Plotly and Cufflinks

June 8, 2025

Plotly and Cufflinks By Isha Borgaonkar

Plotly is a library that allows you to create interactive plots that you can use in dashboards or websites (you can save them as html files or static images).

0.1 Installation

In order for this all to work, you'll need to install plotly and cufflinks to call plots directly off of a pandas dataframe. These libraries are not currently available through **conda** but are available through **pip**. Install the libraries at your command line/terminal using:

```
!pip install plotly
!pip install cufflinks
```

** NOTE: Make sure you only have one installation of Python on your computer when you do this, otherwise the installation may not work. **

0.2 Imports and Set-up

[16]:
```
# Importing the core Plotly library for interactive visualizations
import plotly

# Importing Cufflinks, which bridges Plotly with pandas for easy plotting using␣
 ↪.iplot()
import cufflinks

# Importing Folium, a library for creating interactive leaflet maps in Python
import folium

# Confirming that all libraries have been successfully imported
print("All libraries imported successfully!")
```

```
All libraries imported successfully!
```

[17]:
```
# Installing the matplotlib library using pip
# This is used for 2D plotting in Python (line plots, histograms, scatter␣
 ↪plots, etc.)
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\isha\anaconda3\lib\site-
packages (3.9.4)
```

```
Requirement already satisfied: packaging>=20.0 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: cycler>=0.10 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: pillow>=8 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib) (9.2.0)
Requirement already satisfied: numpy>=1.23 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib) (1.23.5)
Requirement already satisfied: importlib-resources>=3.2.0 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (6.4.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (1.4.2)
Requirement already satisfied: zipp>=3.1.0 in c:\users\isha\anaconda3\lib\site-
packages (from importlib-resources>=3.2.0->matplotlib) (3.8.0)
Requirement already satisfied: six>=1.5 in c:\users\isha\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

[18]:
```
# Importing the pyplot module from matplotlib for creating 2D plots like line␣
 ↪charts, bar charts, etc.
import matplotlib.pyplot as plt

# Importing Axes3D from mpl_toolkits to enable 3D plotting capabilities in␣
 ↪matplotlib
from mpl_toolkits.mplot3d import Axes3D
```

[19]:
```
# Upgrading matplotlib to the latest version to ensure compatibility with other␣
 ↪libraries
!pip install --upgrade matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\isha\anaconda3\lib\site-
packages (3.9.4)
Requirement already satisfied: numpy>=1.23 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib) (1.23.5)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: packaging>=20.0 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: cycler>=0.10 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
```

```
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (1.4.2)
Requirement already satisfied: importlib-resources>=3.2.0 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (6.4.0)
Requirement already satisfied: pillow>=8 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib) (9.2.0)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\isha\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: zipp>=3.1.0 in c:\users\isha\anaconda3\lib\site-
packages (from importlib-resources>=3.2.0->matplotlib) (3.8.0)
Requirement already satisfied: six>=1.5 in c:\users\isha\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

[20]:
```python
# Importing the main matplotlib package to access its metadata and configuration
import matplotlib

# Printing the currently installed version of matplotlib
# Useful to verify if the library was successfully upgraded
print(matplotlib.__version__)
```

```
3.9.4
```

0.3 Using Cufflinks and iplot()

- line
- scatter
- bar
- box
- spread
- ratio
- heatmap
- surface
- histogram
- bubble

[23]:
```python
# import the library
# Importing the folium library to create interactive maps using Leaflet.js in␣
 ↪Python
import folium
import pandas as pd
```

