Time Series Note

DongDong

2024-02-23

Table of contents

python python python
`plotly` `d3.js` `plotly`

1

```python
import plotly.graph_objects as go
import numpy as np
```

1.1 Time Series v.s. Cross Sectional

(time series) (cross sectional)

Table 1.1: Time Series v.s. Cross Sectional

	Time Series	Cross Sectional
	2000/01/01 2020/01/01	2000/01/01 50

1.2

2000/01/01 2020/01/01

1.3

- $U_t, t \in \{1, \cdots, T\}$

U_t U_t

- $U \rightarrow$
- $t \rightarrow$

$U_t = T_t + S_t + C_t + R_t$

- T_t : (Trend component)
- S_t : (Seasonal component)
- C_t : (Cyclical component)
- R_t : (Random component)

1.3.1

- Downward Trend: Television Newspaper
- No Trend: Radio
- Upward Trand: Internet

```
title = 'Main Source for News'
labels = ['Television', 'Newspaper', 'Internet', 'Radio']
colors = ['rgb(67,67,67)', 'rgb(115,115,115)', 'rgb(49,130,189)', 'rgb(189,189,189)']

mode_size = [8, 8, 12, 8]
line_size = [2, 2, 4, 2]

x_data = np.vstack((np.arange(2001, 2014),)*4)

y_data = np.array([
    [74, 82, 80, 74, 73, 72, 74, 70, 70, 66, 66, 69],
    [45, 42, 50, 46, 36, 36, 34, 35, 32, 31, 31, 28],
    [13, 14, 20, 24, 20, 24, 24, 40, 35, 41, 43, 50],
    [18, 21, 18, 21, 16, 14, 13, 18, 17, 16, 19, 23],
])

fig = go.Figure()

for i in range(0, 4):
    fig.add_trace(go.Scatter(x=x_data[i], y=y_data[i], mode='lines',
        name=labels[i],
        line=dict(color=colors[i], width=line_size[i]),
        connectgaps=True,
    ))

    # endpoints
    fig.add_trace(go.Scatter(
        x=[x_data[i][0], x_data[i][-1]],
        y=[y_data[i][0], y_data[i][-1]],
        mode='markers',
        marker=dict(color=colors[i], size=mode_size[i])
    ))

fig.update_layout(
    xaxis=dict(
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=False,
    ),
    autosize=False,
    margin=dict(
        autoexpand=False,
        l=100,
        r=20,
        t=110,
    ),
    showlegend=False,
    plot_bgcolor='white'
```

```
)

annotations = []

# Adding labels
for y_trace, label, color in zip(y_data, labels, colors):
    # labeling the left_side of the plot
    annotations.append(dict(xref='paper', x=0.05, y=y_trace[0],
                                  xanchor='right', yanchor='middle',
                                  text=label + ' {}%'.format(y_trace[0]),
                                  font=dict(family='Arial',
                                            size=16),
                                  showarrow=False))
    # labeling the right_side of the plot
    annotations.append(dict(xref='paper', x=0.95, y=y_trace[11],
                                  xanchor='left', yanchor='middle',
                                  text='{}%'.format(y_trace[11]),
                                  font=dict(family='Arial',
                                            size=16),
                                  showarrow=False))
# Title
annotations.append(dict(xref='paper', yref='paper', x=0.0, y=1.05,
                              xanchor='left', yanchor='bottom',
                              text='Main Source for News',
                              font=dict(family='Arial',
                                        size=30,
                                        color='rgb(37,37,37)'),
                              showarrow=False))
# Source
annotations.append(dict(xref='paper', yref='paper', x=0.5, y=-0.1,
                              xanchor='center', yanchor='top',
                              text='Source: PewResearch Center & ' +
                                   'Storytelling with data',
                              font=dict(family='Arial',
                                        size=12,
                                        color='rgb(150,150,150)'),
                              showarrow=False))

fig.update_layout(annotations=annotations)

fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

1.3.2

```
title = 'Seasonal Effect'

x_data = np.linspace(-10, 10, 100)

y_data = np.array(
    np.sin(x_data) + np.random.normal(size=x_data.shape) * 0.25
)

fig = go.Figure()

fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines+markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
title = 'Cyclical Effect'

x_data = np.linspace(-10, 10, 100)

y_data = np.array(
    np.sin(x_data)*np.exp(x_data/10) + np.random.normal(size=x_data.shape) * 0.25
)

fig = go.Figure()
```

```
fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines+markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.3.3

sin

```
title = 'Seasonal Effect'

x_data = np.linspace(-10, 10, 100)

y_data = np.array(
    np.sin(x_data) + np.random.normal(size=x_data.shape) * 1
)

fig = go.Figure()

fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines+markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.4

