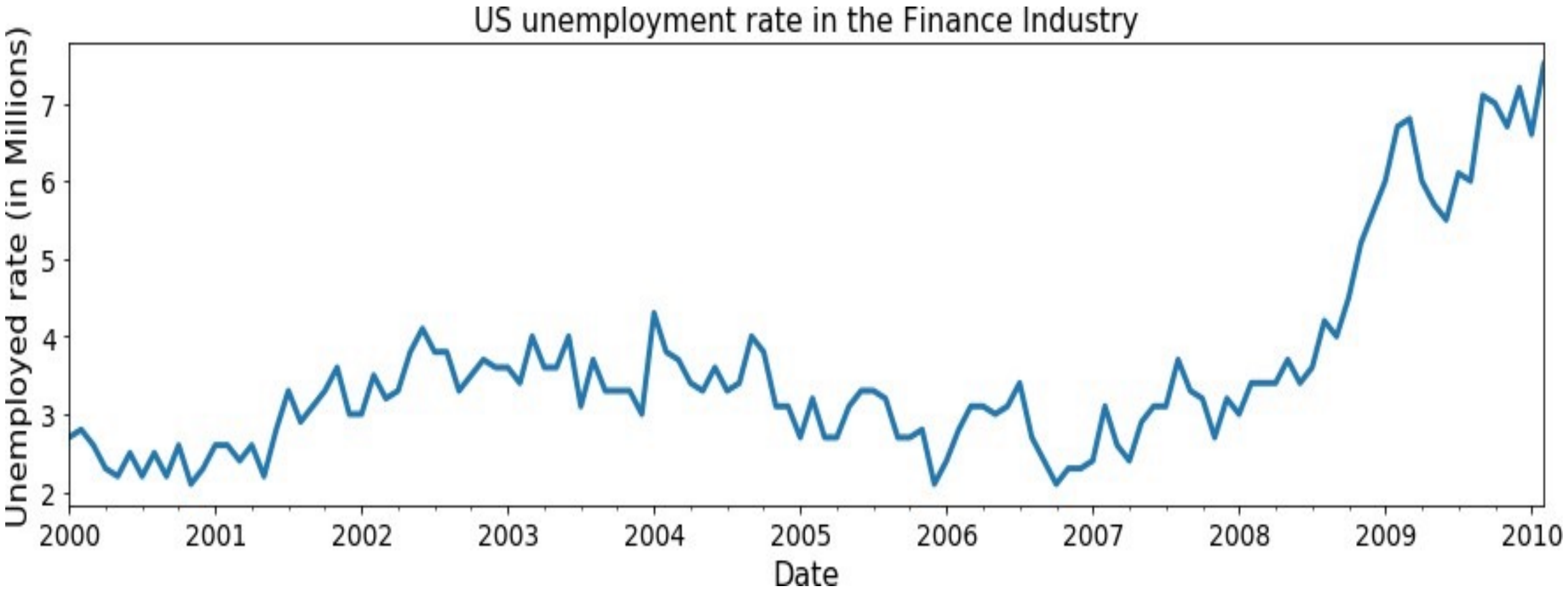
Visualize Time Series Data in Python

Trung-Nghia Cao
Minh-Duc Bui

Outline

- Line Plots
- Summary Statistics and Diagnostics
- Seasonality, Trend and Noise
- Work with Multiple Time Series
- Case Study: Unemployment Rate

Time Series in the Field of Data Science



Time Series in the Field of Data Science

Univariate Time Series

S&P 500 Stock Prices
Currency in USD

Date	Open
Oct 30, 2020	3,293.59
Oct 29, 2020	3,277.17
Oct 28, 2020	3,342.48
Oct 27, 2020	3,403.15
Oct 26, 2020	3,441.42
Oct 23, 2020	3,464.90
Oct 22, 2020	3,438.50
Oct 21, 2020	3,439.91
Oct 20, 2020	3,439.38
Oct 19, 2020	3,493.66
Oct 16, 2020	3,493.50
Oct 15, 2020	3,453.72

Source: www.finance.yahoo.com

Multivariate Time Series

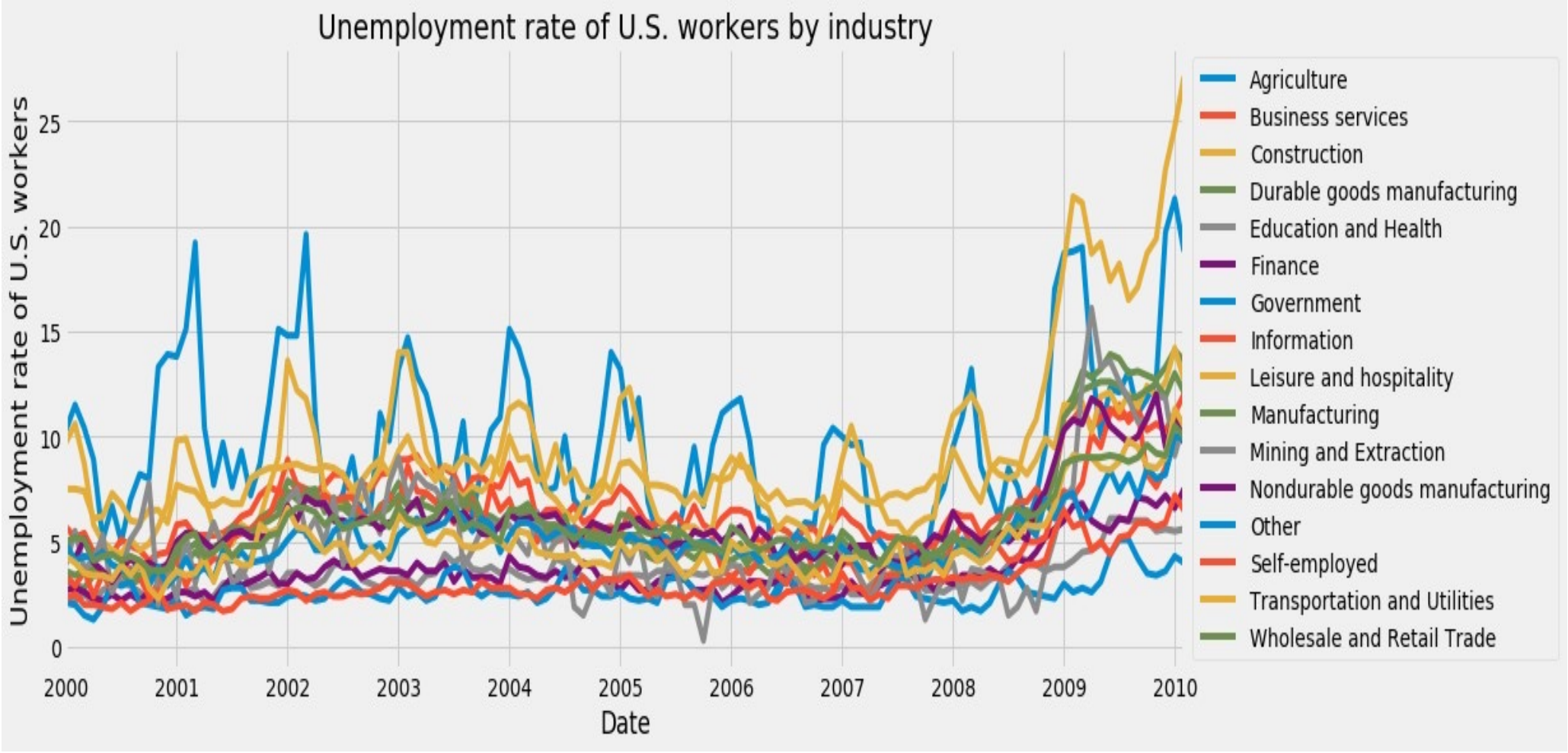
S&P 500 Stock Prices
Currency in USD

Date	Open	High	Low	Close*	Adj Close**	Volume
Oct 30, 2020	3,293.59	3,304.93	3,233.94	3,269.96	3,269.96	4,840,450,000
Oct 29, 2020	3,277.17	3,341.05	3,259.82	3,310.11	3,310.11	4,903,070,000
Oct 28, 2020	3,342.48	3,342.48	3,268.89	3,271.03	3,271.03	5,129,860,000
Oct 27, 2020	3,403.15	3,409.51	3,388.71	3,390.68	3,390.68	3,946,990,000
Oct 26, 2020	3,441.42	3,441.42	3,364.86	3,400.97	3,400.97	3,988,080,000
Oct 23, 2020	3,464.90	3,466.46	3,440.45	3,465.39	3,465.39	3,646,570,000
Oct 22, 2020	3,438.50	3,460.53	3,415.34	3,453.49	3,453.49	4,163,630,000
Oct 21, 2020	3,439.91	3,464.86	3,433.06	3,435.56	3,435.56	4,097,750,000
Oct 20, 2020	3,439.38	3,476.93	3,435.65	3,443.12	3,443.12	3,901,260,000
Oct 19, 2020	3,493.66	3,502.42	3,419.93	3,426.92	3,426.92	4,086,200,000
Oct 16, 2020	3,493.50	3,515.76	3,480.45	3,483.81	3,483.81	4,675,890,000
Oct 15, 2020	3,453.72	3,489.08	3,440.89	3,483.34	3,483.34	3,717,640,000

Source: www.finance.yahoo.com

Time Series in the Field of Data Science

Multiple Time Series



Outline

- **Line Plots**
- Summary Statistics and Diagnostics
- Seasonality, Trend and Noise
- Work with Multiple Time Series
- Case Study: Unemployment Rate

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)
```

The Matplotlib library

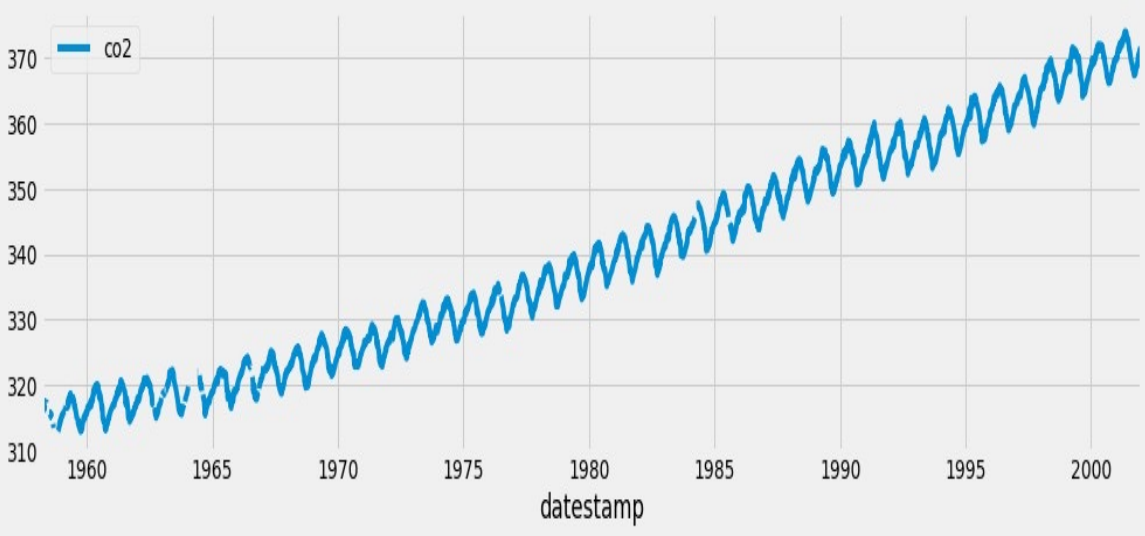
In Python, matplotlib is an extensive package used to plot data

The pyplot submodule of matplotlib is traditionally imported using the `plt` alias

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

Plotting time series data

```
import matplotlib.pyplot as plt
import pandas as pd
plt.style.use('fivethirtyeight')

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




```
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()
```

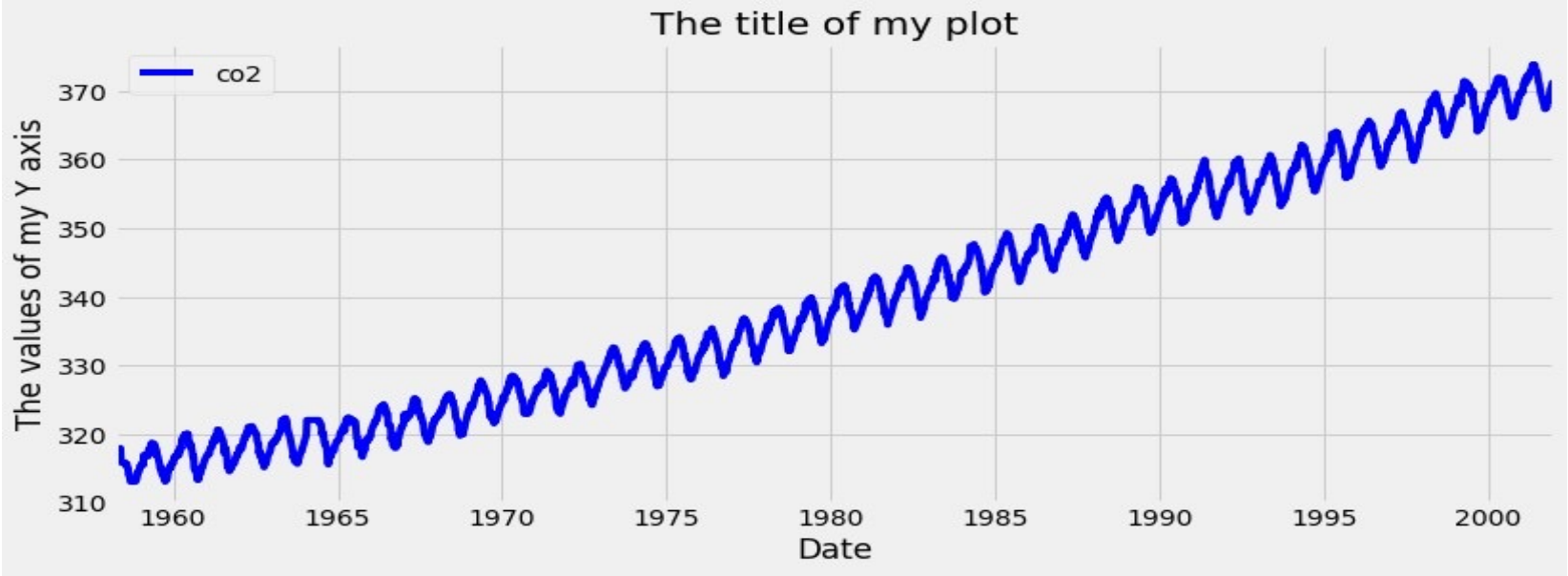
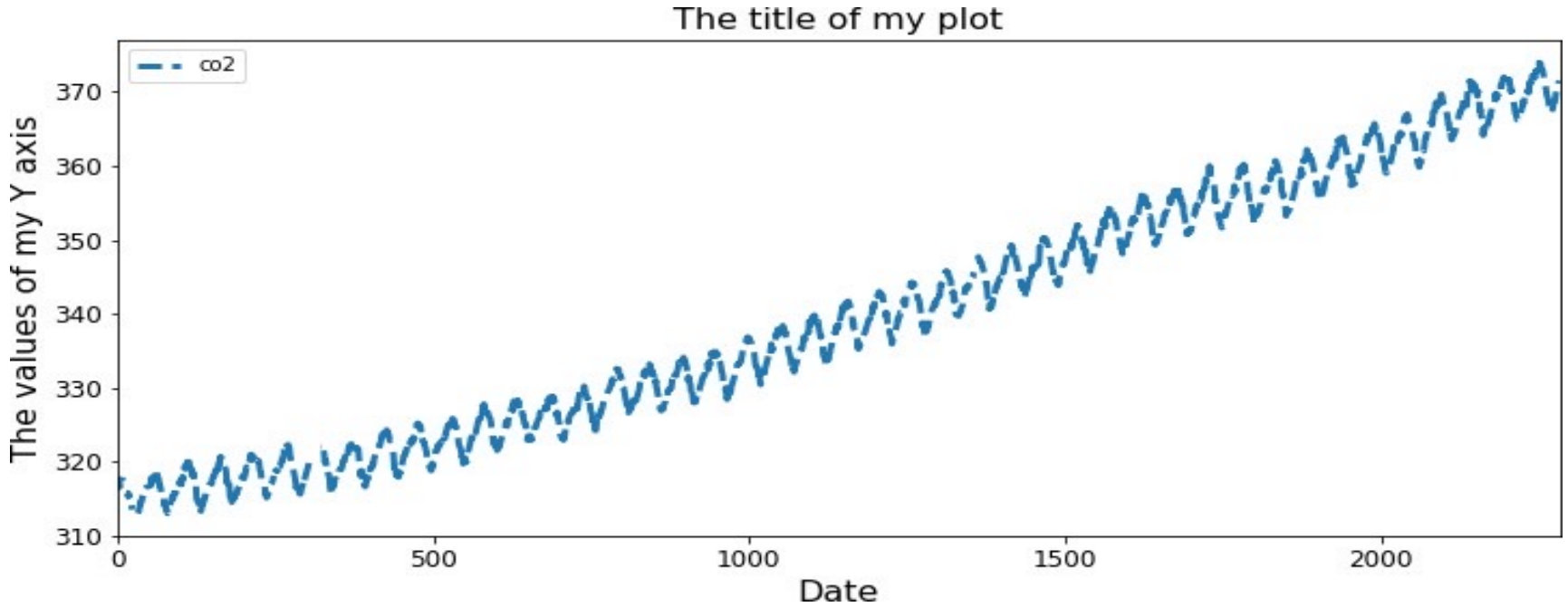


