# pydata_viz

May 20, 2020

[1]:
```
%load_ext autoreload
%autoreload 2
```

## 1 Doing Data Viz in Python

**Ned Letcher**

- nedned.net
- github.com/ned2

Notebook Location: git.io/JfzLL

#### 1.0.1 Goal of data analysis

*Derive information and insights in order to improve our understanding and inform decision making*

#### 1.0.2 Relevance of data visualisation

1. Analysise and explore
2. Communicate findings

This talk is about showing you Python tools for how to achieve these goals, and some tips for how to use them.

### 1.1 Data Viz for exploratory data analysis

**Dataset**: Melbourne City Council Pedestrian Counting System dataset

Contains hourly counts of footfalls across sensors located around Melbourne CDB from 2009 to now.

#### 1.1.1 Data prep

[4]:
```
from pathlib import Path

DATA_PATH = Path("data/
↪Pedestrian_Counting_System___2009_to_Present__counts_per_hour_.csv")
```

[5]:
```
import pandas as pd
```

```
data_df = pd.read_csv(DATA_PATH)
data_df.head()
```

```
[5]:        ID               Date_Time  Year     Month  Mdate     Day  Time  \
     0  2887628  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     1  2887629  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     2  2887630  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     3  2887631  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     4  2887632  11/01/2019 05:00:00 PM  2019  November      1  Friday    17

        Sensor_ID                    Sensor_Name  Hourly_Counts
     0         34           Flinders St-Spark La            300
     1         39                   Alfred Place            604
     2         37                 Lygon St (East)           216
     3         40  Lonsdale St-Spring St (West)            627
     4         36                 Queen St (West)           774
```

```
[6]: def load_and_clean_pedestrian_data(path):
         df = pd.read_csv(path)
         df["datetime"] = pd.to_datetime(
             {
                 "day": df["Mdate"],
                 "year": df["Year"],
                 "hour": df["Time"],
                 "month": pd.to_datetime(df["Month"], format='%B').dt.month
             }
         )
         return df

     data_df = load_and_clean_pedestrian_data(DATA_PATH)
     data_df.head()
```

```
[6]:        ID               Date_Time  Year     Month  Mdate     Day  Time  \
     0  2887628  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     1  2887629  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     2  2887630  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     3  2887631  11/01/2019 05:00:00 PM  2019  November      1  Friday    17
     4  2887632  11/01/2019 05:00:00 PM  2019  November      1  Friday    17

        Sensor_ID                    Sensor_Name  Hourly_Counts            datetime
     0         34           Flinders St-Spark La            300 2019-11-01 17:00:00
     1         39                   Alfred Place            604 2019-11-01 17:00:00
     2         37                 Lygon St (East)           216 2019-11-01 17:00:00
     3         40  Lonsdale St-Spring St (West)            627 2019-11-01 17:00:00
     4         36                 Queen St (West)           774 2019-11-01 17:00:00
```

### 1.1.2 Let's have a look at what's in the data

*Q: How many years does the dataset cover?*

```
[7]: YEARS = sorted(data_df["Year"].unique())
     YEARS
```

```
[7]: [2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
```

*Q: How many sensors are in the datset?*

```
[54]: SENSORS = data_df["Sensor_Name"].unique()
      SENSORS.shape
```

```
[54]: (69,)
```

*Q: How many people are recorded each year?*

```
[55]: year_counts = data_df.groupby("Year")["Hourly_Counts"].sum()
      year_counts
```

```
[55]: Year
      2009     62650110
      2010     93459437
      2011     90571965
      2012    102215521
      2013    117389820
      2014    169500386
      2015    209099687
      2016    228757880
      2017    216586414
      2018    261909318
      2019    266411675
      2020     65584490
      Name: Hourly_Counts, dtype: int64
```

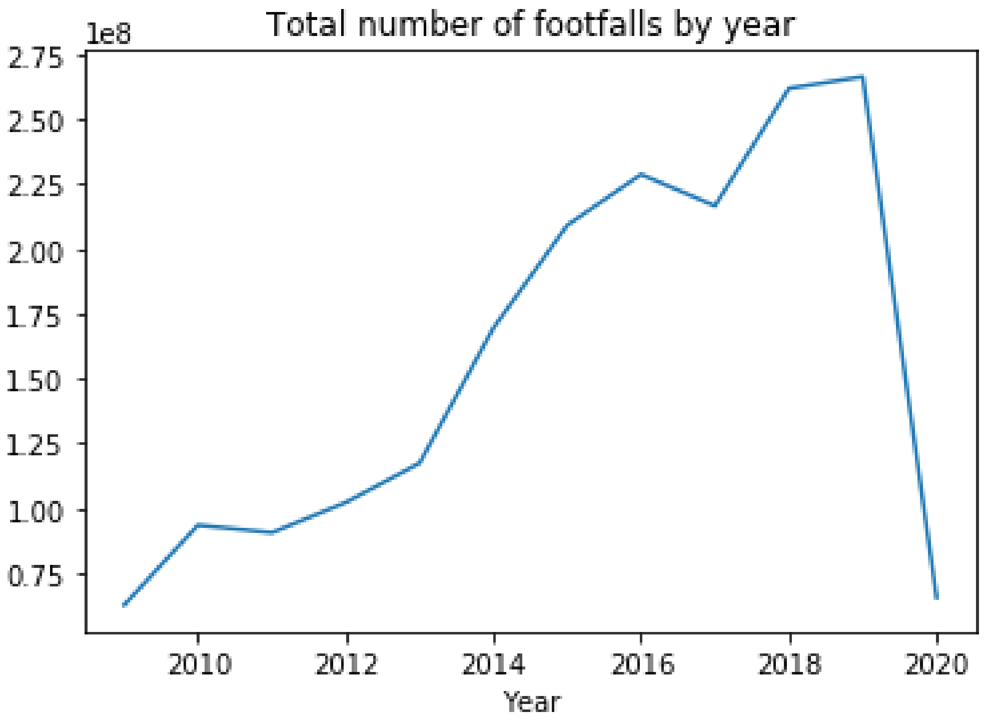
Just eyeballing the data isn't going to cut it. Let's visualise with Pandas' plotting API.

This uses the Matplotlib library, so first set it up and configure. The first line below tells Jupyter to automatically render Matplotlib plots in the cell's output.

```
[8]: %matplotlib inline
     import matplotlib.pyplot as plt
```

```
[9]: year_counts.plot(title="Total number of footfalls by year")
```

```
[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6c52b64640>
```

