting long-term trends or cycles.

The moving average model

```
co2_levels_mean = co2_levels.rolling(window=52).mean()


ax = co2_levels_mean.plot()
ax.set_xlabel("Date")
ax.set_ylabel("The values of my Y axis")
ax.set_title("52 weeks rolling mean of my time series")


plt.show()
```

A plot of the moving average for the CO2 data



Computing aggregate values of your time series

```
co2_levels.index
```

```
DatetimeIndex(['1958-03-29', '1958-04-05',...],
              dtype='datetime64[ns]', name='datestamp',
              length=2284, freq=None)
```

```
print(co2_levels.index.month)
```

```
array([ 3,  4,  4, ..., 12, 12, 12], dtype=int32)
```

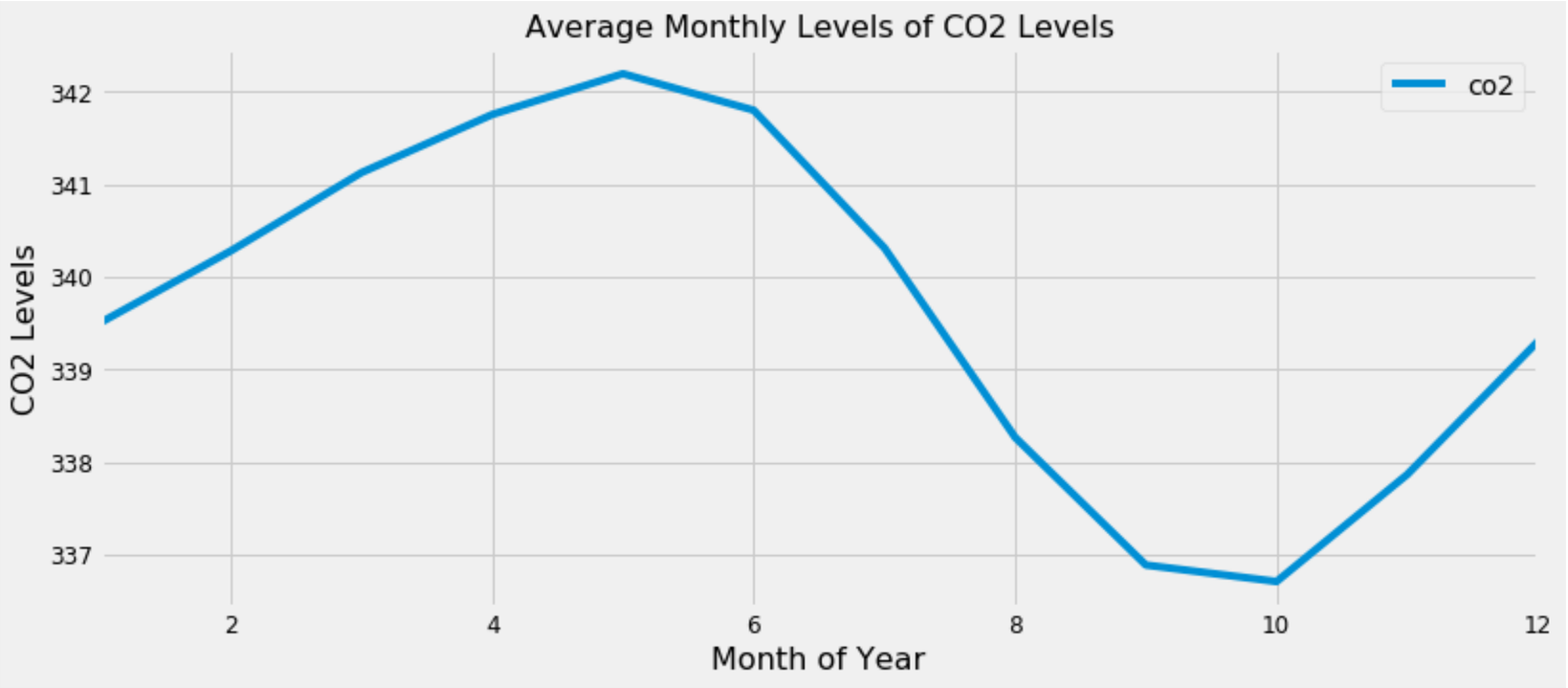
```
print(co2_levels.index.year)
```

```
array([1958, 1958, 1958, ..., 2001,
       2001, 2001], dtype=int32)
```

Plotting aggregate values of your time series

```
index_month = co2_levels.index.month
co2_levels_by_month = co2_levels.groupby(index_month).mean()
co2_levels_by_month.plot()

plt.show()
```

Plotting aggregate values of your time series



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Summarizing the values in your time series data

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

Obtaining numerical summaries of your data

- What is the average value of this data?
- What is the maximum value observed in this time series?

The `.describe()` method automatically computes key statistics of all numeric columns in your DataFrame

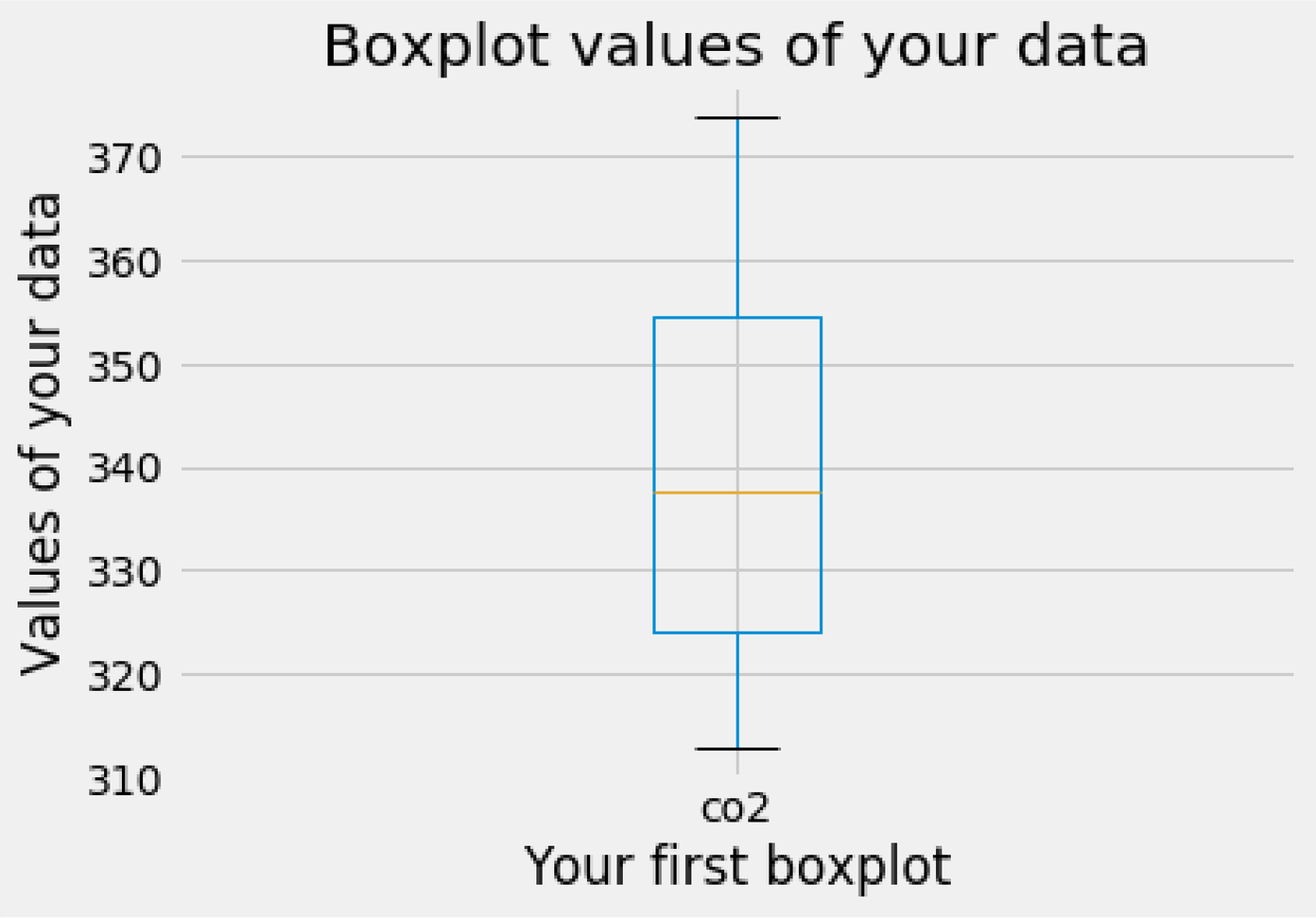
```
print(df.describe())
```

```
                co2
count  2284.000000
mean    339.657750
std      17.100899
min     313.000000
25%     323.975000
50%     337.700000
75%     354.500000
max     373.900000
```

Summarizing your data with boxplots

```
ax1 = df.boxplot()
ax1.set_xlabel('Your first boxplot')
ax1.set_ylabel('Values of your data')
ax1.set_title('Boxplot values of your data')


plt.show()
```

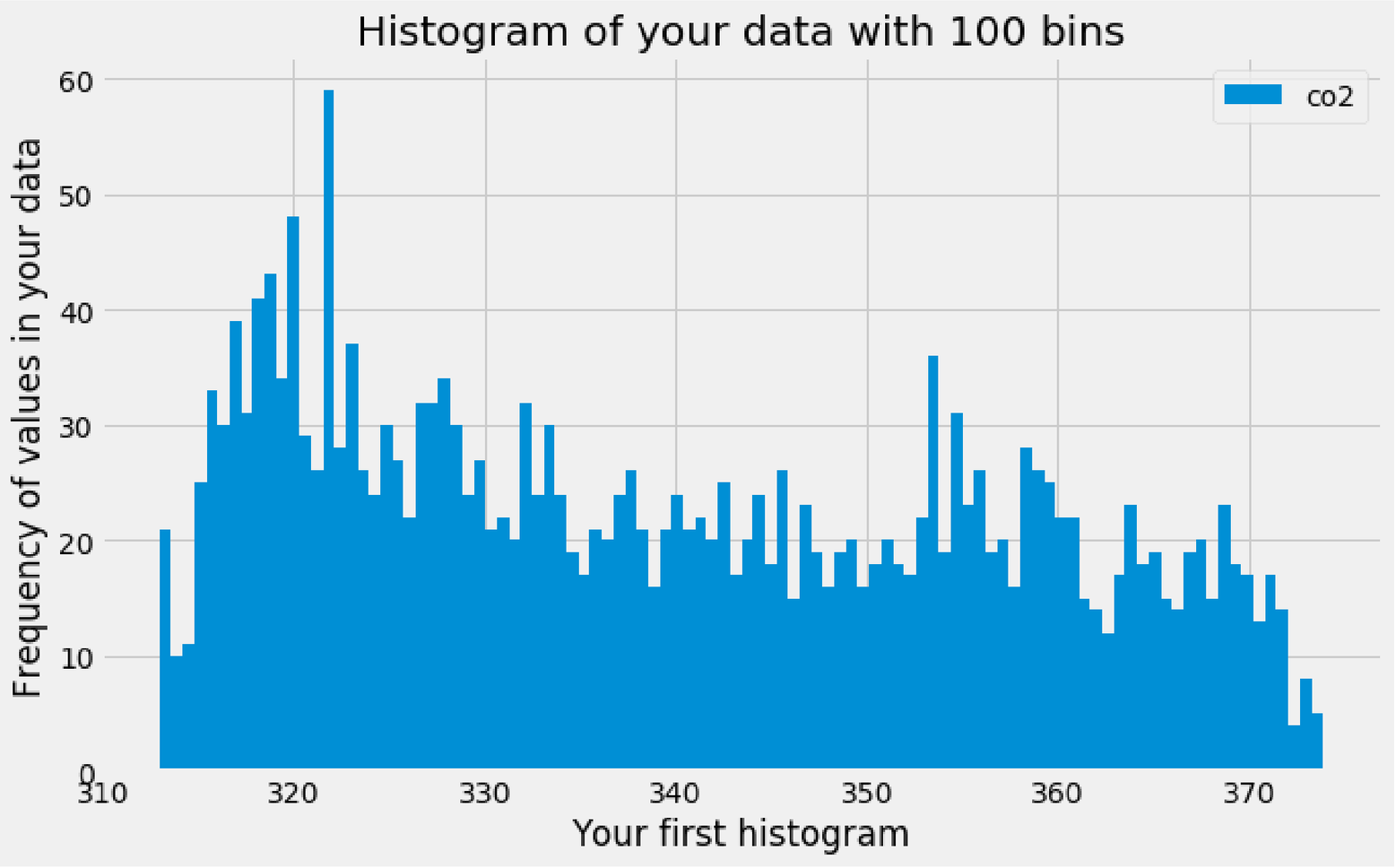
A boxplot of the values in the CO2 data



Summarizing your data with histograms

```
ax2 = df.plot(kind='hist', bins=100)
ax2.set_xlabel('Your first histogram')
ax2.set_ylabel('Frequency of values in your data')
ax2.set_title('Histogram of your data with 100 bins')

plt.show()
```