Figure size, linewidth, linestyle and fontsize

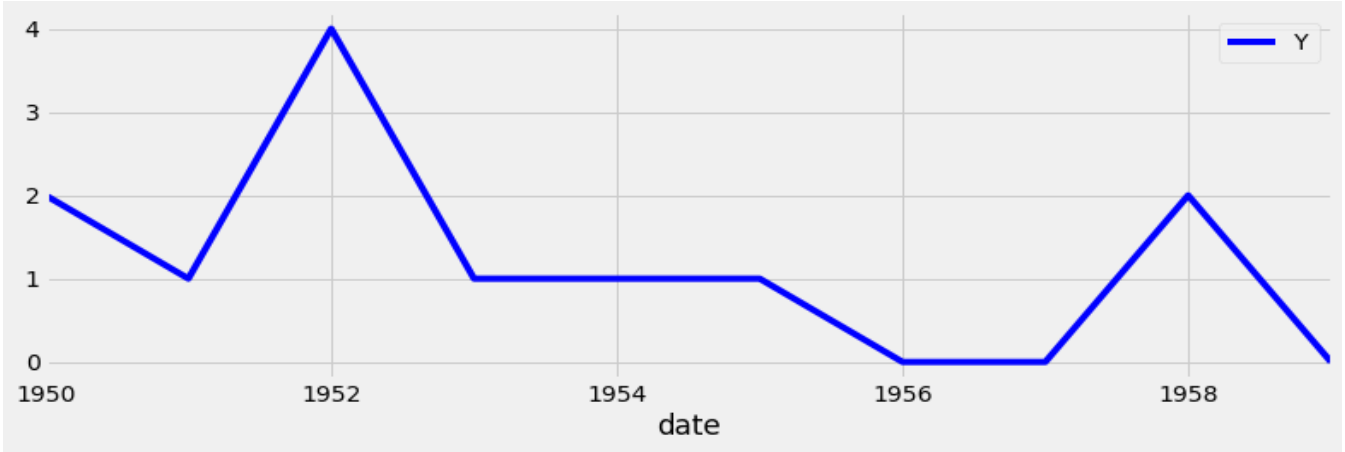
```
ax = df.plot(figsize=(12, 5), fontsize=12,
             linewidth=3, linestyle='--')
ax.set_xlabel('Date', fontsize=16)
ax.set_ylabel('The values of my Y axis', fontsize=16)
ax.set_title('The title of my plot', fontsize=16)
plt.show()
```



Plotting subset of your time series data

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight’)
df_subset = discoveries[‘1950':’1960']
```

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



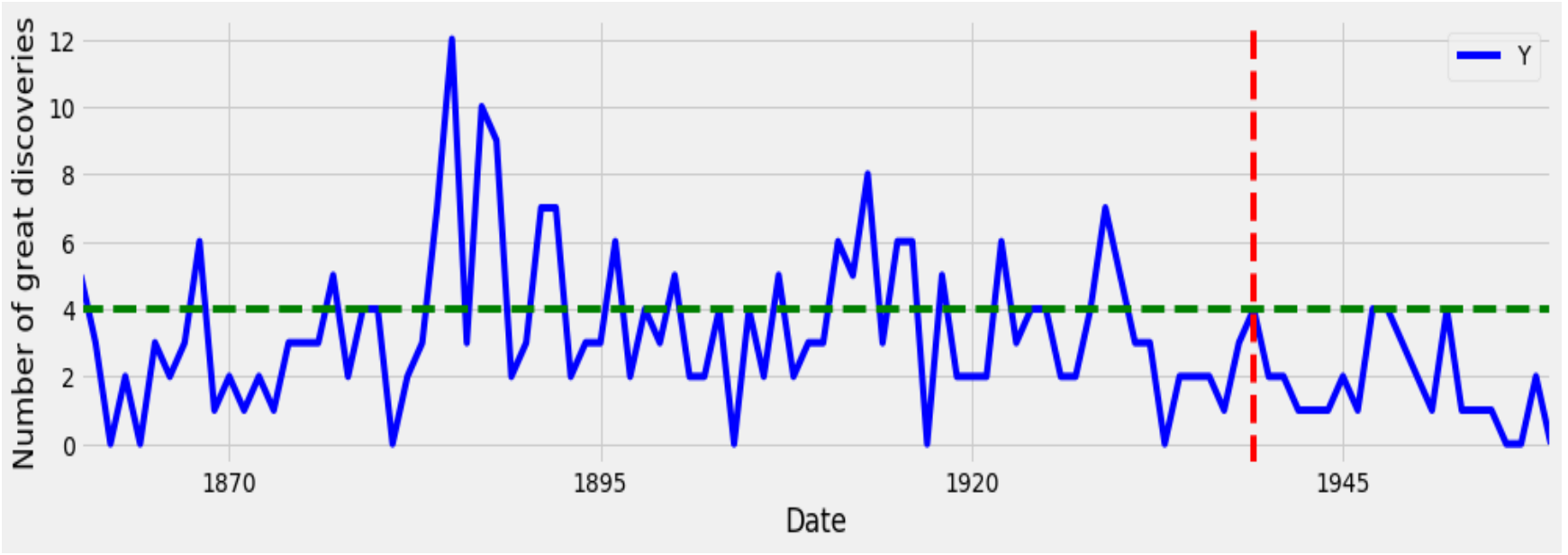

```
discoveries['1960':'1970']
```

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

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

Using markers

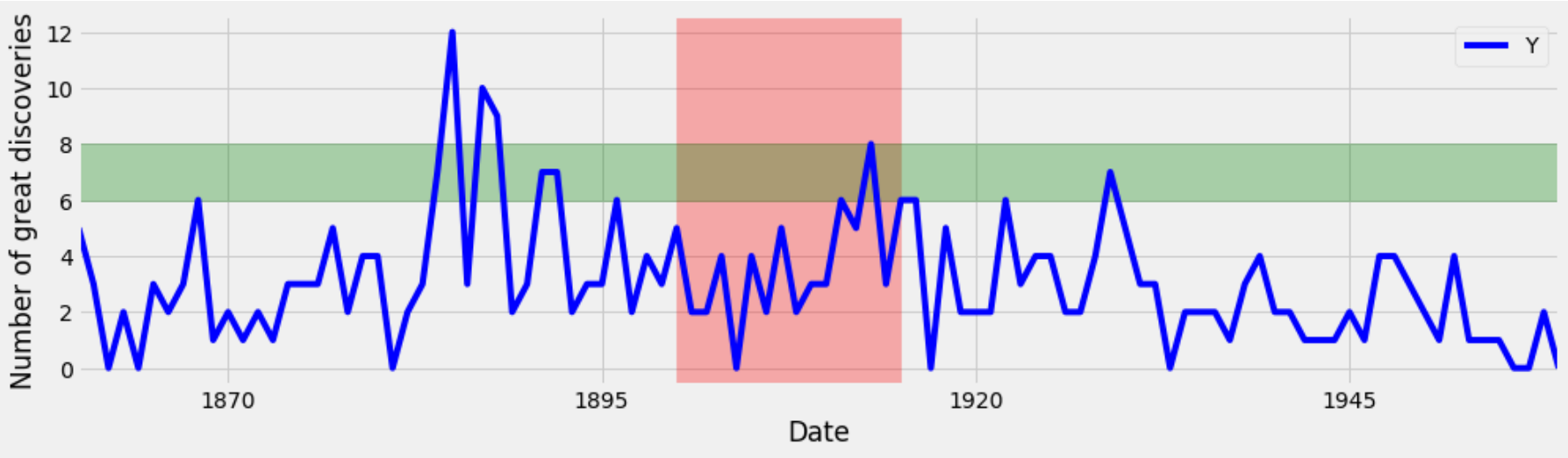
```
ax = discoveries.plot(color='blue’)
ax.set_xlabel('Date’)
ax.set_ylabel('Number of great discoveries’)
ax.axvline(‘1940-01-01', color='red', linestyle='--’)
ax.axhline(4, color='green', linestyle='--')
```



Highlighting regions of interest

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

```
ax.axvspan('1900-01-01', '1915-01-01', color='red',  alpha=0.3)
ax.axhspan(8, 6, color='green', alpha=0.3)
```



Outline

- Line Plots
- **Summary Statistics and Diagnostics**
- Seasonality, Trend and Noise
- Work with Multiple Time Series
- Case Study: Unemployment Rate
- Feature Engineering in General
- Feature Engineering for Time Series Data

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
...
```

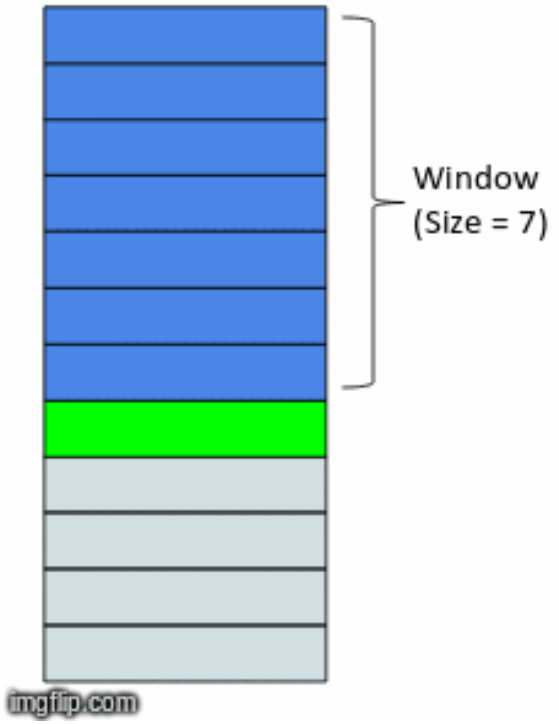
```
df = df.fillna(method='bfill')
print(df)
```

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

Moving averages

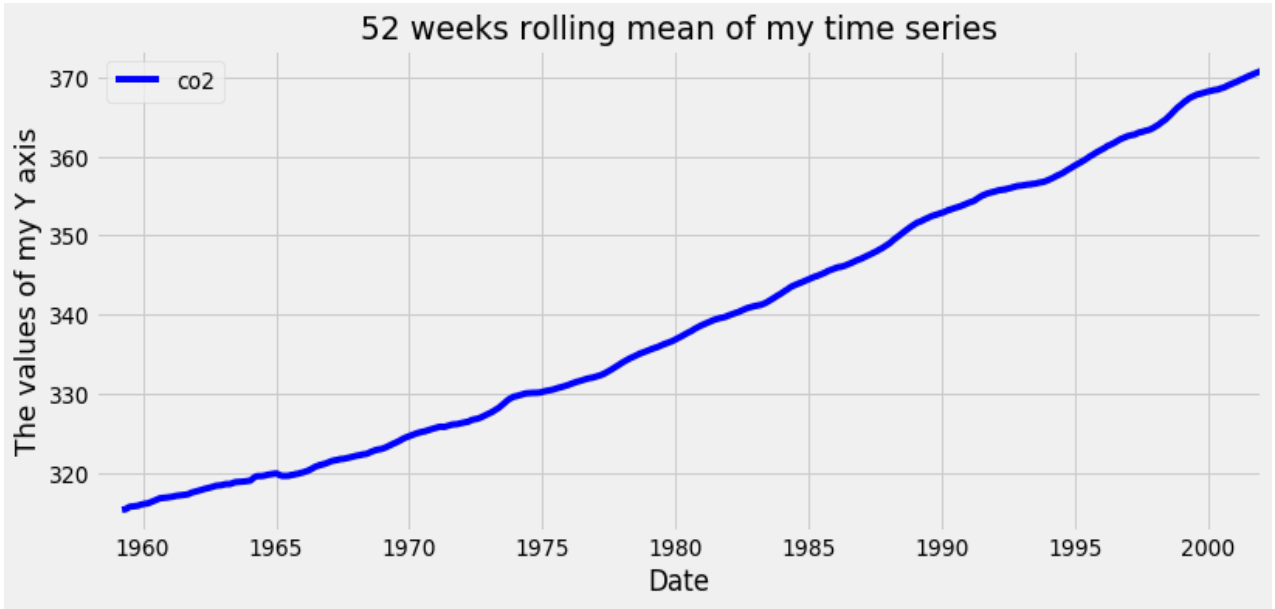
In the field of time series analysis, a moving average can be used for many different purposes:

- 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()
```



Computing aggregate values

```
co2_levels.index
```

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

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

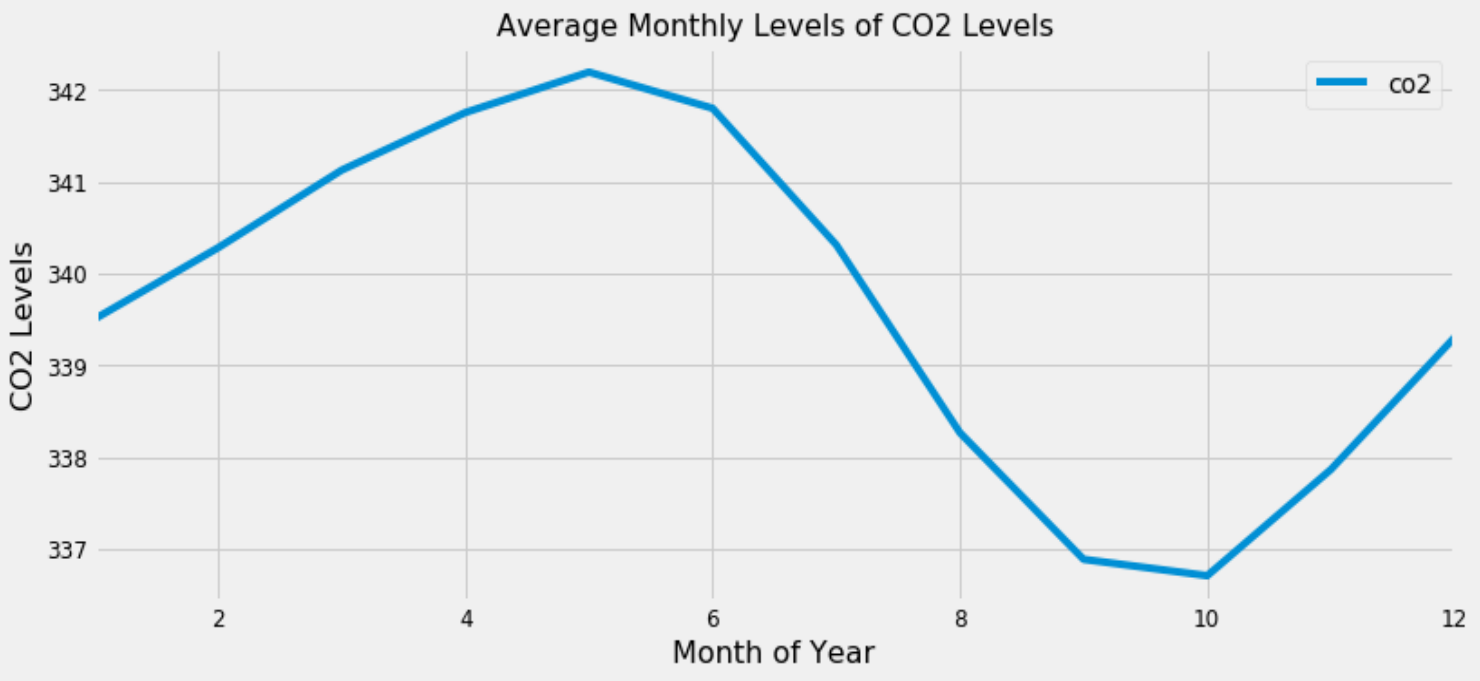
```
array([ 3,  4,  4, ..., 12, 12, 12], dtype=int32)
```

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

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

Plotting aggregate values

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

plt.show()
```



Obtaining numerical summaries

What is the **average value** of this data?

What is the **maximum value** observed in this time series?