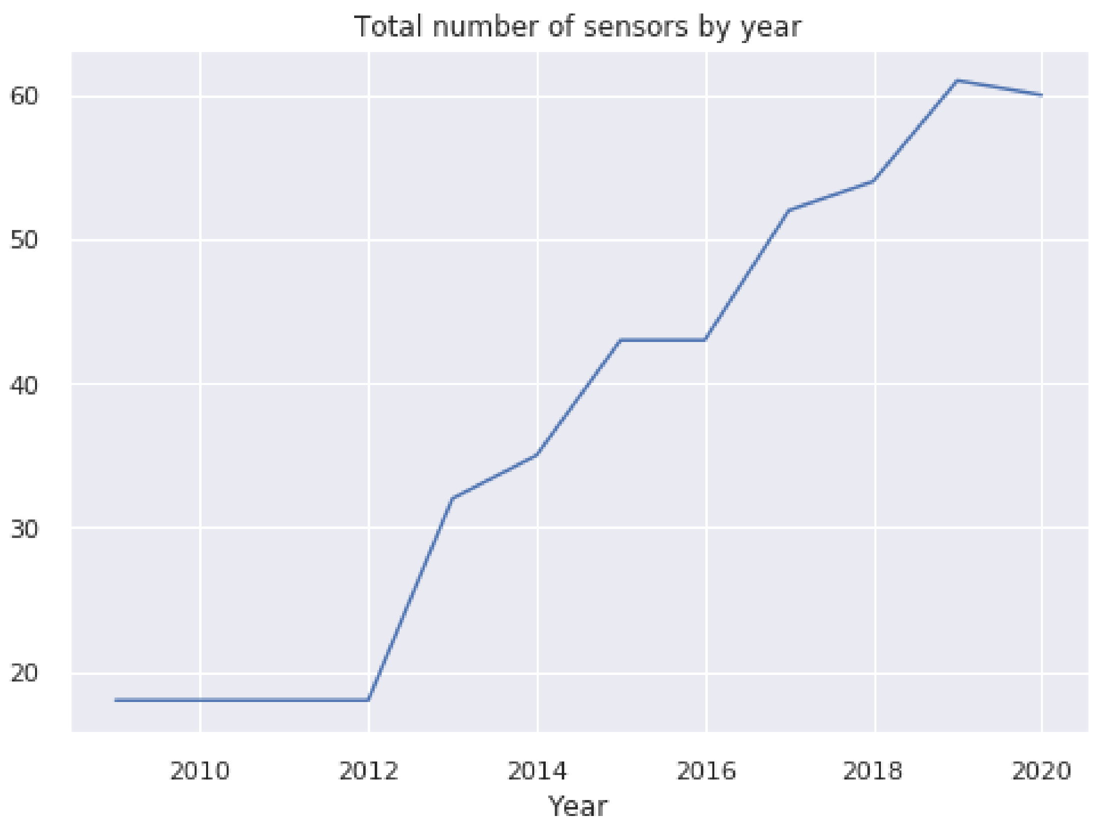


*Tip: always title your plots.*

Problem is that the number of sensors increases.

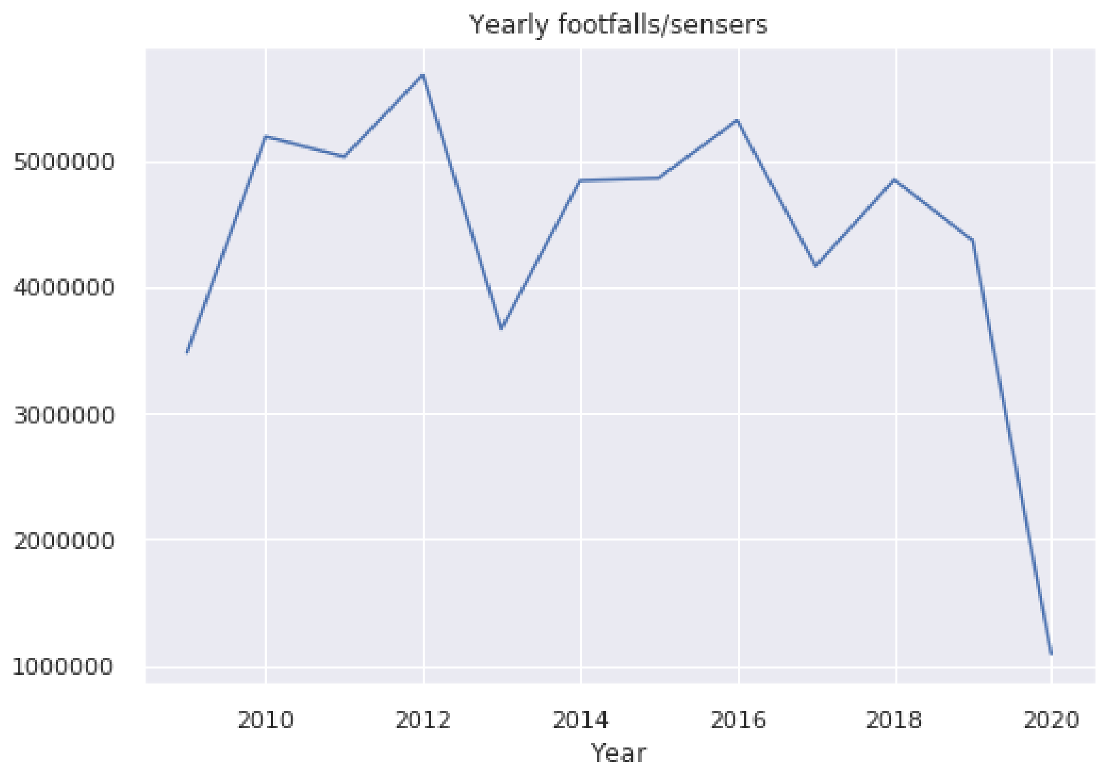
[56]:
```
num_sensors = data_df.groupby("Year")["Sensor_Name"].nunique()
num_sensors.plot(title="Total number of sensors by year");
```

[56]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c1a60da30>
```



Let's normalise by number of sensors.

[57]:
```
year_counts_df = data_df.groupby("Year").agg({"Hourly_Counts":sum,␣
↪"Sensor_Name": "nunique"})
year_counts_df["count_per_sensor"] = year_counts_df["Hourly_Counts"] /␣
↪year_counts_df["Sensor_Name"]
year_counts_df["count_per_sensor"].plot(title="Yearly footfalls/sensers");
```

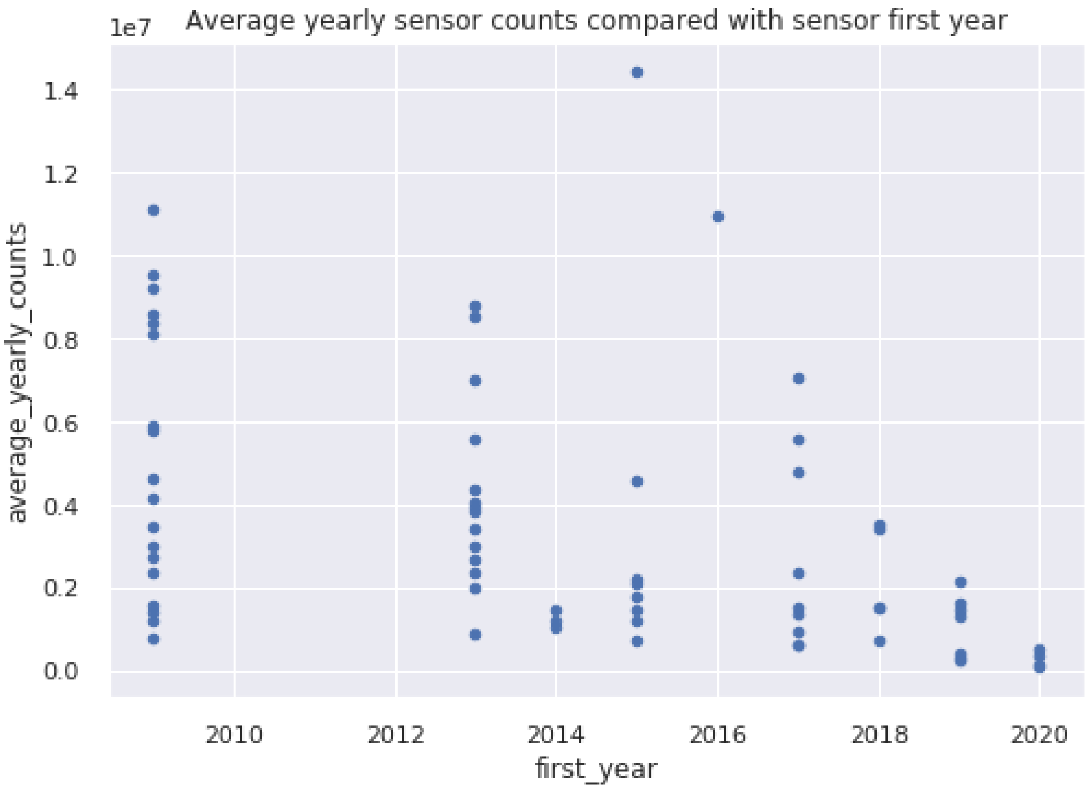


Not clear how helpful this is… maybe newer sensors are more likely to be in less travelled areas and our footfalls shouldn't be spread across them to the same weight as more trafficked sensors.

[58]:
```
# Build up a DataFrame of sensor names with their average yearly counts and␣
 ↪first year of existence:

