ClusterKit

Plotly & Dash Overview

Content

NumPy

NumPy Overview

NumPy is a first-rate library for numerical programming

- Widely used in academia, finance and industry.
- Mature, fast, stable and under continuous development.

The essential problem that NumPy solves is fast array processing.

For example, suppose we want to create an array of 1 million random draws from a uniform distribution and compute the mean.

If we did this in pure Python it would be orders of magnitude slower than C or Fortran.

This is because

- Loops in Python over Python data types like lists carry significant overhead.
- C and Fortran code contains a lot of type information that can be used for optimization.
- Various optimizations can be carried out during compilation when the compiler sees the instructions as a whole.

However, for a task like the one described above, there's no need to switch back to C or Fortran.

Instead, we can use NumPy, where the instructions look like this:

```
import numpy as np

x = np.random.uniform(0, 1, size=1000000)
x
```

```
array([0.14508844, 0.96544075, 0.47891906, ..., 0.02291455, 0.88634353,
       0.24104596])
```

```
x.mean()
```

```
0.49987683598389504
```

The operations of creating the array and computing its mean are both passed out to carefully optimized machine code compiled from C.

More generally, NumPy sends operations *in batches* to optimized C and Fortran code.

This is similar in spirit to Matlab, which provides an interface to fast Fortran routines.

NumPy is great for operations that are naturally *vectorized*.

Vectorized operations are precompiled routines that can be sent in batches, like

- matrix multiplication and other linear algebra routines
- generating a vector of random numbers
- applying a fixed transformation (e.g., sine or cosine) to an entire array

References

https://numpy.org/doc/1.17/reference/index.html

NumPy Arrays

Basic

```
my_list = [0, 1, 2, 3, 4]
my_list
```

```
[0, 1, 2, 3, 4]
```

```
arr = np.array(my_list)
arr
```

```
array([0, 1, 2, 3, 4])
```

```
type(arr)
```

```
numpy.ndarray
```

Arange integers, takes in start, stop, and step size

```
a = np.arange(0, 10)
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
a = np.arange(0, 10, 2)
a
```

```
array([0, 2, 4, 6, 8])
```

Create an array of zeros

```
a = np.zeros((5, 5))
a
```

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

Create an array of ones

```
a = np.ones((2, 4))
a
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

Create an array of random integers

```
a = np.random.randint(0, 10)
a
```

```
3
```

Create 2d matrix of random integers

```
a = np.random.randint(0, 10, (3, 3))
a
```

```
array([[5, 0, 5],
       [5, 4, 0],
       [6, 0, 3]])
```

Create linearly spaced array

```
a = np.linspace(0, 10, 6)
a
```

```
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

Operations On Arrays

```
arr = np.random.randint(0, 100, 10)
arr
```

```
array([50, 16, 20, 51, 51, 47, 74, 94, 20, 43])
```

```
arr.max()
```

```
94
```

```
arr.min()
```

```
16
```

```
arr.mean()
```

```
46.6
```

```
arr.argmin()
```

```
1
```

```
arr.argmax()
```

```
7
```

```
arr.reshape(2, 5)
```

```
array([[50, 16, 20, 51, 51],
       [47, 74, 94, 20, 43]])
```

Additional Functionality

```
mat = np.arange(0, 100).reshape(10, 10)
mat = np.arange(0, 100).reshape(10, 10)
mat
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

```
row = 0
col = 1
```

Select an individual number

```
mat[row, col]
```

```
1
```

Select an entire column

```
mat[:, col]
```

```
array([ 1, 11, 21, 31, 41, 51, 61, 71, 81, 91])
```

Select an entire row

```
mat[row, :]
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Masking

```
mat > 50
```

```
array([[False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True]])
```

```
mat[mat>50]
```

```
array([51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

Masking allows you to use conditional filters to grab elements we'll see this idea used in pandas.

Pandas

Pandas Overview

Pandas is a package of fast, efficient data analysis tools for Python.

Its popularity has surged in recent years, coincident with the rise of fields such as data science and machine learning.

Just as NumPy provides the basic array data type plus core array operations, pandas

1. defines fundamental structures for working with data and
2. endows them with methods that facilitate operations such as
   - reading in data
   - adjusting indices
   - working with dates and time series
   - sorting, grouping, re-ordering and general data munging
   - dealing with missing values, etc., etc.

More sophisticated statistical functionality is left to other packages, such as statsmodels and scikit-learn, which are built on top of pandas.

References

https://pandas.pydata.org/pandas-docs/stable/

This lecture will provide a basic introduction to pandas.

Throughout the lecture, we will assume that the following imports have taken place

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

Series

Two important data types defined by pandas are `Series` and `DataFrame`.

You can think of a `Series` as a “column” of data, such as a collection of observations on a single variable.

A `DataFrame` is an object for storing related columns of data.

Let’s start with Series

```
s = pd.Series(np.random.randn(4), name='daily returns')
s
```

```
0   -0.183012
1   -0.599447
2    1.899475
3    1.684493
Name: daily returns, dtype: float64
```

Here you can imagine the indices `0, 1, 2, 3` as indexing four listed companies, and the values being daily returns on their shares.

Pandas `Series` are built on top of NumPy arrays and support many similar operations

```
s * 100
```

```
0    -18.301209
1    -59.944681
2    189.947488
3    168.449330
Name: daily returns, dtype: float64
```

```
np.abs(s)
```

```
0    0.183012
1    0.599447
2    1.899475
3    1.684493
Name: daily returns, dtype: float64
```

But `Series` provide more than NumPy arrays.

Not only do they have some additional (statistically oriented) methods

```
s.describe()
```

```
count    4.000000
mean     0.700377
std      1.274917
min     -0.599447
25%     -0.287121
50%      0.750741
75%      1.738239
max      1.899475
Name: daily returns, dtype: float64
```

But their indices are more flexible

```
s.index = ['AMZN', 'AAPL', 'MSFT', 'GOOG']
s
```

```
AMZN   -0.183012
AAPL   -0.599447
MSFT    1.899475
GOOG    1.684493
Name: daily returns, dtype: float64
```

Viewed in this way, `Series` are like fast, efficient Python dictionaries (with the restriction that the items in the dictionary all have the same type —in this case, floats).

In fact, you can use much of the same syntax as Python dictionaries

```
s['AMZN']
```

```
-0.18301208606983488
```

```
s['AMZN'] = 0
s
```

```
AMZN    0.000000
AAPL   -0.599447
MSFT    1.899475
GOOG    1.684493
Name: daily returns, dtype: float64
```

```
'AAPL' in s
```

```
True
```

DataFrames

While a `Series` is a single column of data, a `DataFrame` is several columns, one for each variable.

In essence, a `DataFrame` in pandas is analogous to a (highly optimized) Excel spreadsheet.

Thus, it is a powerful tool for representing and analyzing data that are naturally organized into rows and columns, often with descriptive indexes for individual rows and individual columns.

Let’s look at an example that reads data from the CSV file `dataset/salaries.csv`

Here’s the content of `salaries.csv`

```
Name,Salary,Age
John,50000,34
Sally,120000,45
Alyssa,80000,27
```

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/salaries.csv')
type(df)
```

```
pandas.core.frame.DataFrame
```

```
df
```

	Name	Salary	Age
0	John	50000	34
1	Sally	120000	45
2	Alyssa	80000	27

Select particular rows using standard Python array slicing notation

```
df[:2]
```

	Name	Salary	Age
0	John	50000	34
1	Sally	120000	45

Select columns with a bracket call

```
df['Name']
```

```
0       John
1      Sally
2     Alyssa
Name: Name, dtype: object
```

```
df['Salary']
```

```
0      50000
1     120000
2      80000
Name: Salary, dtype: int64
```

Select multiple columns with a list of column names, Since you are passing in a list, you see two sets of []

```
df[['Name', 'Salary']]
```

	Name	Salary
0	John	50000
1	Sally	120000
2	Alyssa	80000

Similar to NumPy, you can calls min(), max(), mean(), etc... on a pandas dataframe

```
df['Age'].mean()
```

```
35.333333333333336
```

Just like Numpy, we can use conditional filtering to select rows that meet certain critera. Like choosing rows where the Age value is greater than 30

```
df['Age'] > 30
```

```
0      True
1      True
2     False
Name: Age, dtype: bool
```

```
df[df['Age'] > 30]
```

	Name	Salary	Age
0	John	50000	34
1	Sally	120000	45

Get list of unique values for Age

```
df['Age'].unique()
```

```
array([34, 45, 27])
```

Get number of unqiue values

```
df['Age'].nunique()
```

```
3
```

General info about dataframe

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
Name      3 non-null object
Salary    3 non-null int64
Age       3 non-null int64
dtypes: int64(2), object(1)
memory usage: 200.0+ bytes
```

Statistics about dataframe

```
df.describe()
```

	Salary	Age
count	3.000000	3.000000
mean	83333.333333	35.333333
std	35118.845843	9.073772
min	50000.000000	27.000000
25%	65000.000000	30.500000
50%	80000.000000	34.000000
75%	100000.000000	39.500000
max	120000.000000	45.000000

Get a list of all columns

```
df.columns
```

```
Index(['Name', 'Salary', 'Age'], dtype='object')
```

Set dataframe index

```
df = df.set_index('Name')
df
```

	Salary	Age
Name		
John	50000	34
Sally	120000	45
Alyssa	80000	27

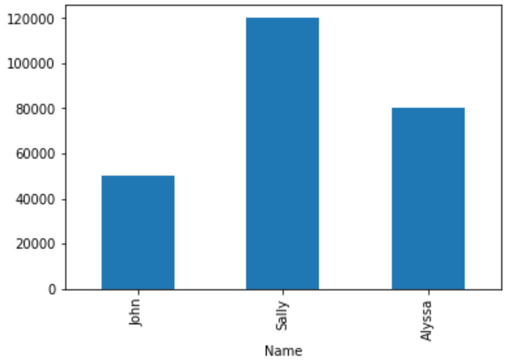
Get a list of indices

```
df.index
```

```
Index(['John', 'Sally', 'Alyssa'], dtype='object', name='Name')
```

Plot graph using matplotlib