Obtaining numerical summaries

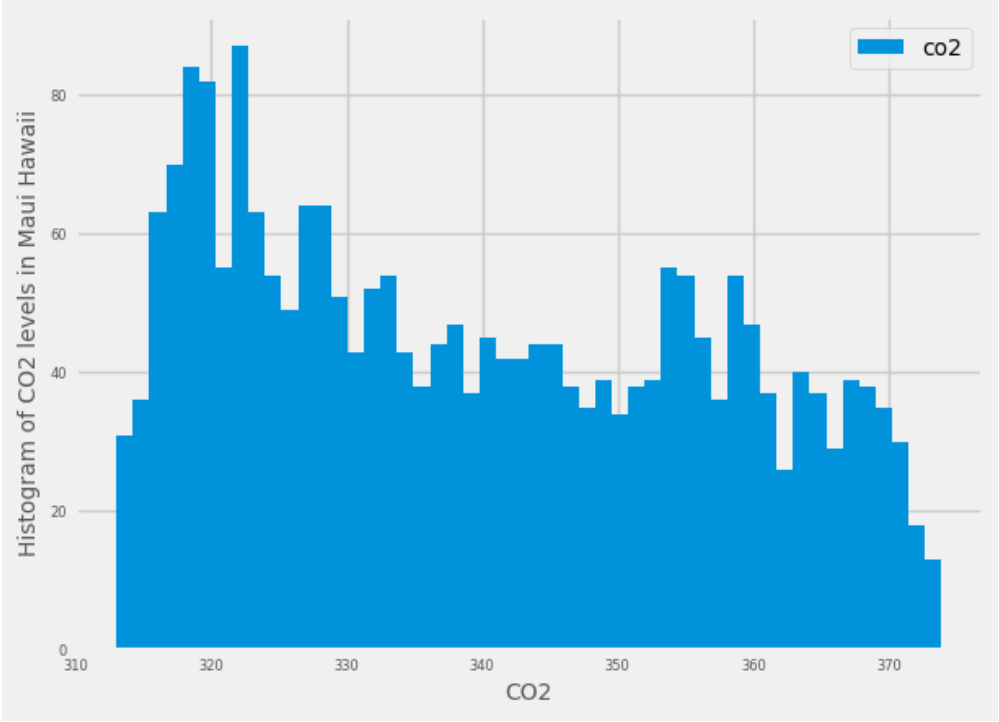
The `.describe()` method automatically computes key statistics of all numeric columns in your DataFrame

```
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
```

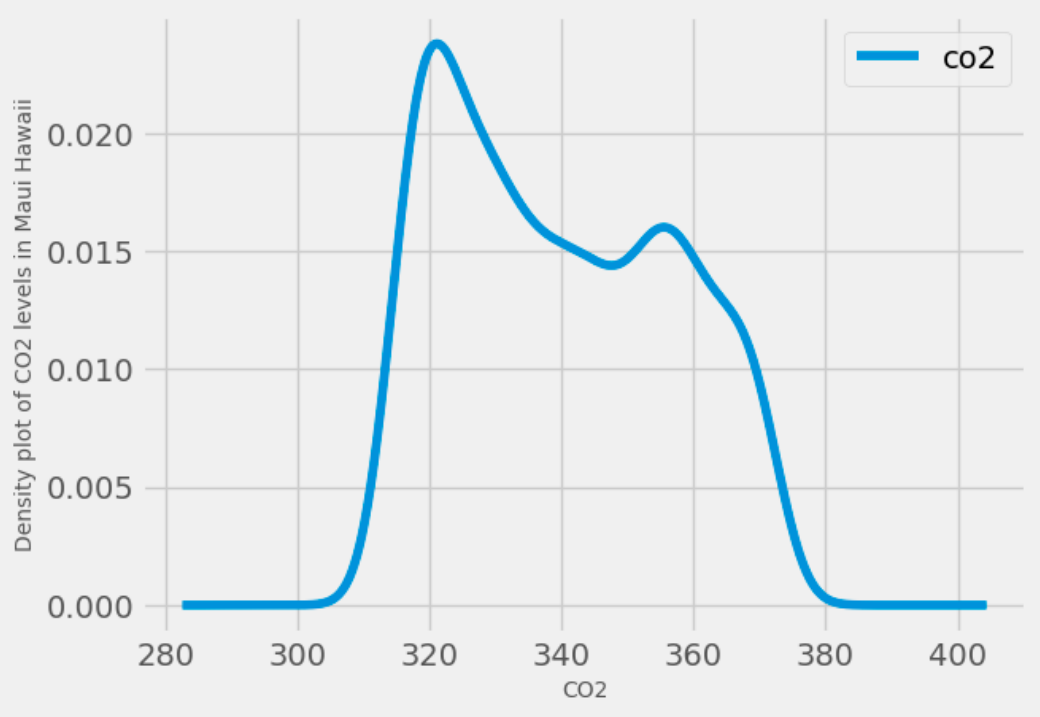
Histogram plot

```
# Generate a histogram
ax = co2_levels.plot(kind='hist', bins=50)

# Annotate labels
ax.set_xlabel('CO2', fontsize=10)
ax.set_ylabel('Histogram plot of CO2 levels in Maui Hawaii', fontsize=10)

plt.show()
```



Density plot

```
# Display density plot of CO2 levels values
ax = co2_levels.plot(kind='density', linewidth=4)

# Annotate labels
ax.set_xlabel('CO2', fontsize=10)
ax.set_ylabel('Density plot of CO2 levels in Maui Hawaii', fontsize=10)

plt.show()
```

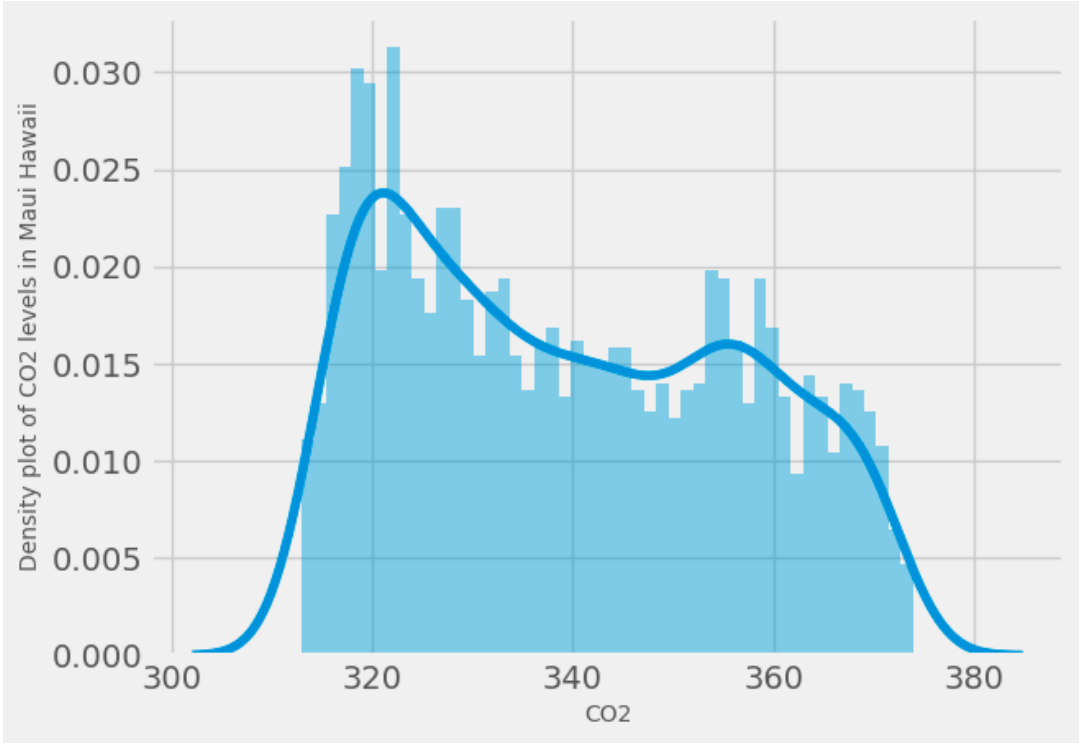


The continuous and smoothed version of the Histogram

Histogram & Density plots

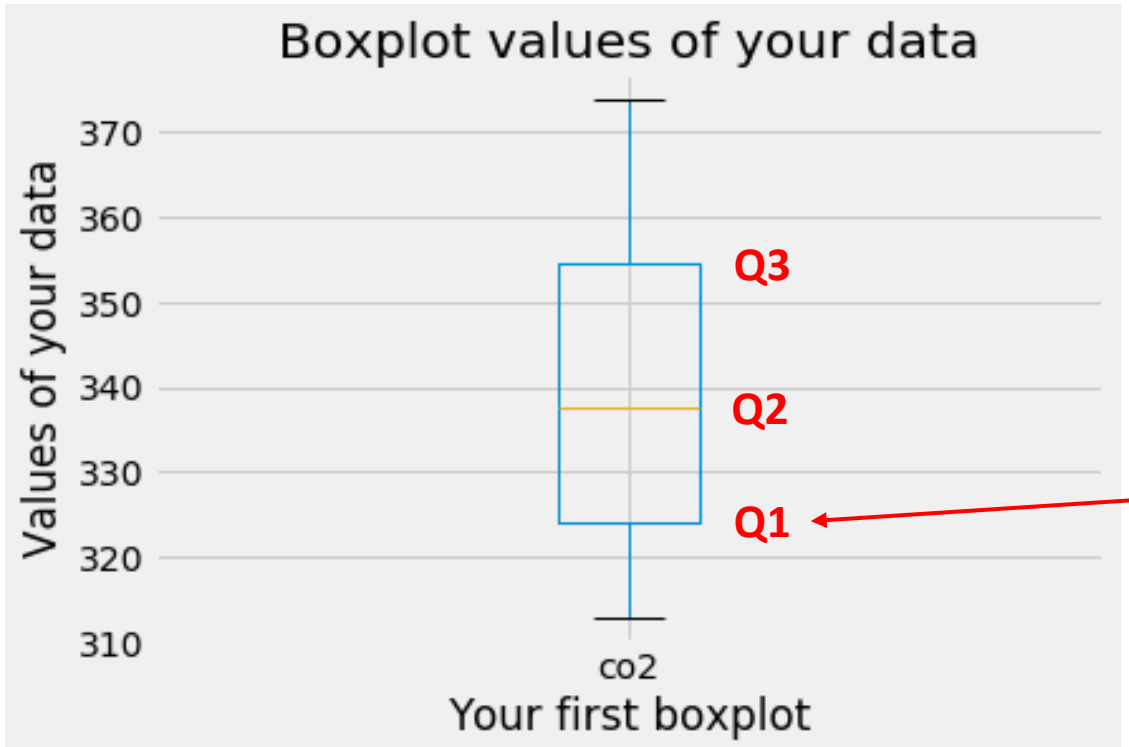
```
import seaborn as sns

ax = sns.distplot(co2_levels, hist=True, kde=True, bins=50)

# Annotate labels
ax.set_xlabel('CO2', fontsize=10)
ax.set_ylabel('Density plot of CO2 levels in Maui Hawaii', fontsize=10)

plt.show()
```

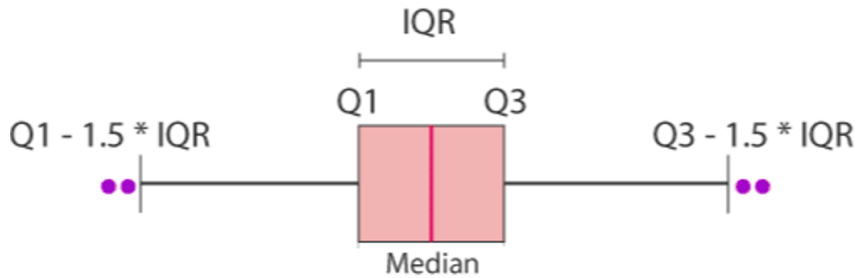
kernel density estimate



Box Plot

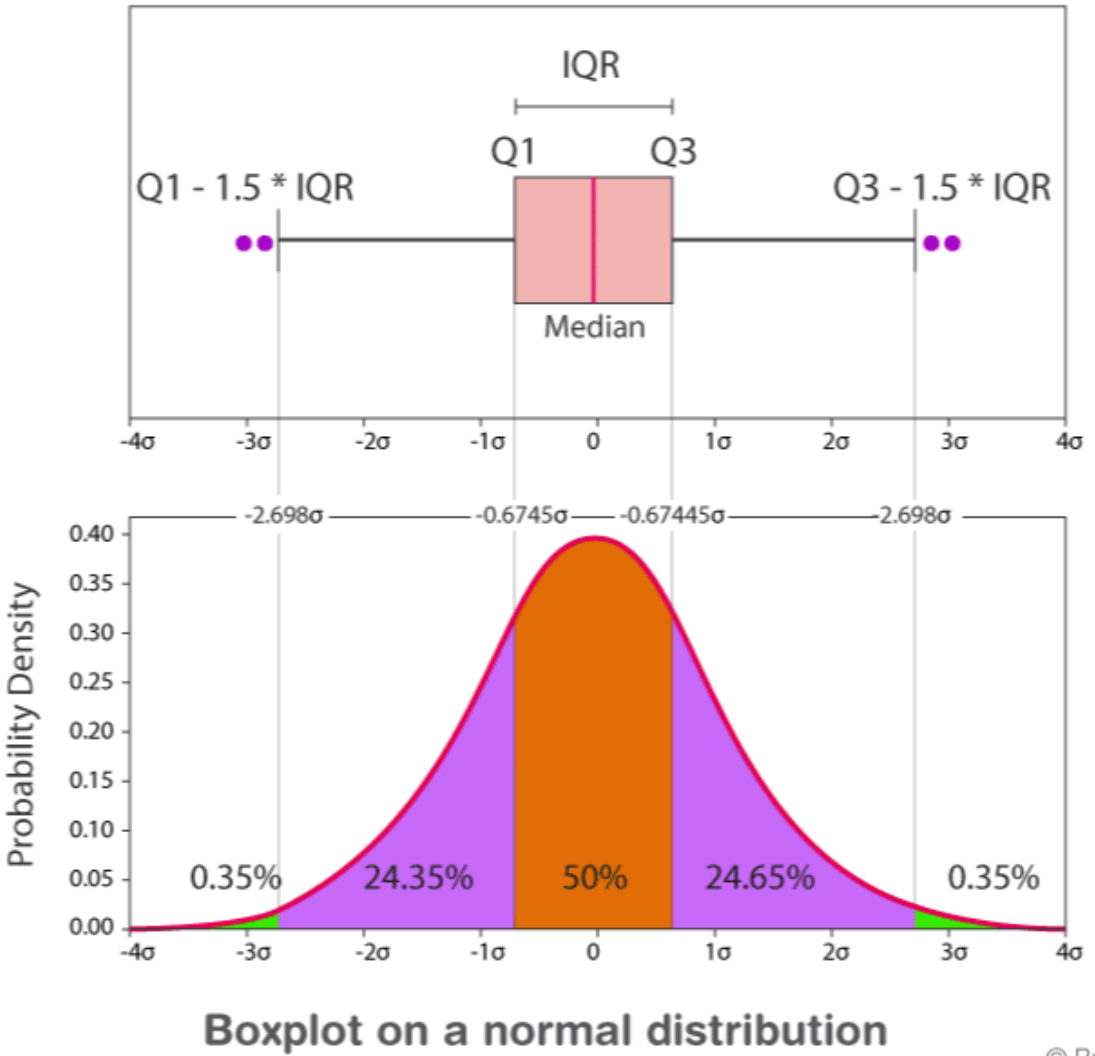
```
ax1 = df.boxplot()
ax1.set_xlabel('Your first boxplot')
ax1.set_ylabel('Values of your data')
ax1.set_title('Boxplot values of your data')

plt.show()
```

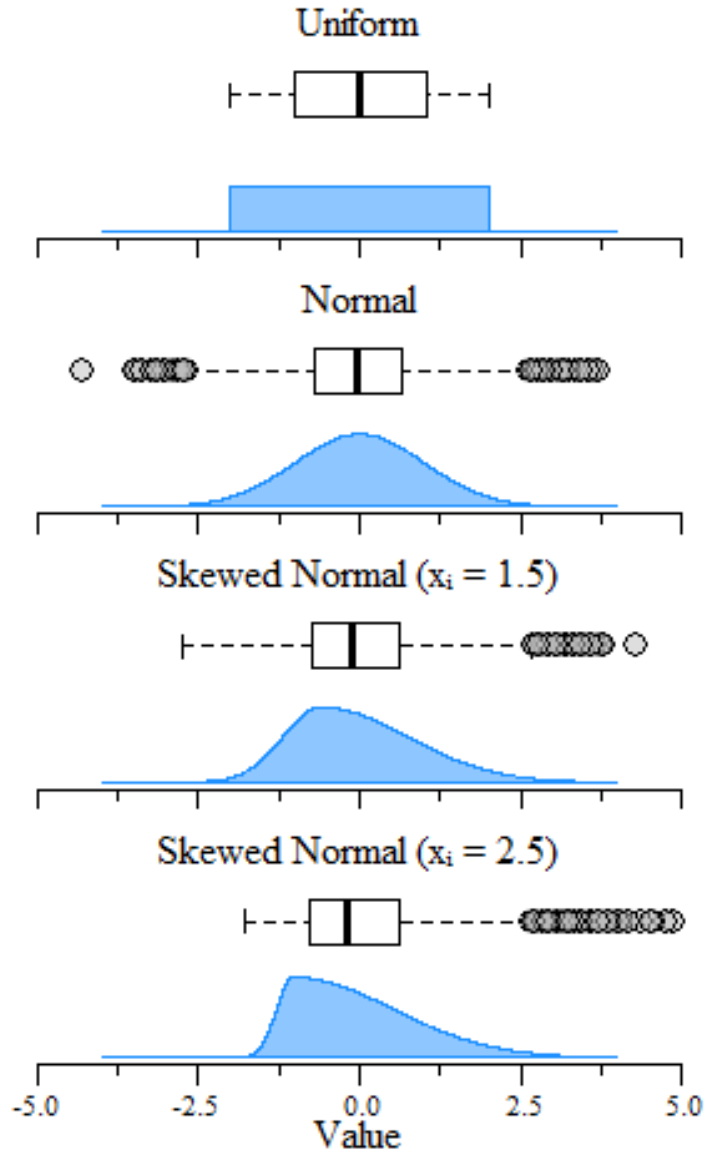




	CO2
count	2284.000000
mean	339.657750
std	17.100899
min	313.000000
25% (Q1)	323.975000
50% (Q2)	337.700000
75% (Q3)	354.500000
max	373.900000

Box Plot


IQR
Q1
Q3
Q1 - 1.5 * IQR
Q3 - 1.5 * IQR
Median
-4σ
-3σ
-2σ
-1σ
0
1σ
2σ
3σ
4σ
-2.698σ
-0.6745σ
-0.67445σ
2.698σ
Probability Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.35%
24.35%
50%
24.65%
0.35%
Boxplot on a normal distribution
© Byjus.com


Uniform
Normal
Skewed Normal ($x_i = 1.5$)
Skewed Normal ($x_i = 2.5$)
-5.0
-2.5
0.0
2.5
5.0
Value

Outline

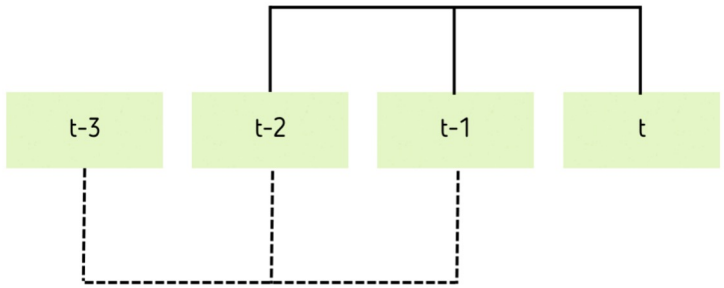
- Line Plots
- Summary Statistics and Diagnostics
- **Seasonality, Trend and Noise**
- Work with Multiple Time Series
- Case Study: Unemployment Rate

Statsmodels

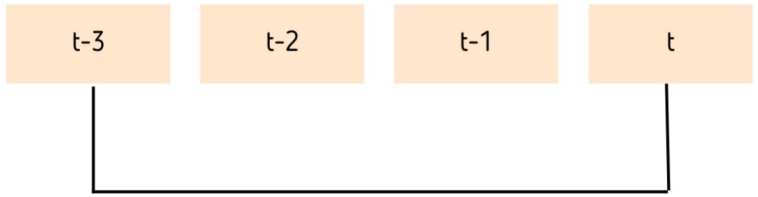
`Statsmodels` is a Python module that provides classes and functions for the estimation of many di erent statistical models, as well as for conducting **statistical tests**, and **statistical data exploration**.