```python
title = 'Cyclical Effect'

x_data = np.linspace(-10, 10, 1000)

y_data = np.array(
    np.sin(x_data)*np.exp(x_data/100)/10 + np.random.normal(size=x_data.shape)
)

fig = go.Figure()

fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```python
import plotly.figure_factory as ff
import numpy as np

hist_data = [y_data]

group_labels = ['Data (no T)']
colors = ['#A56CC1']

# Create distplot with curve_type set to 'normal'
fig = ff.create_distplot(hist_data, group_labels, colors=colors,
                         bin_size=.2, show_rug=False)

# Add title
fig.update_layout(title_text='Hist and Curve Plot')
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.5 Stationary

1.5.1 Weak stationary

$\{Z_t\}$ (second-order or covariance stationary)

- $\mu(t) = C : \mu$
- $\gamma(t, t-k) = \gamma(0, k) :$ (lag)

$$\sim \mathcal{N}(0, \sigma^2 I)$$

$$X_t = X_{t-1} + \epsilon$$

ϵ

- $\mathbb{E}(X_t) = \mathbb{E}(X_{t-1})$
- $\mathrm{Var}(X_t) = t\sigma^2$

```
title = 'Seasonal Effect'

T = 1000
walks = []

loc = 0
for i in range(T):
    loc += + np.random.normal(0,1)
    walks.append(loc)
walks = np.array(walks)

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=walks, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

$$D_t = X_t - X_{t-1} =$$

- $\mathbb{E}(D_t) = 0$
- $\mathrm{Var}(X_t) = \sigma^2$

```
title = 'Seasonal Effect'

D = walks[1:] - walks[:-1]

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T-1)), y=D, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.5.2 Strict stationary

$\{Z_t\}$

$Z_{t_1}, Z_{t_2}, \cdots, Z_{t_n}$ $Z_{t_1-k}, Z_{t_2-k}, \cdots, Z_{t_n-k}$

Note:

-
-

1.5.3 Sample AutoCorrelation Function (ACF)

? **** ** ACF

$$\rho_\ell = \frac{\text{cov}(r_t, r_{t-\ell})}{\text{var}(r_t)}$$

- $\gamma_k = \text{Cov}(r_t, r_{t-k}) = E[(r_t-\mu)(r_{t-k}-\mu)]$: k
- $\rho_0 = 1$: 1
- $\rho_k = \rho_{-k}$:

Lag-ℓ μ ( )

$$\hat{\rho}_\ell = \frac{\Sigma_{t=1}^{T-\ell}(r_t-\bar{r})(r_{t+\ell}-\bar{r})}{\Sigma_{t=1}^{T}(r_t-\bar{r})^2}$$

- $\bar{r}$:
- T :

python `numpy.correlate` `numpy.correlate(a, v, mode)` v filter a v a ( ) `numpy.correlate`

$$c_k = \sum_n a_{n+k} * \bar{v}_n$$

- $\bar{v}_n$ complex conjugation
- `mode`:
  - `valid`: v a a
  - `same`: v a
  - `full`: v a

```python
def acf(series, lag=None):
    """
    Calculate the autocorrelation function (ACF) for a given time series.

    Parameters:
        series (array-like): The time series data.
        lag (int, optional): The maximum lag for which to calculate the ACF.
            If None (default), the ACF is calculated for all possible lags.

    Returns:
        acf_values (array): Autocorrelation values for the given time series.
            If `lag` is provided, returns a single value representing
            autocorrelation at that lag. Otherwise, returns an array of
            autocorrelation values for all lags.
    """
    series = np.asarray(series)
    n = len(series)

    mean = np.mean(series)
    centered_data = series - mean

    # Calculate autocovariance function
    acov = np.correlate(centered_data, centered_data, mode='full')
    acf_values = acov / sum(centered_data ** 2)

    if lag is not None:
        return acf_values[(n-1):(n-1+lag)]
    else:
        return acf_values[(n-1):]
```