# get average yearly counts
res_df = pd.DataFrame(
    data_df.groupby("Sensor_Name").apply(
        lambda df:df.groupby("Year")["Hourly_Counts"].sum().mean()
    ),
    columns=["average_yearly_counts"]
)

# get first year of existence for each sensor
sorted_df = data_df.sort_values(by="Year")
res_df["first_year"] = [
    sorted_df[sorted_df["Sensor_Name"] == sensor].iloc[0]["Year"]
    for sensor in res_df.index
]
```

```
[59]: # can also use df.plot.scatter()
res_df.plot(kind="scatter", x="first_year", y="average_yearly_counts",␣
 ↪title="Average yearly sensor counts compared with sensor first year");
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



### 1.1.3 Pandas' plotting API

- Built on Matplotlib (has its own API, which Pandas hides)
- Really useful for quick exploratory visualisation for `DataFrame` and `Series`

**Limitations**

1. Doesn't look very pretty out of the box
2. Pandas plotting API is limited
3. Static image: can't zoom or toggle visibility of data

### 1.1.4 What are your options?

**1. Improve Aesthetics using Matplotlib Themes**

[14]:
```python
print(plt.style.available)
```

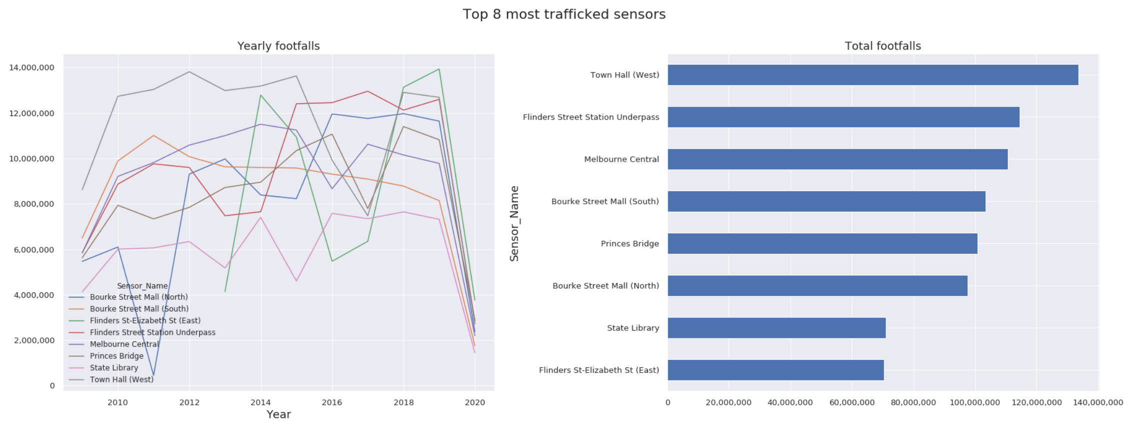
```
['seaborn-notebook', 'seaborn-ticks', 'seaborn', '_classic_test', 'grayscale',
'seaborn-paper', 'ggplot', 'bmh', 'seaborn-deep', 'seaborn-white', 'fast',
'seaborn-muted', 'seaborn-whitegrid', 'seaborn-colorblind', 'seaborn-poster',
'seaborn-dark', 'fivethirtyeight', 'seaborn-bright', 'seaborn-dark-palette',
'tableau-colorblind10', 'seaborn-darkgrid', 'classic', 'seaborn-pastel',
'seaborn-talk', 'dark_background', 'Solarize_Light2']
```

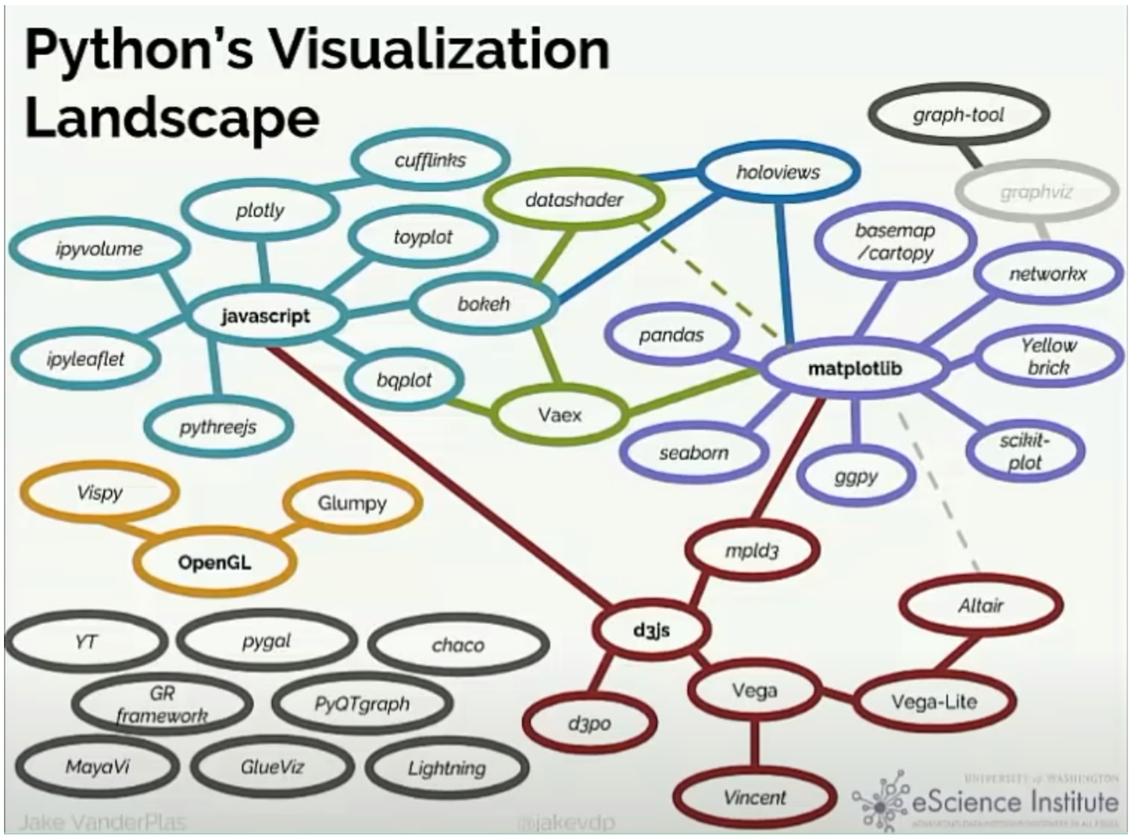
[15]:
```python
plt.style.use('seaborn')
```