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



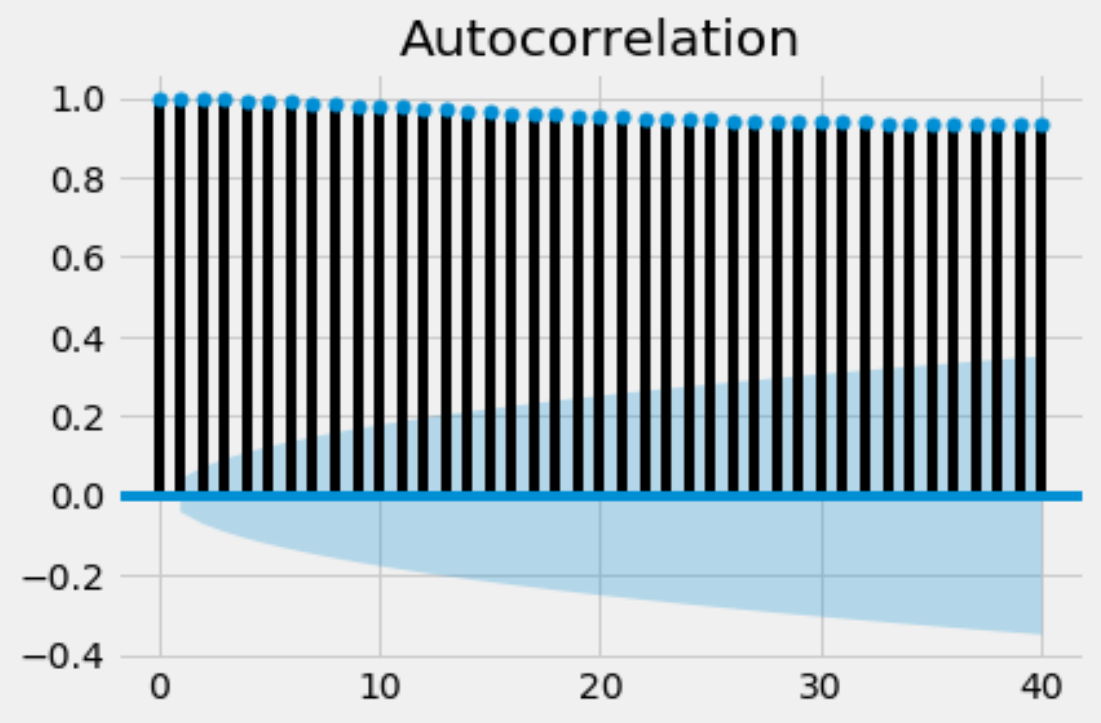
Indirect



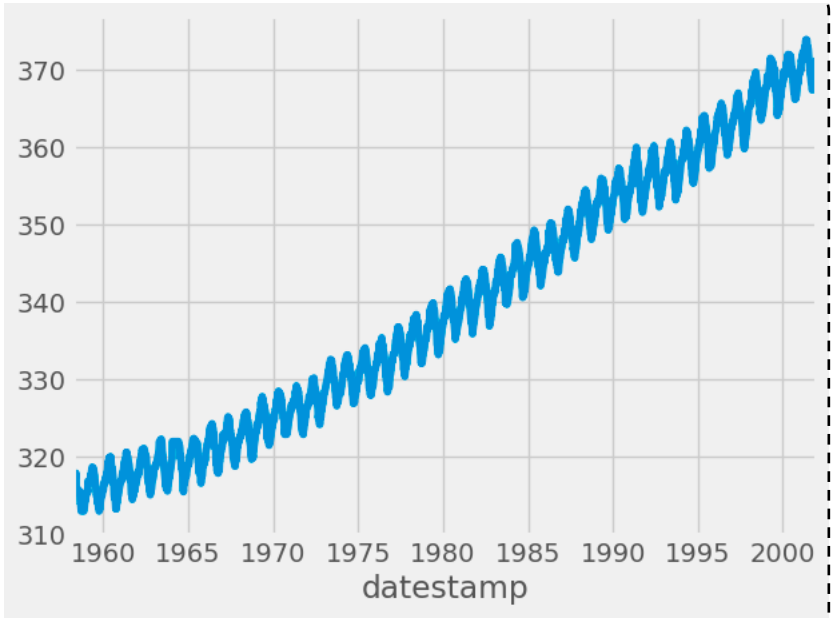
Direct

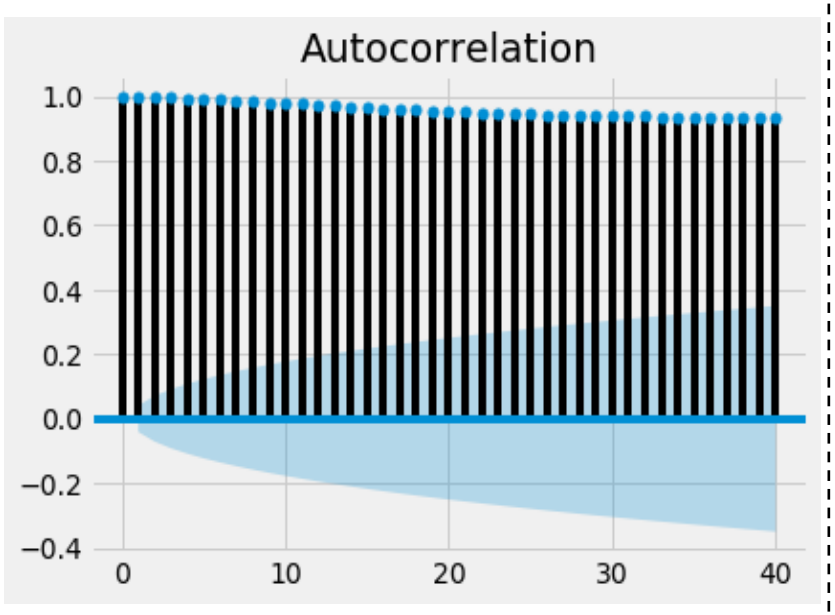
Plotting Autocorrelations

```
import matplotlib.pyplot as plt

from statsmodels.graphics import tsaplots

fig = tsaplots.plot_acf(co2_levels['co2'], lags=40)


plt.show()
```

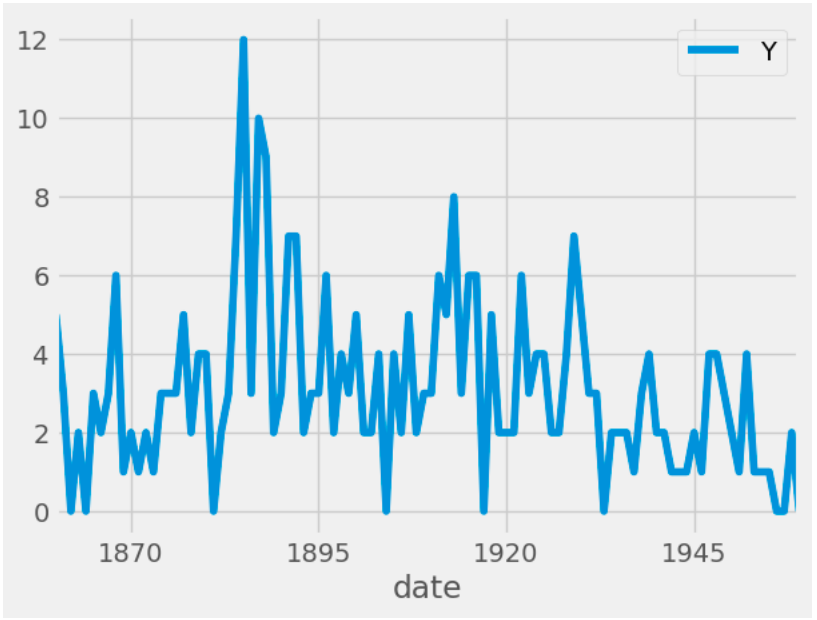


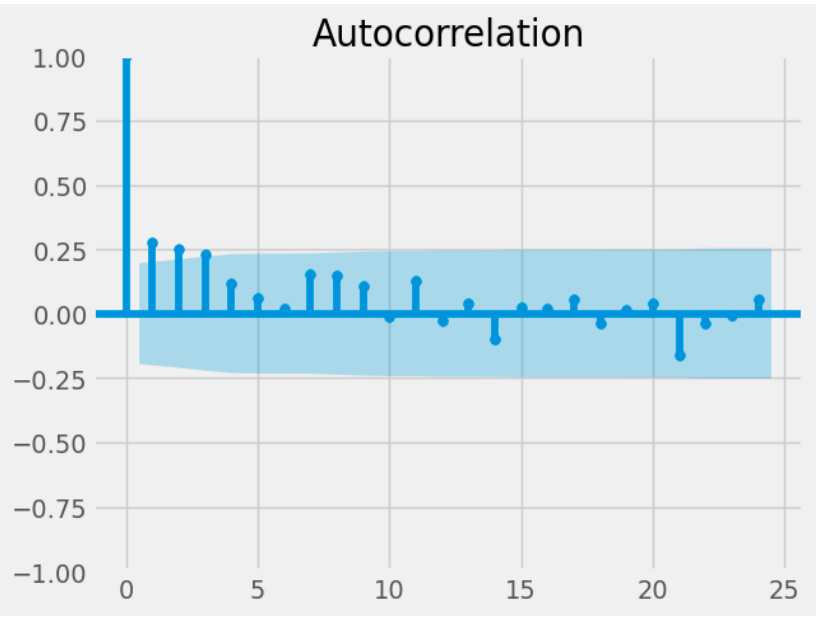
ACF Plot
(Autocorrelations Function)





ch2_co2_levels.csv

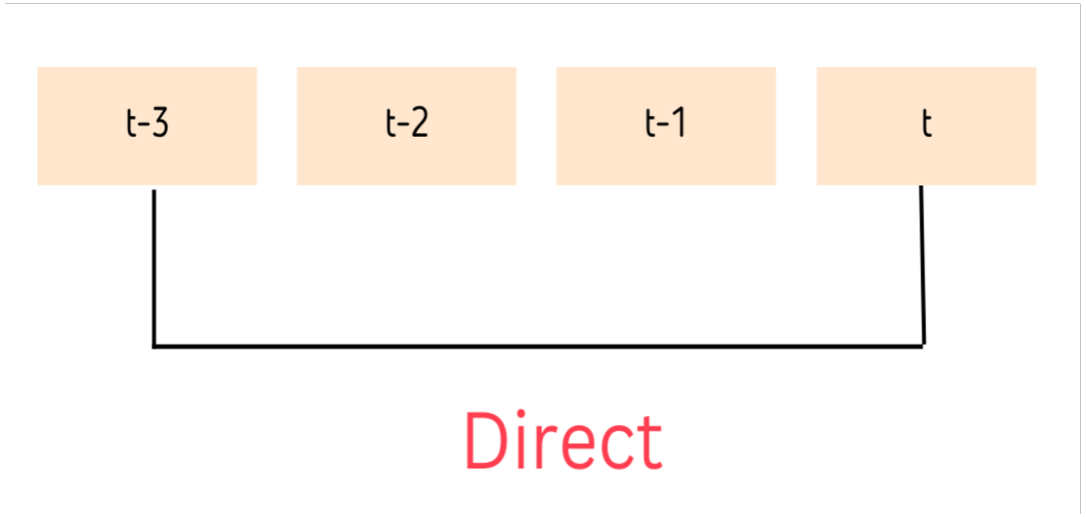




ch1_discoveries.csv

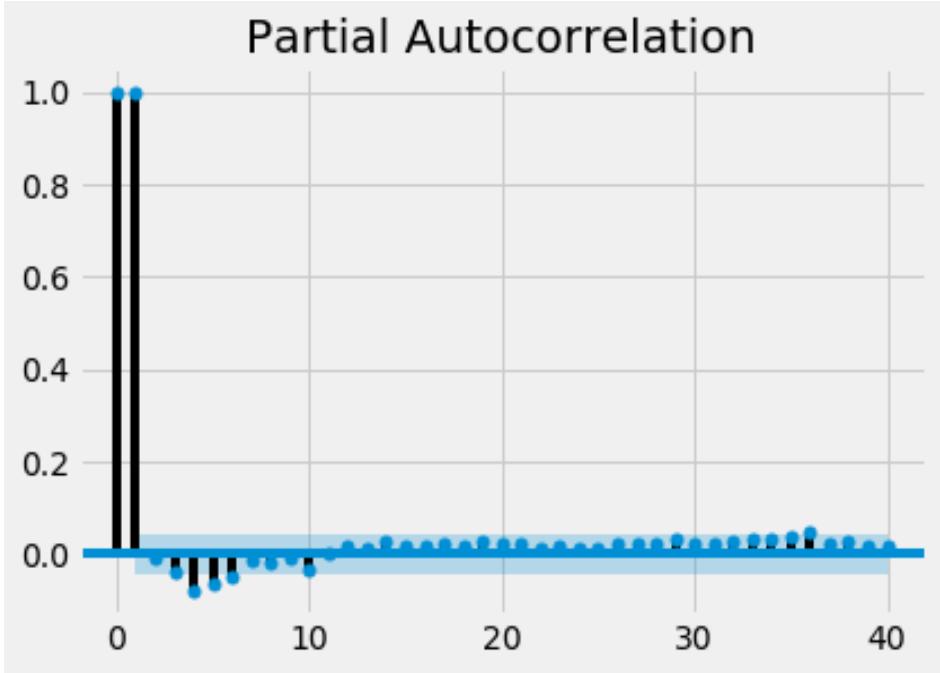
Partial autocorrelation in time series data

- 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 (`t_1`, `t_2`, `t_3`, ...) and lagged values of itself by 3 time points (`t_4`, `t_5`, `t_6`, ...), but only after removing all effects attributable to lags 1 and 2.



Plotting Partial Autocorrelations

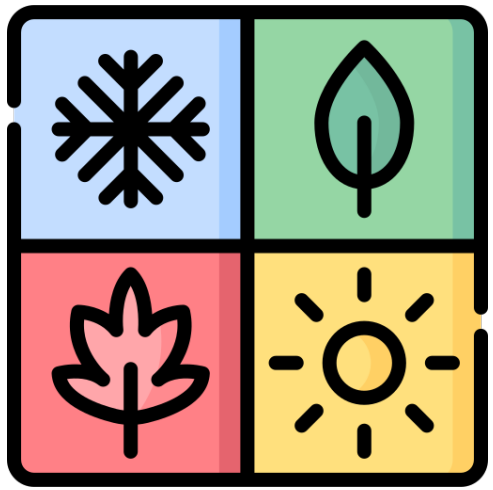
```
import matplotlib.pyplot as plt

from statsmodels.graphics import tsaplots


fig = tsaplots.plot_pacf(co2_levels['co2'], lags=40)

plt.show()
```

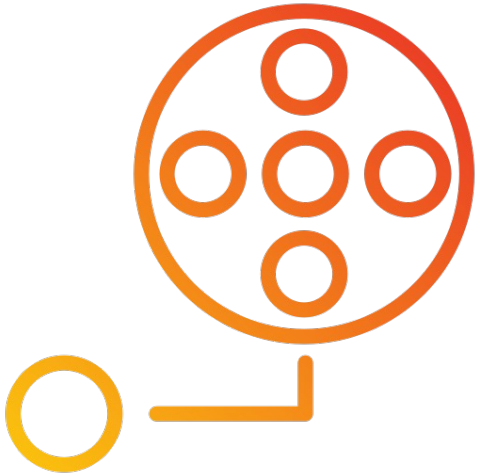


PACF Plot
(Partial Autocorrelations Function)

Properties of time series

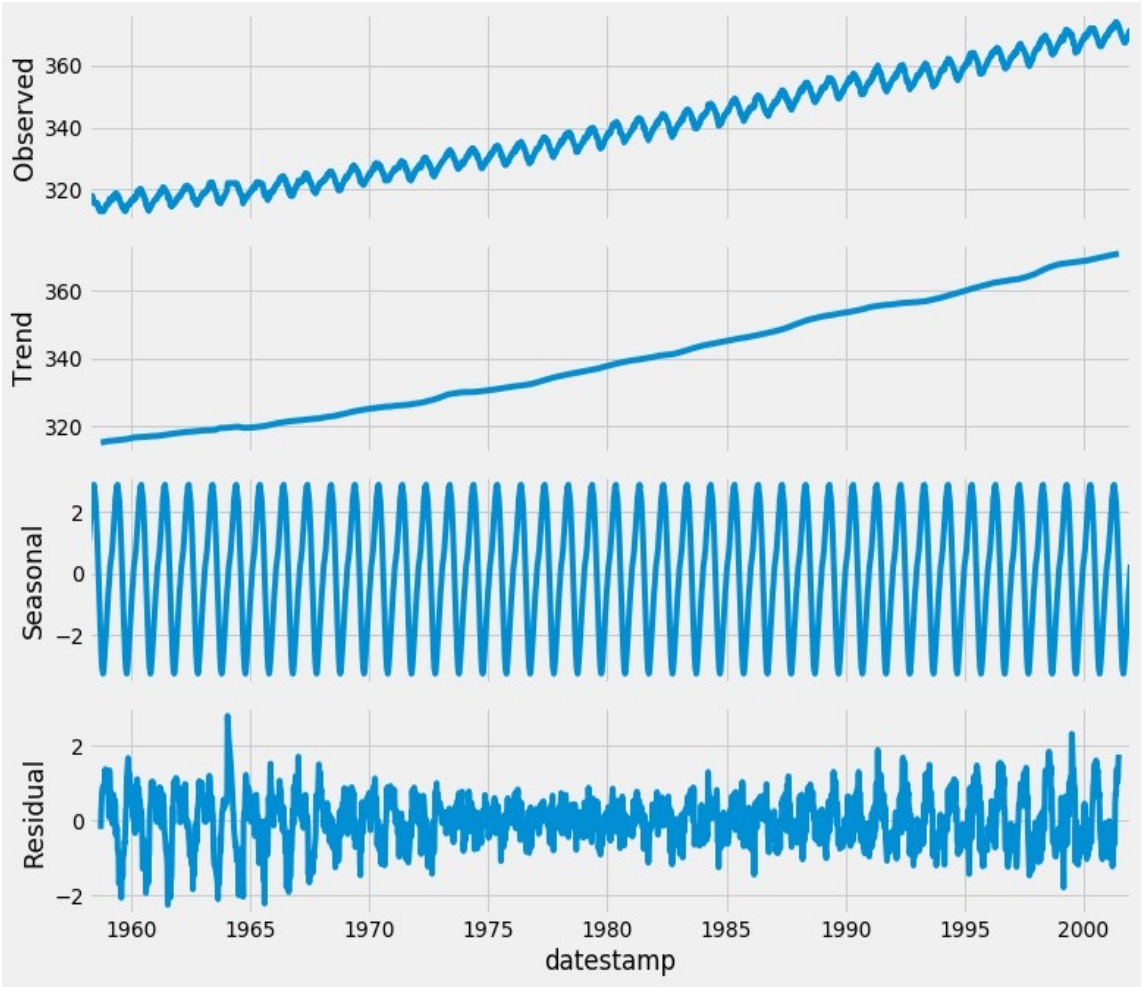
- **Seasonality**: does the data display a clear **periodic pattern**?
- **Trend**: does the data follow a **consistent** upwards or downwards slope?
- **Noise**: are there any **outlier** points or **missing** values that are not consistent with the rest of the data?