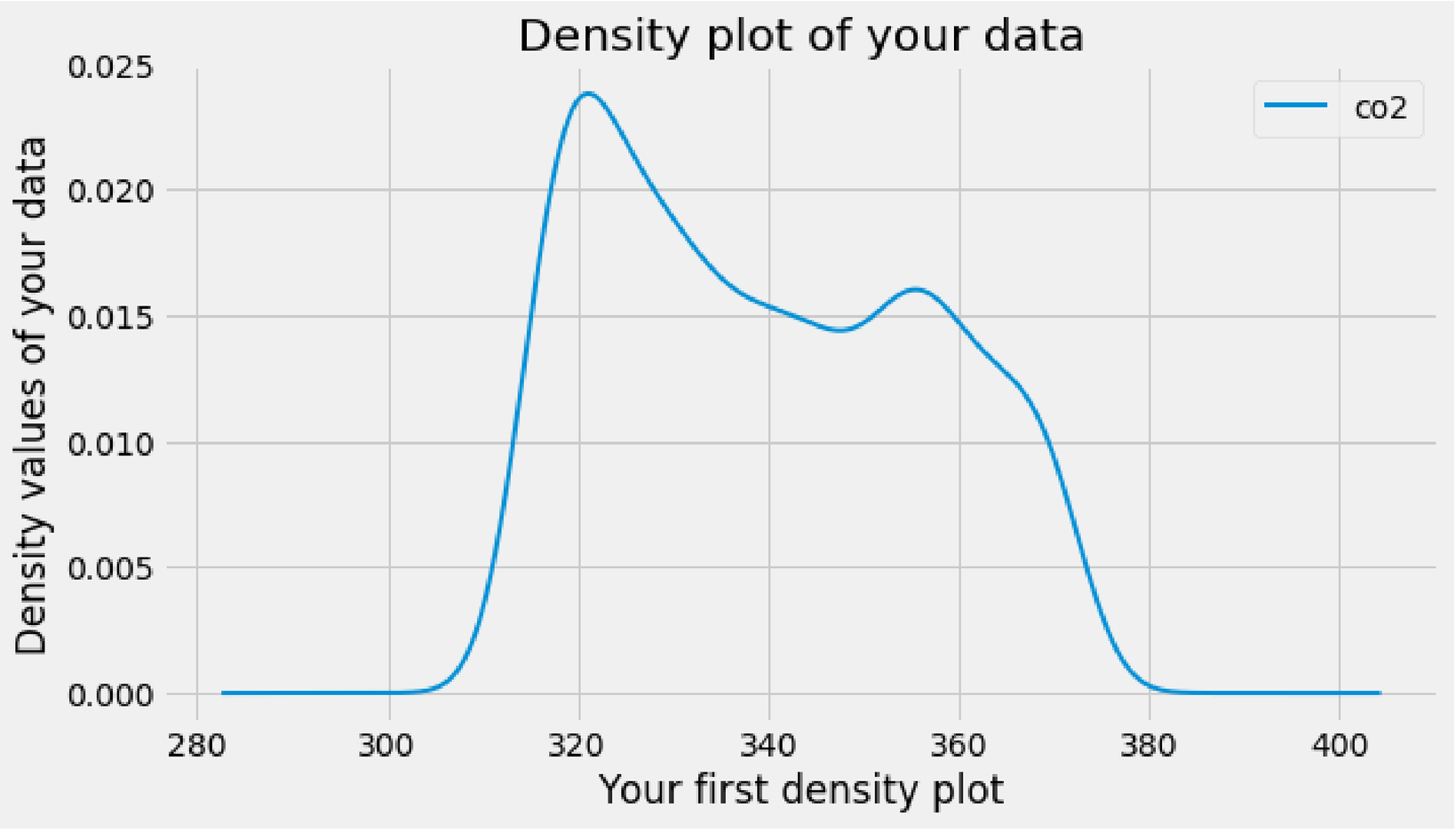
A histogram plot of the values in the CO2 data



Summarizing your data with density plots

```
ax3 = df.plot(kind='density', linewidth=2)
ax3.set_xlabel('Your first density plot')
ax3.set_ylabel('Density values of your data')
ax3.set_title('Density plot of your data')


plt.show()
```

A density plot of the values in the CO2 data



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Autocorrelation and Partial autocorrelation

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

Autocorrelation in time series data

- Autocorrelation is measured as the correlation between a time series and a delayed copy of itself
- For example, an autocorrelation of order 3 returns the correlation between a time series at points (`t_1`, `t_2`, `t_3`, ...) and its own values lagged by 3 time points, i.e. (`t_4`, `t_5`, `t_6`, ...)
- It is used to find repetitive patterns or periodic signal in time series

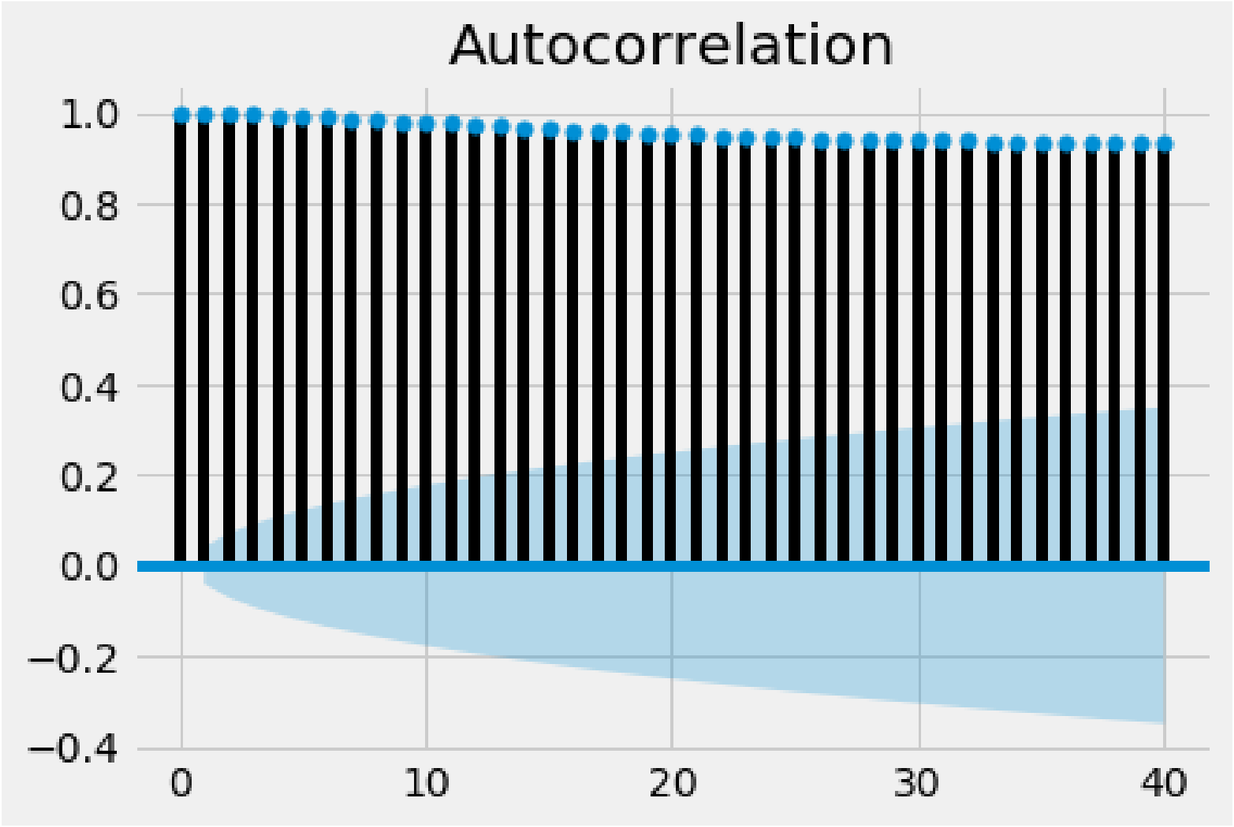
Statsmodels

> `statsmodels` is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.

Plotting autocorrelations

```
import matplotlib.pyplot as plt
from statsmodels.graphics import tsaplots
fig = tsaplots.plot_acf(co2_levels['co2'], lags=40)

plt.show()
```

Interpreting autocorrelation plots



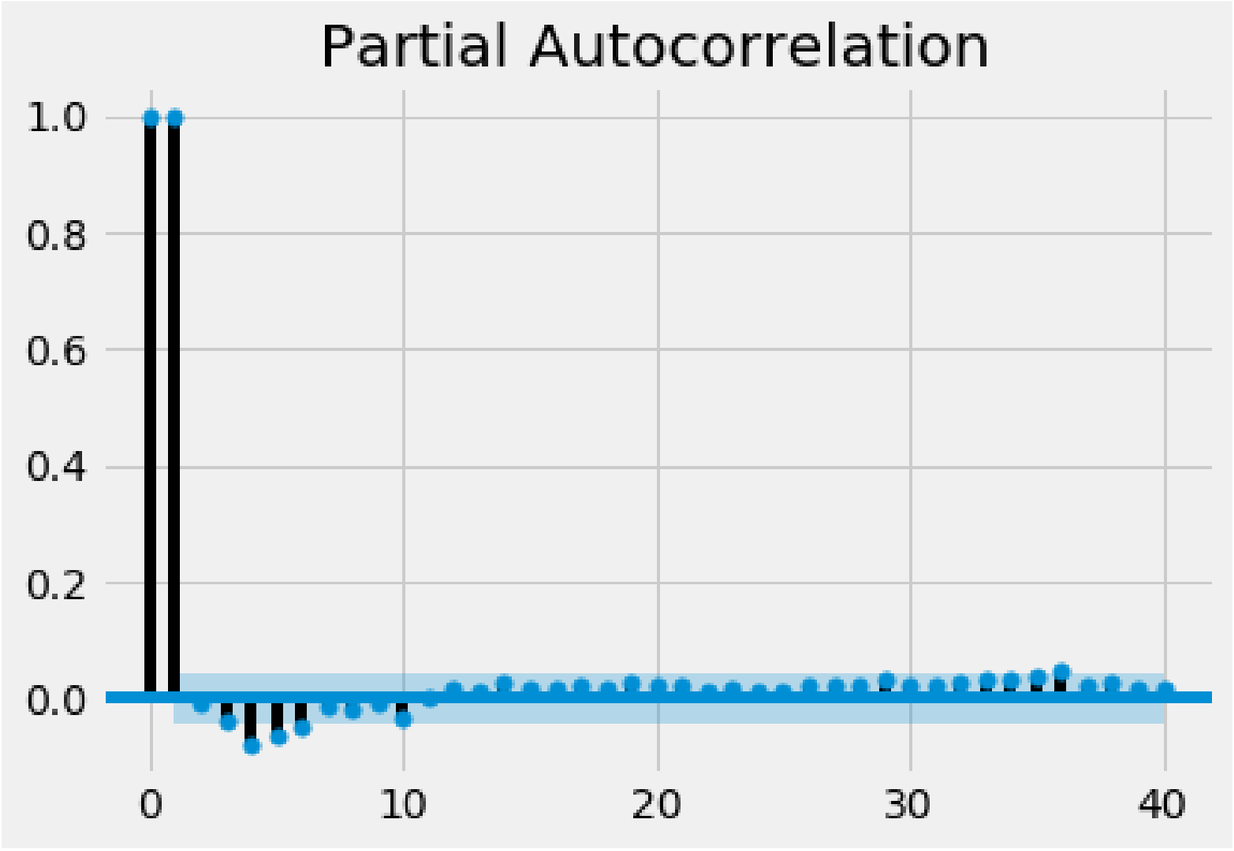
Partial autocorrelation in time series data

- Contrary to autocorrelation, partial autocorrelation removes the effect of previous time points
- For example, a partial autocorrelation function of `order 3` returns the correlation between our time series (`t1`, `t2`, `t3`, ...) and lagged values of itself by 3 time points (`t4`, `t5`, `t6`, ...), but only after removing all effects attributable to lags 1 and 2

Plotting partial autocorrelations

```
import matplotlib.pyplot as plt


from statsmodels.graphics import tsaplots
fig = tsaplots.plot_pacf(co2_levels['co2'], lags=40)


plt.show()
```