### 2. Work with Matplotlib's API

- Pandas's `plot()` produces Matplotlib objects Can break out into full Matplotlib API
- Matplotlib API is very powerful, but API can be hard to learn

[61]:
```python
from matplotlib import ticker

# Tip: wrap up code to make a plot into a function. Useful for:
#      - parameterising you plot
#      - not polluting the global namespace/clobbering other identifiers

def make_top_sensor_plot(df, num_sensors=8):
    # make and configure our split figure
    plt.rcParams.update(
        {"figure.titlesize": 22, "axes.titlesize": 18, "axes.labelsize": 18,
         "legend.fontsize": 12, "xtick.labelsize":13, "ytick.labelsize":13}
    )
    fig, (ax1, ax2) = plt.subplots(1, 2)
    plt.subplots_adjust(wspace=.4)

    fig.suptitle(f"Top {num_sensors} most trafficked sensors")
    ax1.set_title("Yearly footfalls")
    ax2.set_title("Total footfalls")

    # make numeric axes comma separated integers
    ax1.yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))
    ax2.xaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

    # filter the data to the top 8 busiest sensors
    sensor_counts = df.groupby("Sensor_Name")["Hourly_Counts"].sum().nlargest(8)
    top_df = df[df["Sensor_Name"].isin(set(sensor_counts.index))]

    # plot their total counts by year in axis 1
    top_df.groupby("Sensor_Name").apply(
        lambda df:df.groupby("Year")["Hourly_Counts"].sum()
    ).unstack().transpose().plot(ax=ax1, figsize=(30, 10))
    # plot their aggregate counts in axis 2
    sensor_counts.sort_values(ascending=True).plot.barh(ax=ax2);


make_top_sensor_plot(data_df)
```





**3. Use Another Python Data** Jake VanderPlas The Python Visualization Landscape PyCon

2017

### 1.1.5 How to choose?

- What is the intended use case?
- data exploration
- data communication
- decision support
- Does your data have domain-specific needs?
- spatial data
- network data
- big data
- Does it need to be interactive?
- Does it need to be made available to end consumers?

## 1.2 Some Good General-purpose Libraries for Data Analysis

## 1.3 Seaborn

Seaborn is a library built on top of Matplotlib.

Provides: * API designed for statistical visualisation * Recipes for producing specialised plots * Improved visual appearance

Like Matplotlib, only produces static images

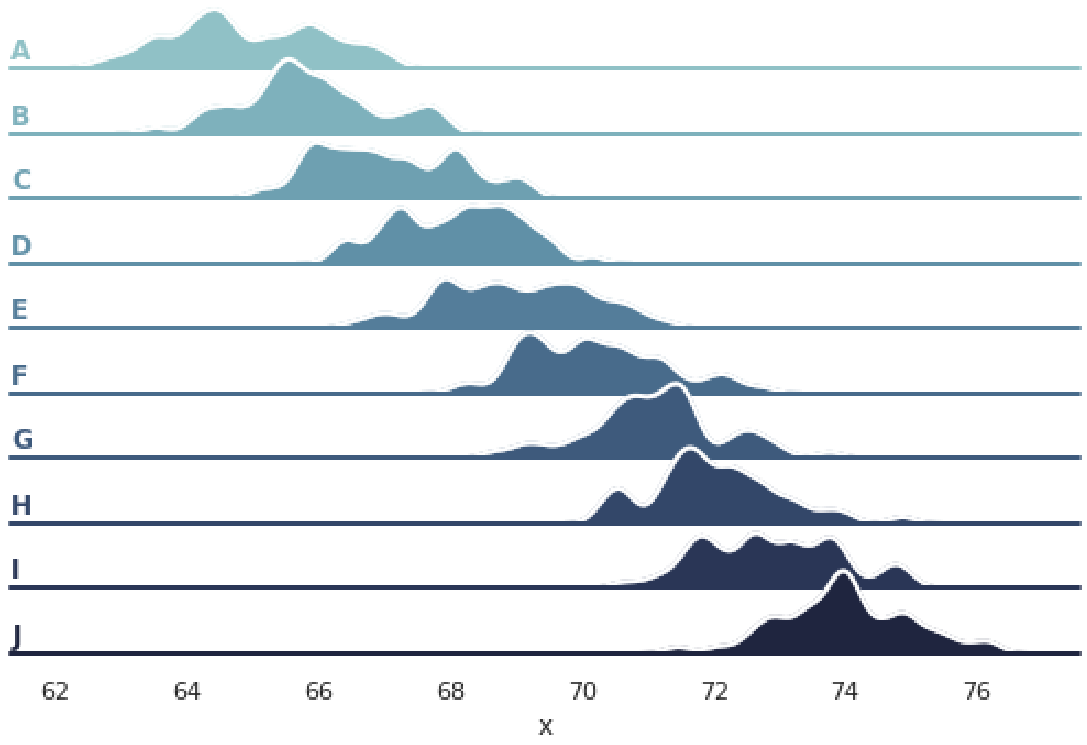
**Links** * Seaborn Overview * Gallery

### 1.3.1 Seaborn Demos

[62]:
```
# Ridgeplot
# https://seaborn.pydata.org/examples/kde_ridgeplot.html

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", rc={"axes.facecolor": (0, 0, 0, 0)})

# Create the data
rs = np.random.RandomState(1979)
x = rs.randn(500)
g = np.tile(list("ABCDEFGHIJ"), 50)
rand_df = pd.DataFrame(dict(x=x, g=g))
m = rand_df.g.map(ord)
rand_df["x"] += m

# Initialize the FacetGrid object
pal = sns.cubehelix_palette(10, rot=-.25, light=.7)
g = sns.FacetGrid(rand_df, row="g", hue="g", aspect=15, height=.5, palette=pal)
```