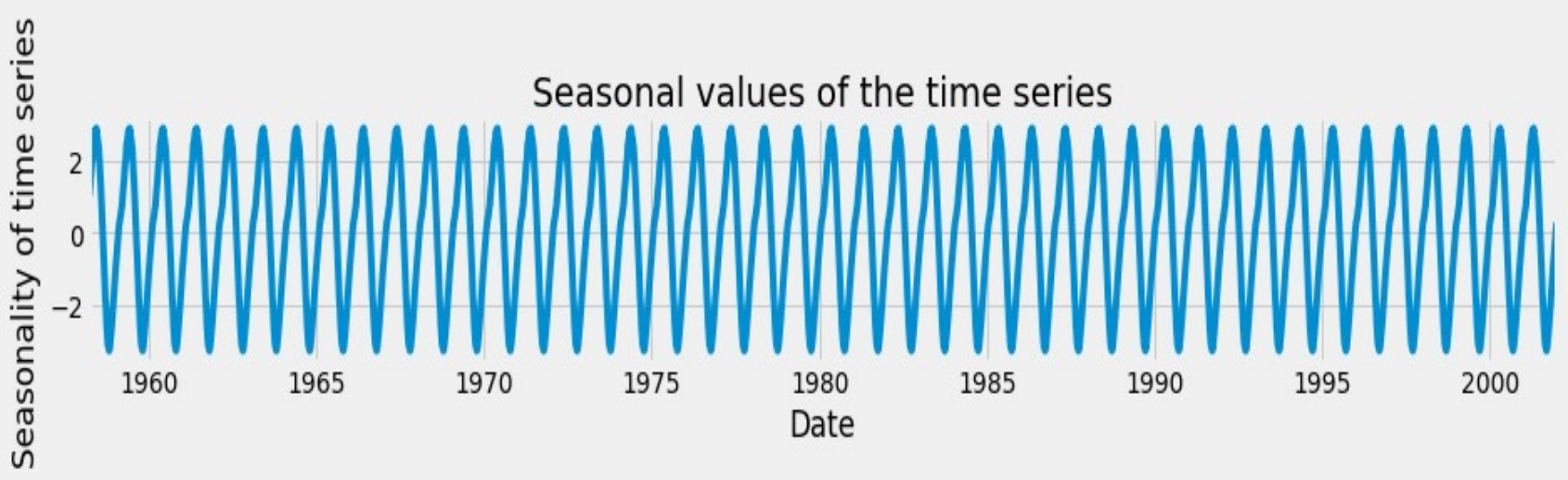
Time Series Decomposition

```
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()
```



Seasonality Component

```
decomposition = sm.tsa.seasonal_decompose(
                        co2_levels['co2']
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()
```



Outline

- Line Plots
- Summary Statistics and Diagnostics
- Seasonality, Trend and Noise
- **Work with Multiple Time Series**
- Case Study: Unemployment Rate

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

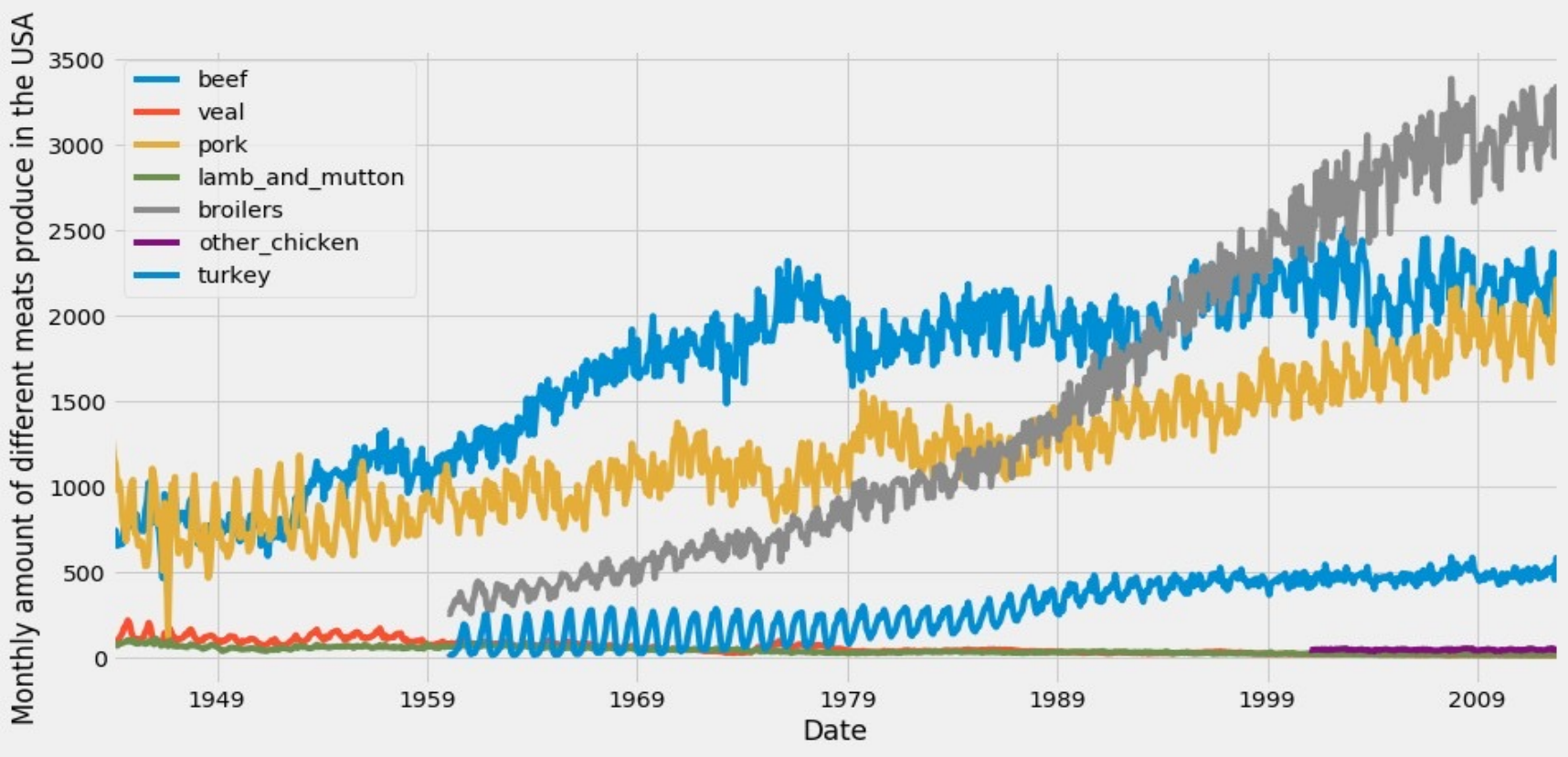
The Meat Production Dataset

```
import pandas as pd
meat = pd.read_csv("meat.csv")
print(meat.head(10))
```

date	beef	veal	pork	lamb_and_mutton	broilers	other_chicken	turkey
1944-01-01	751.0	85.0	1280.0	89.0	NaN	NaN	NaN
1944-02-01	713.0	77.0	1169.0	72.0	NaN	NaN	NaN
1944-03-01	741.0	90.0	1128.0	75.0	NaN	NaN	NaN
1944-04-01	650.0	89.0	978.0	66.0	NaN	NaN	NaN
1944-05-01	681.0	106.0	1029.0	78.0	NaN	NaN	NaN
1944-06-01	658.0	125.0	962.0	79.0	NaN	NaN	NaN
1944-07-01	662.0	142.0	796.0	82.0	NaN	NaN	NaN
1944-08-01	787.0	175.0	748.0	87.0	NaN	NaN	NaN
1944-09-01	774.0	182.0	678.0	91.0	NaN	NaN	NaN
1944-10-01	834.0	215.0	777.0	100.0	NaN	NaN	NaN

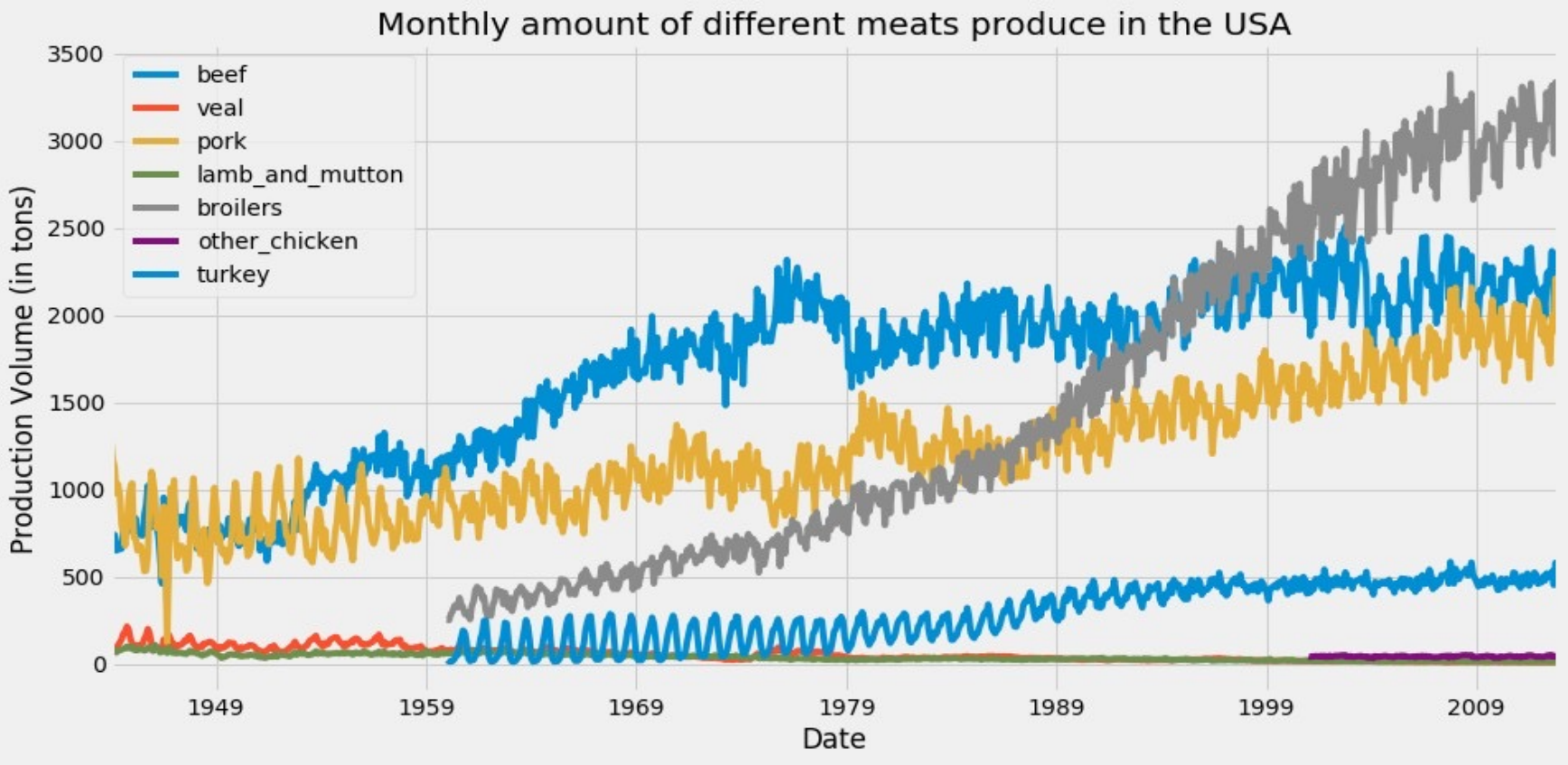
Summarizing and Plotting Multiple Time Series

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

plt.show()
```



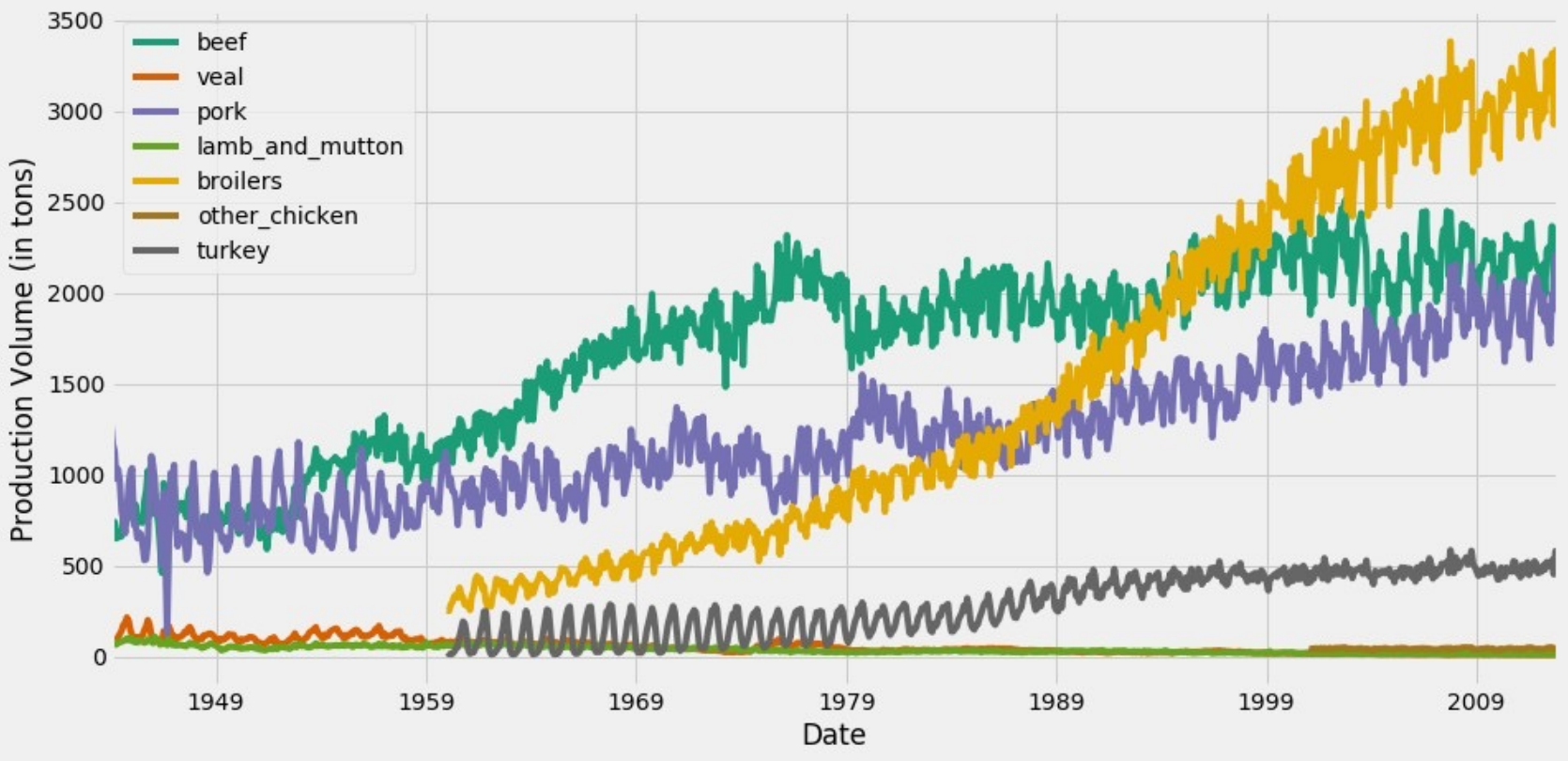
Clarity is key

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()
```