Interpreting partial autocorrelations plot



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

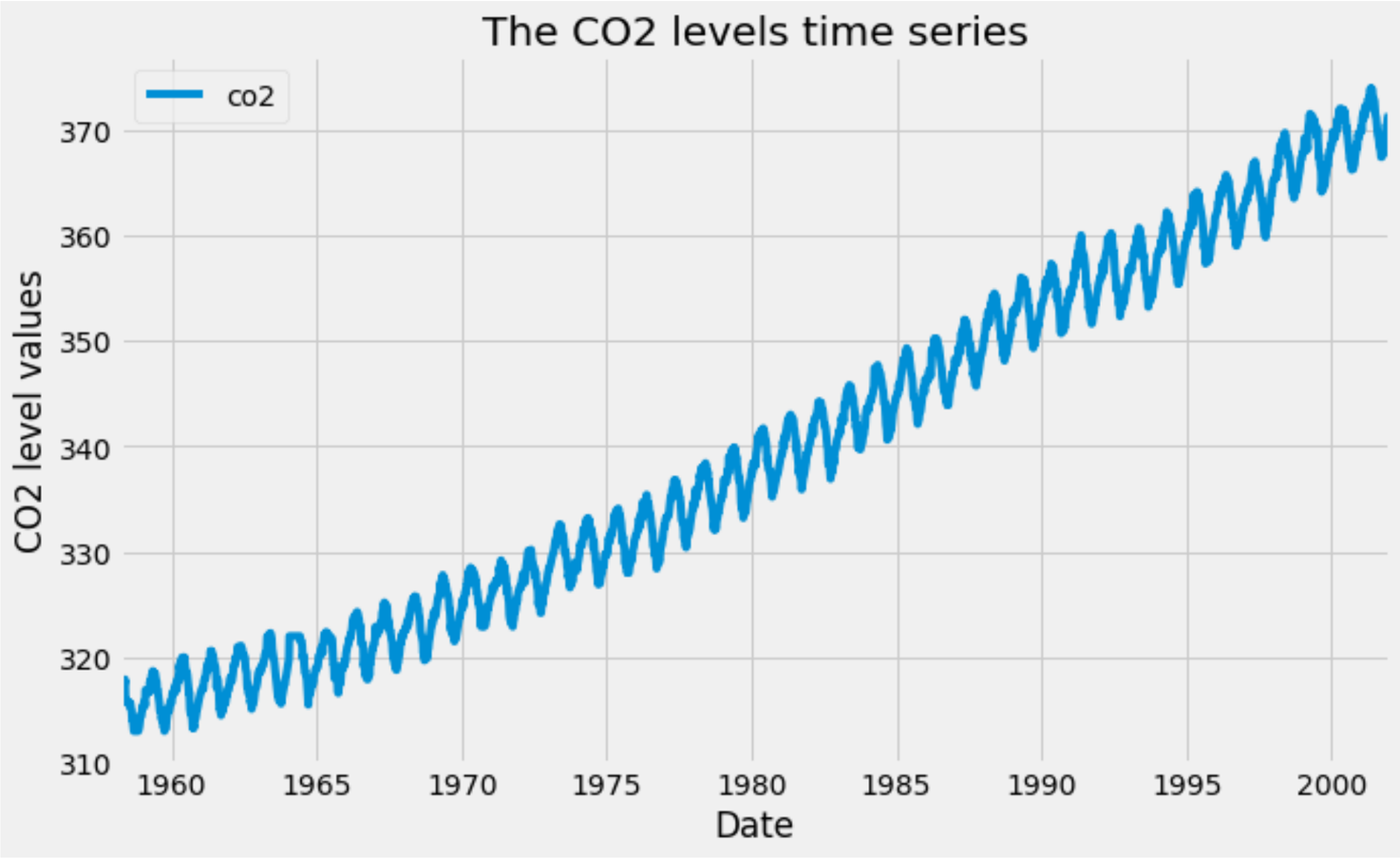
Seasonality, trend and noise in time series data

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

Properties of time series



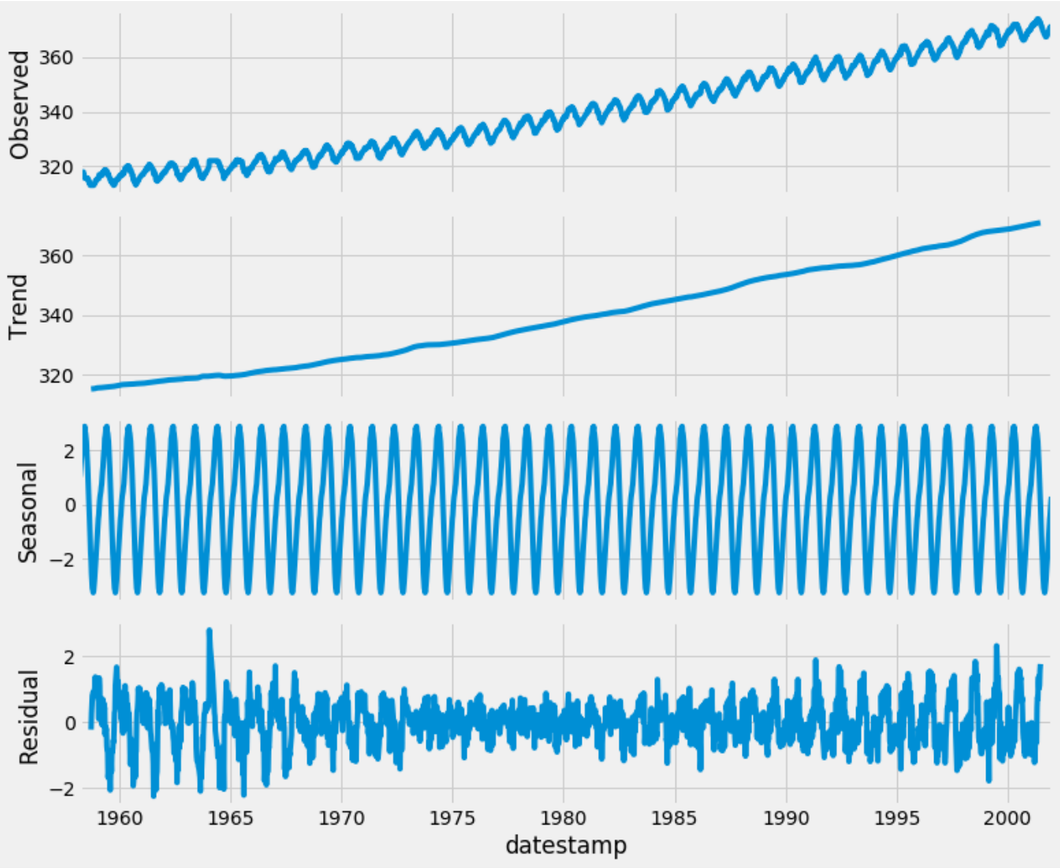
The properties of time series

- Seasonality: does the data display a clear periodic pattern?
- Trend: does the data follow a consistent upwards or downwards slope?
- Noise: are there any outlier points or missing values that are not consistent with the rest of the data?

Time series decomposition

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
from pylab import rcParams

rcParams['figure.figsize'] = 11, 9
decomposition = sm.tsa.seasonal_decompose(
                co2_levels['co2'])
fig = decomposition.plot()

plt.show()
```

A plot of time series decomposition on the CO2 data



Extracting components from time series decomposition

```
print(dir(decomposition))
```

```
['__class__', '__delattr__', '__dict__',
 ... 'plot', 'resid', 'seasonal', 'trend']
```

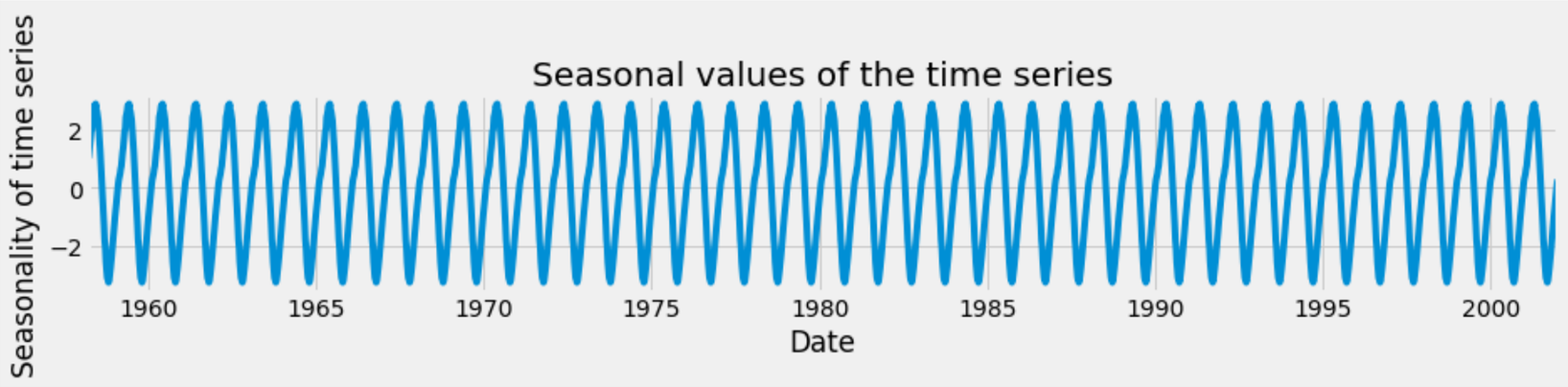
```
print(decomposition.seasonal)
```

```
datestamp
1958-03-29    1.028042
1958-04-05    1.235242
1958-04-12    1.412344
1958-04-19    1.701186
```

Seasonality component in time series

```
decomp_seasonal = decomposition.seasonal


ax = decomp_seasonal.plot(figsize=(14, 2))
ax.set_xlabel('Date')
ax.set_ylabel('Seasonality of time series')
ax.set_title('Seasonal values of the time series')


plt.show()
```

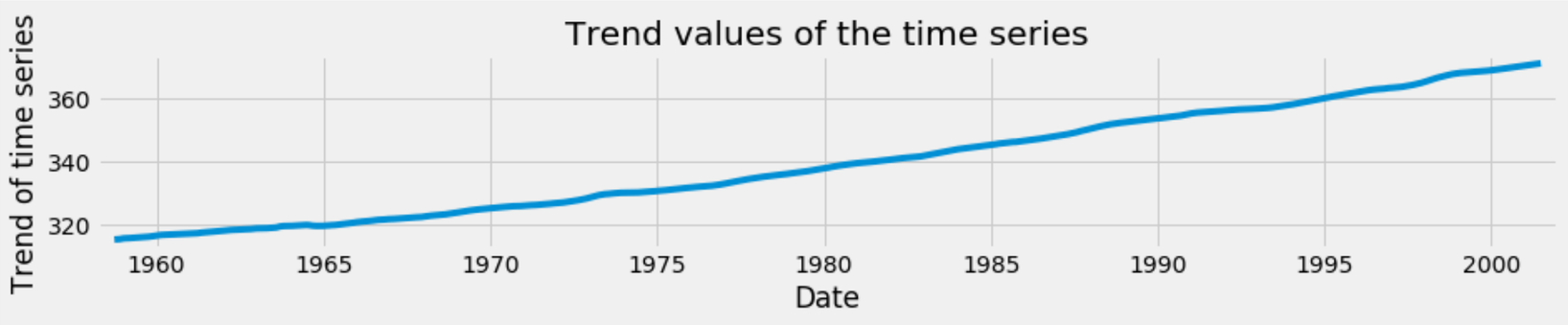
Seasonality component in time series



Trend component in time series

```
decomp_trend = decomposition.trend


ax = decomp_trend.plot(figsize=(14, 2))
ax.set_xlabel('Date')
ax.set_ylabel('Trend of time series')
ax.set_title('Trend values of the time series')


plt.show()
```

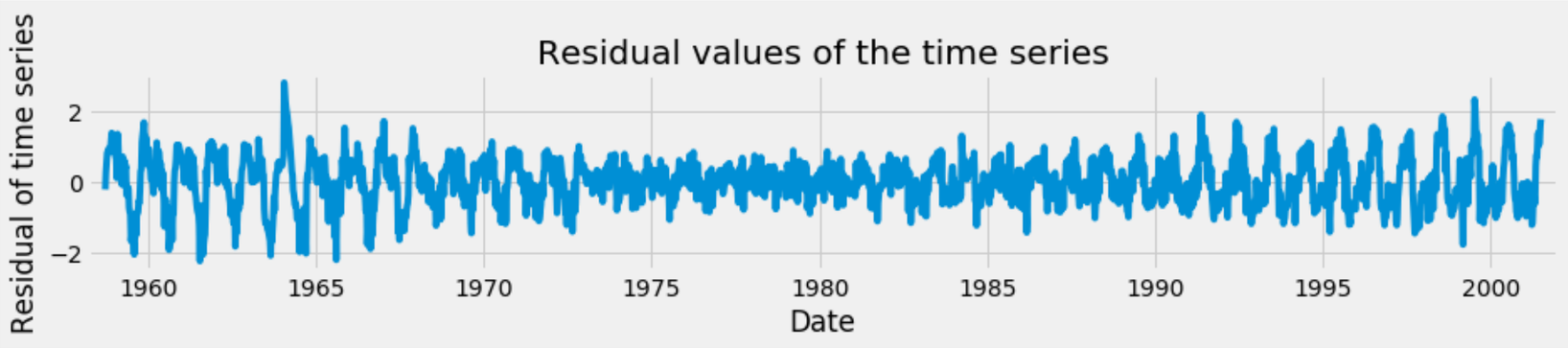
Trend component in time series



Noise component in time series