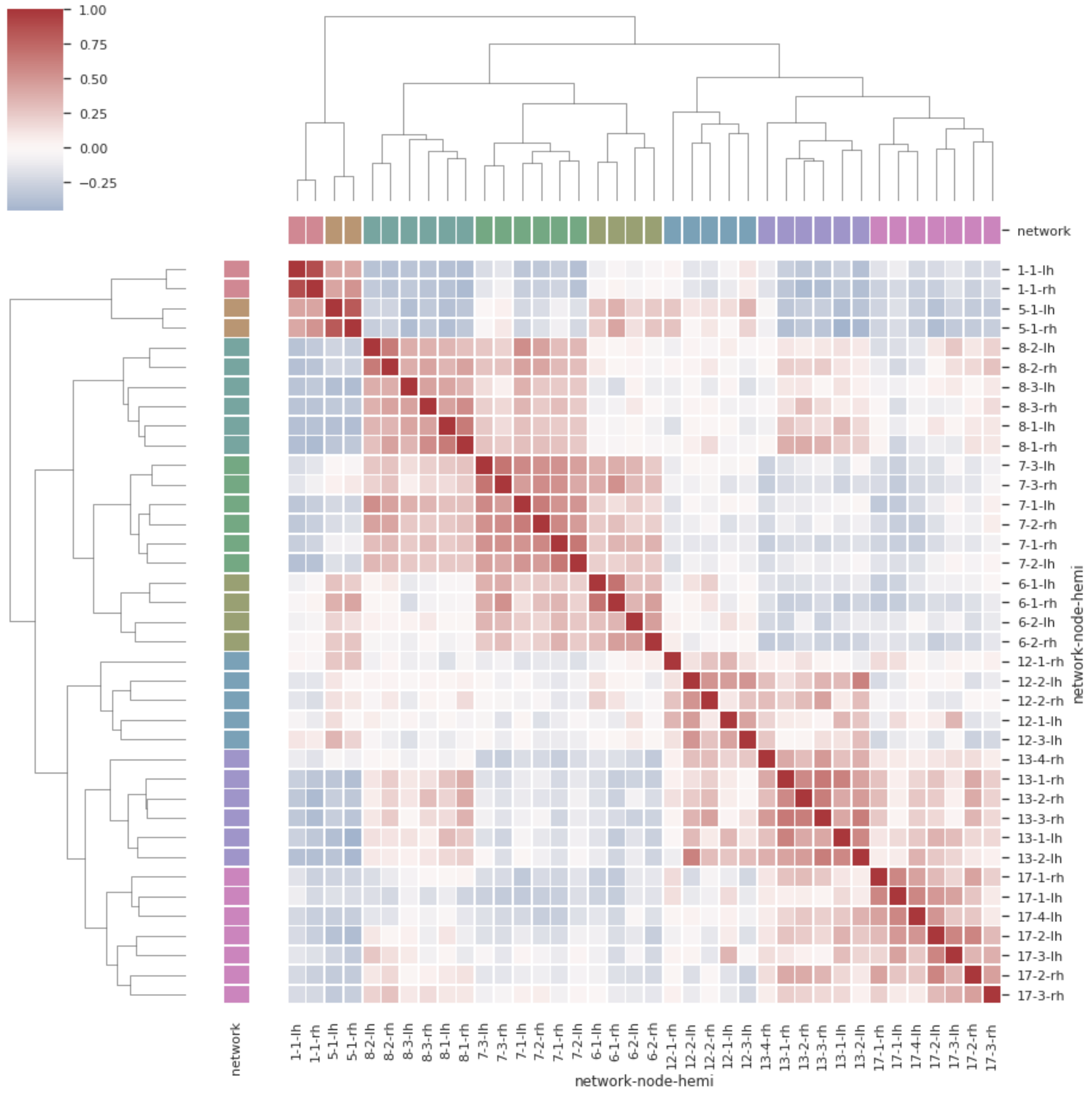
```
# Draw the densities in a few steps
g.map(sns.kdeplot, "x", clip_on=False, shade=True, alpha=1, lw=1.5, bw=.2)
g.map(sns.kdeplot, "x", clip_on=False, color="w", lw=2, bw=.2)
g.map(plt.axhline, y=0, lw=2, clip_on=False)


# Define and use a simple function to label the plot in axes coordinates
def label(x, color, label):
    ax = plt.gca()
    ax.text(0, .2, label, fontweight="bold", color=color,
            ha="left", va="center", transform=ax.transAxes)


g.map(label, "x")

# Set the subplots to overlap
g.fig.subplots_adjust(hspace=-.25)

# Remove axes details that don't play well with overlap
g.set_titles("")
g.set(yticks=[])
g.despine(bottom=True, left=True);
```

```
/home/ned/.pyenv/versions/3.8.3/envs/pydata-viz/lib/python3.8/site-
packages/matplotlib/tight_layout.py:211: UserWarning:

Tight layout not applied. tight_layout cannot make axes height small enough to
accommodate all axes decorations
```



[22]:
```
# clustermap
# https://seaborn.pydata.org/examples/structured_heatmap.html

import pandas as pd
import seaborn as sns
sns.set()

# Load the brain networks example dataset
brain_df = sns.load_dataset("brain_networks", header=[0, 1, 2], index_col=0)

# Select a subset of the networks
used_networks = [1, 5, 6, 7, 8, 12, 13, 17]
used_columns = (brain_df.columns.get_level_values("network")
                          .astype(int)
                          .isin(used_networks))
brain_df = brain_df.loc[:, used_columns]

# Create a categorical palette to identify the networks
network_pal = sns.husl_palette(8, s=.45)
network_lut = dict(zip(map(str, used_networks), network_pal))

# Convert the palette to vectors that will be drawn on the side of the matrix
```

```python
networks = brain_df.columns.get_level_values("network")
network_colors = pd.Series(networks, index=brain_df.columns).map(network_lut)

# Draw the full plot
sns.clustermap(brain_df.corr(), center=0, cmap="vlag",
               row_colors=network_colors, col_colors=network_colors,
               linewidths=.75, figsize=(13, 13));
```



## 1.4 Holoviews and Bokeh

**Bokeh** * A general-purpose interactive visualization library for modern web browsers * Can be embedded in jupyer Notebooks * Can be deployed as dashboards or interacrtive webapps * A number of other libraries are built on top of Bokeh

**Holoviews** * High-level declarative library for producing visualisations with a consise amount of code * Easier to use for data analysis than Bokeh * Can fall back to Bokeh API if needed * Can also target Plotly and Matplotlib backends * Good interactive support

*Recomendation: Use Holoviews for analysis, falling back to Bokeh if needed*

**Links**

- Holoviews Gallery
- Bokeh Gallery
- Jupyer Notebook/Lab setup

### 1.4.1 Holoviews Demos

[23]:
```
# Density Grid
# https://holoviews.org/gallery/demos/bokeh/iris_density_grid.
↪html#demos-bokeh-gallery-iris-density-grid

import holoviews as hv
from holoviews import opts
hv.extension('bokeh')

from bokeh.sampledata.iris import flowers
from holoviews.operation import gridmatrix

iris_ds = hv.Dataset(flowers)
density_grid = gridmatrix(iris_ds, diagonal_type=hv.Distribution, chart_type=hv.
↪Bivariate)
point_grid = gridmatrix(iris_ds, chart_type=hv.Points)

(density_grid * point_grid).opts(
    opts.Bivariate(bandwidth=0.5, cmap='Blues'),
    opts.Points(size=2, tools=['box_select']))
```

[23]:
```
:GridMatrix   [X,Y]
   :Overlay
      .Distribution.I :Distribution   [sepal_width]   (Density)
      .Histogram.I    :Histogram   [sepal_width]   (sepal_width_frequency)
```

[24]:
```
# Chloropleth
# https://holoviews.org/gallery/demos/bokeh/texas_choropleth_example.
↪html#demos-bokeh-gallery-texas-choropleth-example

# uncomment on first run
#import bokeh
#bokeh.sampledata.download()

import holoviews as hv
from holoviews import opts
```

```
hv.extension('bokeh')

from bokeh.sampledata.us_counties import data as counties
from bokeh.sampledata.unemployment import data as unemployment

counties = [dict(county, Unemployment=unemployment[cid])
            for cid, county in counties.items()
            if county["state"] == "tx"]
choropleth = hv.Polygons(counties, ['lons', 'lats'], [('detailed name',␣
 ↪'County'), 'Unemployment'])

choropleth.opts(
    opts.Polygons(logz=True, tools=['hover'], xaxis=None, yaxis=None,
                   show_grid=False, show_frame=False, width=500, height=500,
                   color_index='Unemployment', colorbar=True, toolbar='above',␣
 ↪line_color='white'))
```

