

Plotly Grundlagen

Plotly ist ein Python-Framework zum Erzeugen von interaktiven, hochqualitativen Graphen. Die Graphen werden dabei eingebettet in HTML angezeigt.

Installation

```
sudo pip3 install plotly
```

Imports

Häufig werden NumPy und Pandas zur Datenaufbereitung vor dem Plotting eingesetzt.

```
import numpy as np
import pandas as pd
import plotly.offline as pyo
import plotly.graph_objs as go
import plotly.figure_factory as ff
```

Datenvorbereitung

```
x = np.linspace(0, 5, 50)
y = np.cos(x)
z = np.sin(x)
df = pd.DataFrame(np.random.randn(100,4),
                  columns='A B C D'.split())
```

Beispielplot

```
pyo.plot([{'x': df.index, 'y': df[col],
          'name': col} for col in df.columns])
```

Plot mit Layout und Figures

```
layout = go.Layout(
    title='Liniendiagramm')
trace = go.Scatter(x=x, y=y,
                  mode='lines', name='Linie')
fig = go.Figure(data=[trace],
                layout=layout)
```

Plot anzeigen

```
pyo.plot(fig, filename='plot.html')
```

Linientyp

```
trace = go.Scatter(x=x, y=y,
                  mode='lines',
                  line=dict(color='green', width=25))
```

```
trace = go.Scatter(x=x, y=y,
                  mode='lines', line=dict(dash='dot'))
```

```
trace = go.Scatter(x=x, y=y,
                  mode='lines',
                  line=dict(dash='dashdot'))
```

Annotierung

```
layout = go.Layout(showlegend=False,
                  annotations=[dict(x=0.5, y=1, ax=0, ay=-40,
                                    showarrow=True, arrowhead=1, text='Info')])
trace = go.Scatter(x=[0, 1], y=[1, 1],
                  mode='lines+text', text=['Text A', 'Text B'],
                  textposition='top center')
```

Plotly Plotting

Scatterplot

```
trace = go.Scatter(x=x, y=y,
                  mode='markers', name='Markers')
```

Vertikale Balken

```
trace = go.Bar(y=[5,6,8],
               orientation='v')
```

Horizontale Balken

```
trace = go.Bar(y=[5,6,8],
               orientation='h')
```

Fülle zu x

```
trace = go.Scatter(x=x, y=y,
                  fill='tozeroy')
```

Fülle zu x

```
trace = go.Scatter(x=x, y=y,
                  fill='tozeroy')
```

Horizontale Linie

```
trace = go.Scatter(x=list(range(
    len(x))), y=[0.35]*len(x),
                  mode='lines')
```

Vertikale Linie

```
trace = go.Scatter(y=list(range(
    len(y))), x=[0.69]*len(y),
                  mode='lines')
```

Histogramm

```
trace = go.Histogram(x=y,
                    xbins=go.XBins(start=-1, end=1, )
                    size=0.2)
```

Pfeilplot

```
fig = ff.create_quiver(x, y, y, z)
```

Boxplot

```
trace = go.Box(y=y)
```

Violinplot

```
fig = ff.create_violin(z)
```

Weitere Plots

Kuchendiagramm

```
labels = ['A','B','C','D']
values = [4500, 2500, 1053, 500]
fig = go.Figure(
    data=[go.Pie(labels=labels, values=values)])
```

Heatmap

```
fig = go.Figure(data=go.Heatmap(
    z=[[1, 20, 30], [20, 1, 60], [30, 60, 1]]))
```

Zeitfolgen

```
from datetime import datetime as dt
x = [dt(year=2013, month=10, day=4),
     dt(year=2013, month=11, day=5),
     dt(year=2013, month=12, day=6)]
```

```
fig = go.Figure(
    data=[go.Scatter(x=x, y=[1, 3, 6])])
fig.update_layout(xaxis_range=
    [dt(2013, 10, 17),dt(2013, 11, 20)])
```