```
df['Salary'].plot(kind='bar')
plt.show()
```



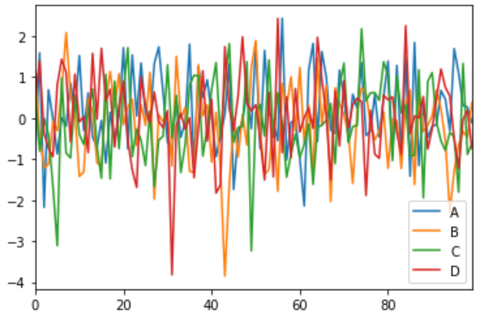
You can convert a numpy matrix to a dataframe with

```
mat = np.random.randn(100, 4)
df = pd.DataFrame(mat, columns=['A', 'B', 'C', 'D'])
df
```

	A	B	C	D
0	-0.913369	1.081135	0.575741	0.516506
1	1.583831	-0.417009	-0.809022	1.387205
2	-2.178086	-1.215231	-0.010966	-0.377157
3	0.686784	-1.126198	-0.393500	-0.737001
4	-0.016711	-0.093503	-1.644211	-0.943567
...	...	...	...	...
95	1.689986	-1.286735	-0.465283	-0.515286
96	1.096142	-0.656740	-1.806754	-1.221919
97	0.306505	-0.326553	1.328013	-0.039685
98	0.263024	-0.475811	-0.889015	0.187923
99	-0.108021	0.371692	-0.635198	-0.736648

100 rows × 4 columns

```
df.plot()
plt.show()
```



Plotly

Plotly Overview

Plotly's Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts.

Plotly.py is free and open source and you can view the source, report issues or contribute on GitHub.

References

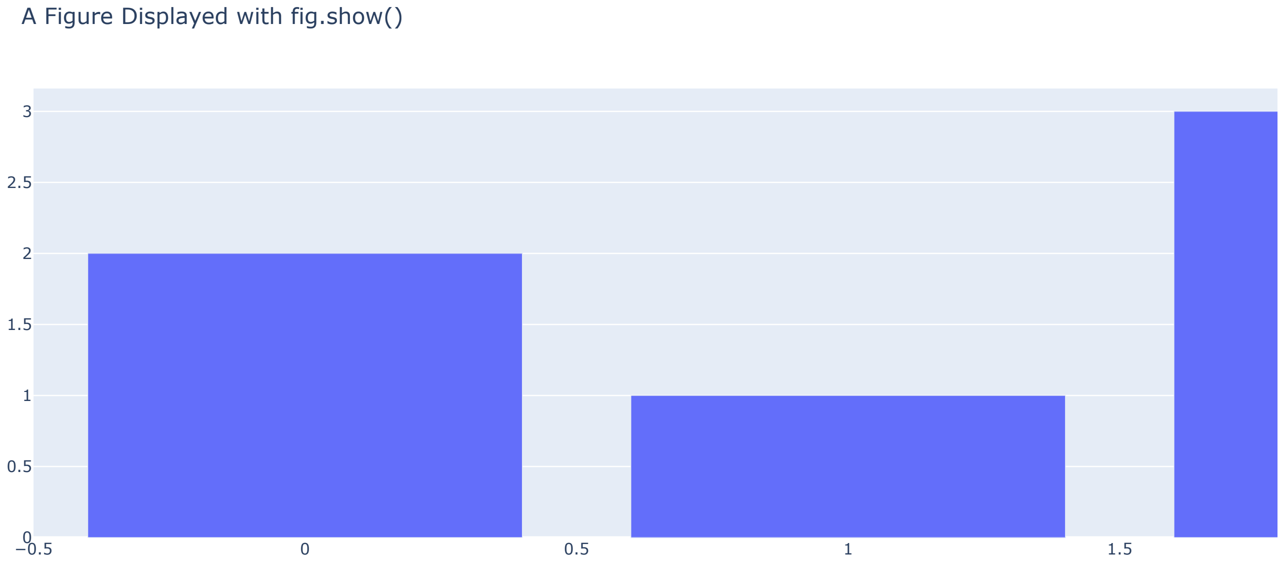
https://plot.ly/python/

Throughout the lecture, we will assume that the following imports have taken place

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
```

Plotly Basic

```
fig = go.Figure(
    data=[go.Bar(y=[2, 1, 3])],
    layout_title_text="A Figure Displayed with fig.show()"
)
fig.show()
```



A Figure Displayed with fig.show()

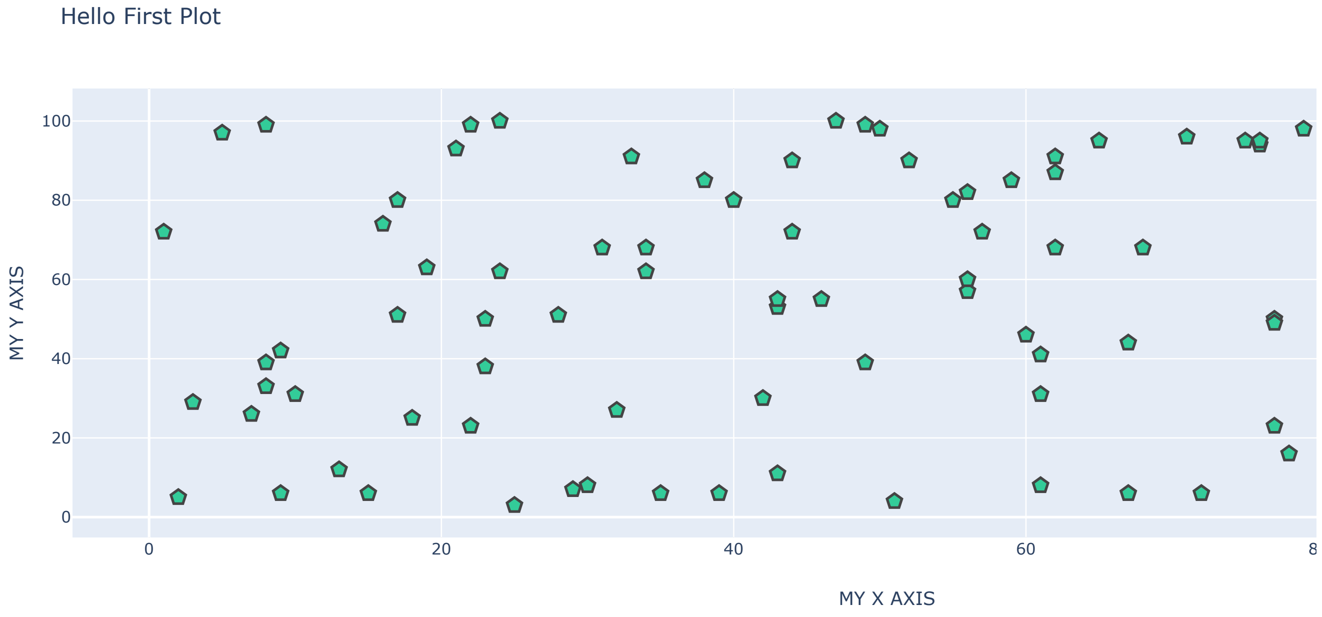
Scatterplot

```
random_x = np.random.randint(1, 101, 100)
random_y = np.random.randint(1, 101, 100)

data = [go.Scatter(x=random_x,
                   y=random_y,
                   mode='markers',
                   marker=dict(
                       size=12,
                       color='rgb(51,204,153)',
                       symbol='pentagon',
                       line=dict(width=2)
                   ))]

layout = go.Layout(title='Hello First Plot',
                   xaxis=dict(title='MY X AXIS'),
                   yaxis=dict(title='MY Y AXIS'),
                   hovermode='closest')

fig = go.Figure(data=data, layout=layout)
fig.show()
```



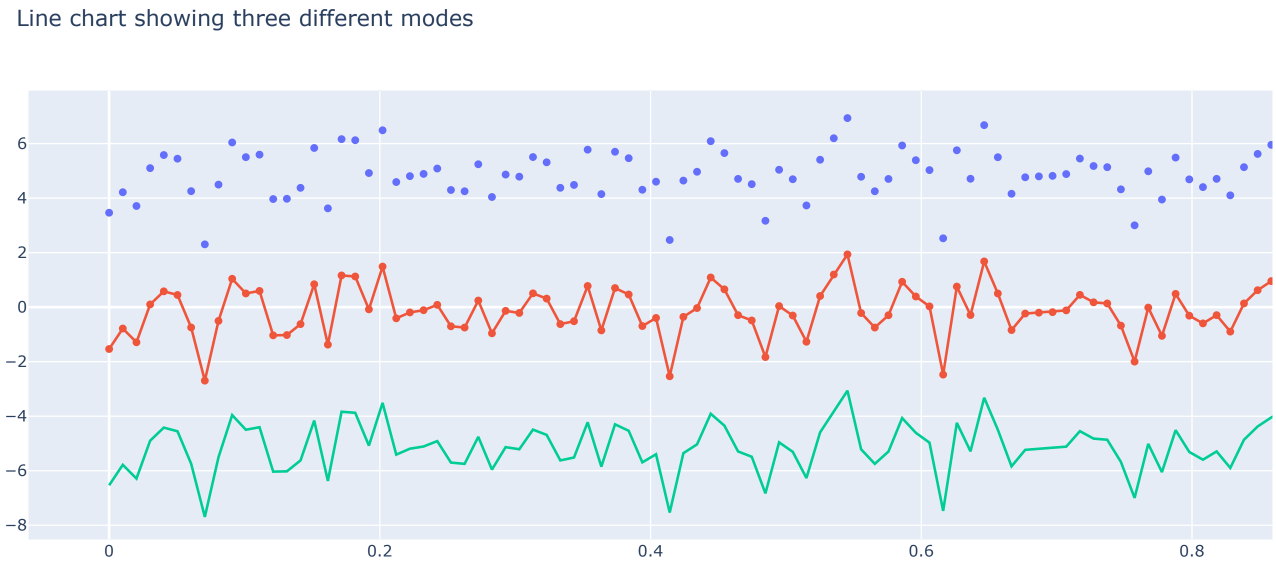
Hello First Plot

Line Chart

Mode

```
x_values = np.linspace(0, 1, 100) # 100 evenly spaced values
y_values = np.random.randn(100)   # 100 random values

# create traces
trace0 = go.Scatter(
    x = x_values,
    y = y_values+5,
    mode = 'markers',
    name = 'markers'
)
trace1 = go.Scatter(
    x = x_values,
    y = y_values,
    mode = 'lines+markers',
    name = 'lines+markers'
)
trace2 = go.Scatter(
    x = x_values,
    y = y_values-5,
    mode = 'lines',
    name = 'lines'
)
data = [trace0, trace1, trace2]  # assign traces to data
layout = go.Layout(
    title = 'Line chart showing three different modes'
)
fig = go.Figure(data=data,layout=layout)
fig.show()
```