[24]:
```
:Polygons   [lons,lats]   (detailed name,Unemployment)
```

[25]:
```
# Topographic hillshading
# https://holoviews.org/gallery/demos/bokeh/topographic_hillshading.
 ↪html#demos-bokeh-gallery-topographic-hillshading

import holoviews as hv
from holoviews import opts
hv.extension('bokeh')

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cbook import get_sample_data
from matplotlib.colors import LightSource

dem = np.load(get_sample_data('jacksboro_fault_dem.npz'))
z = dem['elevation']

dx, dy = dem['dx'], dem['dy']
dy = 111200 * dy
dx = 111200 * dx * np.cos(np.radians(dem['ymin']))

# Shade from the northwest, with the sun 45 degrees from horizontal
ls = LightSource(azdeg=315, altdeg=45)
cmap = plt.cm.gist_earth

# Vary vertical exaggeration and blend mode and plot all combinations
grid = hv.GridMatrix(kdims=['Vertical exaggeration', 'Blend mode', ])
for ve in [0.1, 1, 10]:
    # Show the hillshade intensity image in the first row
    grid['None', ve] = hv.Image(ls.hillshade(z, vert_exag=ve, dx=dx, dy=dy))
    # Place hillshaded plots with different blend modes in the rest of the rows
    for mode in ['hsv', 'overlay', 'soft']:
        rgb = ls.shade(z, cmap=cmap, blend_mode=mode,
                       vert_exag=ve, dx=dx, dy=dy)
        grid[mode, ve] = hv.RGB(rgb)

grid.opts(
    opts.GridMatrix(xaxis='bottom', yaxis='left', shared_xaxis=False,␣
 ↪shared_yaxis=False),
    opts.Image(cmap='gray'))
```

[25]:
```
:GridMatrix   [Vertical exaggeration,Blend mode]
   :RGB   [x,y]   (R,G,B,A)
```

## 1.5 Altair

- declarative statistical visualization library
- Aims to enable you to produce elegant visualisations with minimal amount of code
- Creates JSON specifications of Vega-Lite visualization grammar

**Vega-Lite** * a high-level grammar that enables rapid specification of interactive data visualizations * based on the Grammar of Graphics with extensions for interactivity * provides visual encoding rules and composition algebra for visual components * eg: `plot = data(x,y) + point(opacity=0.5) + line_fitted() + scale_y_log()`

**Links** * Gallery * Vega-Lite * Vega-Lite: A Grammar of Interactive Graphics * Jupyter Notebook/Lab setup * Talk: How to Think about Data Visualization - PyCon 2019

### 1.5.1 Altair Demos

[26]:
```
# Layered Histogram
# https://altair-viz.github.io/gallery/layered_histogram.html

import pandas as pd
import altair as alt
import numpy as np
np.random.seed(42)

# Generating Data
source = pd.DataFrame({
    'Trial A': np.random.normal(0, 0.8, 1000),
    'Trial B': np.random.normal(-2, 1, 1000),
    'Trial C': np.random.normal(3, 2, 1000)
})

alt.Chart(source).transform_fold(
    ['Trial A', 'Trial B', 'Trial C'],
    as_=['Experiment', 'Measurement']
).mark_area(
    opacity=0.3,
    interpolate='step'
).encode(
    alt.X('Measurement:Q', bin=alt.Bin(maxbins=100)),
    alt.Y('count()', stack=None),
    alt.Color('Experiment:N')
)
```

[26]: alt.Chart(…)

[27]:
```
# Chloropleth
# https://altair-viz.github.io/gallery/choropleth.html

import altair as alt
from vega_datasets import data

counties = alt.topo_feature(data.us_10m.url, 'counties')
source = data.unemployment.url

alt.Chart(counties).mark_geoshape().encode(
    color='rate:Q'
).transform_lookup(
    lookup='id',
    from_=alt.LookupData(source, 'id', ['rate'])
).project(
    type='albersUsa'
).properties(
    width=500,
    height=300
)
```

[27]: alt.Chart(…)

[28]:
```
# Selection Histogram
# https://altair-viz.github.io/gallery/selection_histogram.html

import altair as alt
from vega_datasets import data

source = data.cars()

brush = alt.selection(type='interval')

points = alt.Chart(source).mark_point().encode(
    x='Horsepower:Q',
    y='Miles_per_Gallon:Q',
    color=alt.condition(brush, 'Origin:N', alt.value('lightgray'))
).add_selection(
    brush
)

bars = alt.Chart(source).mark_bar().encode(
    y='Origin:N',
    color='Origin:N',
    x='count(Origin):Q'
).transform_filter(
    brush
)

points & bars
```

[28]:
```
alt.VConcatChart(...)
```

## 1.6 Plotly

- Library for making wide range of interactive, publication-quality graphs
- Good Jupyer notebook Support
- Python library is based on plotly.js JavaScript Library
- Declarative figure specification allows supporting multiple clients
- Python
- R
- JavaScript
- Good support for interactive visualisations
- `FigureWidget` and `ipywidgets`
- Dash

*Recommendation: use Plotly Express for exploratory data analysis, falling back to full Plotly API when needed*

Figurewidget

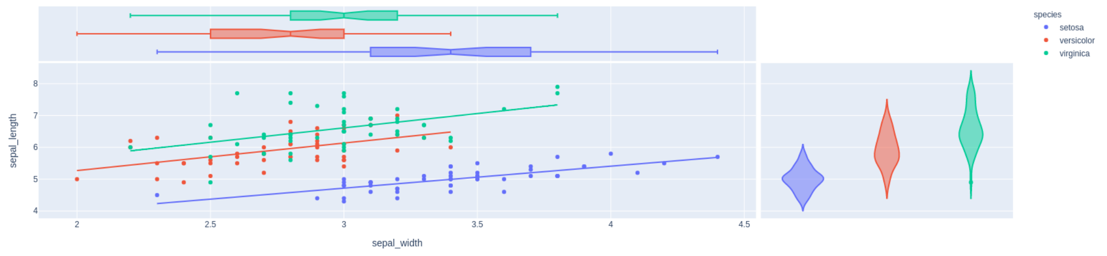
**Links** * Plotly Python Client Reference * Plotly Figure Specification Reference * Plotly Express Reference * Interactive Data Analysis with FigureWidget ipywidgets in Python * Jupyer Notebook/Lab setup * plotly.js

### 1.6.1 Plotly Express Demos

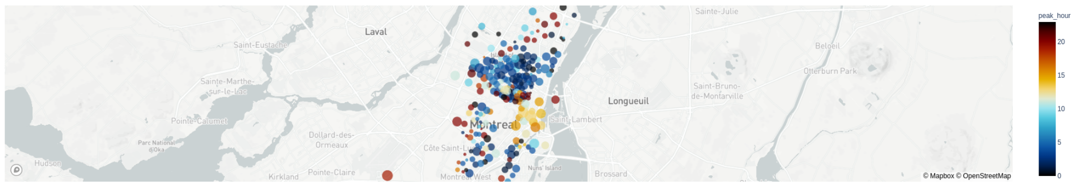
https://plotly.com/python/plotly-express/

[38]:
```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species",␣
 ↪marginal_y="violin",
           marginal_x="box", trendline="ols")
```