[24]:
```python
# Creating a pandas DataFrame containing sample geographic data
# Columns include:
# - 'lat' : latitude values for map markers
# - 'lon' : longitude values for map markers
```

```
# - 'name' : name of the cities corresponding to each coordinate
# - 'value' : some numeric value associated with each location (e.g.,␣
 ↪population, metric, etc.)
data = pd.DataFrame({
    'lat': [-58.0, 2.0, 145.0, 30.32, -4.03, -73.57, 36.82, -38.5],
    'lon': [-34.0, 49.0, -38.0, 59.93, 5.33, 45.52, -1.29, -12.97],
    'name': ['Buenos Aires', 'Paris', 'Melbourne', 'St Petersbourg', 'Abidjan',␣
 ↪'Montreal', 'Nairobi', 'Salvador'],
    'value': [1.0, 1.2, 4.0, 7.0, 2.3, 4.3, 10.0, 43.0]
})

# Displaying the DataFrame to verify contents
data
```

[24]:

	lat	lon	name	value
0	-58.00	-34.00	Buenos Aires	1.0
1	2.00	49.00	Paris	1.2
2	145.00	-38.00	melbourne	4.0
3	30.32	59.93	St Petersbourg	7.0
4	-4.03	5.33	Abidjan	2.3
5	-73.57	45.52	Montreal	4.3
6	36.82	-1.29	Nairobi	10.0
7	-38.50	-12.97	Salvador	43.0

[26]:
```
# Importing the folium library for interactive maps
import folium

# Creating a base world map centered around latitude=20, longitude=0 (roughly␣
 ↪Africa/Atlantic Ocean)
# tiles="CartoDB positron" gives a clean, minimal light-colored map style
# zoom_start=2 sets the initial zoom level (2 means zoomed out to show the␣
 ↪whole world)
m = folium.Map(location=[20, 0], tiles="CartoDB positron", zoom_start=2)

# Displaying the map (works in Jupyter Notebook)
m
```

[26]: <folium.folium.Map at 0x1f99ee7c670>

[79]:
```
# Loop through each row in the DataFrame to add circular markers to the map
for i in range(0, len(data)):
    folium.Circle(
        # Setting the location using longitude and latitude (Note: This seems␣
 ↪reversed, should be [lat, lon])
        location=[data.iloc[i]['lon'], data.iloc[i]['lat']],  #  likely␣
 ↪incorrect order - should be [lat, lon]
```

```
        popup=data.iloc[i]['name'],                                    # Show city name␣
 ↪when clicked
        radius=int(data.iloc[i]['value'] * 10000),                     # Circle size␣
 ↪scaled by 'value' column
        color='crimson',                                               # Circle border␣
 ↪color
        fill=True,                                                     # Enable fill
        fill_color='crimson'                                           # Fill color same␣
 ↪as border
    ).add_to(m)  # Add the circle to the map

# Save the final map as an HTML file (uncomment the next line if you want to␣
 ↪save it)
# m.save('world_map.html')
```

[28]:
```
# Print the currently installed version of the folium library
folium.__version__
```

[28]:
```
'0.19.7'
```

[29]:
```
# Save the current folium map object `m` to an HTML file named 'mymap.html'
# You can open this file in any web browser to view the interactive map
m.save('mymap.html')
```

[30]:
```
# Import required libraries for reading a remote JSON file
from urllib.request import urlopen
import json

# Load US counties GeoJSON file from Plotly's GitHub repository
# This file contains the geographic shapes (borders) for all US counties
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/
 ↪geojson-counties-fips.json') as response:
    counties = json.load(response)

# Import pandas for handling tabular data
import pandas as pd

# Load unemployment data by US county (FIPS codes) from Plotly's GitHub
# Note: dtype={"fips": str} ensures FIPS codes are read as strings (important␣
 ↪for mapping)
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/
 ↪fips-unemp-16.csv",
                 dtype={"fips": str})

# Import Plotly's graph_objects module to create advanced visualizations
import plotly.graph_objects as go
```

```
# Create a Choropleth mapbox figure (for mapping geographic data)
fig = go.Figure(go.Choroplethmapbox(
    geojson=counties,           # US counties geometry
    locations=df.fips,          # FIPS codes for county matching
    z=df.unemp,                 # Unemployment values (used for color scale)
    colorscale="Viridis",       # Color scale used to map unemployment levels
    zmin=0, zmax=12,            # Range of the color scale
    marker_opacity=0.5,         # Transparency of county fills
    marker_line_width=0         # No borders between counties
))

# Configure the map style and initial view (centered on the US)
fig.update_layout(
    mapbox_style="carto-positron",                  # Light-themed map style
    mapbox_zoom=3,                                  # Initial zoom level
    mapbox_center={"lat": 37.0902, "lon": -95.7129} # Centered on continental US
)

# Remove extra margins around the map
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

# Display the interactive map
fig.show()
```