Using population.csv dataset

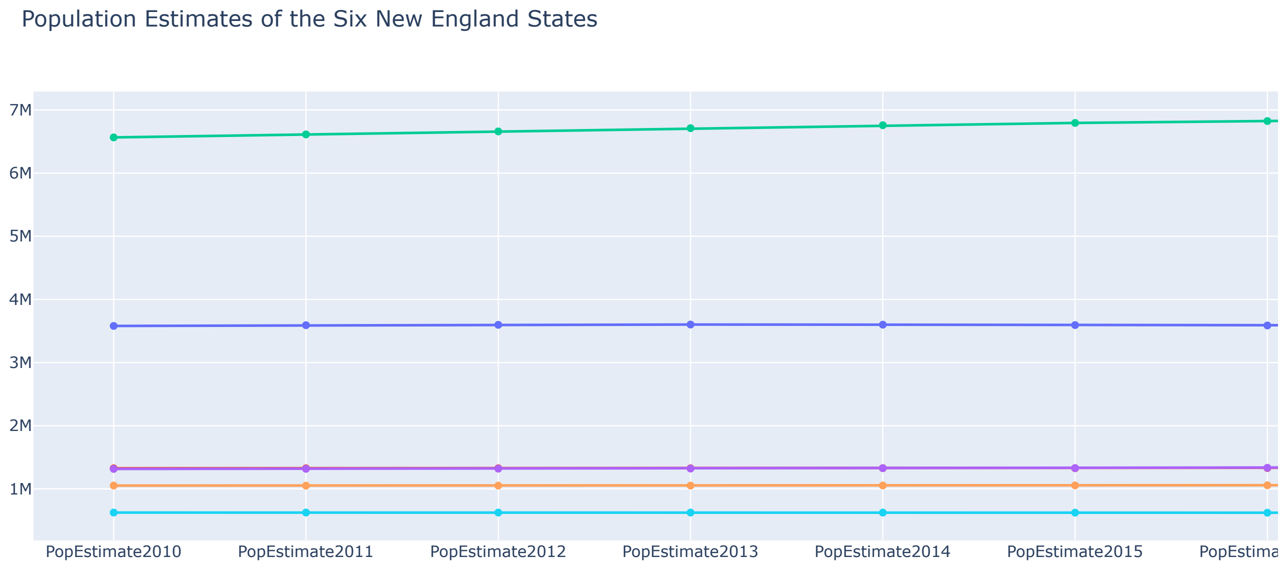
```
# read a .csv file into a pandas DataFrame:
df = pd.read_csv(
    'https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/population.csv',
    index_col=0)
df
```

Name	PopEstimate2010	PopEstimate2011	PopEstimate2012	PopEstimate2013	PopEstimate2014	PopEstimate2015	PopEstimate2016
Connecticut	3580171	3591927	3597705	3602470	3600188	3593862	3587685
Maine	1327568	1327968	1328101	1327975	1328903	1327787	1330232
Massachusetts	6564943	6612178	6659627	6711138	6757925	6794002	6823721
New Hampshire	1316700	1318345	1320923	1322622	1328684	1330134	1335015
Rhode Island	1053169	1052154	1052761	1052784	1054782	1055916	1057566
Vermont	625842	626210	625606	626044	625665	624455	623354

```
# create traces
traces = [go.Scatter(
    x = df.columns,
    y = df.loc[name],
    mode = 'markers+lines',
    name = name
) for name in df.index]

layout = go.Layout(
    title = 'Population Estimates of the Six New England States'
)

fig = go.Figure(data=traces,layout=layout)
fig.show()
```


Population Estimates of the Six New England States

Using 2010YumaAZ.csv dataset

```
# Create a pandas DataFrame from 2010YumaAZ.csv
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2010YumaAZ.csv')
df
```

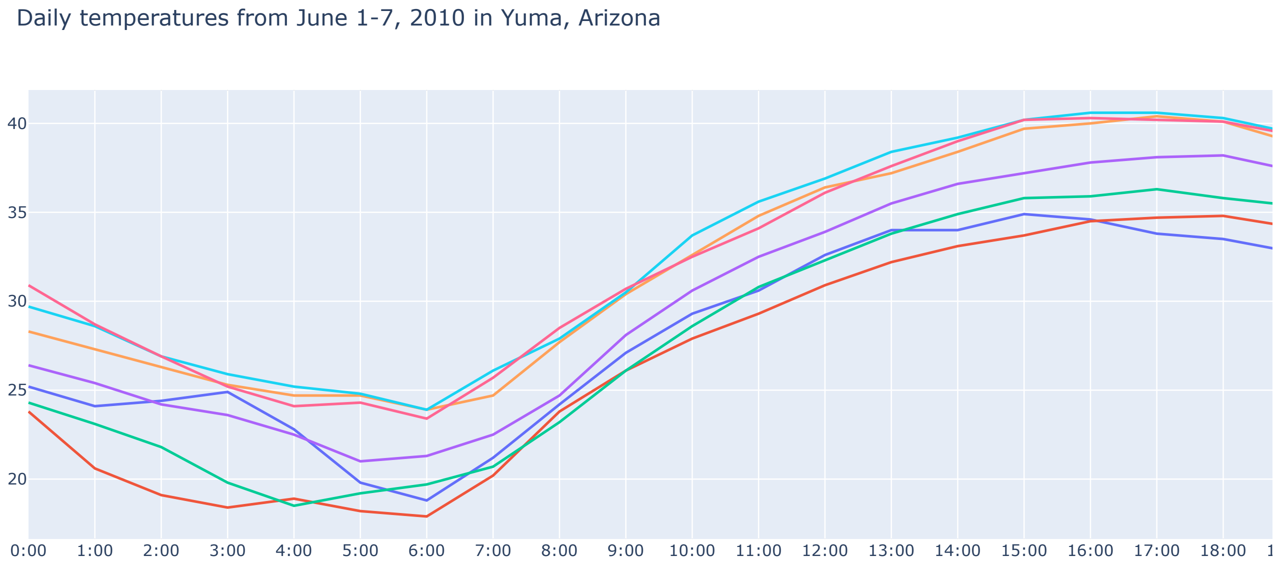
	LST_DATE	DAY	LST_TIME	T_HR_AVG
0	20100601	TUESDAY	0:00	25.2
1	20100601	TUESDAY	1:00	24.1
2	20100601	TUESDAY	2:00	24.4
3	20100601	TUESDAY	3:00	24.9
4	20100601	TUESDAY	4:00	22.8
...	...	...	...	...
163	20100607	MONDAY	19:00	39.4
164	20100607	MONDAY	20:00	38.5
165	20100607	MONDAY	21:00	37.0
166	20100607	MONDAY	22:00	34.7
167	20100607	MONDAY	23:00	32.6

168 rows × 4 columns

```
# Define a data variable
data = [{
    'x': df['LST_TIME'],
    'y': df[df['DAY']==day]['T_HR_AVG'],
    'name': day
} for day in df['DAY'].unique()]

# Define the layout
layout = go.Layout(
    title='Daily temperatures from June 1-7, 2010 in Yuma, Arizona',
    hovermode='closest'
)

# Create a fig from data and layout, and plot the fig
fig = go.Figure(data=data, layout=layout)
fig.show()
```