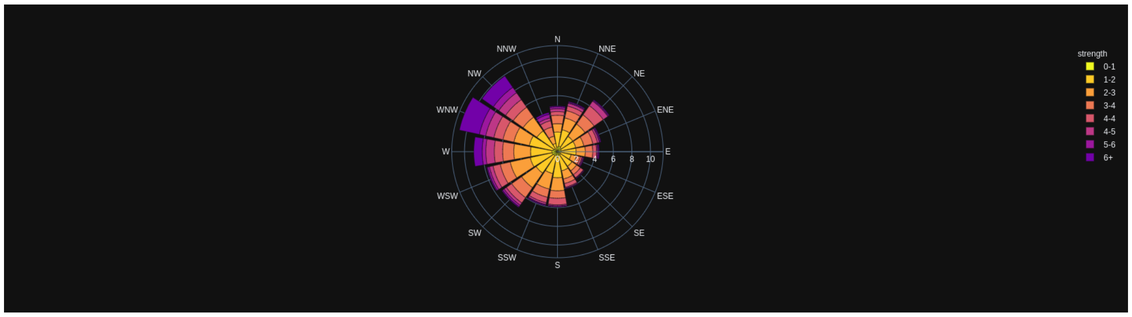
```
fig.show()
```



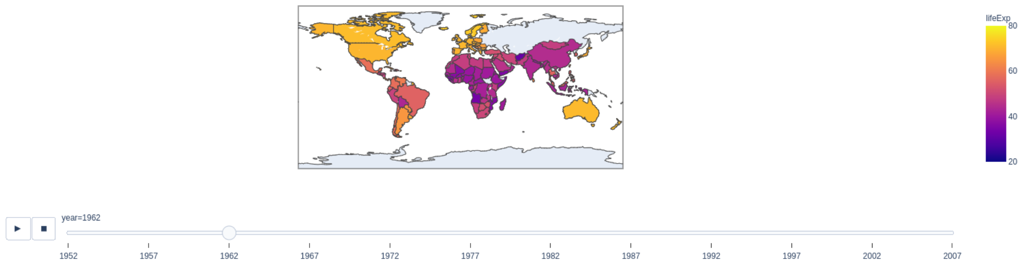
[64]:
```
# NOTE: You need to sign up for an account at https://mapbox.com and put your␣
↪token
# in the file .mapbox_token for this to work

import plotly.express as px
px.set_mapbox_access_token(open(".mapbox_token").read())
df = px.data.carshare()
fig = px.scatter_mapbox(df, lat="centroid_lat", lon="centroid_lon",     ␣
↪color="peak_hour", size="car_hours",
                  color_continuous_scale=px.colors.cyclical.IceFire,␣
↪size_max=15, zoom=10)
fig.show()
```



[39]:
```
import plotly.express as px
df = px.data.wind()
fig = px.bar_polar(df, r="frequency", theta="direction", color="strength",␣
↪template="plotly_dark",
            color_discrete_sequence= px.colors.sequential.Plasma_r)
fig.show()
```



```
[40]: import plotly.express as px
      df = px.data.gapminder()
      fig = px.choropleth(df, locations="iso_alpha", color="lifeExp",␣
       ↪hover_name="country", animation_frame="year", range_color=[20,80])
      fig.show()
```



### 1.6.2 Using Plotly's Figure API

All Plotly plots are described by the Figure API * Python dictionaries/JSON objects * Plotly Express functions all return figures * Can construct figures directly, or modify the figures returned

*Recomendation: Use the `plotly.graph_objects` module to construct figures, for property validation and autocompletion*

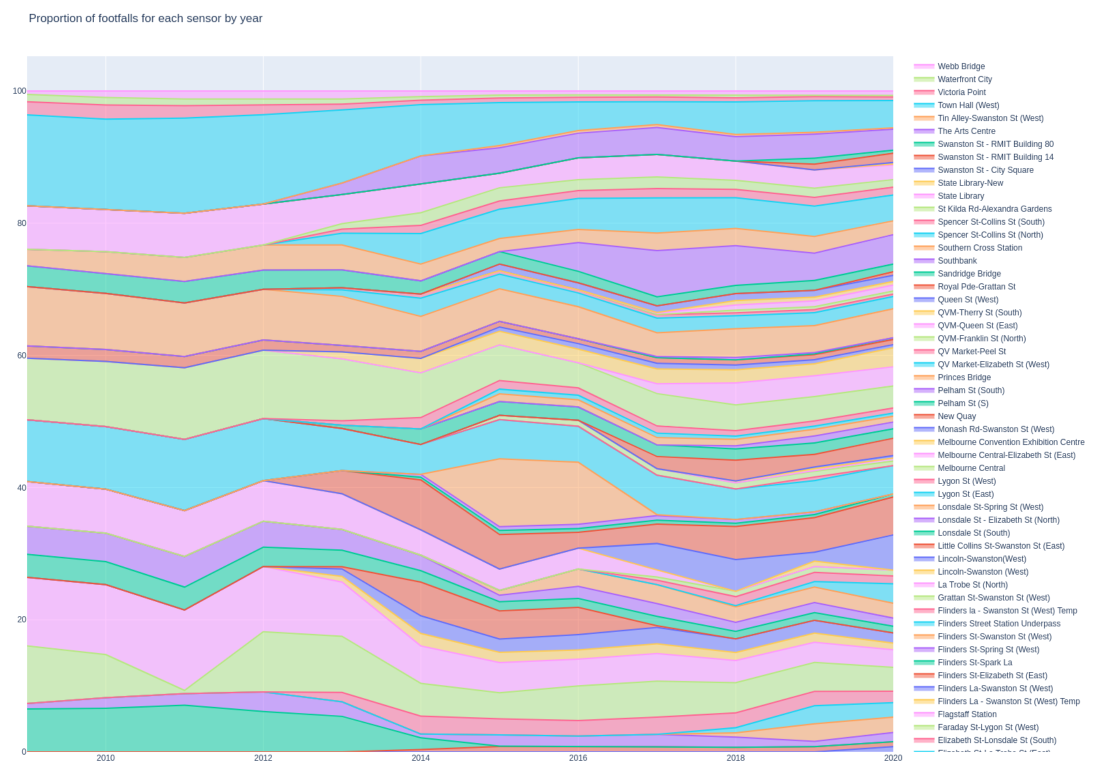
```
[66]: fig_dict = fig.to_dict()
      #print(type(fig_dict))
```

```
<class 'plotly.graph_objs._figure.Figure'>
```

**Plotly Figure API Demo** Stacked Area chart of Sensor Counts by Year

```
[68]: import plotly.graph_objects as go



      def make_stacked_sensor_plot(df, years=None, sensors=None, normalised=True):
```

```
    if years is not None:
        df = df[df["Year"].isin(years)]
    if sensors is not None:
        df = df[df["Sensor_Name"].isin(sensors)]
    sensor_years_s = df.groupby(["Sensor_Name", "Year"])["Hourly_Counts"].sum()
    sensor_dfs = [(sensor, dfx.reset_index("Sensor_Name")) for sensor, dfx in␣
 ↪sensor_years_s.groupby(level=0)]

    fig = go.Figure()
    for sensor, df in sensor_dfs:
        fig.add_trace(go.Scatter(
            x=df.index, y=df["Hourly_Counts"],
            #mode='lines',
            name=sensor,
            stackgroup='one',
            groupnorm='percent' if normalised else "",
        ))

    fig.update_layout(width=1500, height=1200, title="Proportion of footfalls␣
 ↪for each sensor by year")
    return fig

make_stacked_sensor_plot(data_df)
```



## 1.7 Making our Plots Interactive

TODO:

- make this interactive with ipywidgets (and Figure Widget)
- introduce voila
- then copy into separate notebook and deploy with ngrok
- will come back to this later for dashboards