[32]:
```
# Importing the Cufflinks library, which connects Plotly with pandas DataFrames␣
 ↪for easy interactive plotting
import cufflinks as cf

# Setting Cufflinks to offline mode so plots render inside the Jupyter Notebook␣
 ↪without needing an internet connection
cf.go_offline()
```

[34]:
```
# Importing NumPy for numerical operations
import numpy as np

# Creating a DataFrame with 100 rows and 3 columns ('A', 'B', 'C')
# Each cell is filled with a random number drawn from the standard normal␣
 ↪distribution (mean=0, std=1)
df = pd.DataFrame(np.random.randn(100, 3), columns=['A', 'B', 'C'])

# Displaying the first 5 rows to get an initial look at the data
df.head()

# Applying cumulative sum to column 'A' and adding 20 to shift the values up
# This simulates a trending (non-stationary) time series starting around 20
df['A'] = df['A'].cumsum() + 20
```

```
# Applying the same transformation to column 'B'
df['B'] = df['B'].cumsum() + 20

# And to column 'C'
df['C'] = df['C'].cumsum() + 20
```

[35]:
```
# Importing pandas to load and work with tabular data
import pandas as pd

# Loading a CSV dataset of the top 1000 U.S. cities from Plotly's GitHub
us_cities = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/
 ↪master/us-cities-top-1k.csv")

# Importing Plotly Express for high-level interactive plotting
import plotly.express as px

# Creating a scatter plot on a Mapbox map:
# - lat/lon: coordinates for each city
# - hover_name: city name shown on hover
# - hover_data: extra info (state, population)
# - color: all points colored 'fuchsia'
# - zoom=3: zoomed out to show the US
# - height=300: map height in pixels
fig = px.scatter_mapbox(
    us_cities,
    lat="lat",
    lon="lon",
    hover_name="City",
    hover_data=["State", "Population"],
    color_discrete_sequence=["fuchsia"],
    zoom=3,
    height=300
)

# Setting the map style to OpenStreetMap (free and lightweight)
fig.update_layout(mapbox_style="open-street-map")

# Removing margins around the plot area
fig.update_layout(margin={"r": 0, "t": 0, "l": 0, "b": 0})

# Displaying the interactive map
fig.show()
```