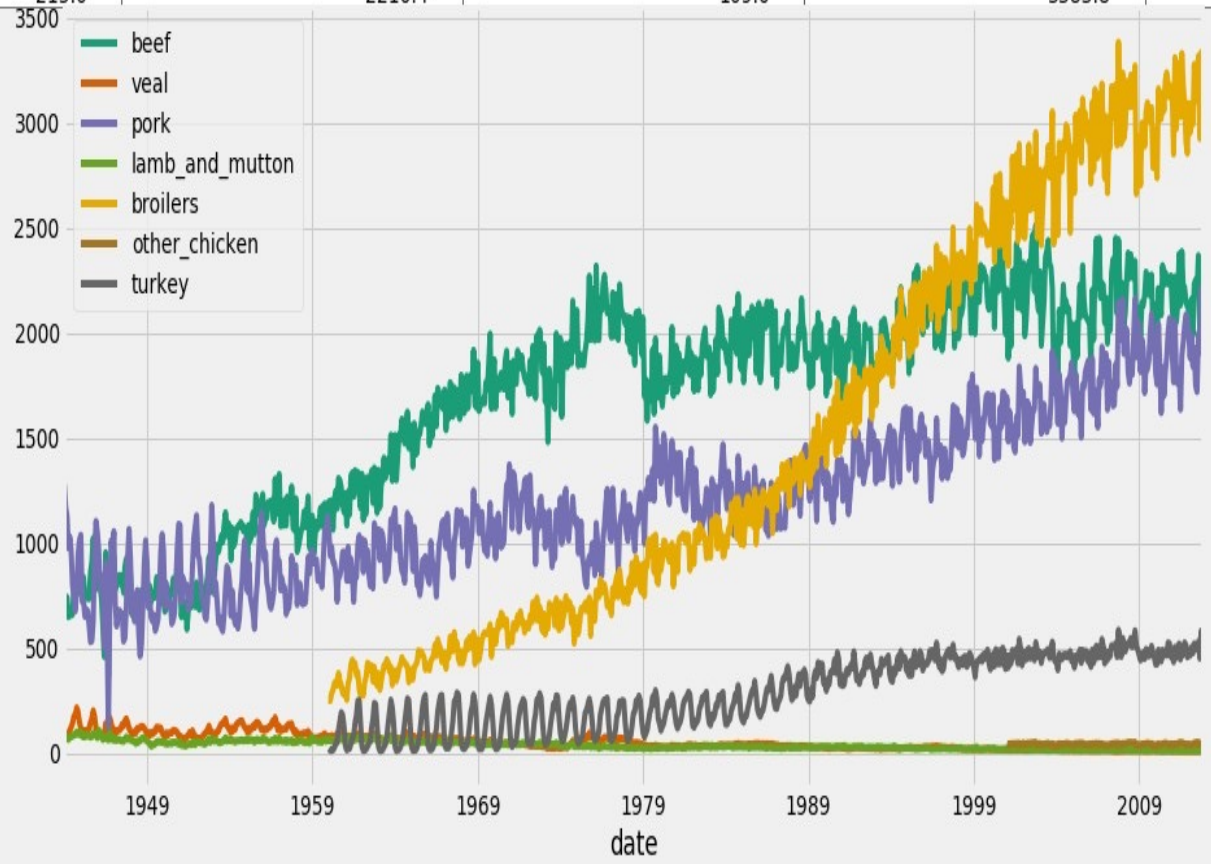
https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html

Enhancing your plot with information

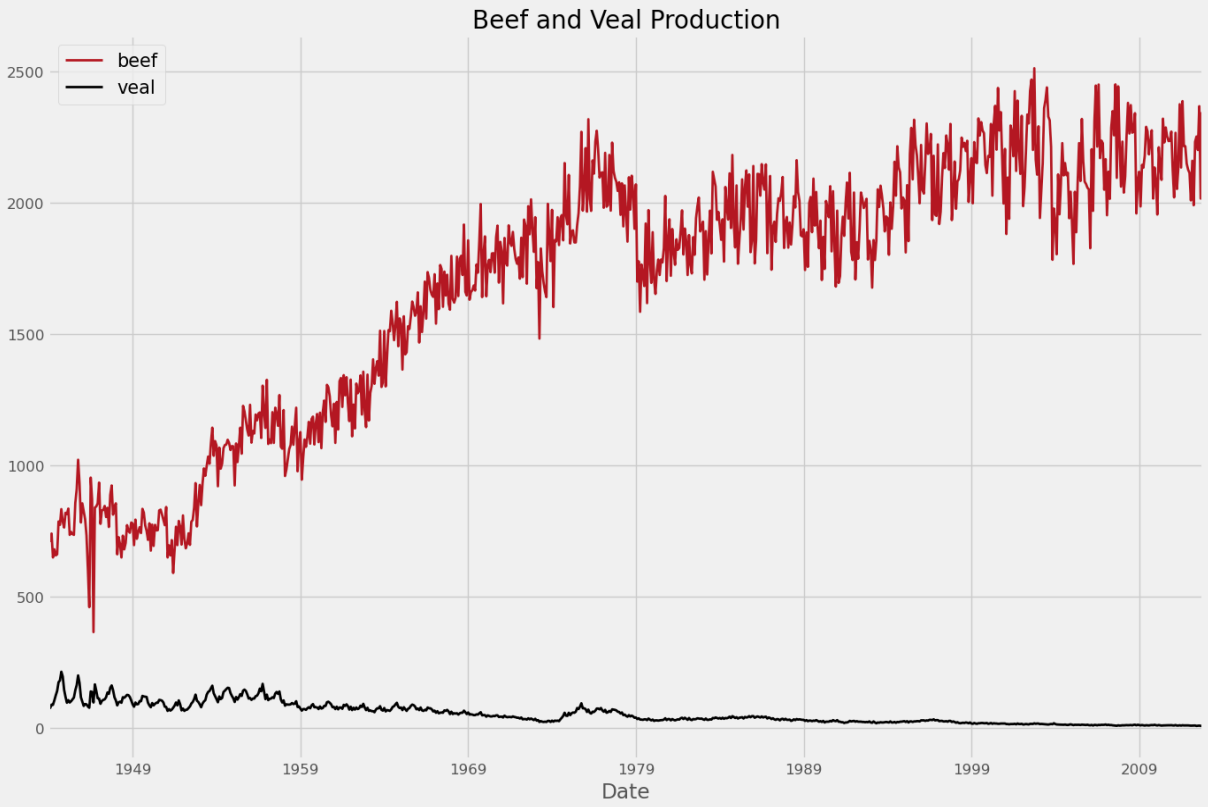
```
ax = df.plot(colormap='Dark2', figsize=(14, 7))
df_summary = df.describe()

# 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()
```

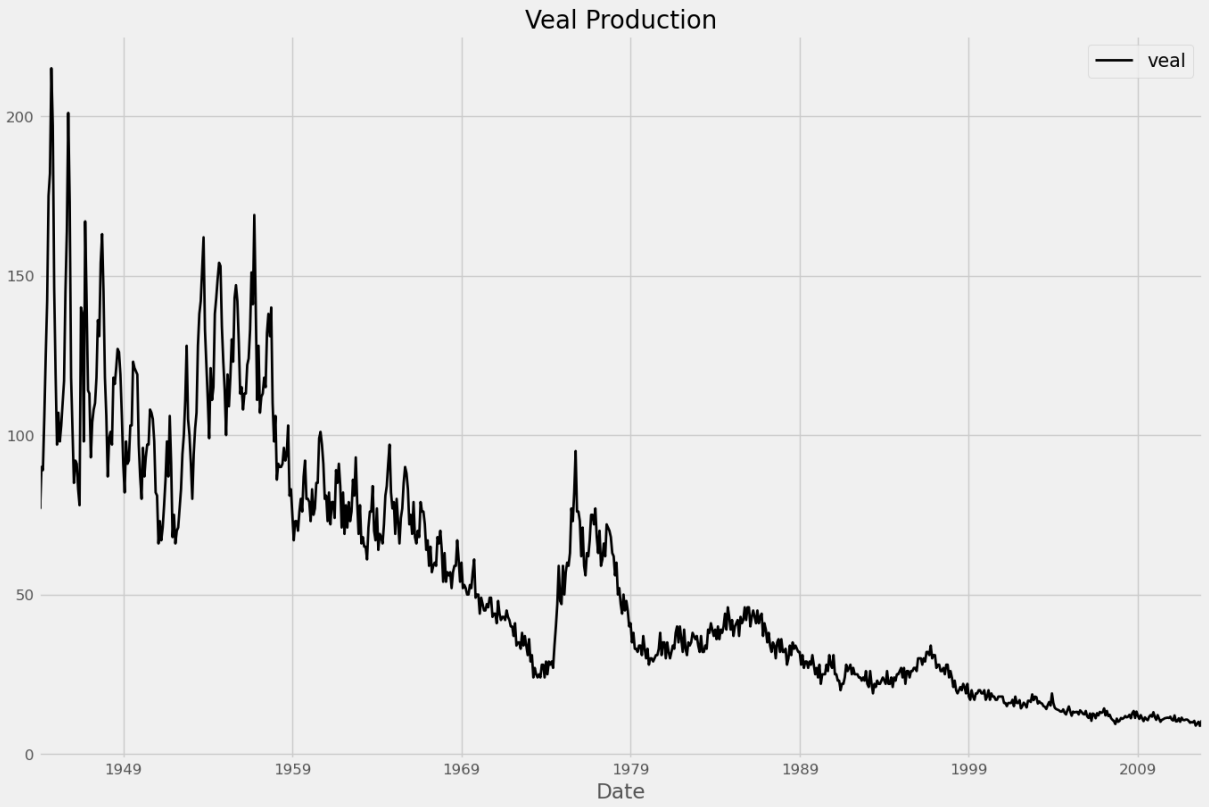
Adding Statistical summaries to your plots

	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



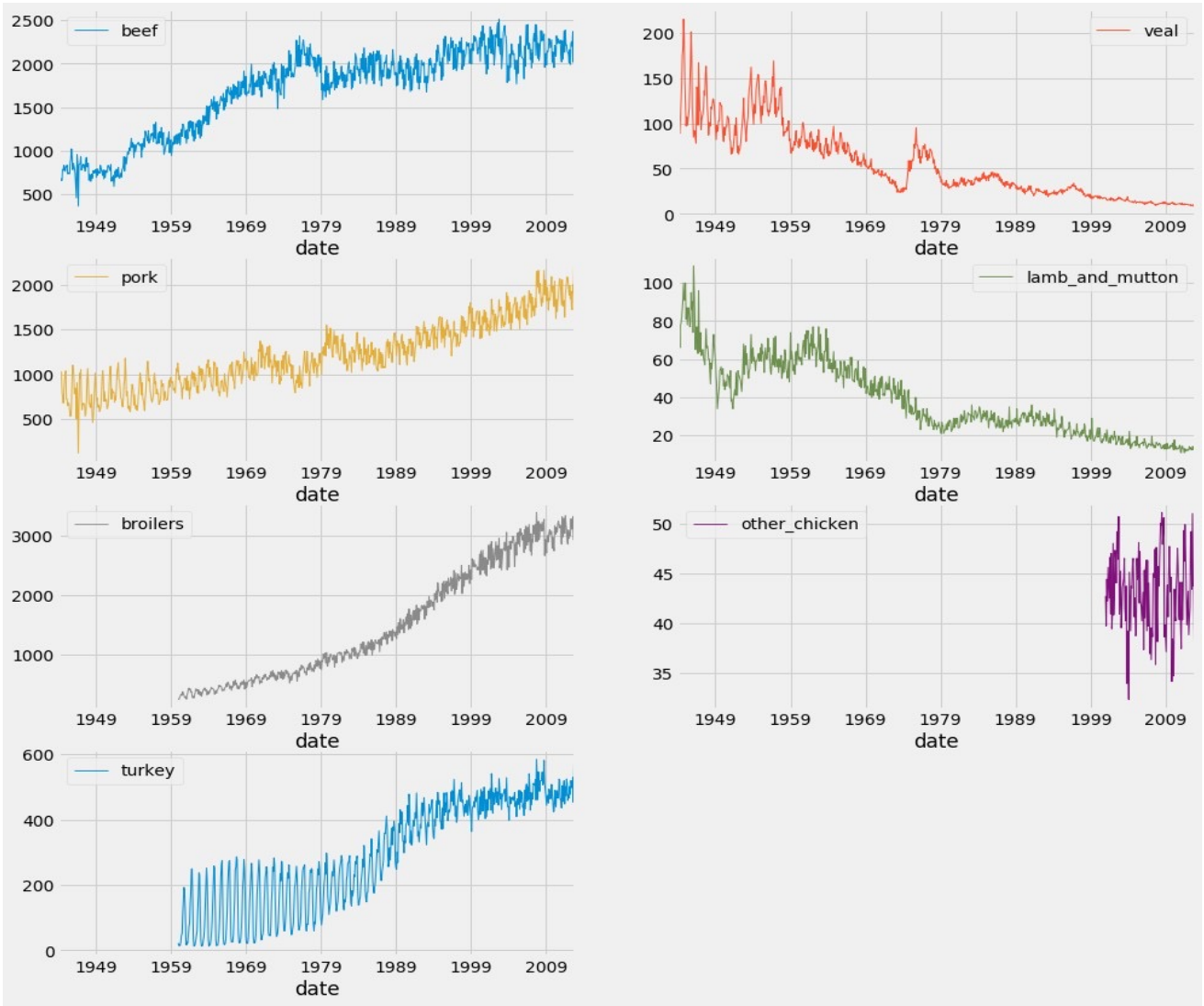
Dealing with different scales





Facet plots

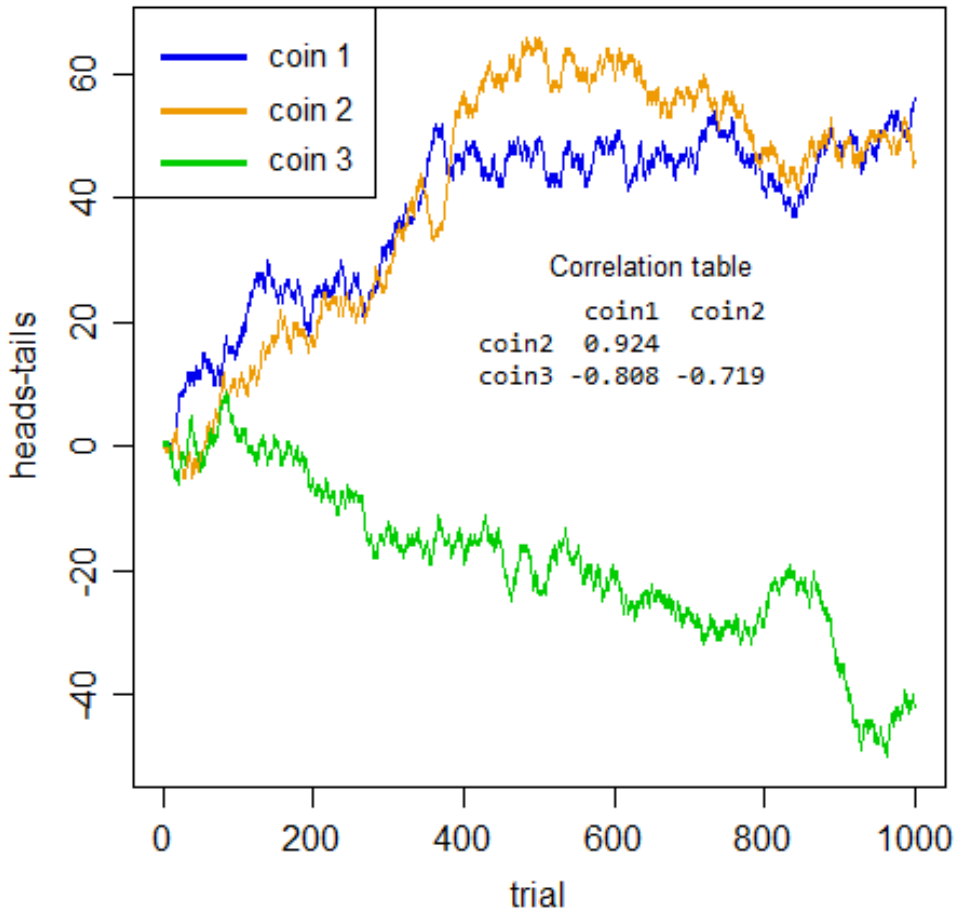
```
df.plot(subplots=True,
        linewidth=0.5,
        layout=(2, 4),
        figsize=(16, 10),
        sharex=False,
        sharey=False)

plt.show()
```