```
decomp_resid = decomp.resid


ax = decomp_resid.plot(figsize=(14, 2))
ax.set_xlabel('Date')
ax.set_ylabel('Residual of time series')
ax.set_title('Residual values of the time series')


plt.show()
```

Noise component in time series



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

A review on what you have learned so far

VISUALIZING TIME SERIES DATA IN PYTHON

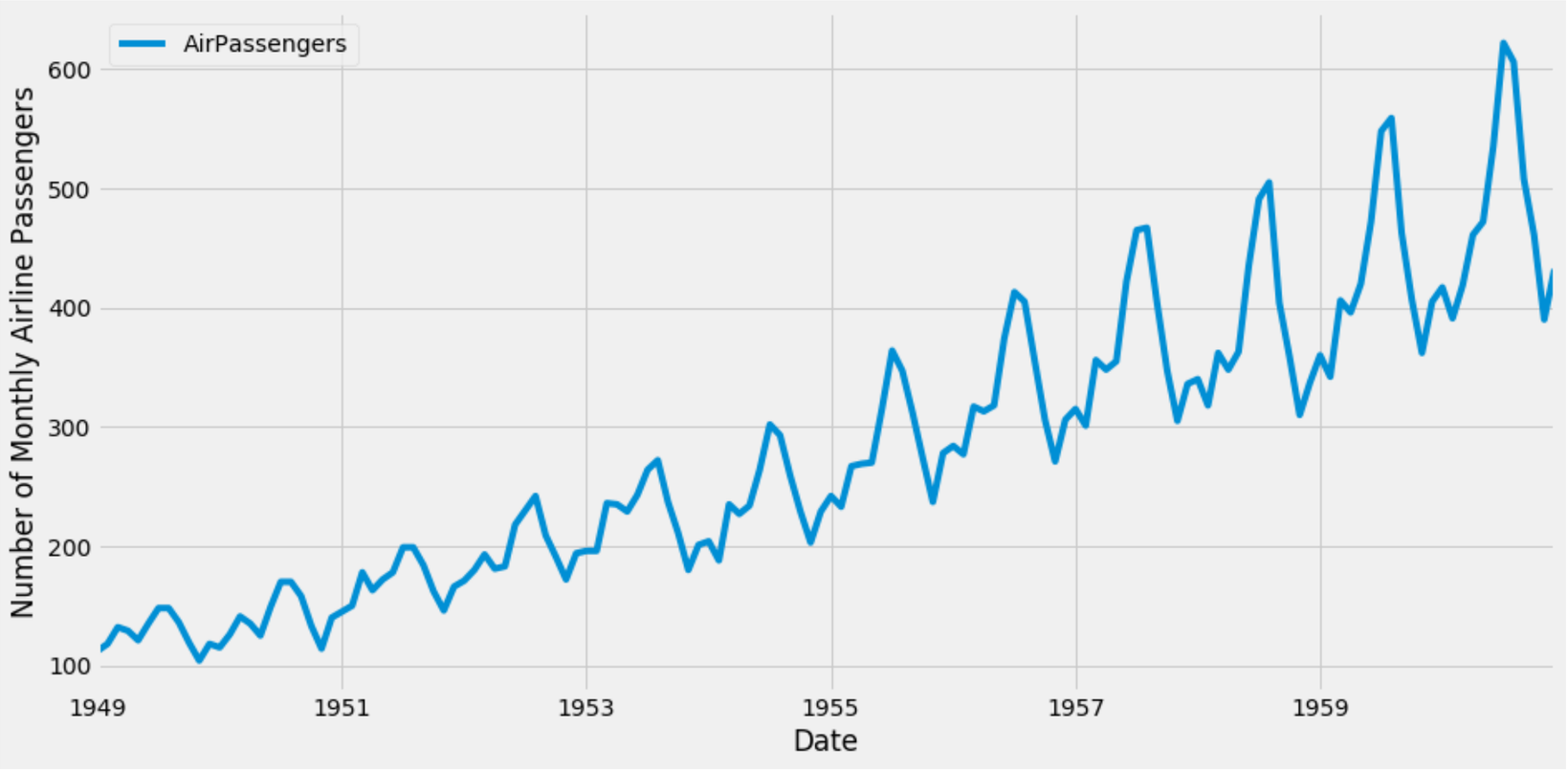
Thomas Vincent
Head of Data Science, Getty Images

datacamp

So far ...

- Visualize aggregates of time series data
- Extract statistical summaries
- Autocorrelation and Partial autocorrelation
- Time series decomposition

The airline dataset



Let's analyze this data!

VISUALIZING TIME SERIES DATA IN PYTHON

Working with more than one time series

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

Working with multiple time series

An isolated time series

date	ts1
1949-01	112
1949-02	118
1949-03	132

A file with multiple time series

date	ts1	ts2	ts3	ts4	ts5	ts6	ts7
2012-01-01	2113.8	10.4	1987.0	12.1	3091.8	43.2	476.7
2012-02-01	2009.0	9.8	1882.9	12.3	2954.0	38.8	466.8
2012-03-01	2159.8	10.0	1987.9	14.3	3043.7	40.1	502.1

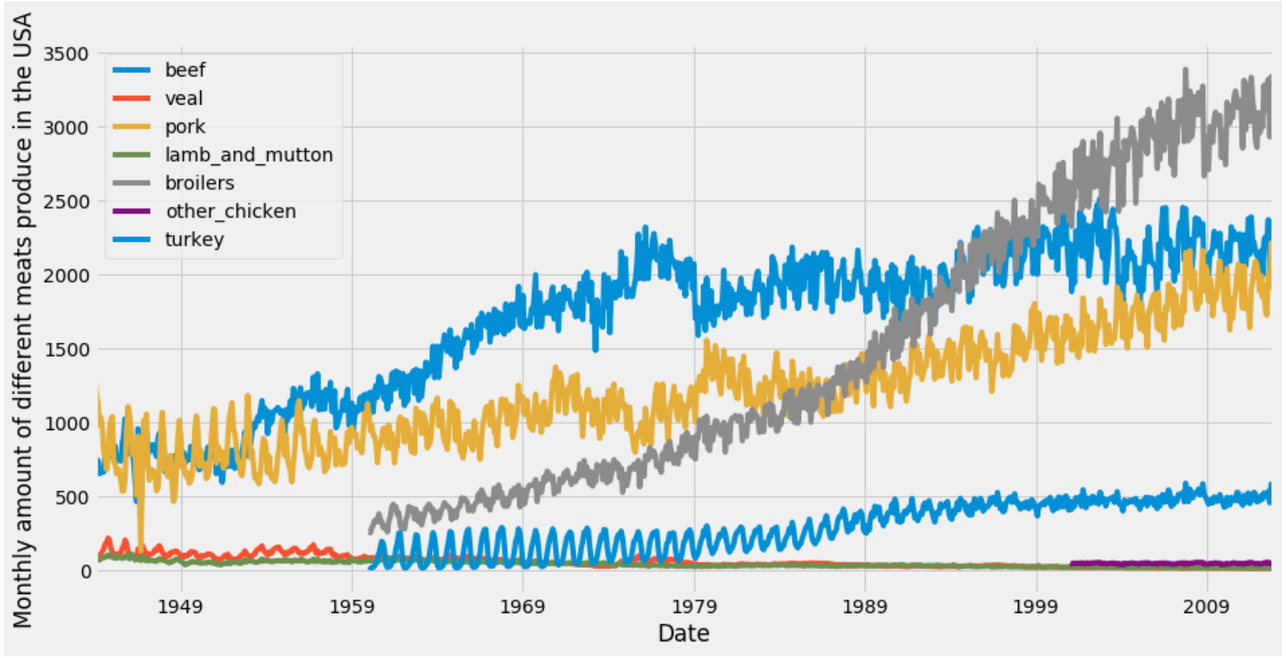
The Meat production dataset

```
import pandas as pd
meat = pd.read_csv("meat.csv")
print(meat.head(5))
```

```
         date   beef   veal    pork  lamb_and_mutton  broilers
0  1944-01-01  751.0   85.0  1280.0             89.0       NaN
1  1944-02-01  713.0   77.0  1169.0             72.0       NaN
2  1944-03-01  741.0   90.0  1128.0             75.0       NaN
3  1944-04-01  650.0   89.0   978.0             66.0       NaN
4  1944-05-01  681.0  106.0  1029.0             78.0       NaN

   other_chicken  turkey
0            NaN     NaN
1            NaN     NaN
2            NaN     NaN
3            NaN     NaN
4            NaN     NaN
```

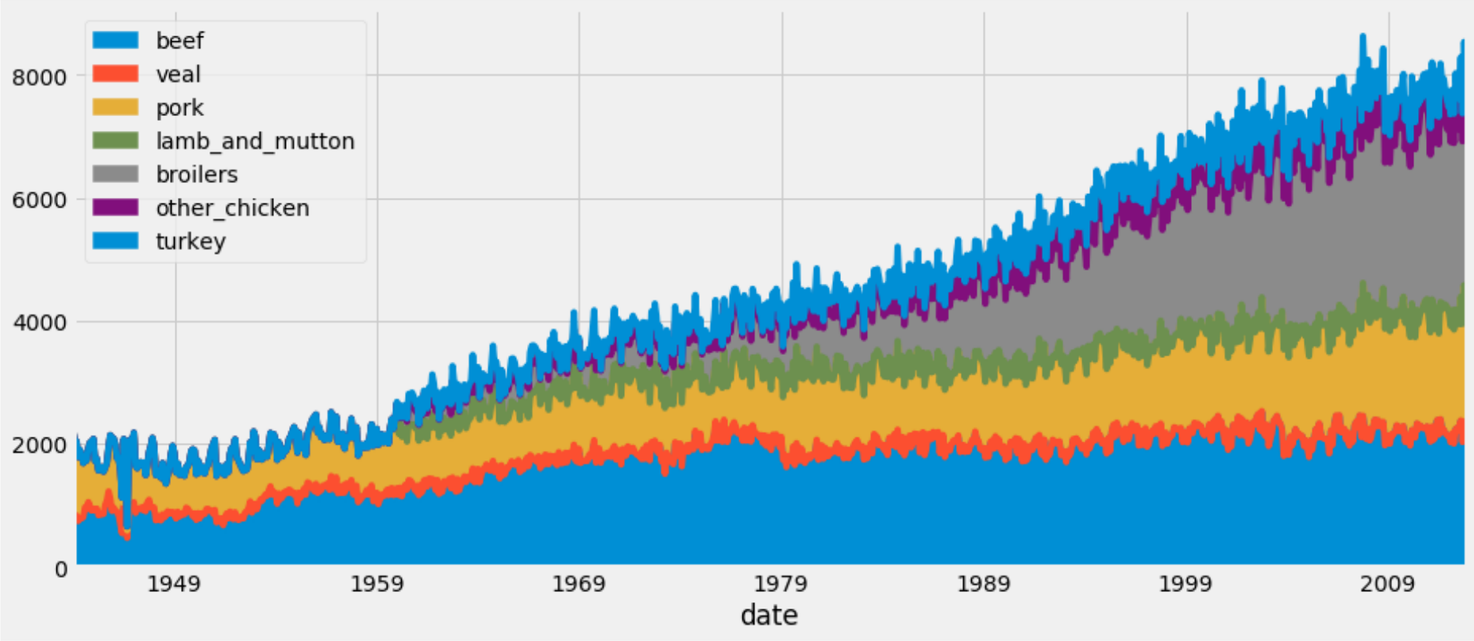
Summarizing and plotting multiple time series

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
ax = df.plot(figsize=(12, 4), fontsize=14)

plt.show()
```