[36]:
```
#printing first few rows
df.head()
```

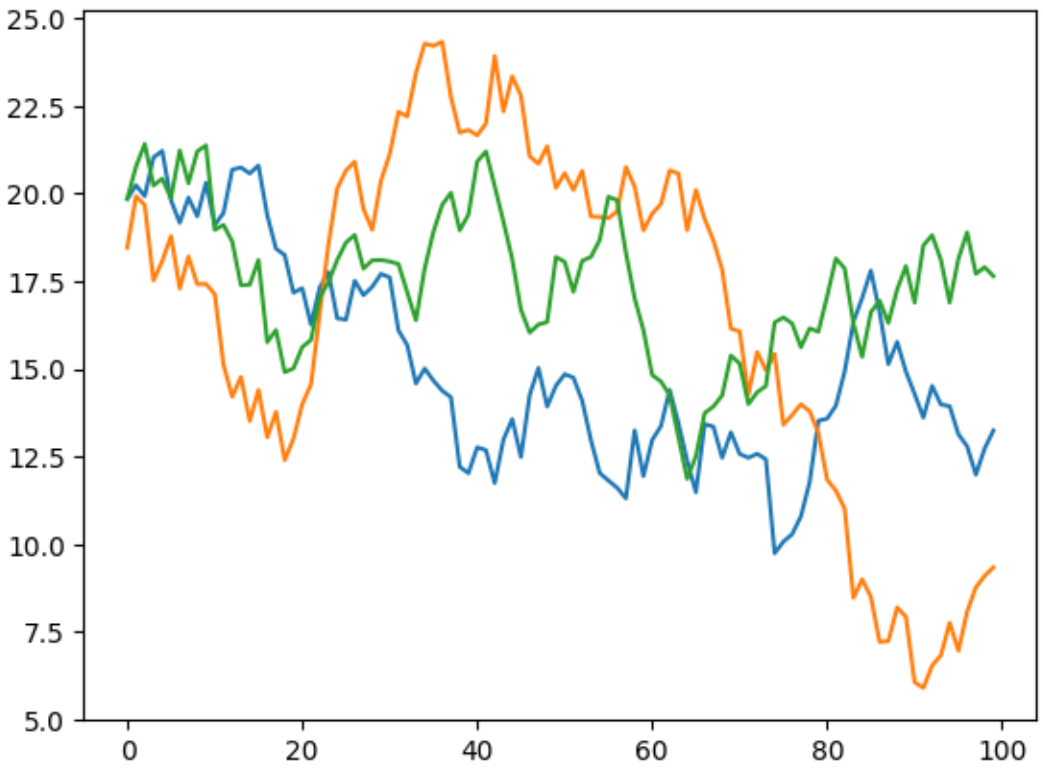
[36]:

	A	B	C
0	19.843075	18.458270	19.854691
1	20.237494	19.920759	20.768882
2	19.926224	19.672327	21.416884
3	21.030729	17.519520	20.227271
4	21.215417	18.074202	20.427433

[37]:
```
# Plotting all columns of the DataFrame `df` as interactive line plots using␣
 ↪Cufflinks and Plotly
# Each column will be plotted as a separate line on the same chart
df.iplot()
```

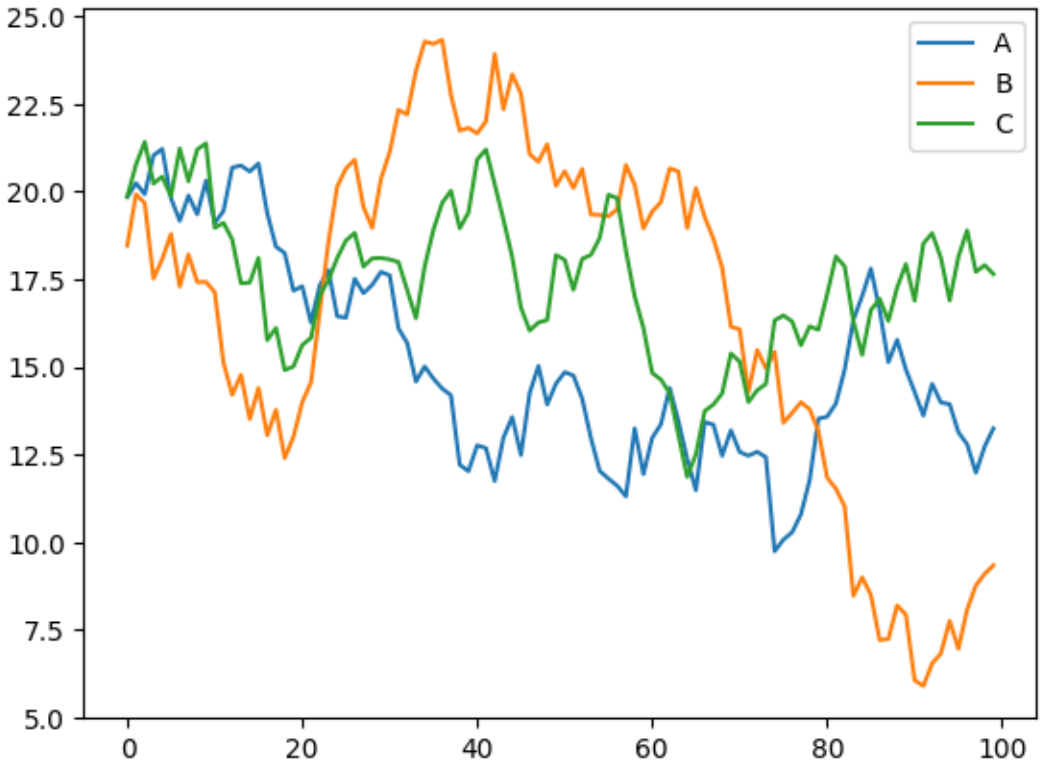
[38]:
```
# Plotting all columns of the DataFrame `df` using matplotlib's default line␣
 ↪plot
# Each column ('A', 'B', 'C') will be plotted as a separate line
plt.plot(df)
```