Bar Chart

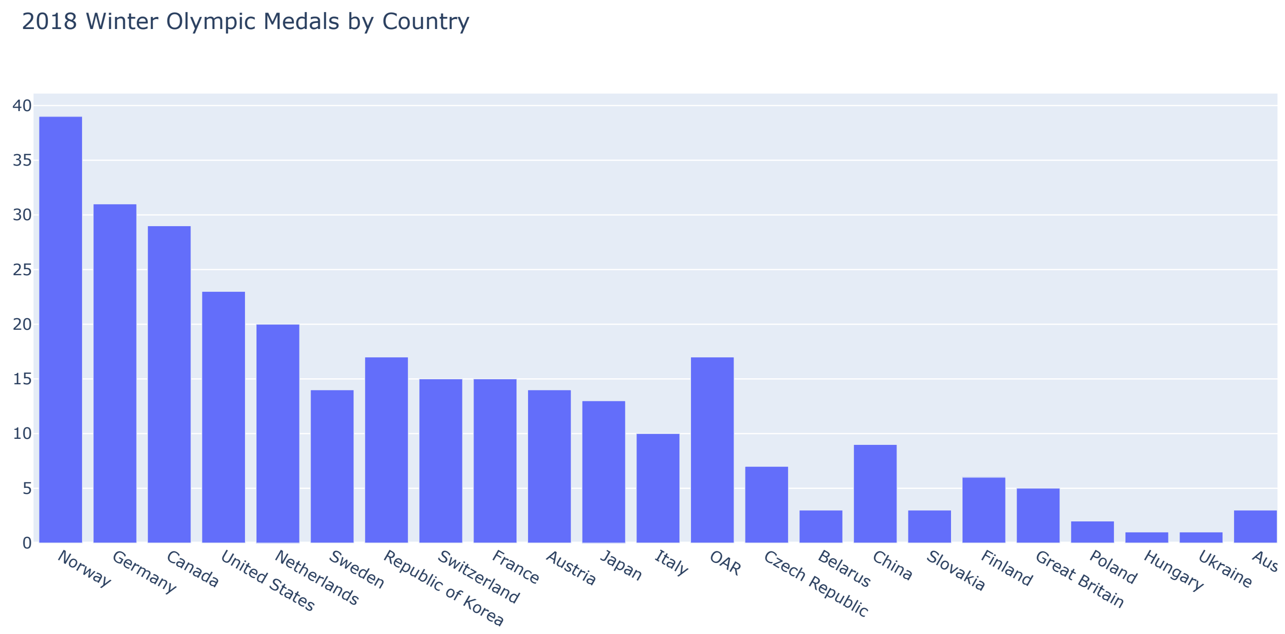
Using 2018WinterOlympics.csv dataset

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2018WinterOlympics.csv')
df
```

	Rank	NOC	Gold	Silver	Bronze	Total
0	1	Norway	14	14	11	39
1	2	Germany	14	10	7	31
2	3	Canada	11	8	10	29
3	4	United States	9	8	6	23
4	5	Netherlands	8	6	6	20
5	6	Sweden	7	6	1	14
6	7	Republic of Korea	5	8	4	17
7	8	Switzerland	5	6	4	15
8	9	France	5	4	6	15
9	10	Austria	5	3	6	14
10	11	Japan	4	5	4	13
11	12	Italy	3	2	5	10
12	13	OAR	2	6	9	17
13	14	Czech Republic	2	2	3	7
14	15	Belarus	2	1	0	3
15	16	China	1	6	2	9
16	17	Slovakia	1	2	0	3
17	18	Finland	1	1	4	6
18	19	Great Britain	1	0	4	5
19	20	Poland	1	0	1	2
20	21	Hungary	1	0	0	1
21	21	Ukraine	1	0	0	1
22	23	Australia	0	2	1	3
23	24	Slovenia	0	1	1	2
24	25	Belgium	0	1	0	1
25	26	Spain	0	0	2	2
26	26	New Zealand	0	0	2	2
27	28	Kazakhstan	0	0	1	1
28	28	Latvia	0	0	1	1
29	28	Liechtenstein	0	0	1	1

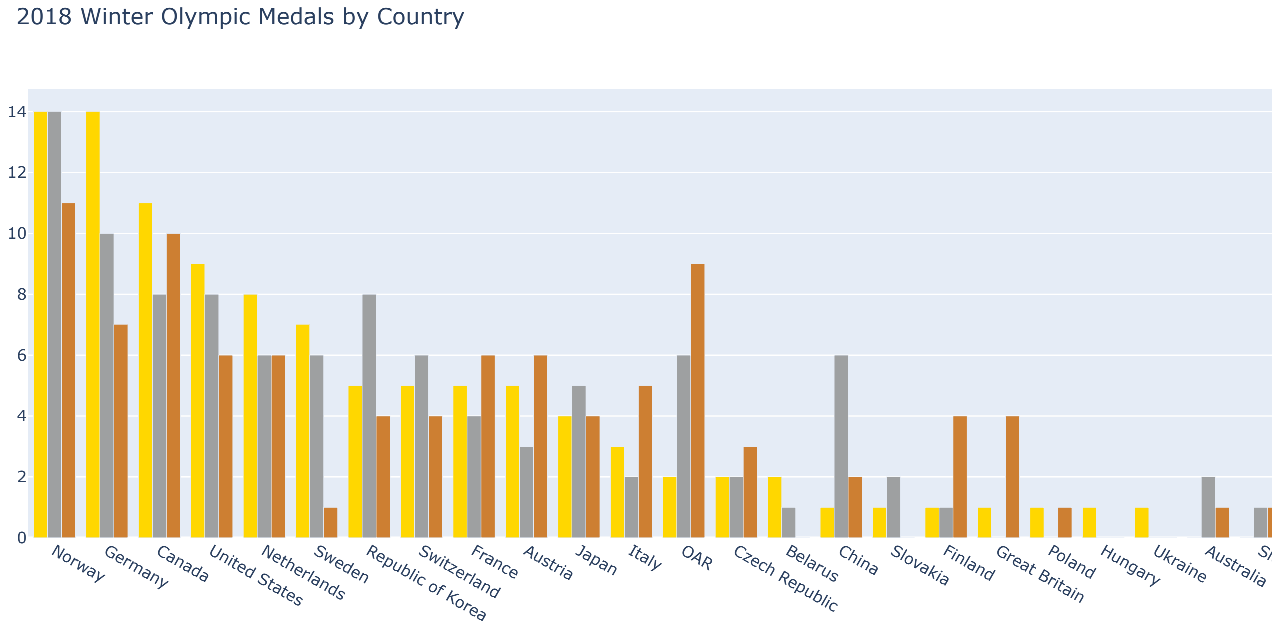
basic bar chart showing the total number of 2018 Winter Olympics Medals won by Country.

```
data = [go.Bar(
    x=df['NOC'],  # NOC stands for National Olympic Committee
    y=df['Total']
)]
layout = go.Layout(
    title='2018 Winter Olympic Medals by Country'
)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



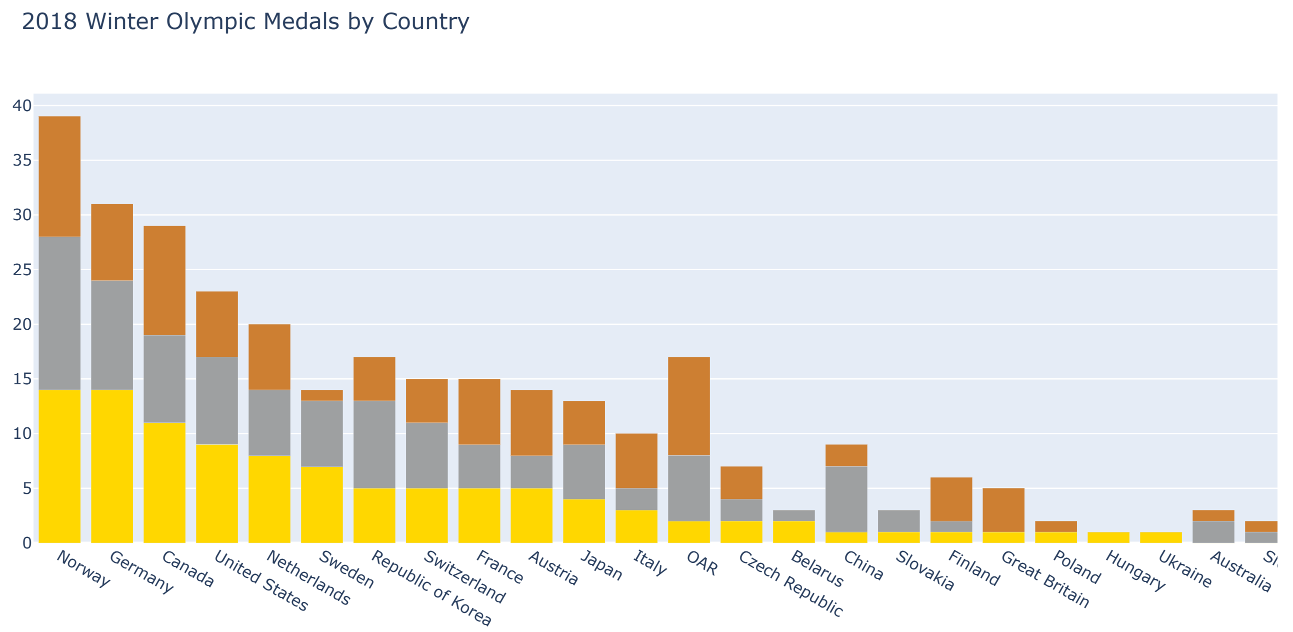
grouped bar chart showing three traces (gold, silver and bronze medals won) for each country

```
trace1 = go.Bar(
    x=df['NOC'],  # NOC stands for National Olympic Committee
    y=df['Gold'],
    name = 'Gold',
    marker=dict(color='#FFD700') # set the marker color to gold
)
trace2 = go.Bar(
    x=df['NOC'],
    y=df['Silver'],
    name='Silver',
    marker=dict(color='#9EA0A1') # set the marker color to silver
)
trace3 = go.Bar(
    x=df['NOC'],
    y=df['Bronze'],
    name='Bronze',
    marker=dict(color='#CD7F32') # set the marker color to bronze
)
data = [trace1, trace2, trace3]
layout = go.Layout(
    title='2018 Winter Olympic Medals by Country'
)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



stacked bar chart showing three traces (gold, silver and bronze medals won) for each country

```
trace1 = go.Bar(
    x=df['NOC'],  # NOC stands for National Olympic Committee
    y=df['Gold'],
    name = 'Gold',
    marker=dict(color='#FFD700') # set the marker color to gold
)
trace2 = go.Bar(
    x=df['NOC'],
    y=df['Silver'],
    name='Silver',
    marker=dict(color='#9EA0A1') # set the marker color to silver
)
trace3 = go.Bar(
    x=df['NOC'],
    y=df['Bronze'],
    name='Bronze',
    marker=dict(color='#CD7F32') # set the marker color to bronze
)
data = [trace1, trace2, trace3]
layout = go.Layout(
    title='2018 Winter Olympic Medals by Country',
    barmode='stack'
)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



Bubble Chart

Using mpg.csv dataset

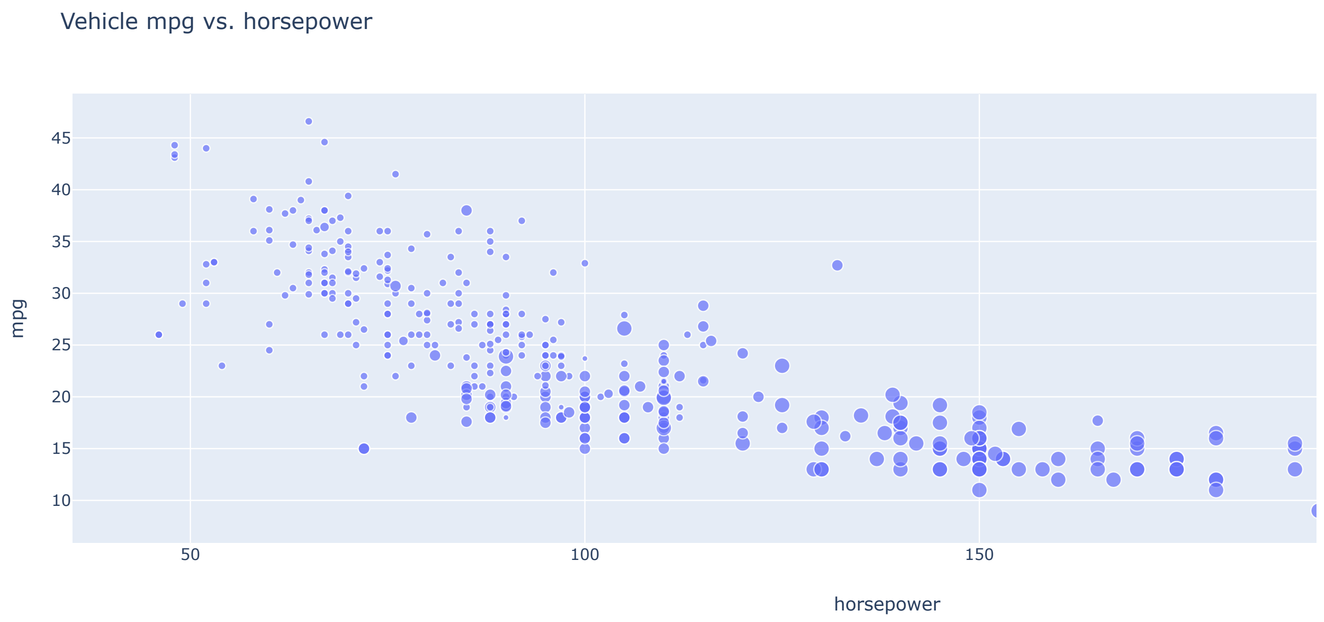
```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/mpg.csv')
df
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