Area charts

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
ax = df.plot.area(figsize=(12, 4), fontsize=14)

plt.show()
```



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Plot multiple time series

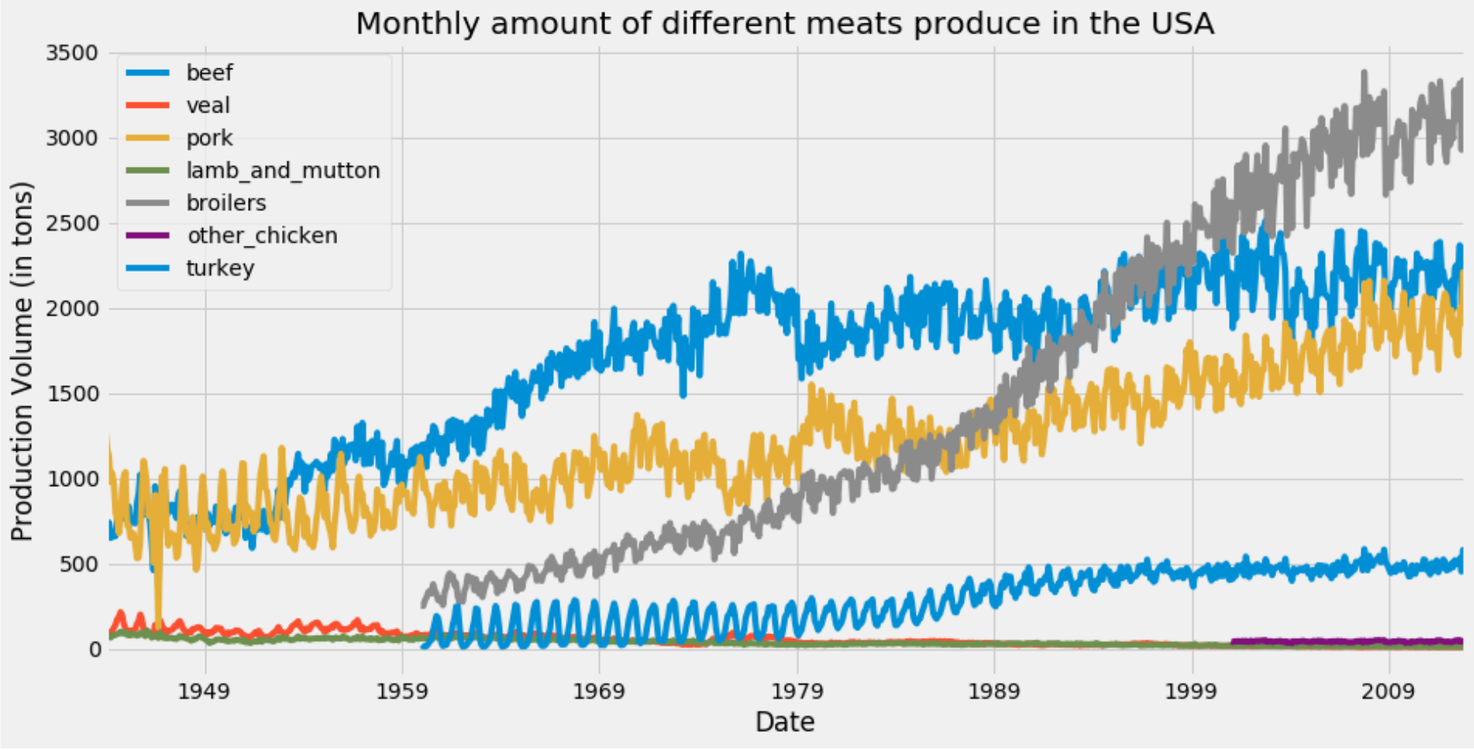
VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

Clarity is key

In this plot, the default `matplotlib` color scheme assigns the same color to the `beef` and `turkey` time series.

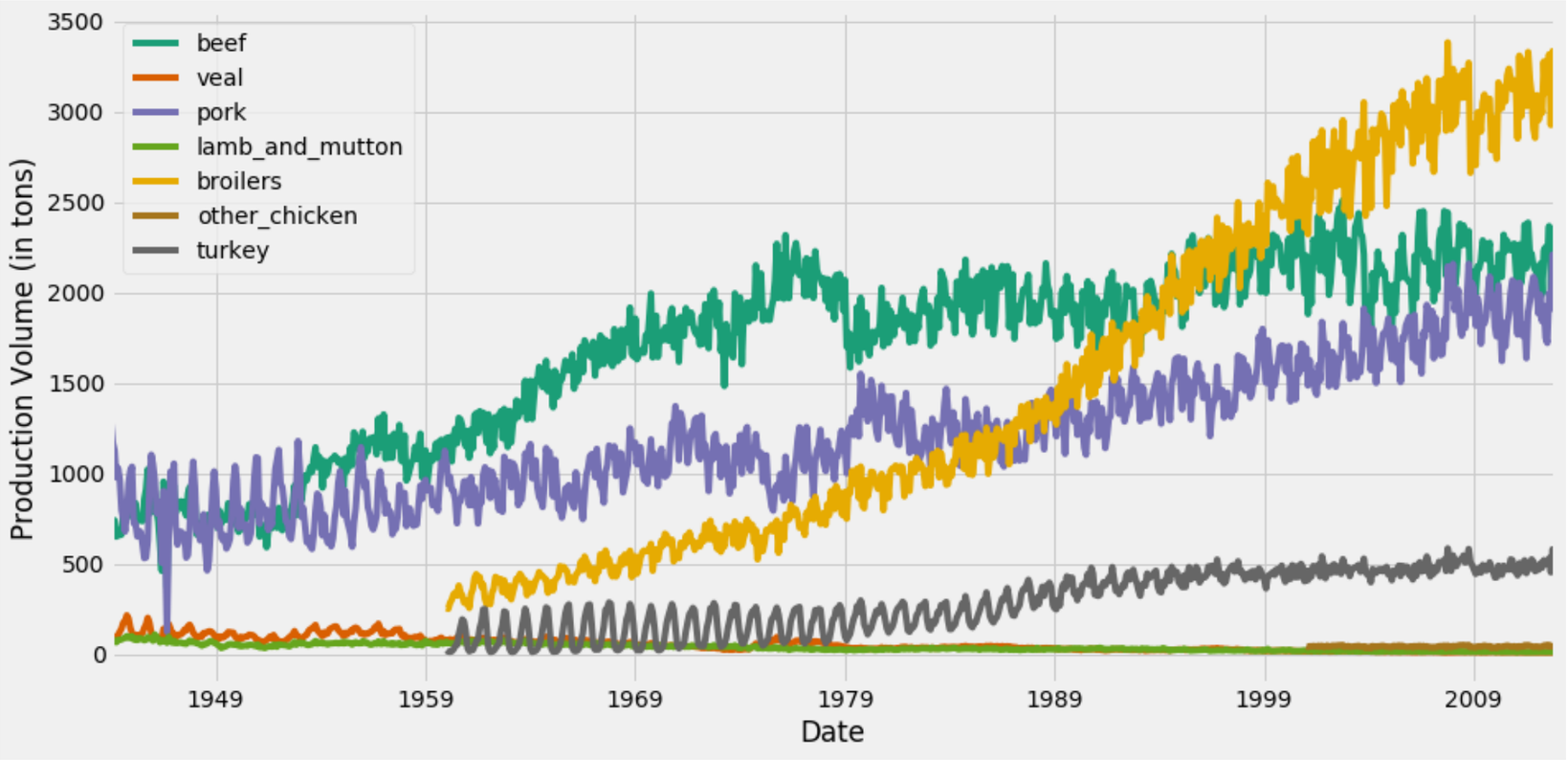


The colormap argument

```
ax = df.plot(colormap='Dark2', figsize=(14, 7))
ax.set_xlabel('Date')
ax.set_ylabel('Production Volume (in tons)')


plt.show()
```

For the full set of available colormaps, click **here**.

Changing line colors with the colormap argument

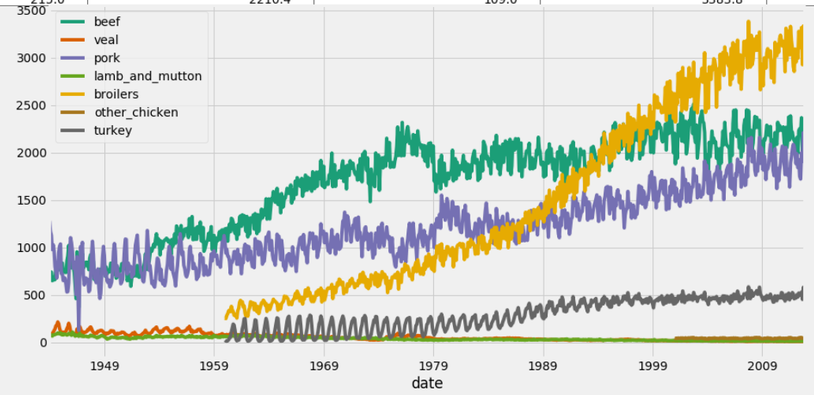


Enhancing your plot with information

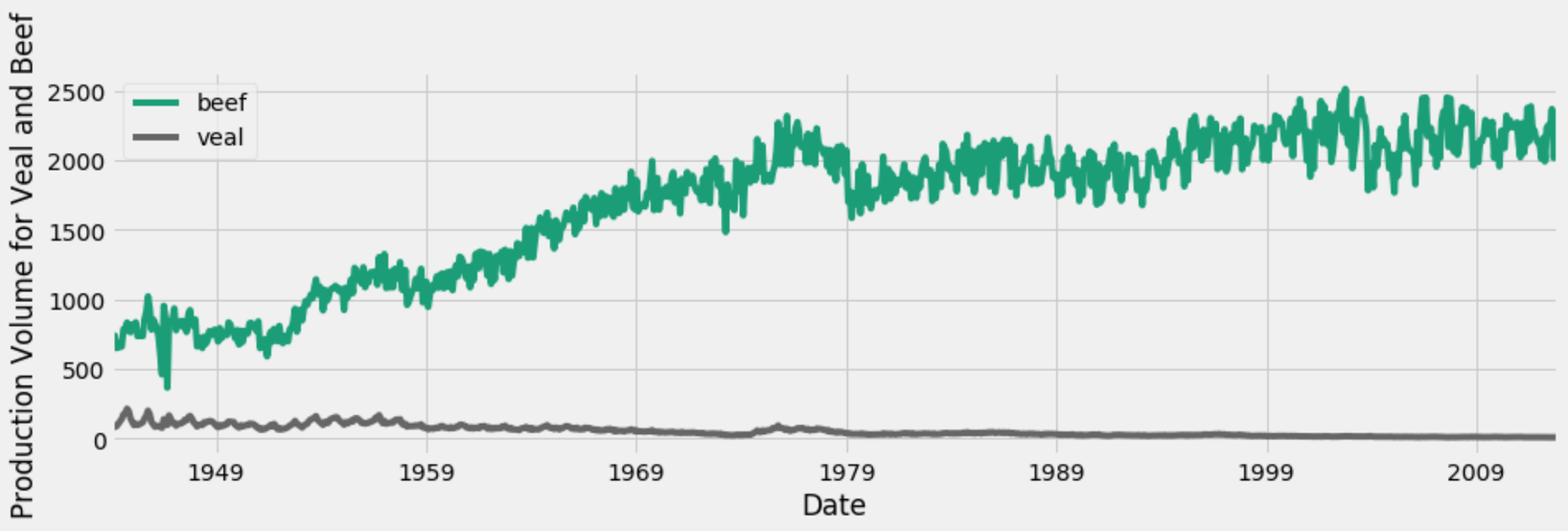
```
ax = df.plot(colormap='Dark2', figsize=(14, 7))
df_summary = df.describe()

# Specify values of cells in the table
ax.table(cellText=df_summary.values,
         # Specify width of the table
         colWidths=[0.3]*len(df.columns),
         # Specify row labels
         rowLabels=df_summary.index,
         # Specify column labels
         colLabels=df_summary.columns,
         # Specify location of the table
         loc='top')