[38]:
```
[<matplotlib.lines.Line2D at 0x1f9a278dca0>,
 <matplotlib.lines.Line2D at 0x1f9a278dd90>,
 <matplotlib.lines.Line2D at 0x1f9a278de80>]
```



[39]:
```
# Plotting all columns of the DataFrame `df` using pandas' built-in `.plot()`
 ↪method
# This internally uses matplotlib and creates a line plot for each column
df.plot()
```

[39]: <Axes: >



[40]:
```
# Creating an interactive scatter plot using Cufflinks (.iplot)
# x-axis: column 'A'
# y-axis: column 'B'
# mode='markers' indicates a scatter plot with dots (no lines)
# size=25 sets the default marker size (optional, only works if enabled via
 ↪layout)
df.iplot(
    x='A',              # Set 'A' as x-axis
    y='B',              # Set 'B' as y-axis
    mode='markers',     # Marker mode for scatter plot
    size=25             # Marker size (this parameter is ignored by default -
 ↪see note below)
)
```

[81]:
```
# Installing the seaborn library using pip
# Seaborn is a powerful statistical data visualization library built on top of␣
 ↪matplotlib
!pip install seaborn
```

[41]:
```
# This command installs the seaborn library using the same Python interpreter␣
 ↪that's running this notebook
# It's the safest way to ensure seaborn is installed in the correct environment␣
 ↪(especially in Jupyter)
import sys
!{sys.executable} -m pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\isha\anaconda3\lib\site-
packages (0.13.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
c:\users\isha\anaconda3\lib\site-packages (from seaborn) (3.9.4)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
c:\users\isha\anaconda3\lib\site-packages (from seaborn) (1.23.5)
Requirement already satisfied: pandas>=1.2 in c:\users\isha\anaconda3\lib\site-
packages (from seaborn) (2.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: importlib-resources>=3.2.0 in
c:\users\isha\anaconda3\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (6.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\isha\anaconda3\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.0.9)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\isha\anaconda3\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.3.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\isha\anaconda3\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.25.0)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\isha\anaconda3\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pillow>=8 in c:\users\isha\anaconda3\lib\site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (9.2.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\isha\anaconda3\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.2)
Requirement already satisfied: packaging>=20.0 in
c:\users\isha\anaconda3\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (21.3)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\isha\anaconda3\lib\site-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\isha\anaconda3\lib\site-
```

```
packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: zipp>=3.1.0 in c:\users\isha\anaconda3\lib\site-
packages (from importlib-resources>=3.2.0->matplotlib!=3.6.1,>=3.4->seaborn)
(3.8.0)
Requirement already satisfied: six>=1.5 in c:\users\isha\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
```