```
data = [go.Scatter(          # start with a normal scatter plot
    x=df['horsepower'],
    y=df['mpg'],
    text=df['name'],
    mode='markers',
    marker=dict(size=1.5*df['cylinders']) # set the marker size
)]

layout = go.Layout(
    title='Vehicle mpg vs. horsepower',
    xaxis = dict(title = 'horsepower'), # x-axis label
    yaxis = dict(title = 'mpg'),        # y-axis label
    hovermode='closest'
)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



A bubble chart is simply a scatter plot with the added feature that the size of the marker can be set by the data.

Heatmap

Heatmap of temperatures for Santa Barbara, California

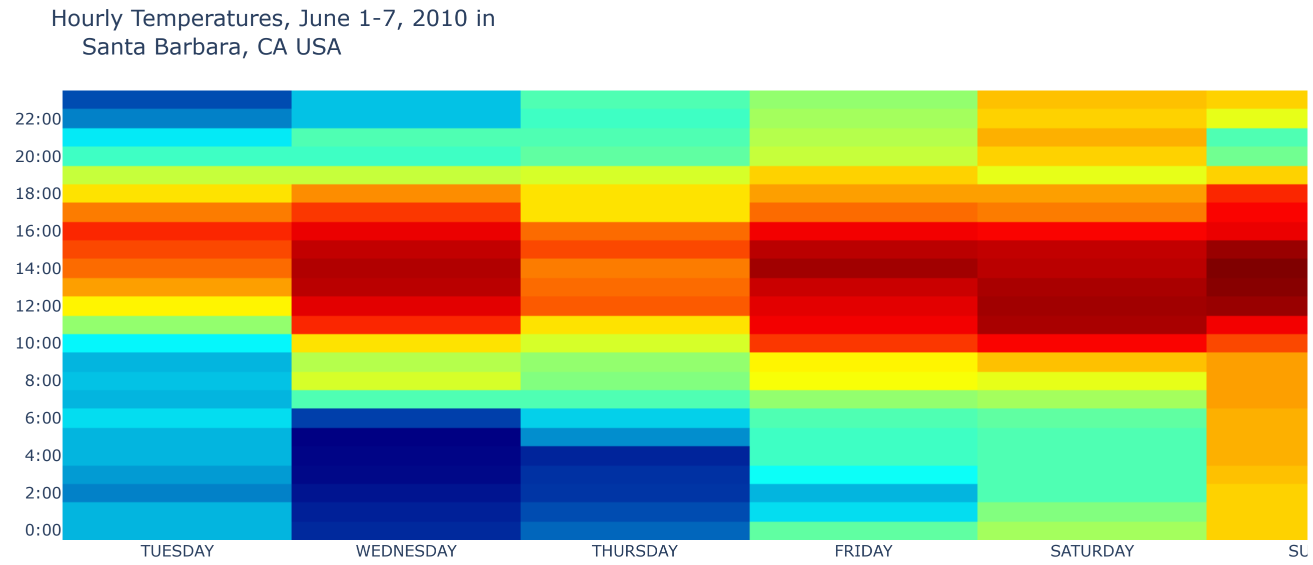
```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2010SantaBarbaraCA.csv')
df
```

	LST_DATE	DAY	LST_TIME	T_HR_AVG
0	20100601	TUESDAY	0:00	12.7
1	20100601	TUESDAY	1:00	12.7
2	20100601	TUESDAY	2:00	12.3
3	20100601	TUESDAY	3:00	12.5
4	20100601	TUESDAY	4:00	12.7
...	...	...	...	...
163	20100607	MONDAY	19:00	15.6
164	20100607	MONDAY	20:00	14.8
165	20100607	MONDAY	21:00	14.3
166	20100607	MONDAY	22:00	14.4
167	20100607	MONDAY	23:00	14.6

168 rows × 4 columns

```
data = [go.Heatmap(
    x=df['DAY'],
    y=df['LST_TIME'],
    z=df['T_HR_AVG'],
    colorscale='Jet'
)]

layout = go.Layout(
    title='Hourly Temperatures, June 1-7, 2010 in<br>\
    Santa Barbara, CA USA'
)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



Heatmap of temperatures for Yuma, Arizona

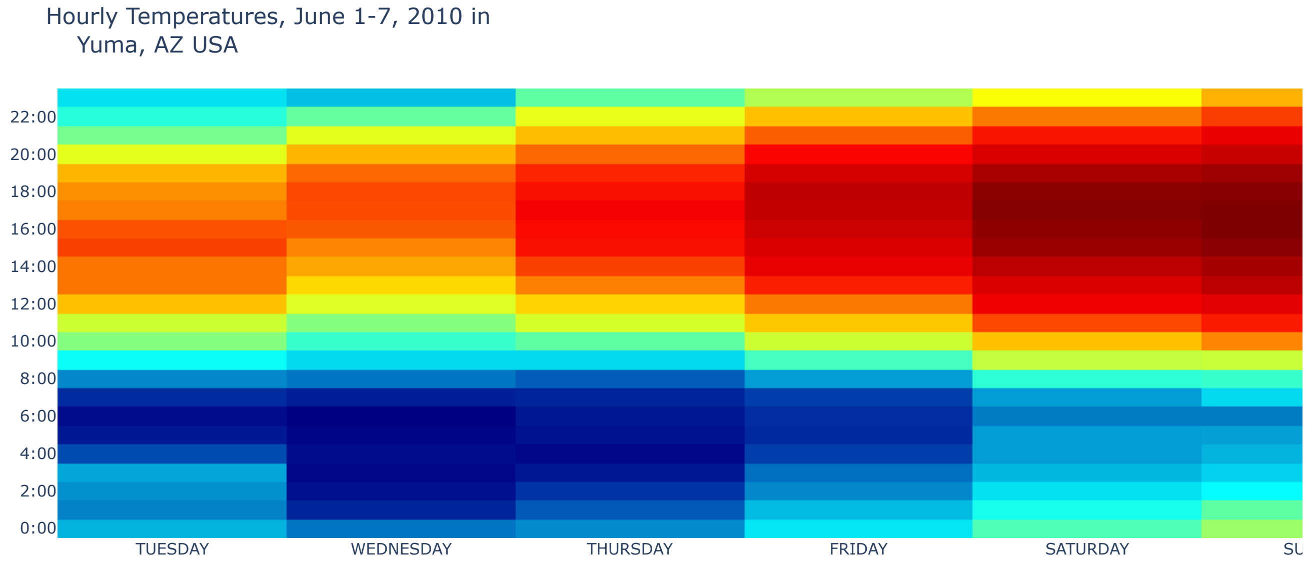
```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2010YumaAZ.csv')
df
```

	LST_DATE	DAY	LST_TIME	T_HR_AVG
0	20100601	TUESDAY	0:00	25.2
1	20100601	TUESDAY	1:00	24.1
2	20100601	TUESDAY	2:00	24.4
3	20100601	TUESDAY	3:00	24.9
4	20100601	TUESDAY	4:00	22.8
...	...	...	...	...
163	20100607	MONDAY	19:00	39.4
164	20100607	MONDAY	20:00	38.5
165	20100607	MONDAY	21:00	37.0
166	20100607	MONDAY	22:00	34.7
167	20100607	MONDAY	23:00	32.6

168 rows × 4 columns

```
data = [go.Heatmap(
    x=df['DAY'],
    y=df['LST_TIME'],
    z=df['T_HR_AVG'],
    colorscale='Jet'
)]

layout = go.Layout(
    title='Hourly Temperatures, June 1-7, 2010 in<br>\
    Yuma, AZ USA'
)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



Heatmap of temperatures for Sitka, Alaska

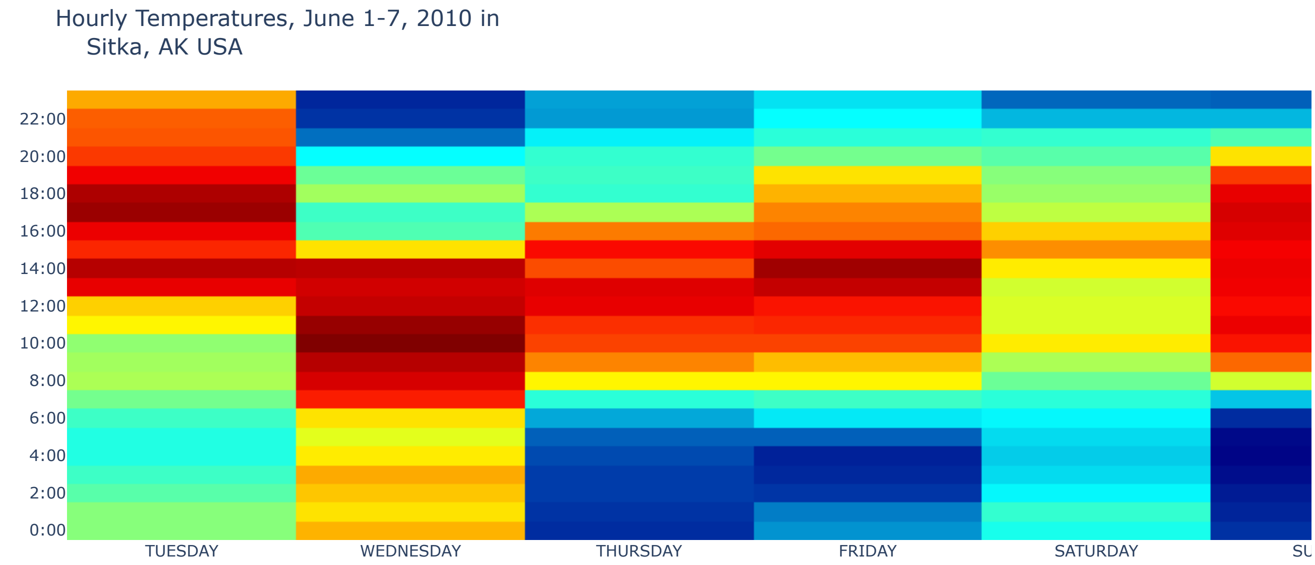
```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2010SitkaAK.csv')
df
```