plt.show()
```

Adding Statistical summaries to your plots

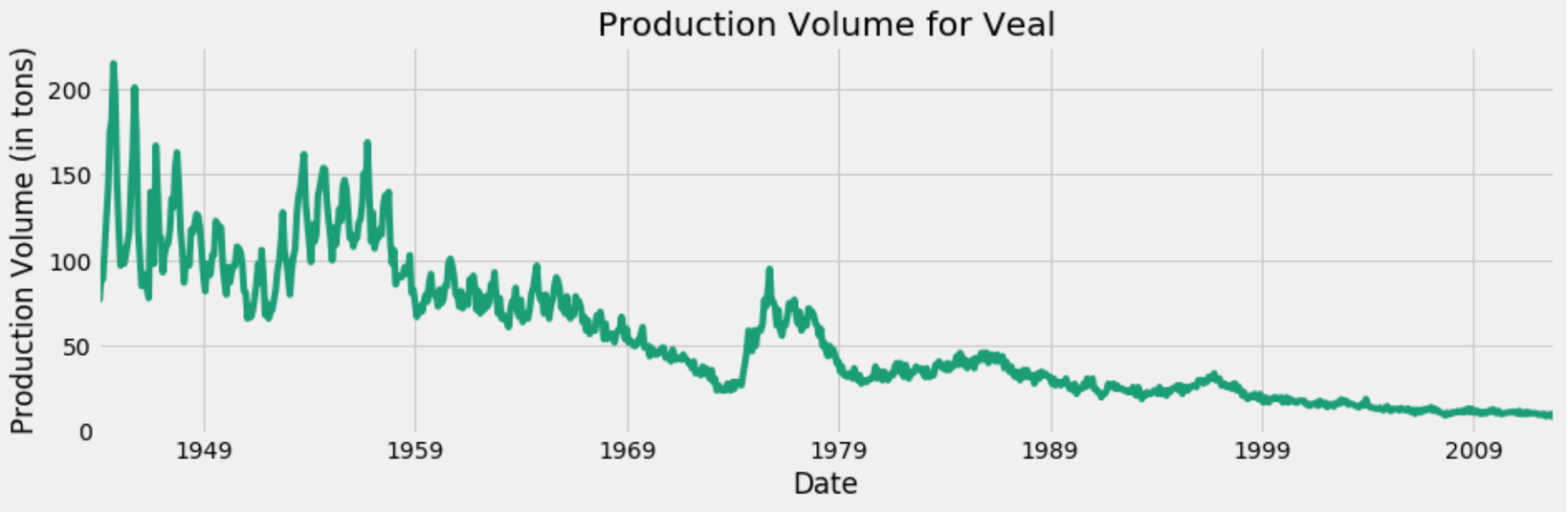
	beef	veal	pork	lamb_and_mutton	broilers	other_chicken	turkey
count	827.0	827.0	827.0	827.0	635.0	143.0	635.0
mean	1683.46336155	54.1985489722	1211.68379686	38.3607013301	1516.58251969	43.0335664336	292.814645669
std	501.698479574	39.0628037539	371.311802273	19.6243398061	963.012101054	3.86714117313	162.482637819
min	366.0	8.8	124.0	10.9	250.9	32.3	12.4
25%	1231.5	24.0	934.5	23.0	636.35	40.2	154.15
50%	1853.0	40.0	1156.0	31.0	1211.3	43.4	278.3
75%	2070.0	79.0	1466.0	55.0	2426.65	45.65	449.15
max	2512.0	215.0	2210.4	109.0	3383.8	51.1	585.1



Dealing with different scales

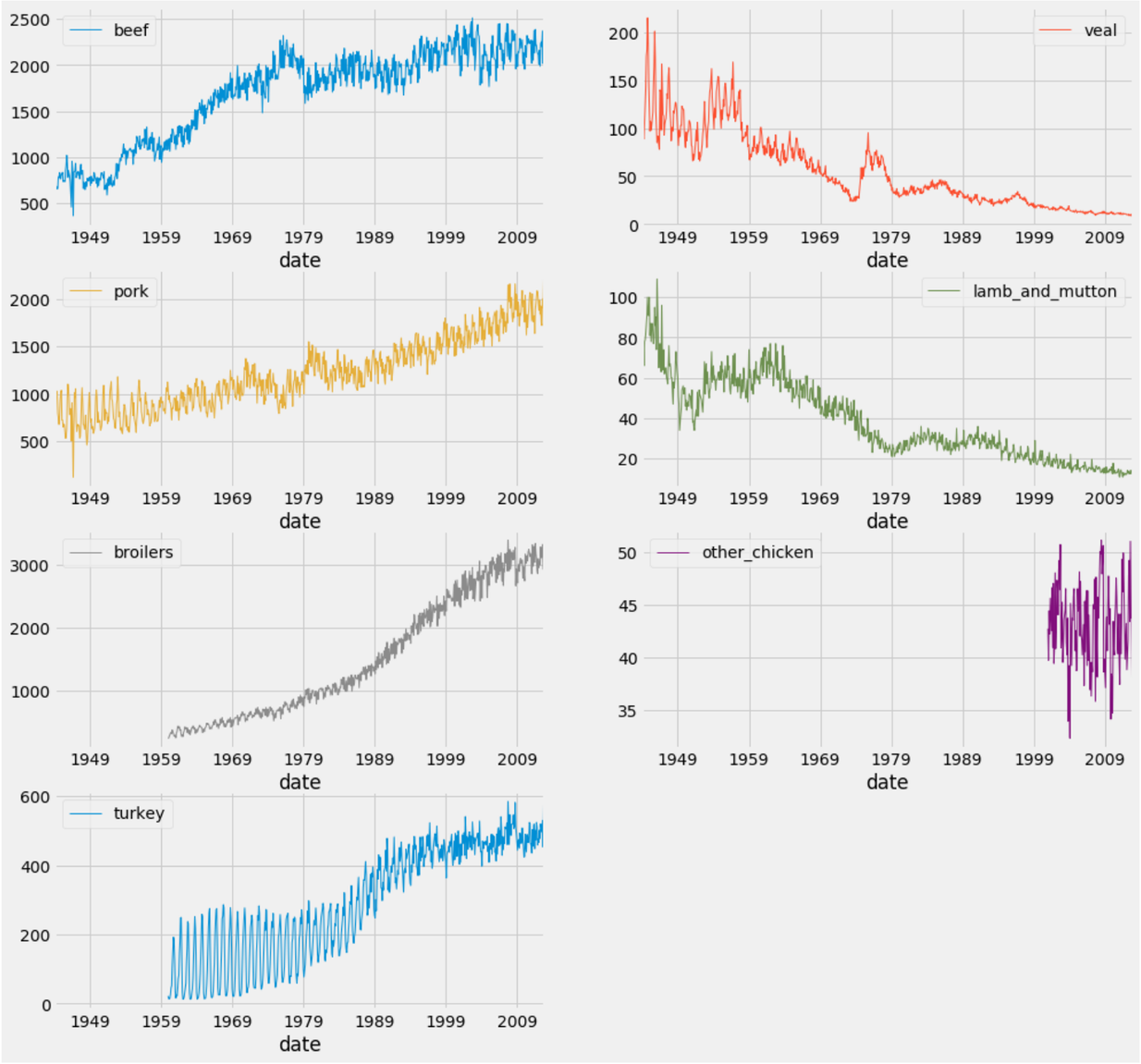


Only veal


Production Volume for Veal
Production Volume (in tons)
200
150
100
50
0
1949
1959
1969
1979
1989
1999
2009
Date

Facet plots

```
df.plot(subplots=True,
                  linewidth=0.5,
                  layout=(2, 4),
                  figsize=(16, 10),
                  sharex=False,
                  sharey=False)


plt.show()
```


beef
veal
pork
lamb_and_mutton
broilers
other_chicken
turkey
date
1949
1959
1969
1979
1989
1999
2009

Time for some action!

VISUALIZING TIME SERIES DATA IN PYTHON

Find relationships between multiple time series

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

Correlations between two variables

- In the field of Statistics, the correlation coefficient is a measure used to determine the strength or lack of relationship between two variables:
  - Pearson's coefficient can be used to compute the correlation coefficient between variables for which the relationship is thought to be linear
  - Kendall Tau or Spearman rank can be used to compute the correlation coefficient between variables for which the relationship is thought to be non-linear

Compute correlations

```
from scipy.stats.stats import pearsonr
from scipy.stats.stats import spearmanr
from scipy.stats.stats import kendalltau
x = [1, 2, 4, 7]
y = [1, 3, 4, 8]
pearsonr(x, y)
```

```
SpearmanrResult(correlation=0.9843, pvalue=0.01569)
```

```
spearmanr(x, y)
```

```
SpearmanrResult(correlation=1.0, pvalue=0.0)
```

```
kendalltau(x, y)
```

```
KendalltauResult(correlation=1.0, pvalue=0.0415)
```

What is a correlation matrix?

- When computing the correlation coefficient between more than two variables, you obtain a correlation matrix
  - Range: [-1, 1]
  - 0: no relationship
  - 1: strong positive relationship
  - -1: strong negative relationship

What is a correlation matrix?

- A correlation matrix is always "symmetric"
- The diagonal values will always be equal to 1

```
   x     y     z
x  1.00 -0.46  0.49
y -0.46  1.00 -0.61
z  0.49 -0.61  1.00
```

Computing Correlation Matrices with Pandas

```
corr_p = meat[['beef', 'veal','turkey']].corr(method='pearson')
print(corr_p)
```

```
          beef     veal     turkey
beef      1.000   -0.829    0.738
veal     -0.829    1.000   -0.768
turkey    0.738   -0.768    1.000
```

```
corr_s = meat[['beef', 'veal','turkey']].corr(method='spearman')
print(corr_s)
```

```
          beef     veal     turkey
beef      1.000   -0.812    0.778
veal     -0.812    1.000   -0.829
turkey    0.778   -0.829    1.000
```

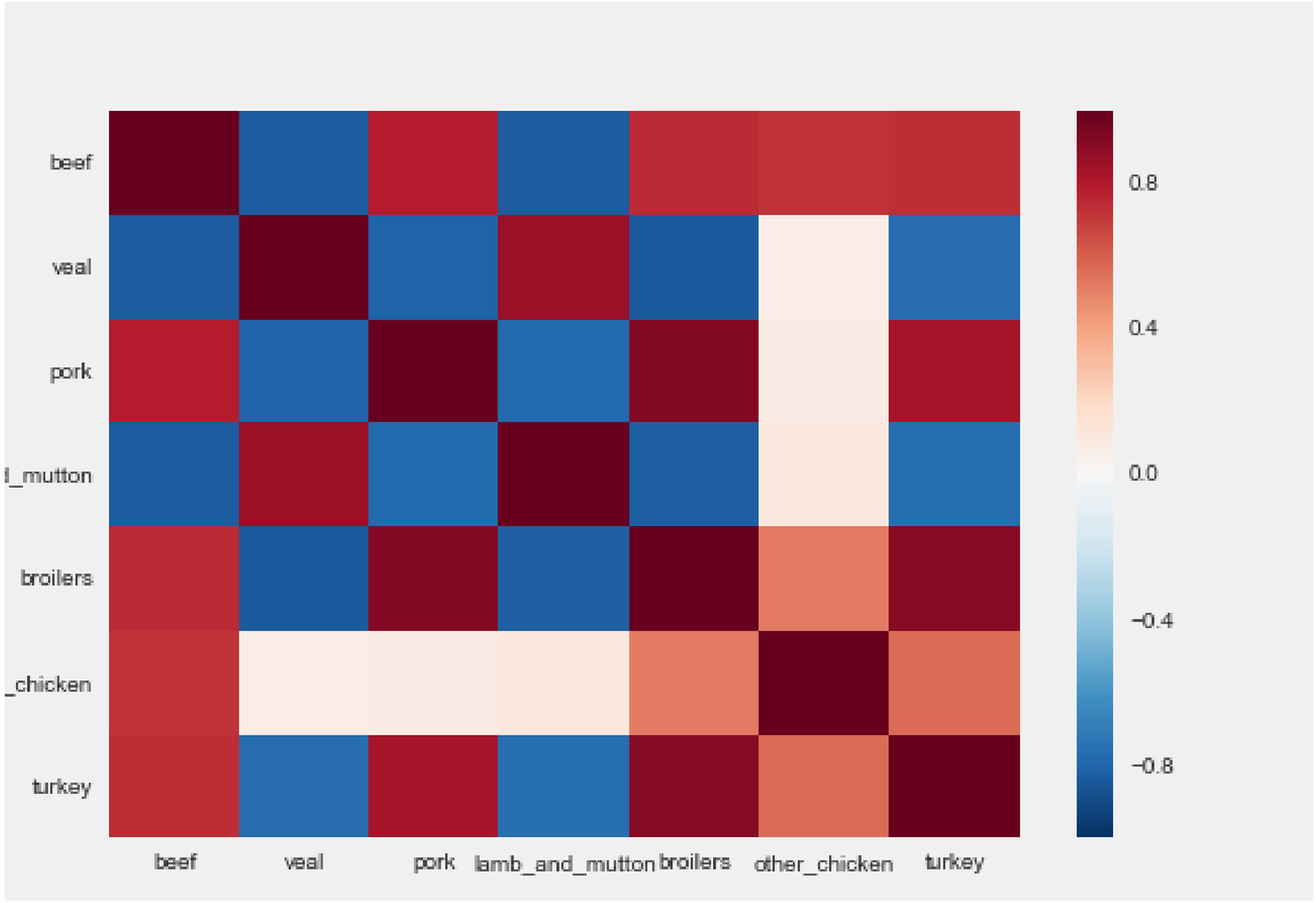
Computing Correlation Matrices with Pandas