[82]:
```
# Printing the path to the Python executable currently in use
# This helps confirm which Python environment is running your notebook
import sys
print(sys.executable)
```

```
C:\Users\ISHA\anaconda3\python.exe
```

[43]:
```
# Importing the seaborn library for statistical data visualization
import seaborn as sns

# Importing pandas to work with the dataset in tabular format
import pandas as pd

# Loading the built-in Titanic dataset from Seaborn
# This dataset contains information about passengers aboard the Titanic (e.g.,␣
 ↪age, sex, class, survival)
titanic = sns.load_dataset('titanic')

# Displaying the first 5 rows of the dataset to inspect its structure and␣
 ↪content
print(titanic.head())
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

[44]:
```
# Grouping the data by 'sex' and summing the number of survivors per gender
grouped = titanic.groupby('sex')['survived'].sum().reset_index()

# Plotting the grouped data as a bar chart using Cufflinks
grouped.iplot(
```

```
    kind='bar',
    x='sex',
    y='survived',
    title='Survived by Sex',
    xTitle='Sex',
    yTitle='# Survived',
    colors='green'
)
```

[45]:
```
# Counting the number of passengers for each gender in the 'sex' column of the␣
 ↪Titanic dataset
titanic['sex'].value_counts()
```

[45]:
```
sex
male      577
female    314
Name: count, dtype: int64
```

[46]:
```
# Displaying the list of available Cufflinks themes for styling interactive␣
 ↪plots
cf.getThemes()
```

[46]:
```
['ggplot', 'pearl', 'solar', 'space', 'white', 'polar', 'henanigans']
```

[47]:
```
# Setting the visual theme for all future Cufflinks plots to 'polar'
# This controls colors, background, gridlines, and font styling
cf.set_config_file(theme='polar')

# Creating a stacked bar chart using the DataFrame `df`
# kind='bar'          : bar chart
# barmode='stack'     : stacks values from columns A, B, and C on top of each␣
 ↪other
# bargap=0.5          : sets spacing between bars (0.0 = no gap, 1.0 = full gap)
df.iplot(
    kind='bar',
    barmode='stack',
    bargap=0.5
)
```

[48]:
```
# Plotting a stacked bar chart of all columns in the DataFrame `df` using␣
 ↪Cufflinks and Plotly
df.iplot(
    kind='bar',       # Set plot type to bar chart
    barmode='stack',  # Stack values from different columns on top of each other
    bargap=0.5        # Set spacing between bars (0 = no gap, 1 = full gap)
)
```

[49]:
```
# Creating a stacked horizontal bar chart using Cufflinks (Plotly wrapper for␣
 ↪pandas)
df.iplot(
    kind='barh',       # 'barh' indicates horizontal bars instead of vertical
    barmode='stack',   # Stack values from columns ('A', 'B', 'C') horizontally␣
 ↪for each row
    bargap=0.5         # Set the gap between bars (0 = no gap, 1 = max gap)
)
```

[50]:
```
1,2,3,4,5,6,7
```

[50]:
```
(1, 2, 3, 4, 5, 6, 7)
```

[51]:
```
# Creating an interactive box plot using Cufflinks and Plotly
# This will generate a separate box for each column in the DataFrame (`df`)
df.iplot(kind='box')
```

[52]:
```
# Creating an interactive line plot for all columns in the DataFrame `df` using␣
 ↪Cufflinks/Plotly
# Each column (e.g., 'A', 'B', 'C') is plotted as a separate line
df.iplot()
```

[53]:
```
# Creating an interactive area chart using Cufflinks and Plotly
# This plots each column in the DataFrame as a filled area under the line
df.iplot(kind='area')
```

[54]:
```
# Creating a new DataFrame `df3` with 5 rows and 3 columns: 'X', 'Y', and 'Z'
# Each column contains a symmetrical sequence of integers
df3 = pd.DataFrame({
    'X': [10, 20, 30, 20, 10],
    'Y': [10, 20, 30, 20, 10],
    'Z': [10, 20, 30, 20, 10]
})