	LST_DATE	DAY	LST_TIME	T_HR_AVG
0	20100601	TUESDAY	0:00	10.5
1	20100601	TUESDAY	1:00	10.5
2	20100601	TUESDAY	2:00	10.0
3	20100601	TUESDAY	3:00	9.7
4	20100601	TUESDAY	4:00	9.4
...	...	...	...	...
163	20100607	MONDAY	19:00	11.9
164	20100607	MONDAY	20:00	10.8
165	20100607	MONDAY	21:00	9.2
166	20100607	MONDAY	22:00	7.8
167	20100607	MONDAY	23:00	6.7

168 rows × 4 columns

```
data = [go.Heatmap(
    x=df['DAY'],
    y=df['LST_TIME'],
    z=df['T_HR_AVG'],
    colorscale='Jet'
)]

layout = go.Layout(
    title='Hourly Temperatures, June 1-7, 2010 in<br>\
    Sitka, AK USA'
)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



Side-by-side heatmaps for Sitka, Alaska, Santa Barbara, California and Yuma, Arizona using a shared temperature scale.

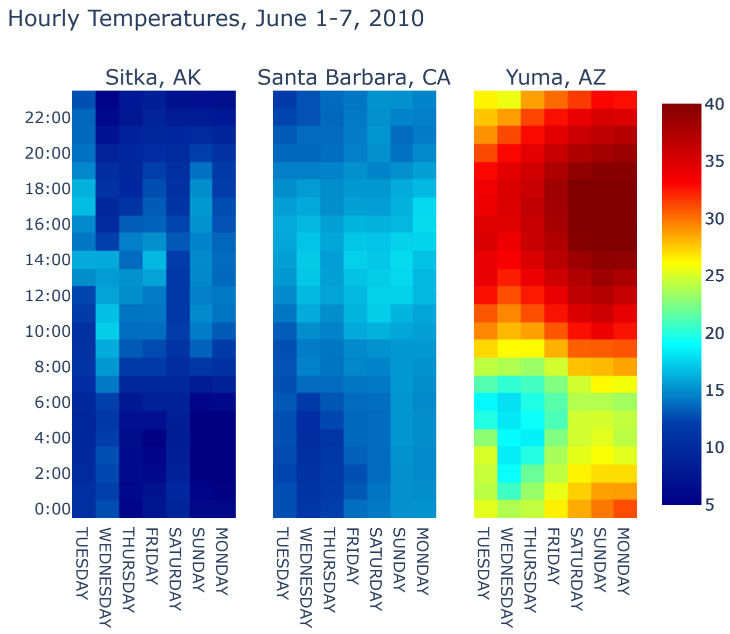
```
import plotly.subplots as sp

df1 = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2010SitkaAK.csv')
df2 = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2010SantaBarbaraCA.csv')
df3 = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/2010YumaAZ.csv')

trace1 = go.Heatmap(
    x=df1['DAY'],
    y=df1['LST_TIME'],
    z=df1['T_HR_AVG'],
    colorscale='Jet',
    zmin = 5, zmax = 40 # add max/min color values to make each plot consistent
)
trace2 = go.Heatmap(
    x=df2['DAY'],
    y=df2['LST_TIME'],
    z=df2['T_HR_AVG'],
    colorscale='Jet',
    zmin = 5, zmax = 40
)
trace3 = go.Heatmap(
    x=df3['DAY'],
    y=df3['LST_TIME'],
    z=df3['T_HR_AVG'],
    colorscale='Jet',
    zmin = 5, zmax = 40
)

fig = sp.make_subplots(rows=1, cols=3,
    subplot_titles=('Sitka, AK','Santa Barbara, CA', 'Yuma, AZ'),
    shared_yaxes = True,  # this makes the hours appear only on the left
)
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig.append_trace(trace3, 1, 3)

fig['layout'].update(      # access the layout directly!
    title='Hourly Temperatures, June 1-7, 2010'
)
fig.show()
```



Dash

Dash Overview

Dash is a productive Python framework for building web applications.

Written on top of Flask, Plotly.js, and React.js, Dash is ideal for building data visualization apps with highly custom user interfaces in pure Python. It's particularly suited for anyone who works with data in Python.

Through a couple of simple patterns, Dash abstracts away all of the technologies and protocols that are required to build an interactive web-based application. Dash is simple enough that you can bind a user interface around your Python code in an afternoon.

Dash apps are rendered in the web browser. You can deploy your apps to servers and then share them through URLs. Since Dash apps are viewed in the web browser, Dash is inherently cross-platform and mobile ready.

References

https://dash.plot.ly/

Since we using dash in jupyter notebook, we need to using jupyter_plotly_dash lib to display.

```
!pip install jupyter_plotly_dash
```

Throughout the lecture, we will assume that the following imports have taken place

```
from jupyter_plotly_dash import JupyterDash
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objs as go
from dash.dependencies import Input, Output, State
import numpy as np
```

Dash Layout

```
app = JupyterDash('SimpleExample')

app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),
    html.Div(children='Dash: A web application framework for Python.'),

    dcc.Graph(
        id='example-graph',
        figure={
            'data': [
            'data': [
                {'x': [1, 2, 3], 'y': [4, 1, 2], 'type': 'bar', 'name': 'SF'},
                {'x': [1, 2, 3], 'y': [2, 4, 5],
                    'type': 'bar', 'name': 'Montréal'},
            ],
            'layout': {
                'title': 'Dash Data Visualization'
            }
        }
    )
])
app
```

Add style to layout

```
app = JupyterDash('SimpleExample')

colors = {
    'background': '#111111',
    'text': '#7FDBFF'
}

app.layout = html.Div(children=[
    html.H1(
        children='Hello Dash',
        style={
            'textAlign': 'center',
            'color': colors['text']
        }
    ),

    html.Div(
        children='Dash: A web application framework for Python.',
        style={
            'textAlign': 'center',
            'color': colors['text']
        }
    ),

    dcc.Graph(
        id='example-graph',
        figure={
            'data': [
                {'x': [1, 2, 3], 'y': [4, 1, 2], 'type': 'bar', 'name': 'SF'},
                {'x': [1, 2, 3], 'y': [2, 4, 5], 'type': 'bar', 'name': u'Montréal'},
            ],
            'layout': {
                'plot_bgcolor': colors['background'],
                'paper_bgcolor': colors['background'],
                'font': {
                    'color': colors['text']
                },
                'title': 'Dash Data Visualization'
            }
        }
    )],
    style={'backgroundColor': colors['background']}
)
app
```

Converting Plotly To Dash

```
app = JupyterDash('SimpleExample')

random_x = np.random.randint(1, 101, 100)
random_y = np.random.randint(1, 101, 100)

app.layout = html.Div([
    dcc.Graph(
        id='scatter3',
        figure={
            'data': [
                go.Scatter(
                    x=random_x,
                    y=random_y,
                    mode='markers',
                    marker={
                        'size': 12,
                        'color': 'rgb(51,204,153)',
                        'symbol': 'pentagon',
                        'line': {'width': 2}
                    }
                )
            ],
            'layout': go.Layout(
                title='Random Data Scatterplot',
                xaxis={'title': 'Some random x-values'},
                yaxis={'title': 'Some random y-values'},
                hovermode='closest'
            )
        }
    )
])
app
```

Multiple Graph in single dashboard

```
app = JupyterDash('SimpleExample')

random_x = np.random.randint(1, 101, 100)
random_y = np.random.randint(1, 101, 100)

scatterplot1 = dcc.Graph(id='scatterplot1',
                         figure={'data': [
                             go.Scatter(
                                 x=random_x,
                                 y=random_y,
                                 mode='markers',
                                 marker={
                                     'size': 12,
                                     'color': 'rgb(51,204,153)',
                                     'symbol': 'pentagon',
                                     'line': {'width': 2}
                                 }
                             )],
                             'layout': go.Layout(title='My Scatterplot',
                                                 xaxis={'title': 'Some X title'})}
                         )

scatterplot2 = dcc.Graph(id='scatterplot2',
                         figure={'data': [
                             go.Scatter(
                                 x=random_x,
                                 y=random_y,
                                 mode='markers',
                                 marker={
                                     'size': 12,
                                     'color': 'rgb(200,204,53)',
                                     'symbol': 'pentagon',
                                     'line': {'width': 2}
                                 }
                             )],
                             'layout': go.Layout(title='Second Plot',
                                                 xaxis={'title': 'Some X title'})}
                         )

app.layout = html.Div([scatterplot1, scatterplot2])
app
```

Dash Components

Core Components

Documentation: https://dash.plot.ly/dash-core-components

```
app = JupyterDash('SimpleExample')

app.layout = html.Div([

    # DROPDOWN https://dash.plot.ly/dash-core-components/dropdown
    html.Label('Dropdown'),
    dcc.Dropdown(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': 'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value='MTL'
    ),
    html.Label('Multi-Select Dropdown'),
    dcc.Dropdown(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': u'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value=['MTL', 'SF'],
        multi=True
    ),

    # SLIDER https://dash.plot.ly/dash-core-components/slider
    html.Label('Slider'),
    html.P(
        dcc.Slider(
            min=-5,
            max=10,
            step=0.5,
            marks={i: i for i in range(-5, 11)},
            value=-3
        )),

    # RADIO ITEMS https://dash.plot.ly/dash-core-components/radioitems
    html.Label('Radio Items'),
    dcc.RadioItems(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': 'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value='MTL'
    )
], style={'width': '50%'})
app
```

HTML Components

Documentation: https://dash.plot.ly/dash-html-components

```
app = JupyterDash('SimpleExample')

app.layout = html.Div([
    'This is the outermost Div',
    html.Div(
        'This is an inner Div',
        style={'color': 'blue', 'border': '2px blue solid', 'borderRadius': 5,
               'padding': 10, 'width': 220}
    ),
    html.Div(
        'This is another inner Div',
        style={'color': 'green', 'border': '2px green solid',
               'margin': 10, 'width': 220}
    ),
],
    # this styles the outermost Div:
    style={'width': 500, 'height': 200, 'color': 'red', 'border': '2px red dotted'})
app
```

Markdown

```
app = JupyterDash('SimpleExample')

markdown_text = '''
### Dash and Markdown

Dash apps can be written in Markdown.
Dash uses the [CommonMark](http://commonmark.org/) specification of Markdown.