Correlations between two variables

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

```
          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

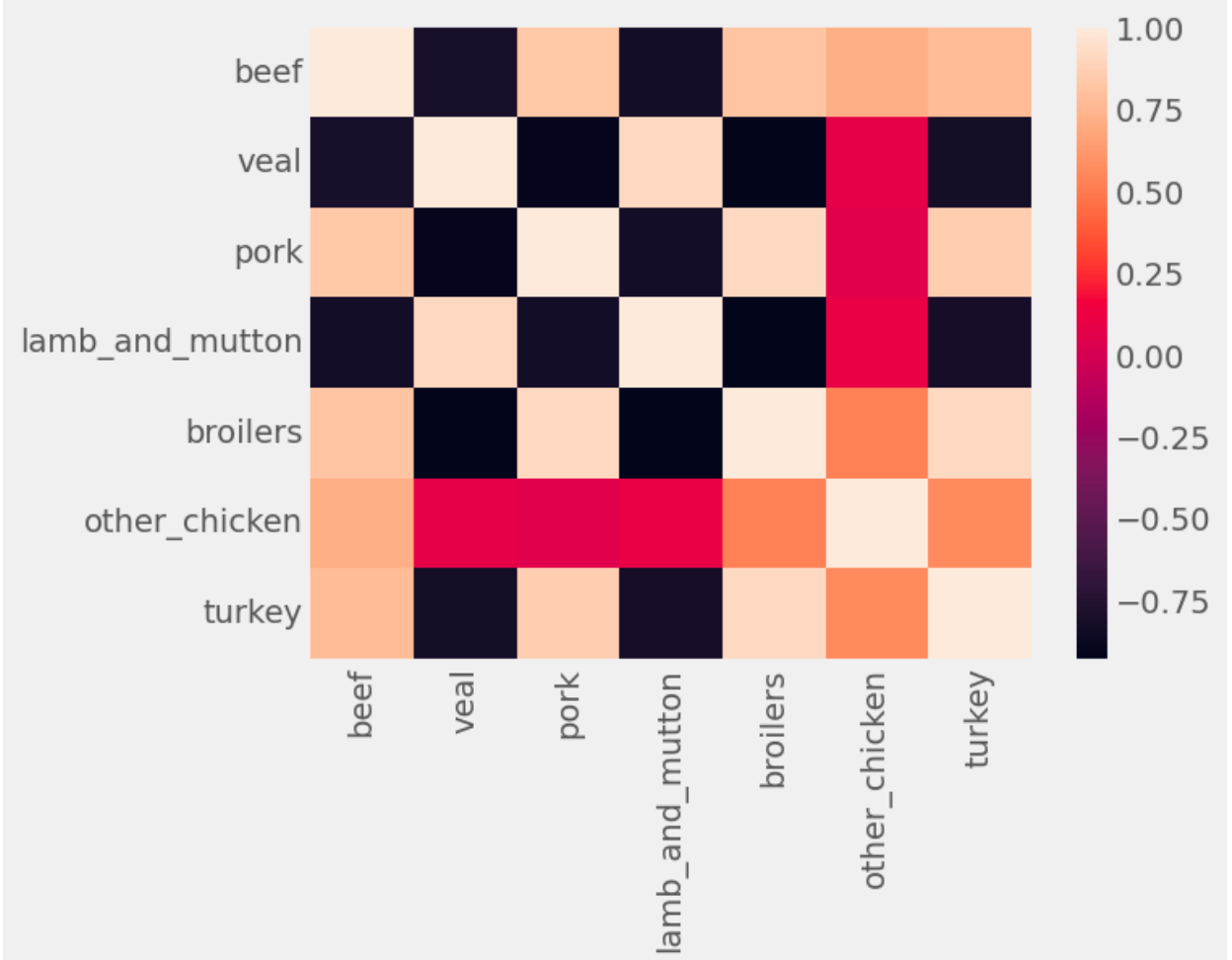
If you want to compute the correlation between all time series in your DataFrame, simply remove the references to the columns.

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

date	beef	veal	pork	lamb_and_mutton	broilers	other_chicken	turkey
1944-01-01	751.0	85.0	1280.0	89.0	NaN	NaN	NaN
1944-02-01	713.0	77.0	1169.0	72.0	NaN	NaN	NaN
1944-03-01	741.0	90.0	1128.0	75.0	NaN	NaN	NaN
1944-04-01	650.0	89.0	978.0	66.0	NaN	NaN	NaN
1944-05-01	681.0	106.0	1029.0	78.0	NaN	NaN	NaN
1944-06-01	658.0	125.0	962.0	79.0	NaN	NaN	NaN
1944-07-01	662.0	142.0	796.0	82.0	NaN	NaN	NaN
1944-08-01	787.0	175.0	748.0	87.0	NaN	NaN	NaN
1944-09-01	774.0	182.0	678.0	91.0	NaN	NaN	NaN
1944-10-01	834.0	215.0	777.0	100.0	NaN	NaN	NaN

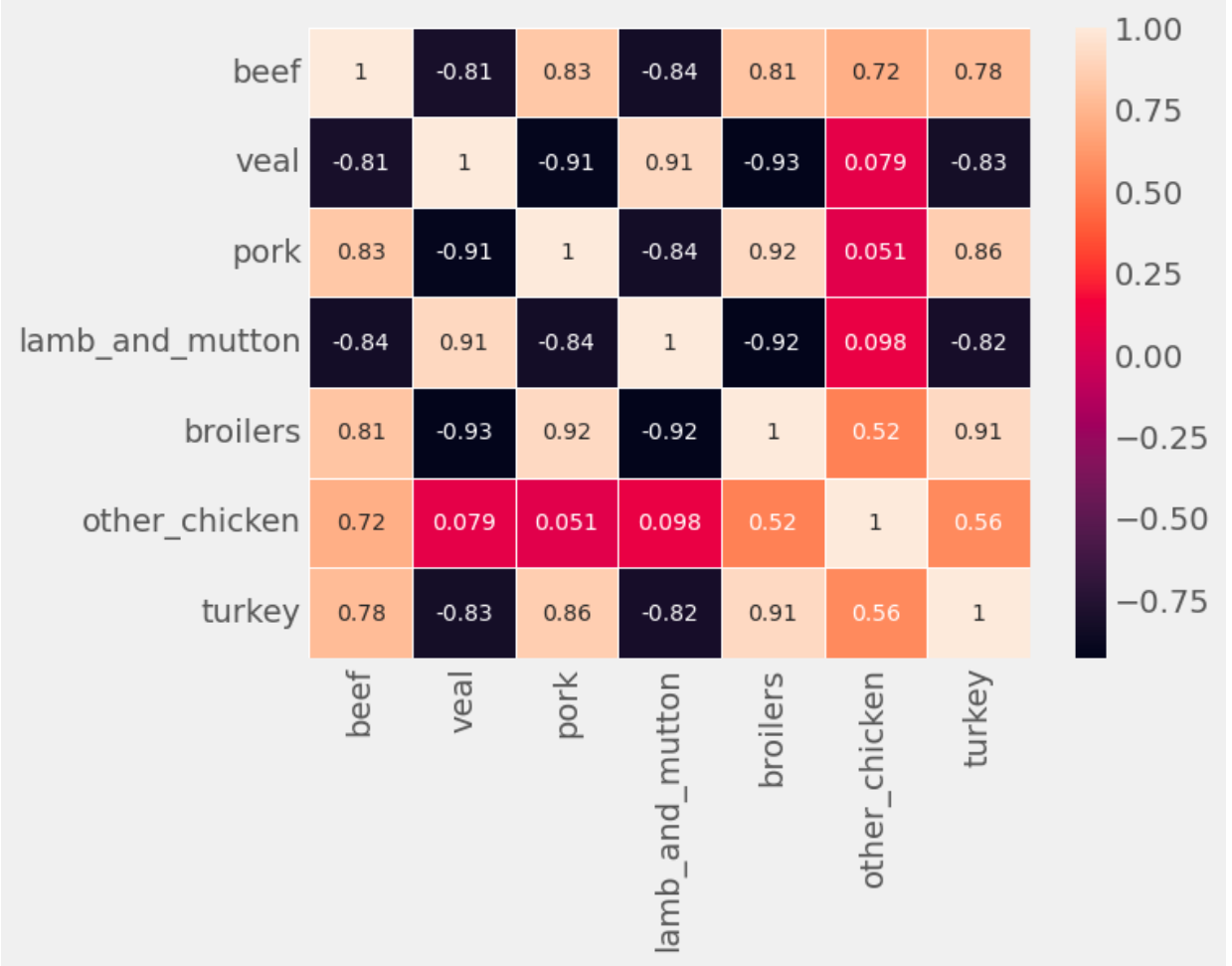
Heatmap

```
import seaborn as sns

sns.heatmap(corr_mat)
```



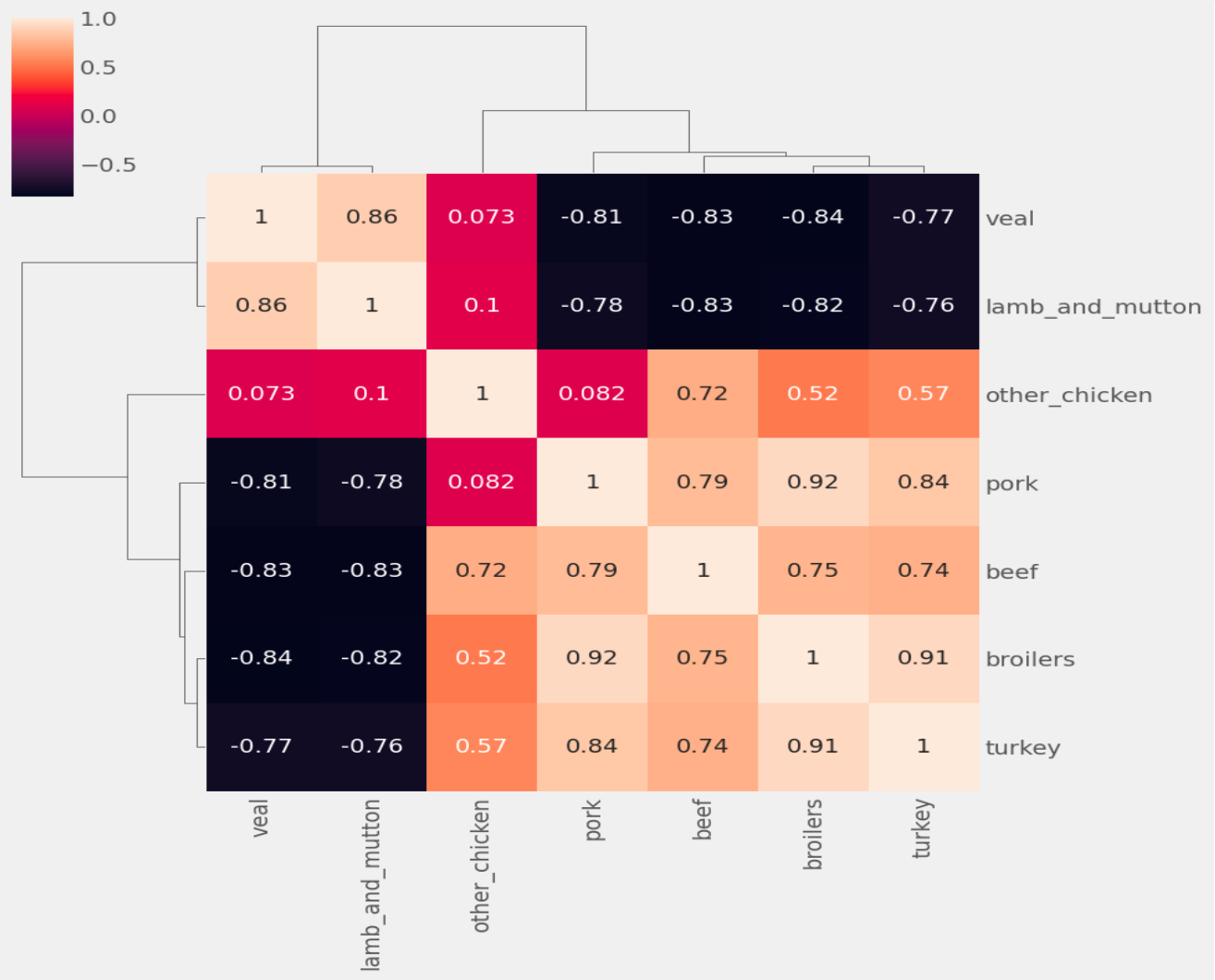
Heatmap

```
import seaborn as sns

sns.heatmap(corr_mat, annot=True)
```



Clustermap

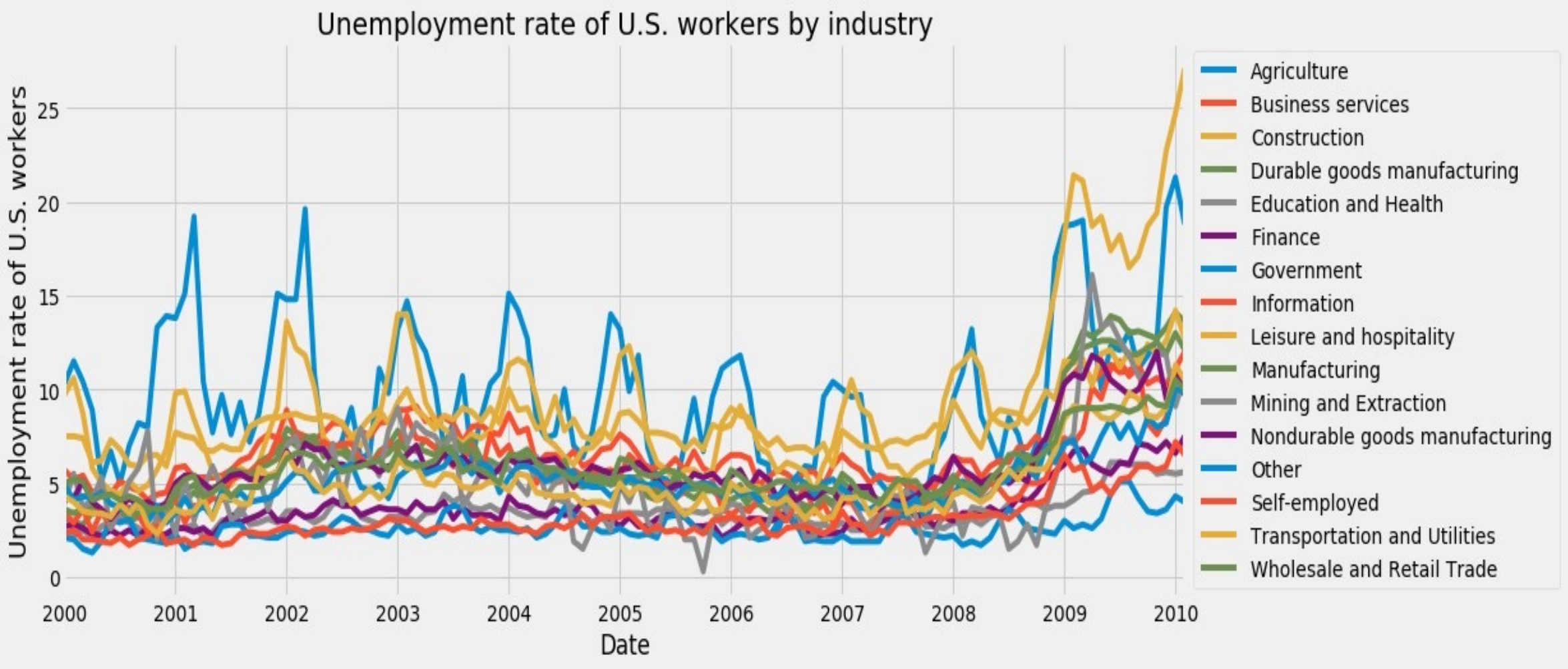
```
sns.clustermap(corr_mat)
```



Outline

- Line Plots
- Summary Statistics and Diagnostics
- Seasonality, Trend and Noise
- Work with Multiple Time Series
- **Case Study: Unemployment Rate**

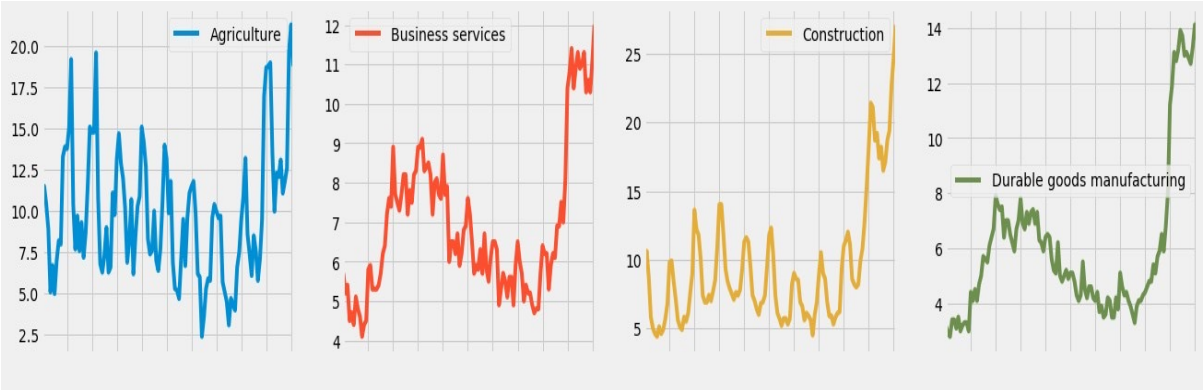
The Jobs dataset

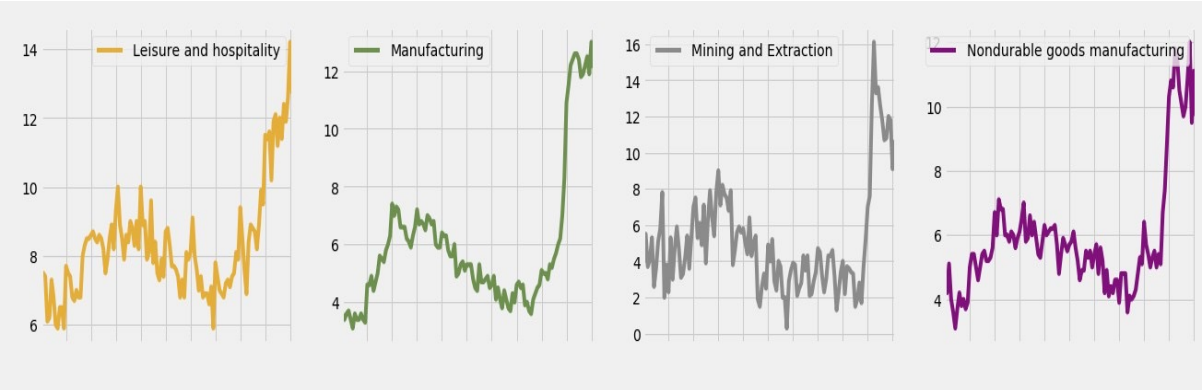


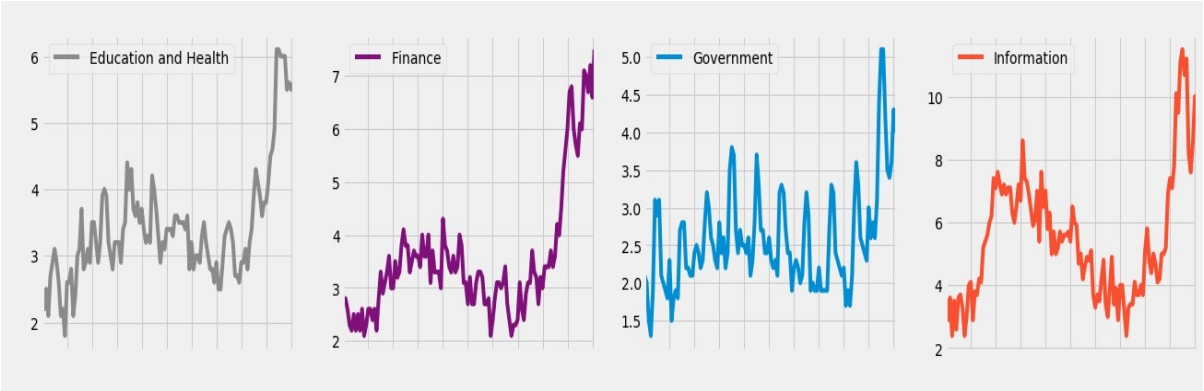
Facet plots of the jobs dataset

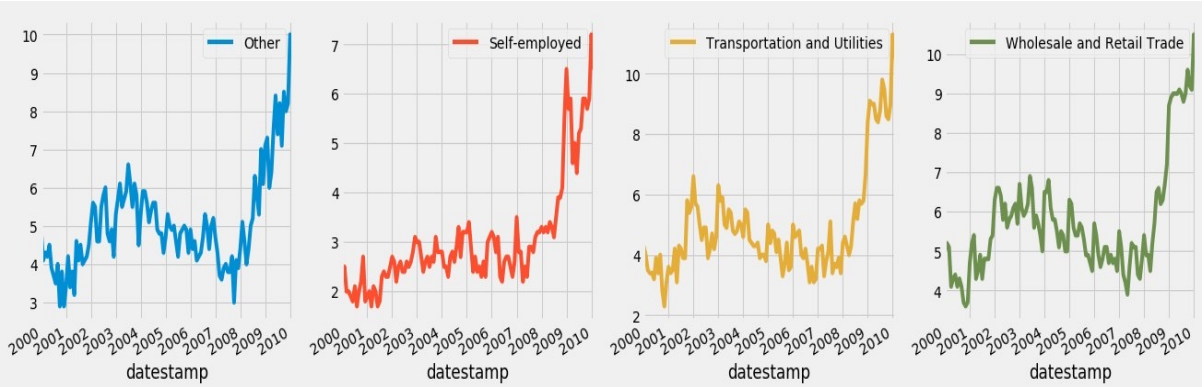
```
jobs.plot(subplots=True,
          layout=(4, 4),
          figsize=(20, 16),
          sharex=True,
          sharey=False)