Gantt-Diagramm

```
df = [dict(Task="A", Start='2009-01-01',
           Finish='2009-02-28'),
      dict(Task="B", Start='2009-03-05',
           Finish='2009-04-15'),
      dict(Task="C", Start='2009-02-20',
           Finish='2009-05-30')]
```

```
fig = ff.create_gantt(df)
```

Tabellen

```
fig = go.Figure(data=[go.Table(
    header=dict(values=['A', 'B']),
    cells=dict(values=[[100, 90, 80, 90],
                      [95, 85, 75, 95]]))])
```

Sankey-Diagramm

```
fig = go.Figure(data=[go.Sankey(node=dict(
    label = ["A1", "A2", "B1", "B2", "C1", "C2"],),
    link = dict(source = [0, 1, 0, 2, 3, 3],
                target = [2, 3, 3, 4, 4, 5],
                value = [8, 4, 2, 8, 4, 2]))])
```

Konturplot

```
fig = go.Figure(data=go.Contour(
    z=[[10, 10.625, 12.5, 15.625, 20],
       [5.625, 6.25, 8.125, 11.25, 15.625],
       [2.5, 3.125, 5., 8.125, 12.5],
       [0.625, 1.25, 3.125, 6.25, 10.625],
       [0, 0.625, 2.5, 5.625, 10]]))
```

Dendrogramm

```
X = np.random.rand(15, 12)
fig = ff.create_dendrogram(X)
```

Logarithmische Achsen

```
fig.add_trace(go.Scatter(x=[0, 1, 2, 3, 4, 5, 6, 7],
                        y=[8, 7, 6, 5, 4, 3, 2, 1]))
fig.add_trace(go.Scatter(x=[0, 1, 2, 3, 4, 5, 6, 7],
                        y=[0, 1, 2, 3, 4, 5, 6, 7]))
fig.update_layout(
    xaxis_type="log", yaxis_type="log")
```

Dash

Dash ist ein Python-Framework für Webapplikationen und zusammen mit Plotly hervorragend für die Entwicklung webbasierte Anwendungen zur Datenanalyse geeignet.

Installation

```
sudo pip install dash==0.21.1 \
dash-renderer==0.13.0
dash-html-components==0.11.0
dash-core-components==0.23.0
```

Imports

```
import dash
import dash_core_components as dcc
import dash_html_components as html
```

App-Aufbau

Zuerst wird eine Instanz der Basisklasse **Dash** erstellt. Anschliessend werden dieser HTML-Elemente und Graphen hinzugefügt. Zuletzt wird die Applikation ausgeführt.

```
app = dash.Dash()
html.Div(children='Dash Beispielanwendung'),
dcc.Graph(figure={'data': [{'x': [1, 2, 3],
                           'y': [6, 4, 1], 'type': 'bar', 'name': 'EUR'},
                        {'x': [1, 2, 3], 'y': [2, 3, 3], 'type': 'bar',
                           'name': 'USD'}]}))
if __name__ == '__main__':
    app.run_server(debug=True)
```

CSS

CSS-Styles können als Parameter übergeben werden.

```
html.H1(children='Hallo', style={
    'textAlign': 'center', 'color': '#999999'}),
```

Scatterplot

```
dcc.Graph(figure={'data': [go.Scatter(x=x, y=y,
                                       mode='markers')]}))
```

Linienplot

Ändere den Parameter **mode** von **markers** zu **lines**, um einen Linienplot anstelle eines Scatterplots darzustellen.

Marker- und Linienstil

Füge der **data**-Liste des obigen Graphen folgenden Parameter hinzu, um den Marker- und Linienstil zu ändern:

```
marker={'size': 15,
        'line': {'width': 5, 'color': 'green'}}
```

Layout

Füge dem **figure**-Dictionary des obigen Graphen einen Parameter **layout** hinzu, um das Layout des Graphen anzupassen:

```
'layout': go.Layout(hovermode='closest',
                    xaxis={'type': 'log', 'title': 'Log'},
                    yaxis={'title': 'Cosinus'},
                    margin={'l': 40, 'b': 40, 't': 10, 'r': 10})
```