```
corr_mat = meat.corr(method='pearson')
```

Heatmap

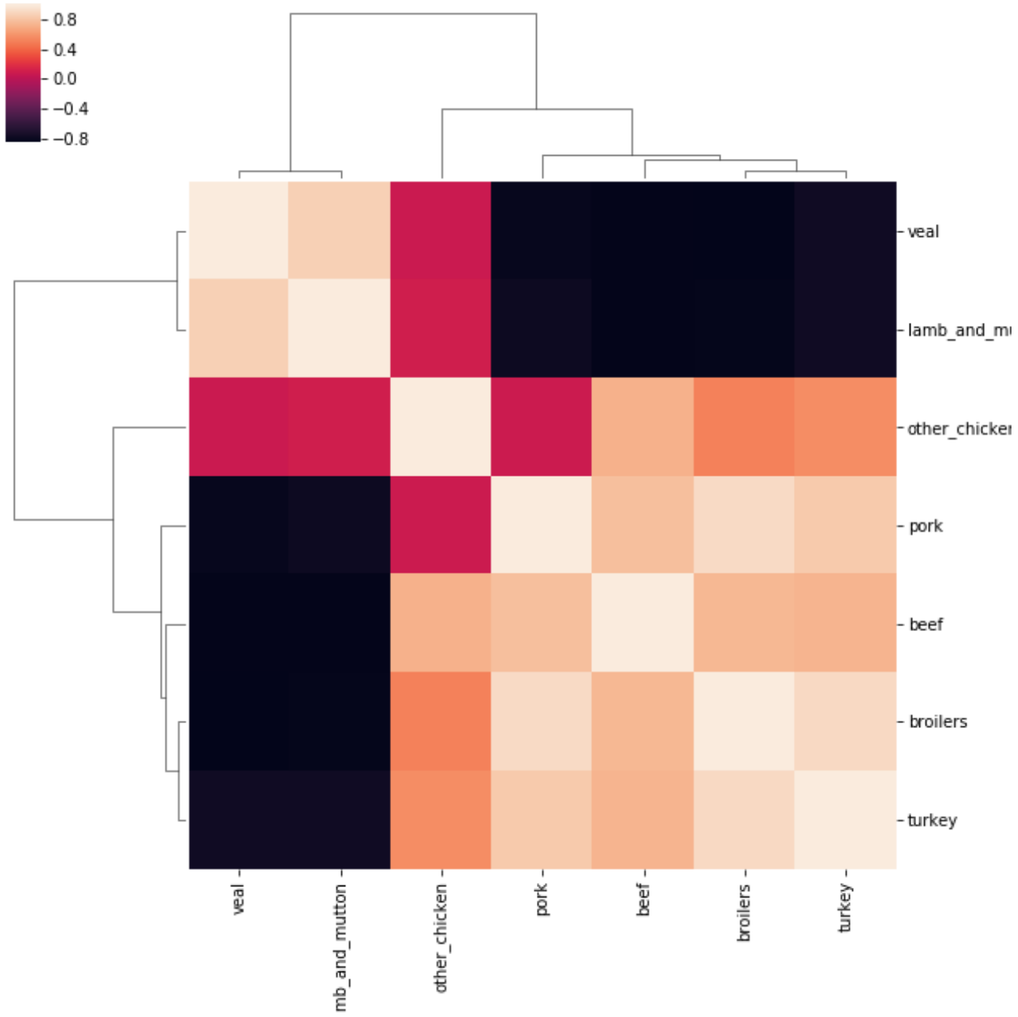
```
import seaborn as sns
sns.heatmap(corr_mat)
```

Heatmap


beef
veal
pork
l_mutton
broilers
_chicken
turkey
beef
veal
pork
lamb_and_mutton
broilers
other_chicken
turkey
0.8
0.4
0.0
−0.4
−0.8

Clustermap

```
sns.clustermap(corr_mat)
```


0.8
0.4
0.0
−0.4
−0.8
veal
lamb_and_mu
other_chicken
pork
beef
broilers
turkey
veal
mb_and_mutton
other_chicken
pork
beef
broilers
turkey

Let's practice!

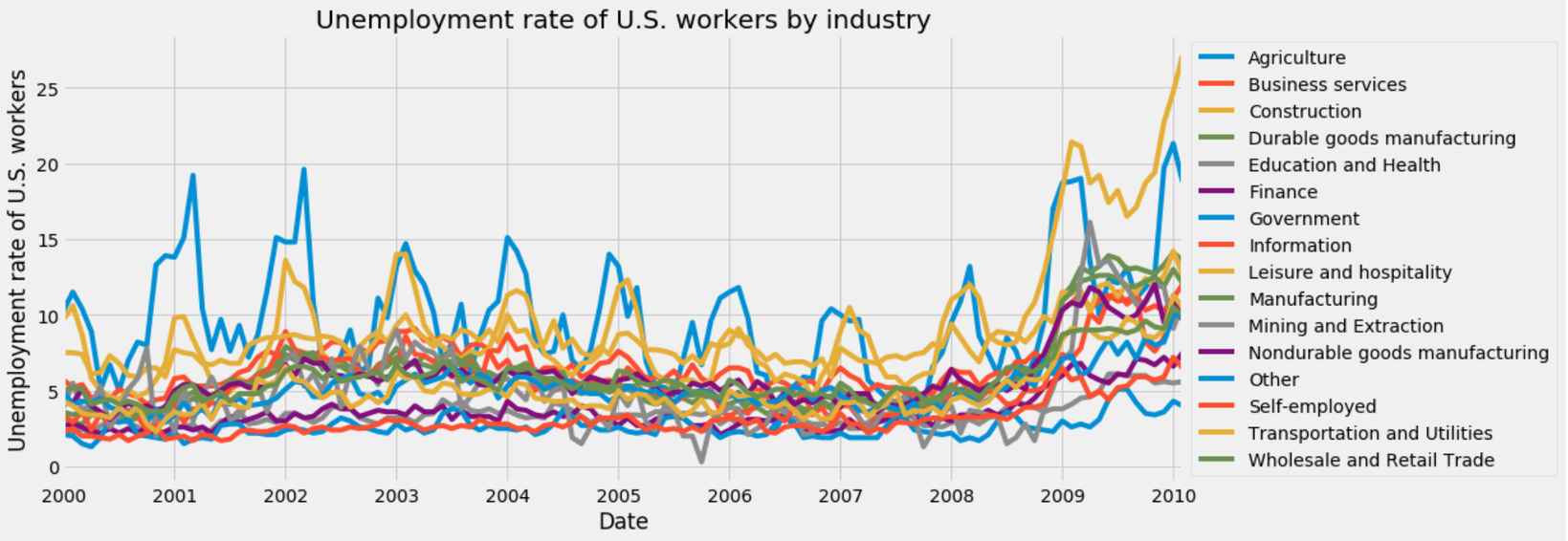
VISUALIZING TIME SERIES DATA IN PYTHON

Apply your knowledge to a new dataset

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

The Jobs dataset



Let's get started!

VISUALIZING TIME SERIES DATA IN PYTHON

Beyond summary statistics

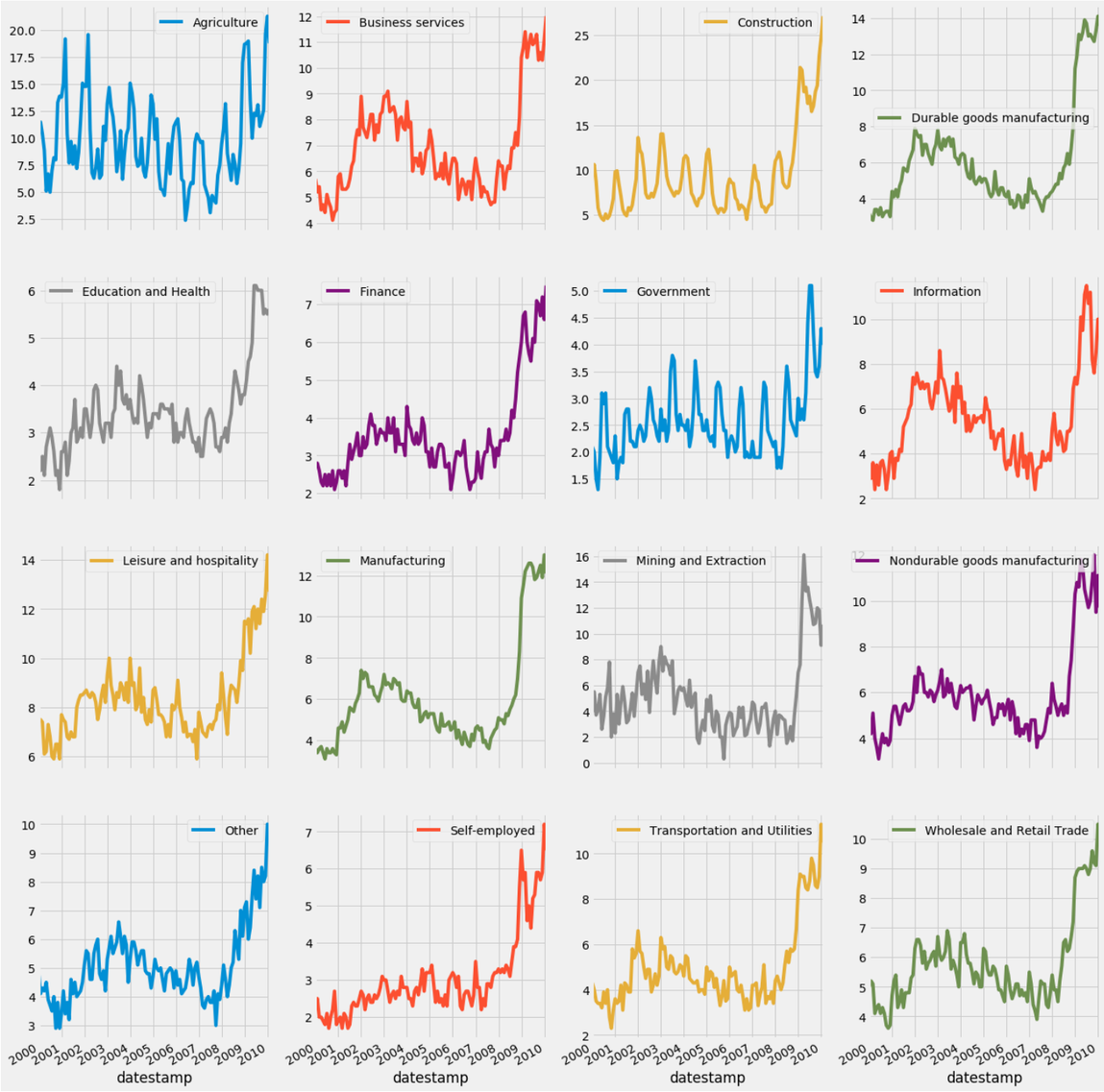
VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

datacamp

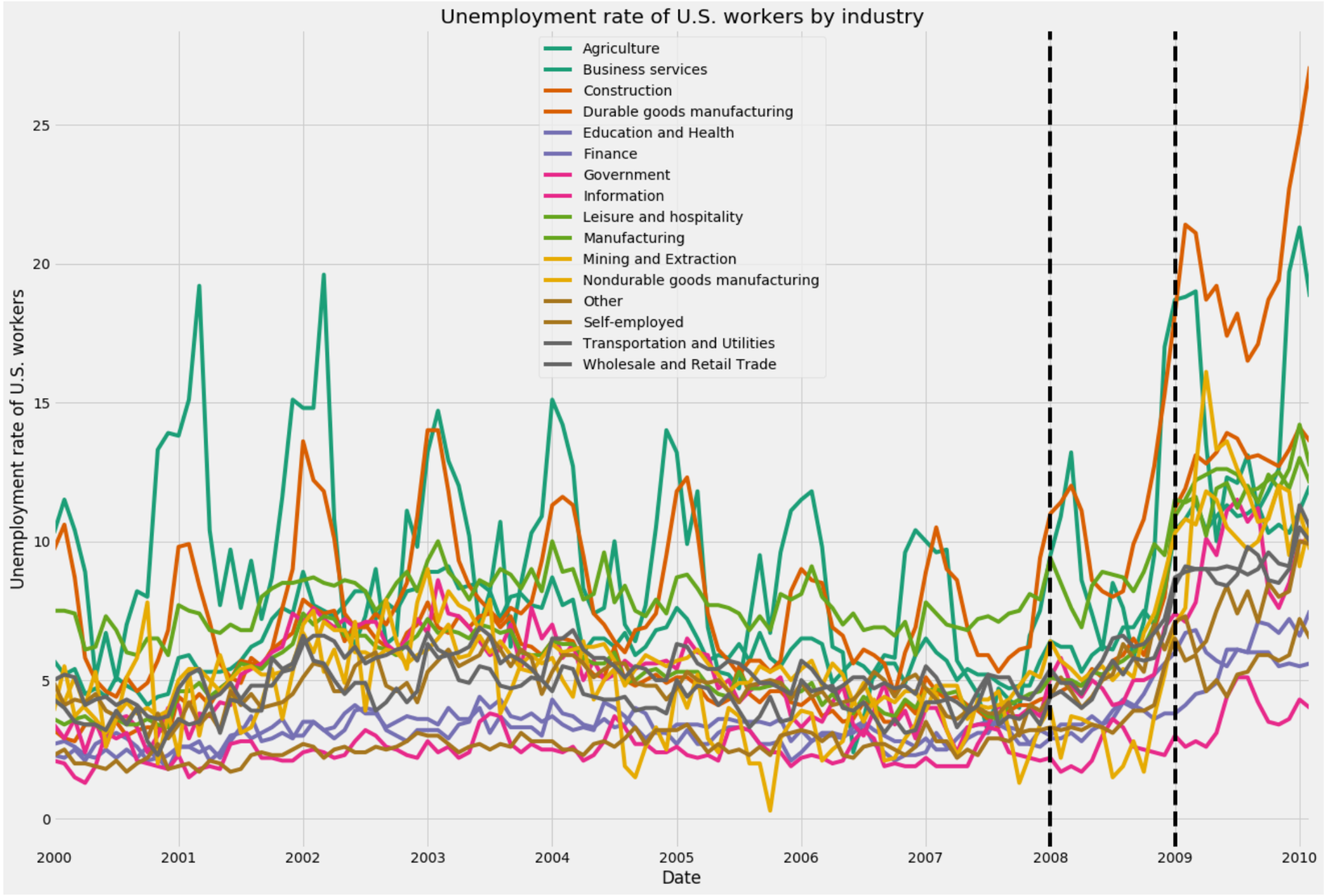
Facet plots of the jobs dataset