Check out their [60 Second Markdown Tutorial](http://commonmark.org/help/)
if this is your first introduction to Markdown!

Markdown includes syntax for things like **bold text** and *italics*,
[links](http://commonmark.org/help), inline `code` snippets, lists,
quotes, and more.
'''

app.layout = html.Div([
    dcc.Markdown(children=markdown_text)
])
```

```
app
```

Dash Callbacks

Basic

```
app = JupyterDash('SimpleExample')

app.layout = html.Div([
    dcc.Input(id='my-id', value='initial value', type='text'),
    html.Div(id='my-div')
])


@app.callback(
    Output(component_id='my-div', component_property='children'),
    [Input(component_id='my-id', component_property='value')]
)
def update_output_div(input_value):
    return 'You\'ve entered "{}"'.format(input_value)

app
```

Using gapminderDataFiveYear.csv

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/gapminderDataFiveYear.csv')
df
```

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106
...	...	...	...	...	...	...
1699	Zimbabwe	1987	9216418.0	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340.0	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948.0	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563.0	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143.0	Africa	43.487	469.709298

1704 rows × 6 columns

```
app = JupyterDash('SimpleExample')

# https://dash.plot.ly/dash-core-components/dropdown
# We need to construct a dictionary of dropdown values for the years
year_options = []
for year in df['year'].unique():
    year_options.append({'label':str(year),'value':year})

app.layout = html.Div([
    dcc.Graph(id='graph'),
    dcc.Dropdown(id='year-picker',options=year_options,value=df['year'].min())
])

@app.callback(Output('graph', 'figure'),
              [Input('year-picker', 'value')])
def update_figure(selected_year):
    filtered_df = df[df['year'] == selected_year]
    traces = []
    for continent_name in filtered_df['continent'].unique():
        df_by_continent = filtered_df[filtered_df['continent'] == continent_name]
        traces.append(go.Scatter(
            x=df_by_continent['gdpPercap'],
            y=df_by_continent['lifeExp'],
            text=df_by_continent['country'],
            mode='markers',
            opacity=0.7,
            marker={'size': 15},
            name=continent_name
        ))
    return {
        'data': traces,
        'layout': go.Layout(
            xaxis={'type': 'log', 'title': 'GDP Per Capita'},
            yaxis={'title': 'Life Expectancy'},
            hovermode='closest'
        )
    }

app
```

Multiple Inputs

Using mpg.csv

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/mpg.csv')
df
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

```
app = JupyterDash('SimpleExample')

features = df.columns

app.layout = html.Div([

    html.Div([
        dcc.Dropdown(
            id='xaxis',
            options=[{'label': i.title(), 'value': i} for i in features],
            value='displacement'
        )
    ],
        style={'width': '48%', 'display': 'inline-block'}),

    html.Div([
        dcc.Dropdown(
            id='yaxis',
            options=[{'label': i.title(), 'value': i} for i in features],
            value='acceleration'
        )
    ], style={'width': '48%', 'float': 'right', 'display': 'inline-block'}),

    dcc.Graph(id='feature-graphic')
], style={'padding': 10})


@app.callback(
    Output('feature-graphic', 'figure'),
    [Input('xaxis', 'value'),
     Input('yaxis', 'value')])
def update_graph(xaxis_name, yaxis_name):
    return {
        'data': [go.Scatter(
            x=df[xaxis_name],
            y=df[yaxis_name],
            text=df['name'],
            mode='markers',
            marker={
                'size': 15
                'size': 15,
                'opacity': 0.5,
                'line': {'width': 0.5, 'color': 'white'}
            }
        )],
        'layout': go.Layout(
            xaxis={'title': xaxis_name.title()},
            yaxis={'title': yaxis_name.title()},
            margin={'l': 40, 'b': 40, 't': 10, 'r': 0},
            hovermode='closest'
        )
    }
app
```

Multiple Outputs

Using wheels.csv

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/wheels.csv')
df
```

	wheels	color	image
0	1	red	redunicycle.jpg
1	1	blue	blueunicycle.jpg
2	1	yellow	yellowunicycle.jpg
3	2	red	redbicycle.jpg
4	2	blue	bluemotorcycle.jpg
5	2	yellow	yellowscooter.jpg
6	3	red	redtricycle.jpg
7	3	blue	bluetricycle.jpg
8	3	yellow	yellowrickshaw.jpg

```
app = JupyterDash('SimpleExample')
app.layout = html.Div([
    dcc.RadioItems(
        id='wheels',
        options=[{'label': i, 'value': i} for i in df['wheels'].unique()],
        value=1
    ),
    html.Div(id='wheels-output'),

    html.Hr(),  # add a horizontal rule
    dcc.RadioItems(
        id='colors',
        options=[{'label': i, 'value': i} for i in df['color'].unique()],
        value='blue'
    ),
    html.Div(id='colors-output')
], style={'fontFamily':'helvetica', 'fontSize':18})

@app.callback(
    Output('wheels-output', 'children'),
    [Input('wheels', 'value')])
def callback_a(wheels_value):
    return 'You\'ve selected "{}"'.format(wheels_value)

@app.callback(
    Output('colors-output', 'children'),
    [Input('colors', 'value')])
def callback_b(colors_value):
    return 'You\'ve selected "{}"'.format(colors_value)

app
```

```
import base64

app = JupyterDash('SimpleExample')

def encode_image(image_file):
    encoded = base64.b64encode(open(image_file, 'rb').read())
    return 'data:image/png;base64,{}'.format(encoded.decode())


app.layout = html.Div([
    dcc.RadioItems(
    dcc.RadioItems(
        id='wheels',
        options=[{'label': i, 'value': i} for i in df['wheels'].unique()],
        value=1
    ),
    html.Div(id='wheels-output'),

    html.Hr(),  # add a horizontal rule
    dcc.RadioItems(
        id='colors',
        options=[{'label': i, 'value': i} for i in df['color'].unique()],
        value='blue'
    ),
    html.Div(id='colors-output'),
    html.Img(id='display-image', src='children', height=300)
], style={'fontFamily': 'helvetica', 'fontSize': 18})


@app.callback(
    Output('wheels-output', 'children'),
    [Input('wheels', 'value')])
def callback_a(wheels_value):
    return 'You\'ve selected "{}"'.format(wheels_value)


@app.callback(
    Output('colors-output', 'children'),
    [Input('colors', 'value')])
def callback_b(colors_value):
    return 'You\'ve selected "{}"'.format(colors_value)


@app.callback(
    Output('display-image', 'src'),
    [Input('wheels', 'value'),
     Input('colors', 'value')])
def callback_image(wheel, color):
    path = '../assets/Images/'
    return encode_image(path+df[(df['wheels'] == wheel) &
                                (df['color'] == color)]['image'].values[0])
app
```

Controlling Callback With State

Output update immediately

```
app = JupyterDash('SimpleExample')

app.layout = html.Div([
    dcc.Input(
        id='number-in',
        value=1,
        style={'fontSize': 28}
    ),
    html.H1(id='number-out')
])


@app.callback(
    Output('number-out', 'children'),
    [Input('number-in', 'value')])
def output(number):
    return number

app
```

Output update only when hit submit button

```
app = JupyterDash('SimpleExample')

app.layout = html.Div([
    dcc.Input(
        id='number-in',
        value=1,
        style={'fontSize': 28}
    ),
    html.Button(
        id='submit-button',
        n_clicks=0,
        children='Submit',
        children='Submit',
        style={'fontSize': 28}
    ),
    html.H1(id='number-out')
])


@app.callback(
    Output('number-out', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('number-in', 'value')])
def output(n_clicks, number):
    return number

app
```

```
app = JupyterDash('SimpleExample')

app.layout = html.Div([
    dcc.Input(
        id='number-in',
        value=1,
        style={'fontSize':28}
    ),
    html.Button(
        id='submit-button',
        n_clicks=0,
        children='Submit',
        style={'fontSize':28}
    ),
    html.H1(id='number-out')
])