plt.show()
```

Facet plots


Agriculture
Business services
Construction
Durable goods manufacturing

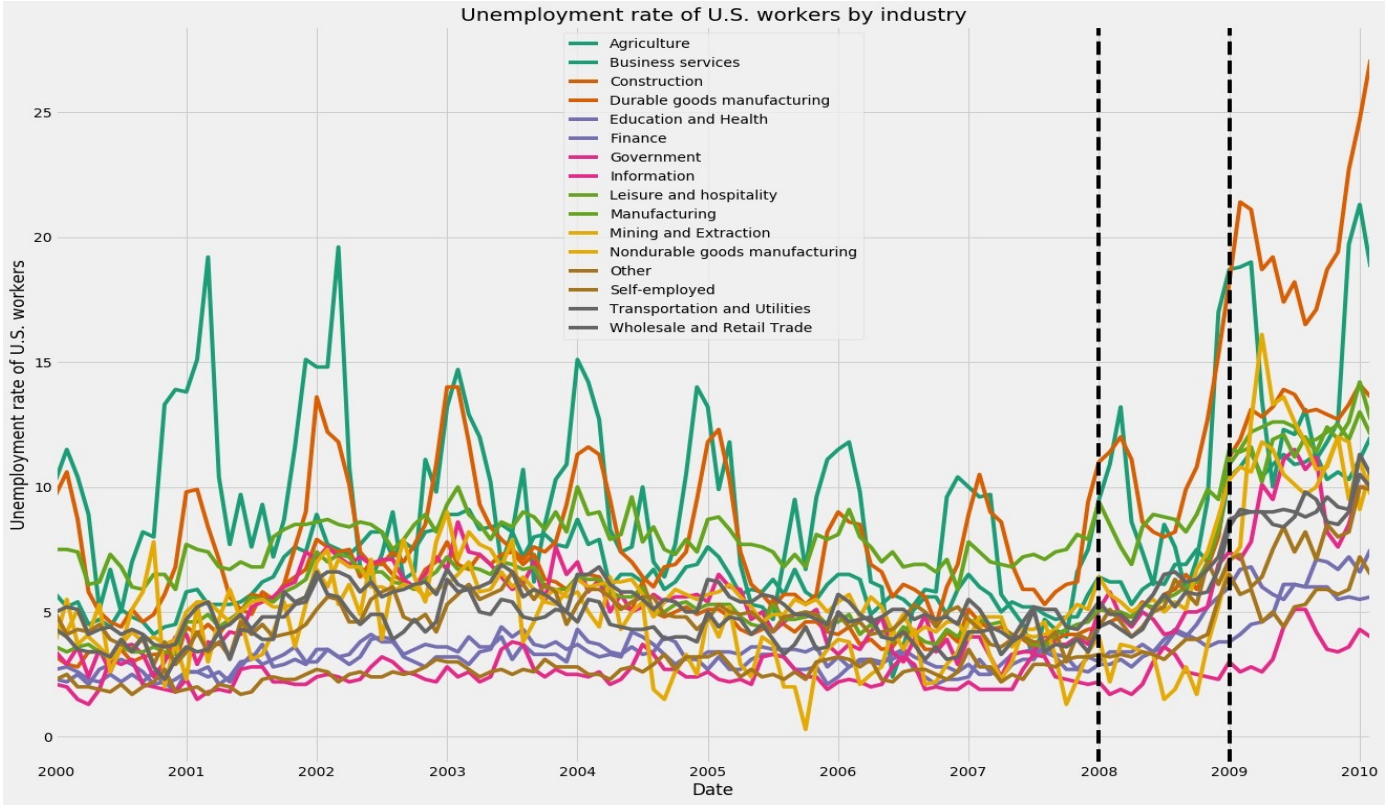

Leisure and hospitality
Manufacturing
Mining and Extraction
Nondurable goods manufacturing


Education and Health
Finance
Government
Information


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='--')
```



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)
```

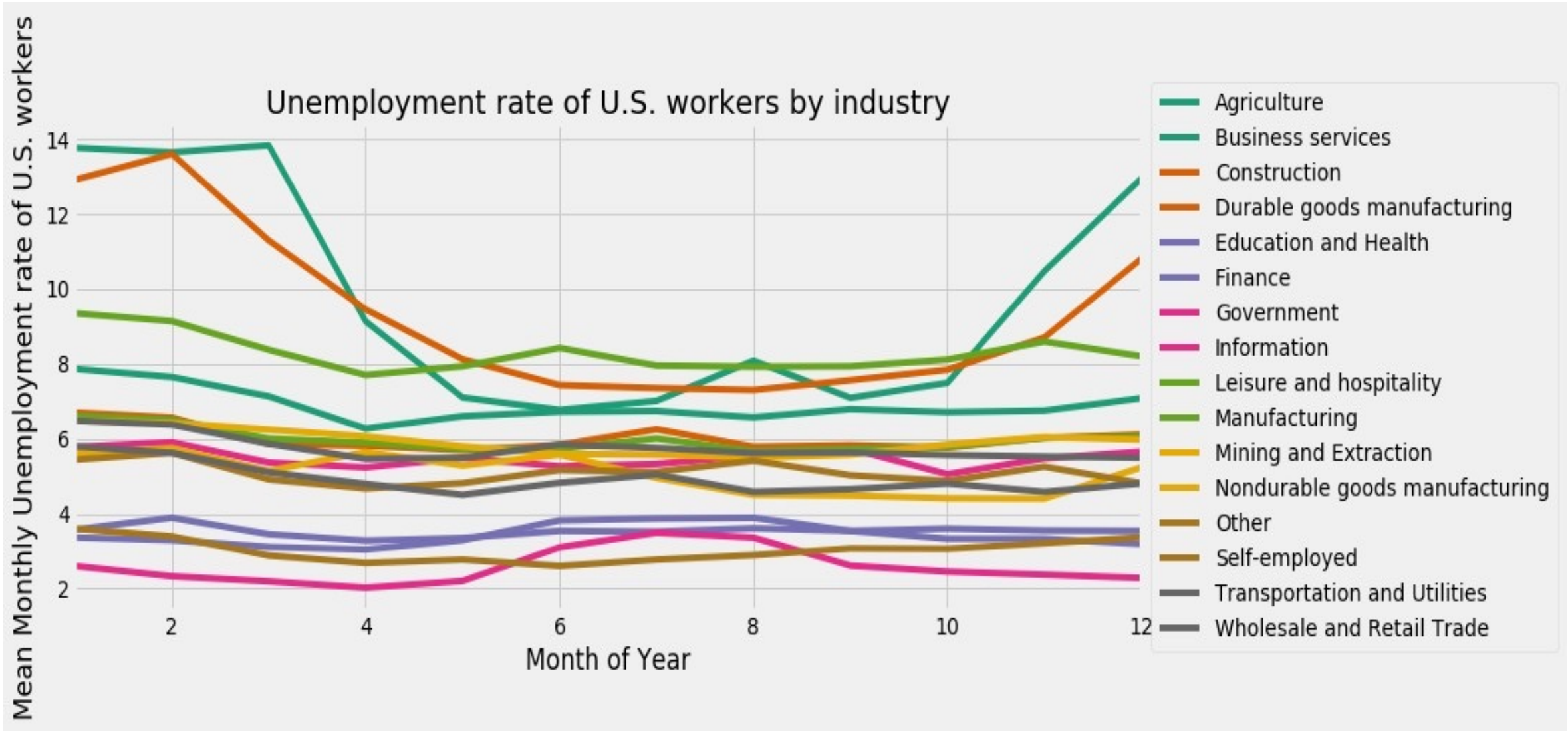
```
datestamp    Agriculture  Business services  Construction
1            13.763636           7.863636     12.909091
2            13.645455           7.645455     13.600000
3            13.830000           7.130000     11.290000
4             9.130000           6.270000      9.450000
5             7.100000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



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
...
```

Trends in Jobs data

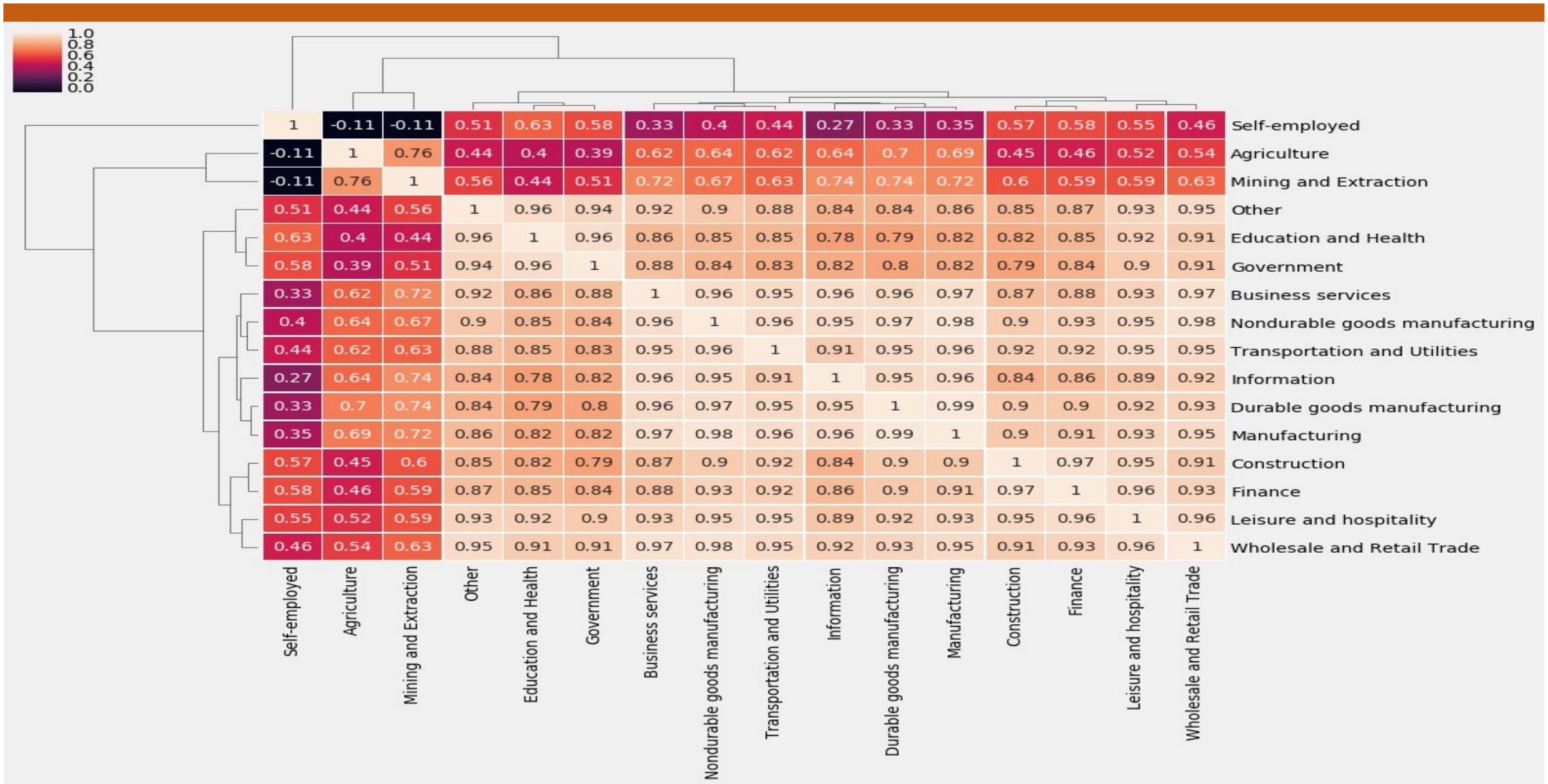
```
print(trend_df)
```

```
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



	Self-employed	Agriculture	Mining and Extraction	Other	Education and Health	Government	Business services	Nondurable goods manufacturing	Transportation and Utilities	Information	Durable goods manufacturing	Manufacturing	Construction	Finance	Leisure and hospitality	Wholesale and Retail Trade
Self-employed	1	-0.11	-0.11	0.51	0.63	0.58	0.33	0.4	0.44	0.27	0.33	0.35	0.57	0.58	0.55	0.46
Agriculture	-0.11	1	0.76	0.44	0.4	0.39	0.62	0.64	0.62	0.64	0.7	0.69	0.45	0.46	0.52	0.54
Mining and Extraction	-0.11	0.76	1	0.56	0.44	0.51	0.72	0.67	0.63	0.74	0.74	0.72	0.6	0.59	0.59	0.63
Other	0.51	0.44	0.56	1	0.96	0.94	0.92	0.9	0.88	0.84	0.84	0.86	0.85	0.87	0.93	0.95
Education and Health	0.63	0.4	0.44	0.96	1	0.96	0.86	0.85	0.85	0.78	0.79	0.82	0.82	0.85	0.92	0.91
Government	0.58	0.39	0.51	0.94	0.96	1	0.88	0.84	0.83	0.82	0.8	0.82	0.79	0.84	0.9	0.91
Business services	0.33	0.62	0.72	0.92	0.86	0.88	1	0.96	0.95	0.96	0.96	0.97	0.87	0.88	0.93	0.97
Nondurable goods manufacturing	0.4	0.64	0.67	0.9	0.85	0.84	0.96	1	0.96	0.95	0.97	0.98	0.9	0.93	0.95	0.98
Transportation and Utilities	0.44	0.62	0.63	0.88	0.85	0.83	0.95	0.96	1	0.91	0.95	0.96	0.92	0.92	0.95	0.95
Information	0.27	0.64	0.74	0.84	0.78	0.82	0.96	0.95	0.91	1	0.95	0.96	0.84	0.86	0.89	0.92
Durable goods manufacturing	0.33	0.7	0.74	0.84	0.79	0.8	0.96	0.97	0.95	0.95	1	0.99	0.9	0.9	0.92	0.93
Manufacturing	0.35	0.69	0.72	0.86	0.82	0.82	0.97	0.98	0.96	0.96	0.99	1	0.9	0.91	0.93	0.95
Construction	0.57	0.45	0.6	0.85	0.82	0.79	0.87	0.9	0.92	0.84	0.9	0.9	1	0.97	0.95	0.91
Finance	0.58	0.46	0.59	0.87	0.85	0.84	0.88	0.93	0.92	0.86	0.9	0.91	0.97	1	0.96	0.93
Leisure and hospitality	0.55	0.52	0.59	0.93	0.92	0.9	0.93	0.95	0.95	0.89	0.92	0.93	0.95	0.96	1	0.96
Wholesale and Retail Trade	0.46	0.54	0.63	0.95	0.91	0.91	0.97	0.98	0.95	0.92	0.93	0.95	0.91	0.93	0.96	1