ACF

```
rw_acf = acf(walks, lag=90)
title = 'ACF Random Walks'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=rw_acf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
d_acf = acf(D, lag=90)
title = 'ACF Random Walks'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T-1)), y=d_acf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

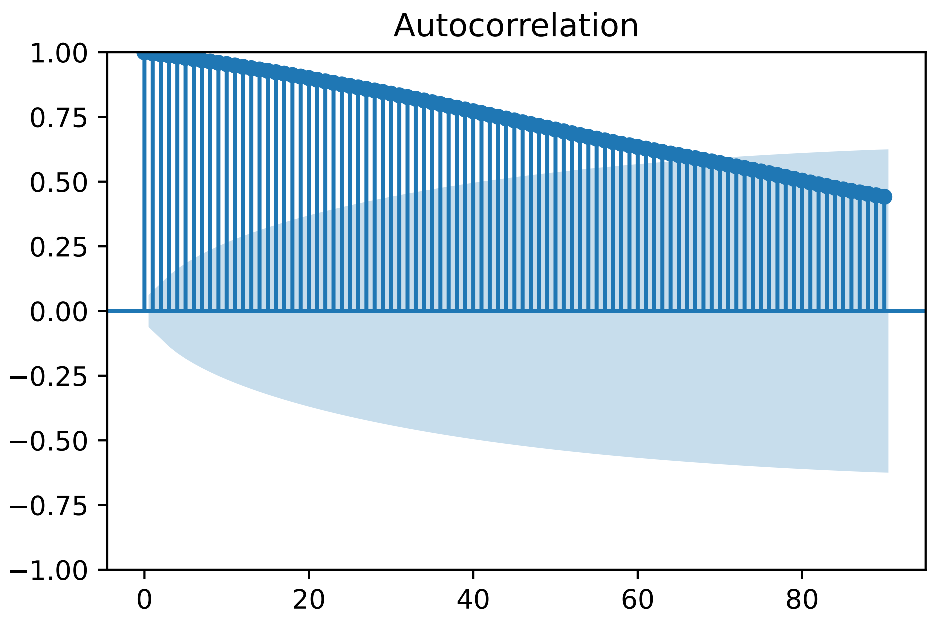
```
Unable to display output for mime type(s): text/html
```

python `statsmodels.graphics.tsaplots.plot_acf`

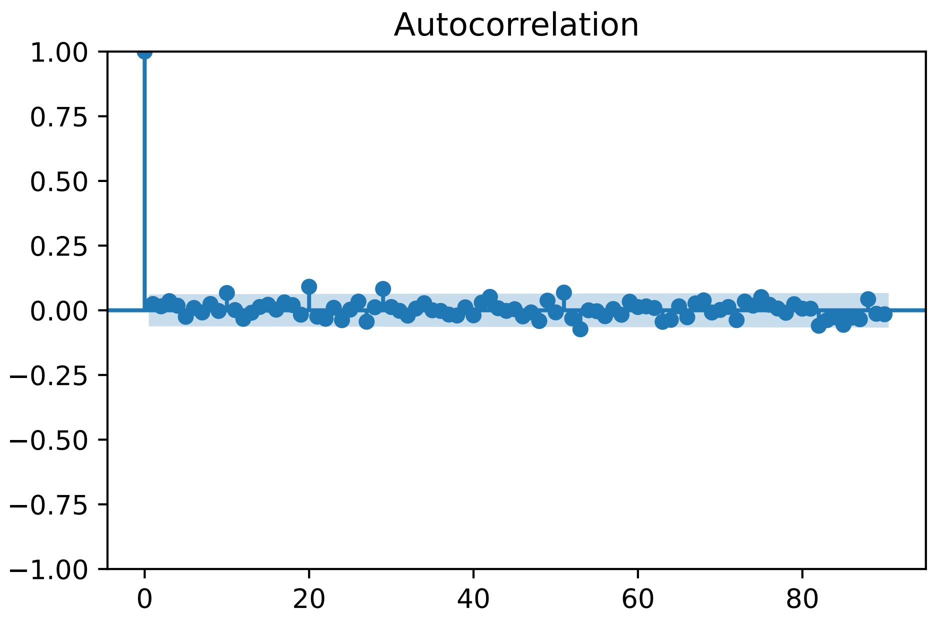
```
from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plot_acf(walks, lags=90)
plt.show()

plot_acf(D, lags=90)
plt.show()
```



Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80



Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80

References