@app.callback(
    Output('number-out', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('number-in', 'value')])
def output(n_clicks, number):
    return '{} displayed after {} clicks'.format(number,n_clicks)

app
```

Hover Over Data

Using wheels.csv dataset

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/wheels.csv')
df
```

	wheels	color	image
0	1	red	redunicycle.jpg
1	1	blue	blueunicycle.jpg
2	1	yellow	yellowunicycle.jpg
3	2	red	redbicycle.jpg
4	2	blue	bluemotorcycle.jpg
5	2	yellow	yellowscooter.jpg
6	3	red	redtricycle.jpg
7	3	blue	bluetricycle.jpg
8	3	yellow	yellowrickshaw.jpg

```
import json

app = JupyterDash('SimpleExample')

app.layout = html.Div([
    html.Div([
        dcc.Graph(
            id='wheels-plot',
            figure={
                'data': [
                    go.Scatter(
                        x=df['color'],
                        y=df['wheels'],
                        dy=1,
                        mode='markers',
                        mode='markers',
                        marker={
                            'size': 12,
                            'color': 'rgb(51,204,153)',
                            'line': {'width': 2}
                        }
                    )
                ],
                'layout': go.Layout(
                    title='Wheels & Colors Scatterplot',
                    xaxis={'title': 'Color'},
                    yaxis={'title': '# of Wheels', 'nticks': 3},
                    hovermode='closest'
                )
            }
        )], style={'width': '30%', 'float': 'left'}),

    html.Div([
        html.Pre(id='hover-data', style={'paddingTop': 35})
    ], style={'width': '30%'})
])


@app.callback(
    Output('hover-data', 'children'),
    [Input('wheels-plot', 'hoverData')])
def callback_image(hoverData):
    return json.dumps(hoverData, indent=2)

app
```

```
app = JupyterDash('SimpleExample')

def encode_image(image_file):
    encoded = base64.b64encode(open(image_file, 'rb').read())
    return 'data:image/png;base64,{}'.format(encoded.decode())


app.layout = html.Div([
    html.Div([
        dcc.Graph(
            id='wheels-plot',
            figure={
                'data': [
                    go.Scatter(
                        x=df['color'],
                        y=df['wheels'],
                        dy=1,
                        mode='markers',
                        marker={
                            'size': 12,
                            'color': 'rgb(51,204,153)',
                            'line': {'width': 2}
                        }
                    )
                ],
                'layout': go.Layout(
                    title='Wheels & Colors Scatterplot',
                    xaxis={'title': 'Color'},
                    yaxis={'title': '# of Wheels', 'nticks': 3},
                    hovermode='closest'
                )
            }
        )], style={'width': '30%', 'float': 'left'}),

    html.Div([
        html.Img(id='hover-image', src='children', height=300)
    ], style={'paddingTop': 35})
])


@app.callback(
    Output('hover-image', 'src'),
    [Input('wheels-plot', 'hoverData')])
def callback_image(hoverData):
    wheel = hoverData['points'][0]['y']
    color = hoverData['points'][0]['x']
    path = '../assets/images/'
    return encode_image(path+df[(df['wheels'] == wheel) &
                                (df['color'] == color)]['image'].values[0])

app
```

Click Data

Using wheels.csv dataset

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/wheels.csv')
df
```

	wheels	color	image
0	1	red	redunicycle.jpg
1	1	blue	blueunicycle.jpg
2	1	yellow	yellowunicycle.jpg
3	2	red	redbicycle.jpg
4	2	blue	bluemotorcycle.jpg
5	2	yellow	yellowscooter.jpg
6	3	red	redtricycle.jpg
7	3	blue	bluetricycle.jpg
8	3	yellow	yellowrickshaw.jpg

```
app = JupyterDash('SimpleExample')

def encode_image(image_file):
    encoded = base64.b64encode(open(image_file, 'rb').read())
    return 'data:image/png;base64,{}'.format(encoded.decode())


app.layout = html.Div([
    html.Div([
        dcc.Graph(
            id='wheels-plot',
            figure={
                'data': [
                    go.Scatter(
                        x=df['color'],
                        y=df['wheels'],
                        dy=1,
                        mode='markers',
                        marker={
                            'size': 12,
                            'color': 'rgb(51,204,153)',
                            'line': {'width': 2}
                        }
                    )
                ],
                'layout': go.Layout(
                    title='Wheels & Colors Scatterplot',
                    xaxis={'title': 'Color'},
                    yaxis={'title': '# of Wheels', 'nticks': 3},
                    hovermode='closest'
                )
            }
        )], style={'width': '30%', 'float': 'left'}),

    html.Div([
        html.Img(id='click-image', src='children', height=300)
    ], style={'paddingTop': 35})
])


@app.callback(
    Output('click-image', 'src'),
    [Input('wheels-plot', 'clickData')])
def callback_image(clickData):
    wheel = clickData['points'][0]['y']
    color = clickData['points'][0]['x']
    path = '../assets/images/'
    return encode_image(path+df[(df['wheels'] == wheel) &
                                (df['color'] == color)]['image'].values[0])

app
```

Selected Data

Using wheels.csv dataset

```
df = pd.read_csv('https://raw.githubusercontent.com/paiboon15721/clusterkit-dash-training/master/dataset/wheels.csv')
df
```

	wheels	color	image
0	1	red	redunicycle.jpg
1	1	blue	blueunicycle.jpg
2	1	yellow	yellowunicycle.jpg
3	2	red	redbicycle.jpg
4	2	blue	bluemotorcycle.jpg
5	2	yellow	yellowscooter.jpg
6	3	red	redtricycle.jpg
7	3	blue	bluetricycle.jpg
8	3	yellow	yellowrickshaw.jpg

```
import json

app = JupyterDash('SimpleExample')

app.layout = html.Div([
    html.Div([
        dcc.Graph(
            id='wheels-plot',
            figure={
                'data': [
                    go.Scatter(
                        x=df['color'],
                        y=df['wheels'],
                        dy=1,
                        mode='markers',
                        marker={
                            'size': 12,
                            'color': 'rgb(51,204,153)',
                            'line': {'width': 2}
                        }
                    )
                ],
                'layout': go.Layout(
                    title='Wheels & Colors Scatterplot',
                    xaxis={'title': 'Color'},
                    yaxis={'title': '# of Wheels', 'nticks': 3},
                    hovermode='closest'
                )
            }
        )], style={'width': '50%', 'display': 'inline-block'}),

    html.Div([
        html.Pre(id='selection', style={'paddingTop': 25})
    ], style={'width': '50%', 'display': 'inline-block', 'verticalAlign': 'top'})
])


@app.callback(
    Output('selection', 'children'),
    [Input('wheels-plot', 'selectedData')])
def callback_image(selectedData):
    return json.dumps(selectedData, indent=2)

app
```

Live Updating

This page updates automatically

```
app = JupyterDash('SimpleExample')

app.layout = html.Div([
    html.H1(id='live-update-text'),
    dcc.Interval(
        id='interval-component',
        interval=2000,  # 2000 milliseconds = 2 seconds
        n_intervals=0
    )
])
```

```
@app.callback(Output('live-update-text', 'children'),
              [Input('interval-component', 'n_intervals')])
def update_layout(n):
    return 'Crash free for {} refreshes'.format(n)

app
```

Using interval component with API call

Data from: https://www.flightradar24.com/

```
import requests

app = JupyterDash('SimpleExample')

app.layout = html.Div([
    html.Div([
        html.Pre(
            id='counter_text',
            children='Active flights worldwide:'
        ),
        dcc.Graph(id='live-update-graph', style={'width': 1200}),
        dcc.Interval(
            id='interval-component',
            interval=6000,  # 6000 milliseconds = 6 seconds
            n_intervals=0
        )])
])
counter_list = []

@app.callback(Output('counter_text', 'children'),
              [Input('interval-component', 'n_intervals')])
def update_layout(n):
    url = "https://data-live.flightradar24.com/zones/fcgi/feed.js?faa=1\
           &mlat=1&flarm=1&adsb=1&gnd=1&air=1&vehicles=1&estimated=1&stats=1"
    # A fake header is necessary to access the site:
    res = requests.get(url, headers={'User-Agent': 'Mozilla/5.0'})
    data = res.json()
    counter = 0
    for element in data["stats"]["total"]:
        counter += data["stats"]["total"][element]
    counter_list.append(counter)
    return 'Active flights worldwide: {}'.format(counter)

@app.callback(Output('live-update-graph', 'figure'),
              [Input('interval-component', 'n_intervals')])
def update_graph(n):
    fig = go.Figure(
        data=[go.Scatter(
            x=list(range(len(counter_list))),
            y=counter_list,
            mode='lines+markers'
        )])
    return fig

app
```

Authentication

Authentication for dash apps is provided through a separate dash-auth package.

dash-auth provides two methods of authentication: HTTP Basic Auth and Plotly OAuth.

HTTP Basic Auth is one of the simplest forms of authentication on the web. As a Dash developer, you hardcode a set of usernames and passwords in your code and send those usernames and passwords to your viewers. There are a few limitations to HTTP Basic Auth:

Users can not log out of applications You are responsible for sending the usernames and passwords to your viewers over a secure channel Your viewers can not create their own account and cannot change their password You are responsible for safely storing the username and password pairs in your code. Plotly OAuth provides authentication through your online Plotly account or through your company's Plotly Enterprise server. As a Dash developer, this requires a paid Plotly subscription. Here's where you can subscribe to Plotly Cloud, and here's where you can contact us about Plotly Enterprise. The viewers of your app will need a Plotly account but they do not need to upgrade to a paid subscription.

Plotly OAuth allows you to share your apps with other users who have Plotly accounts. With Plotly Enterprise, this includes sharing apps through the integrated LDAP system. Apps that you have saved will appear in your list of files at https://plot.ly/organize and you can manage the permissions of the apps there. Viewers create and manage their own accounts.

Reference

https://dash.plot.ly/authentication

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

# Keep this out of source code repository - save in a file or a database
VALID_USERNAME_PASSWORD_PAIRS = {
    'hello': 'world'
}

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
auth = dash_auth.BasicAuth(
    app,
    VALID_USERNAME_PASSWORD_PAIRS
)

app.layout = html.Div([
    html.H1('Welcome to the app'),
    html.H3('You are successfully authorized'),
    dcc.Dropdown(
        id='dropdown',
        options=[{'label': i, 'value': i} for i in ['A', 'B']],
        value='A'
    ),
    dcc.Graph(id='graph')
], className='container')

@app.callback(
    dash.dependencies.Output('graph', 'figure'),
    [dash.dependencies.Input('dropdown', 'value')])
def update_graph(dropdown_value):
    return {
        'layout': {
            'title': 'Graph of {}'.format(dropdown_value),
            'margin': {
                'l': 20,
                'b': 20,
                'r': 10,
                't': 60
            }
        },
        'data': [{'x': [1, 2, 3], 'y': [4, 1, 2]}]
    }

if __name__ == '__main__':
    app.run_server(debug=True)
```

Deployment

By default, Dash apps run on localhost - you can only access them on your own machine. To share a Dash app, you need to "deploy" your Dash app to a server and open up the server's firewall to the public or to a restricted set of IP addresses.

Dash uses Flask under the hood. This makes deployment easy: you can deploy a Dash app just like you would deploy a Flask app. Almost every cloud server provider has a guide for deploying Flask apps. For more, see the official Flask Guide to Deployment

Reference

https://dash.plot.ly/deployment

https://flask.palletsprojects.com/en/1.1.x/deploying