# Displaying the first 5 rows of df3 to verify its contents
df3.head()
```

[54]:

	X	Y	Z
0	10	10	10
1	20	20	20
2	30	30	30
3	20	20	20
4	10	10	10

[55]:
```
# Creating a 3D surface plot using Cufflinks and Plotly
# kind='surface' renders a 3D surface (like a terrain map)
# colorscale='rdylbu' sets the color gradient (Red-Yellow-Blue theme)
```

```
df3.iplot(
    kind='surface',
    colorscale='rdylbu'
)
```

[60]:
```
# Enable offline mode for Cufflinks to render plots inside Jupyter Notebook␣
 ↪without an internet connection
cf.go_offline()

# Generate a sample DataFrame with random line data using Cufflinks' built-in␣
 ↪data generator
# This returns a DataFrame with columns like 'A', 'B', 'C', etc., containing␣
 ↪synthetic time-series data
df = cf.datagen.lines()

# Create an interactive line plot of the generated data
# Each column is plotted as a separate line
df.iplot(title="Random Line Plot from Cufflinks DataGen")
```

[84]:
```
# Generate a sinusoidal wave dataset using Cufflinks' data generator
# Arguments:
# - 10     : number of points
# - 0.25   : frequency factor
# The result is a DataFrame representing a sine wave surface
cf.datagen.sinwave(10, 0.25).iplot(kind='surface')
```

[62]:
```
# Generate a 3D scatter dataset using Cufflinks' built-in data generator
# Parameters:
# - 2     : number of series
# - 150   : number of points per series
# - mode='stocks' : simulate stock-like movement for realistic-looking data
cf.datagen.scatter3d(2, 150, mode='stocks').iplot(
    kind='scatter3d',  # Type of plot: interactive 3D scatter
    x='x',             # Assign 'x' column to X-axis
    y='y',             # Assign 'y' column to Y-axis
    z='z'              # Assign 'z' column to Z-axis
)
```

[85]:
```
#printing columns in dataframe
print(df.columns)
```

```
Index(['URZ.YV', 'MGZ.RA', 'UGY.VT', 'FSU.PW', 'CBQ.BF'], dtype='object')
```

[76]:
```
# Creating an interactive histogram plot using Cufflinks and Plotly
df.iplot(
    kind='hist',      # Plot type: histogram
    bins=25,          # Number of bins to divide the data range into
```

```
    barmode='overlay',# Bars from different columns will overlap instead of␣
 ↪stacking side-by-side
    bargap=0.5        # Gap between bars (0 = no gap, 1 = full gap)
)
```

[77]:
```
# Generate a 3D bubble dataset using Cufflinks' built-in data generator
# Parameters:
# - 5     : number of data points per series
# - 4     : number of series (or categories)
# - mode='stocks' : generates stock-like patterns for x, y, z, and size
df = cf.datagen.bubble3d(5, 4, mode='stocks')

# Create a 3D bubble chart using the generated data
df.iplot(
    kind='bubble3d',  # Type of chart: interactive 3D bubbles
    x='x',            # x-axis values
    y='y',            # y-axis values
    z='z',            # z-axis values (depth)
    size='size'       # size of each bubble is determined by the 'size' column
)
```

[78]:
```
# Generate a 20x20 heatmap dataset using Cufflinks' built-in data generator
# This simulates a 2D grid of values like a correlation matrix or intensity map
df = cf.datagen.heatmap(20, 20)

# Create an interactive heatmap using Plotly via Cufflinks
df.iplot(
    kind='heatmap',           # Set plot type to heatmap
    colorscale='spectral',    # Use the 'spectral' color gradient (rainbow-like)
    title='Cufflinks - Heatmap'  # Set chart title
)
```