```
jobs.plot(subplots=True,
             layout=(4, 4),
             figsize=(20, 16),
             sharex=True,
             sharey=False)

plt.show()
```


Agriculture
Business services
Construction
Durable goods manufacturing
Education and Health
Finance
Government
Information
Leisure and hospitality
Manufacturing
Mining and Extraction
Nondurable goods manufacturing
Other
Self-employed
Transportation and Utilities
Wholesale and Retail Trade
datestamp
datestamp
datestamp
datestamp

Annotating events in the jobs dataset

```
ax = jobs.plot(figsize=(20, 14), colormap='Dark2')
ax.axvline('2008-01-01', color='black',
           linestyle='--')
ax.axvline('2009-01-01', color='black',
           linestyle='--')
```


Unemployment rate of U.S. workers by industry
Agriculture
Business services
Construction
Durable goods manufacturing
Education and Health
Finance
Government
Information
Leisure and hospitality
Manufacturing
Mining and Extraction
Nondurable goods manufacturing
Other
Self-employed
Transportation and Utilities
Wholesale and Retail Trade
Unemployment rate of U.S. workers
0
5
10
15
20
25
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
Date

Taking seasonal average in the jobs dataset

```
print(jobs.index)
```

```
DatetimeIndex(['2000-01-01', '2000-02-01', '2000-03-01',
'2000-04-01', '2009-09-01','2009-10-01',
'2009-11-01', '2009-12-01','2010-01-01', '2010-02-01'],
dtype='datetime64[ns]', name='datestamp',
length=122, freq=None)
```

```
index_month = jobs.index.month
jobs_by_month = jobs.groupby(index_month).mean()
print(jobs_by_month)
```

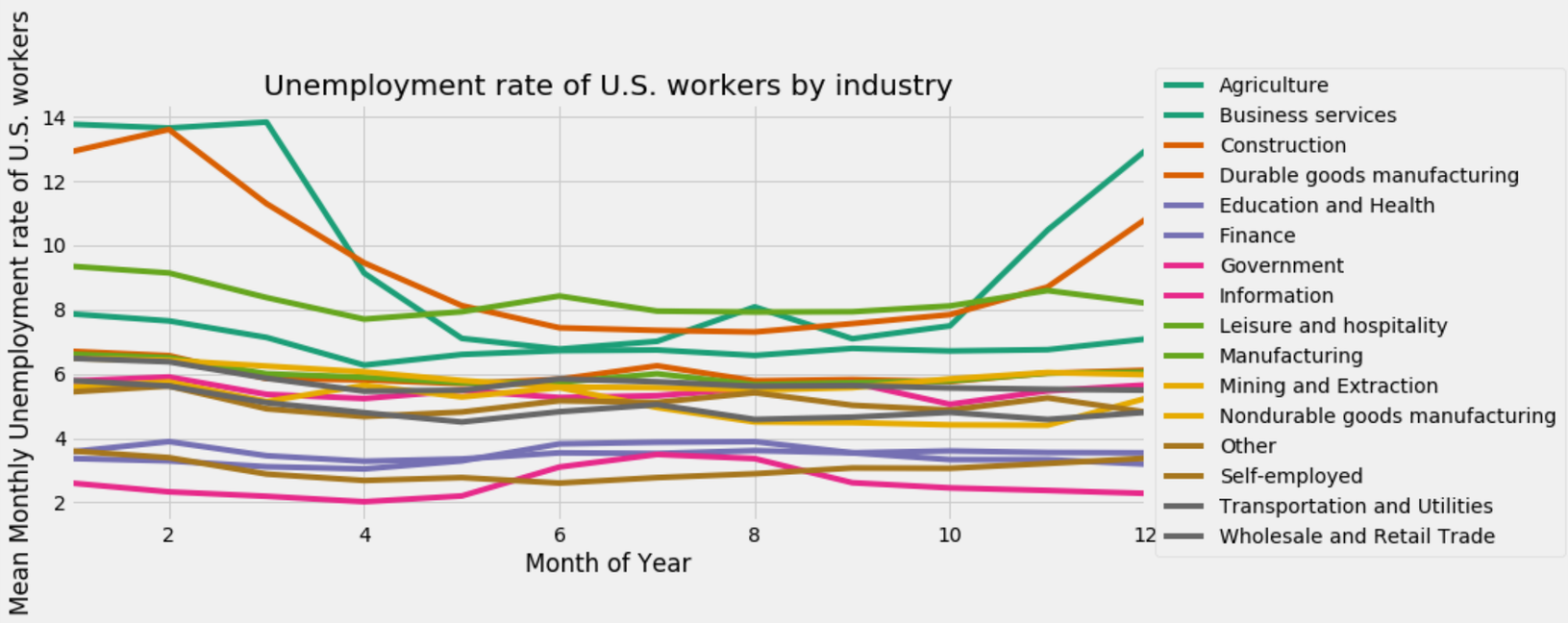
```
datestamp    Agriculture  Business services  Construction
1            13.763636           7.863636     12.909091
2            13.645455           7.645455     13.600000
3            13.830000           7.130000     11.290000
4             9.130000           6.270000      9.450000
5             7.100000           6.600000      8.120000
...
```

Monthly averages in the jobs dataset

```
ax = jobs_by_month.plot(figsize=(12, 5),
colormap='Dark2')
```

```
ax.legend(bbox_to_anchor=(1.0, 0.5),
loc='center left')
```

Monthly averages in the jobs dataset



Time to practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Decompose time series data

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

Python dictionaries

```
# Initialize a Python dictionnary
my_dict = {}

# Add a key and value to your dictionnary
my_dict['your_key'] = 'your_value'

# Add a second key and value to your dictionnary
my_dict['your_second_key'] = 'your_second_value'

# Print out your dictionnary
print(my_dict)
```

```
{'your_key': 'your_value',
 'your_second_key': 'your_second_value'}
```

Decomposing multiple time series with Python dictionaries

```
# Import the statsmodel library
import statsmodels.api as sm
# Initialize a dictionary
my_dict = {}
# Extract the names of the time series
ts_names = df.columns
print(ts_names)
```

```
['ts1', 'ts2', 'ts3']
```

```
# Run time series decomposition
for ts in ts_names:
    ts_decomposition = sm.tsa.seasonal_decompose(jobs[ts])
    my_dict[ts] = ts_decomposition
```

Extract decomposition components of multiple time series

```
# Initialize a new dictionnary
my_dict_trend = {}
# Extract the trend component
for ts in ts_names:
    my_dict_trend[ts] = my_dict[ts].trend
# Convert to a DataFrame
trend_df = pd.DataFrame.from_dict(my_dict_trend)
print(trend_df)
```

```
            ts1  ts2 ts3
datestamp
2000-01-01  2.2  1.3 3.6
2000-02-01  3.4  2.1 4.7
...
```

Python dictionaries for the win!

VISUALIZING TIME SERIES DATA IN PYTHON

Compute correlations between time series

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

Trends in Jobs data

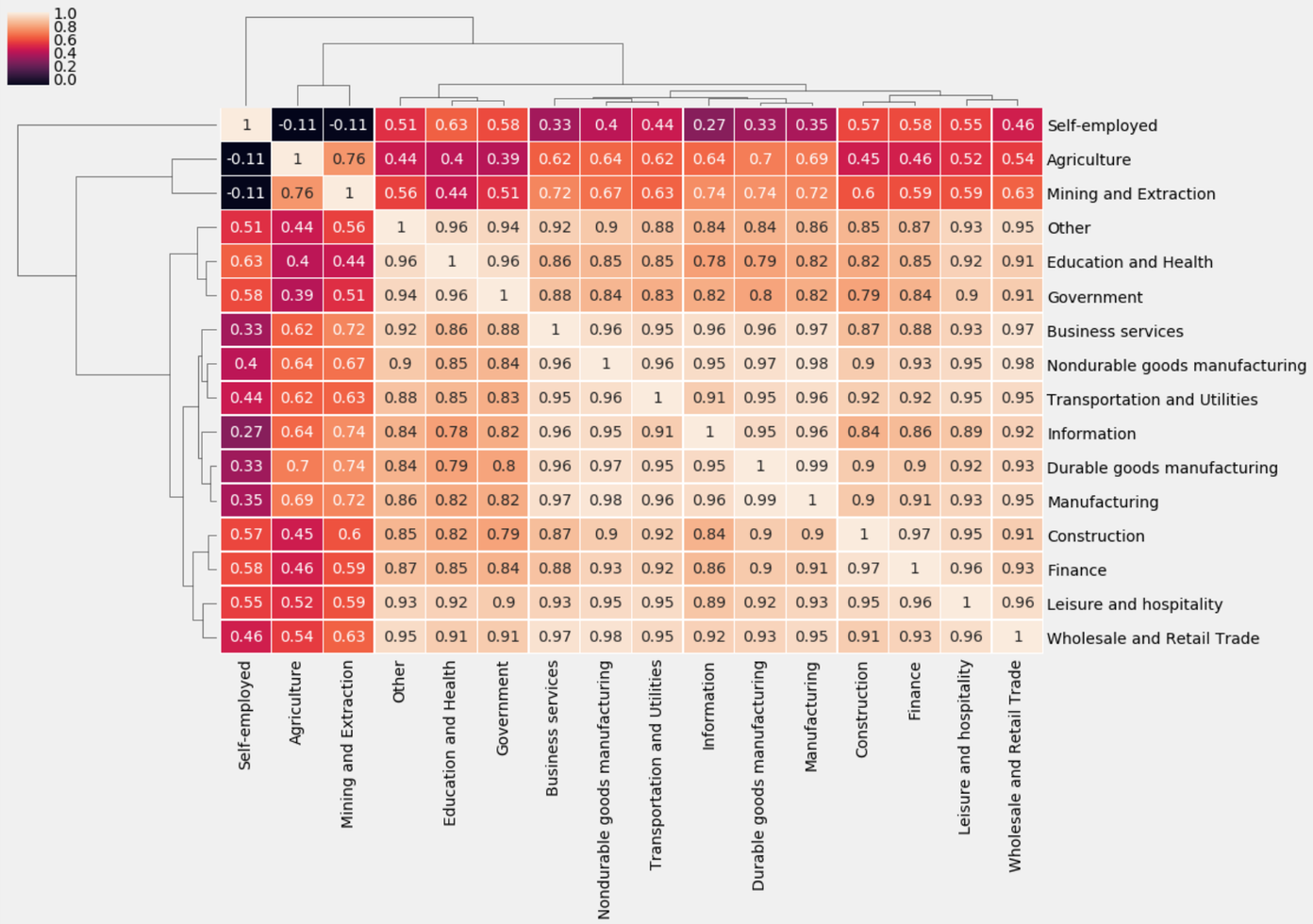
```
print(trend_df)
```

```
datestamp  Agriculture  Business services  Construction

2000-01-01          NaN                NaN           NaN
2000-02-01          NaN                NaN           NaN
2000-03-01          NaN                NaN           NaN
2000-04-01          NaN                NaN           NaN
2000-05-01          NaN                NaN           NaN
2000-06-01          NaN                NaN           NaN
2000-07-01     9.170833           4.787500      6.329167
2000-08-01     9.466667           4.820833      6.304167
...
```

Plotting a clustermap of the jobs correlation matrix

```
# Get correlation matrix of the seasonality_df DataFrame
trend_corr = trend_df.corr(method='spearman')

# Customize the clustermap of the seasonality_corr
correlation matrix
fig = sns.clustermap(trend_corr, annot=True, linewidth=0.4)

plt.setp(fig.ax_heatmap.yaxis.get_majorticklabels(),
rotation=0)

plt.setp(fig.ax_heatmap.xaxis.get_majorticklabels(),
rotation=90)
```

The jobs correlation matrix



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Congratulations!

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent
Head of Data Science, Getty Images

Going further with time series

- Data from Zillow Research
- Kaggle competitions
- Reddit Data

Going further with time series

- The importance of time series in business:
  - to identify seasonal patterns and trends
  - to study past behaviors
  - to produce robust forecasts
  - to evaluate and compare company achievements

Getting to the next level

- Manipulating Time Series Data in Python
- Importing & Managing Financial Data in Python
- Statistical Thinking in Python (Part 1)
- Supervised Learning with scikit-learn

Thank you!

VISUALIZING TIME SERIES DATA IN PYTHON