## 1.8 Spatial Information Specific Libraries

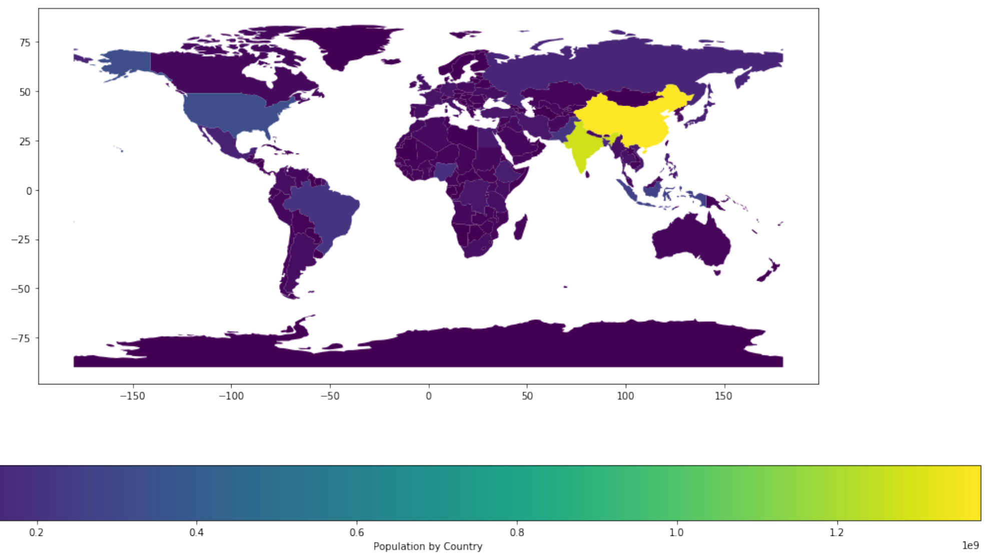
Python Libraries that can helpvisualisang Spatial information

- GeoPandas
- GeoPlot
- Folium
- ipyleaflet
- GeoViews
- CartoPy
- QGIS

### 1.8.1 GeoPandas and Matplotlib

https://geopandas.org/mapping.html

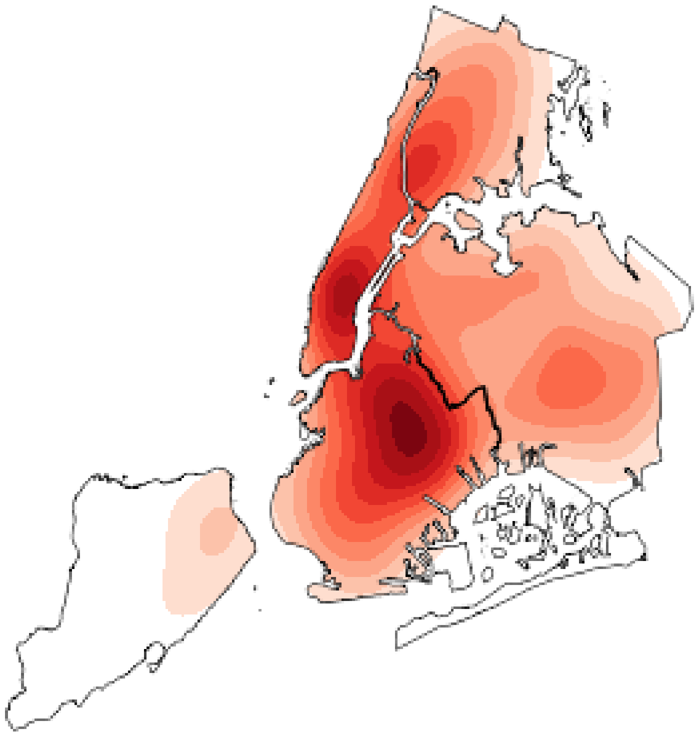
[10]:
```
import matplotlib.pyplot as plt
import geopandas

world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowres'))

fig, ax = plt.subplots(1, 1, figsize=(20,10))

world.plot(
    column='pop_est',
    ax=ax,
    legend=True,
    legend_kwds={
        'label': "Population by Country",
        'orientation': "horizontal"
    }
);
```



### 1.8.2 GeoPandas and GeoPlot

- GeoPlot Gallery
- CartoPy

[2]:
```
# note this needs CartoPy installed

import geopandas
import geoplot

boroughs = geopandas.read_file(
    geoplot.datasets.get_path('nyc_boroughs')
)
collisions = geopandas.read_file(
    geoplot.datasets.get_path('nyc_injurious_collisions')
)

ax = geoplot.kdeplot(
    collisions.head(1000), clip=boroughs.geometry,
    shade=True, cmap='Reds',
    projection=geoplot.crs.AlbersEqualArea())
geoplot.polyplot(boroughs, ax=ax, zorder=1);
```



### 1.8.3 Demo: Datashader

Datashader is a graphics pipeline system for creating meaningful representations of large datasets quickly and flexibly.

US Census Data Visualisation

### 1.8.4 An exercise for the Reader: *Pedestrian Footfall Spatial Visualisation*

**Part 1**: Create a spatial visualisation using the Pedestrian footfall dataset on

**Part 2**: Investigate the impact of COVID-19 on pedestrian traffic in Melbourne CBD

*Note: For part 1 (and 2, depending on how you approach it), you will need to geocode the sensor names to lat/longs*

```
[10]: data_df["Sensor_Name"].unique()
```

```
[10]: array(['Flinders St-Spark La', 'Alfred Place', 'Lygon St (East)',
             'Lonsdale St-Spring St (West)', 'Queen St (West)',
             'St Kilda Rd-Alexandra Gardens', 'Grattan St-Swanston St (West)',
             'Monash Rd-Swanston St (West)', 'Tin Alley-Swanston St (West)',
             'Southbank', 'Little Collins St-Swanston St (East)',
             'Pelham St (S)', 'Melbourne Central-Elizabeth St (East)',
             'QVM-Queen St (East)', 'QVM-Therry St (South)',
             'Faraday St-Lygon St (West)', 'QVM-Franklin St (North)',
             'Elizabeth St-Lonsdale St (South)', 'Lincoln-Swanston(West)',
             'Elizabeth St-La Trobe St (East)',
             'Lonsdale St - Elizabeth St (North)', 'Bourke St Bridge',
             'Bourke St - Spencer St (North)', 'Swanston St - RMIT Building 80',
             'Swanston St - RMIT Building 14', 'La Trobe St (North)',
             'Town Hall (West)', 'Collins Place (South)',
             'Collins Place (North)', 'Collins St (North)',
             'Bourke Street Mall (South)', 'Bourke Street Mall (North)',
             'Melbourne Central', 'State Library', 'Southern Cross Station',
             'Victoria Point', 'New Quay', 'Waterfront City', 'Webb Bridge',
             'Birrarung Marr', 'Princes Bridge',
             'Flinders Street Station Underpass', 'Sandridge Bridge',
             'QV Market-Elizabeth St (West)', 'Spencer St-Collins St (North)',
             'Spencer St-Collins St (South)',
             'Melbourne Convention Exhibition Centre',
             'Bourke St-Russell St (West)', 'Chinatown-Lt Bourke St (South)',
             'Chinatown-Swanston St (North)', 'Flinders St-Elizabeth St (East)',
             'QV Market-Peel St', 'The Arts Centre', 'Lygon St (West)',
             'Lonsdale St (South)', 'Flinders La-Swanston St (West)',
             'Flagstaff Station', 'Australia on Collins', 'City Square',
             'Flinders St-Spring St (West)', 'Flinders St-Swanston St (West)',
             'Pelham St (South)', 'Lincoln-Swanston (West)',
             'Flinders La - Swanston St (West) Temp',
             'Flinders la - Swanston St (West) Temp', '231 Bourke St',
             'Royal Pde-Grattan St', 'Swanston St - City Square',
             'State Library-New'], dtype=object)
```

## 1.9 What Makes a Good Data Visualisation?

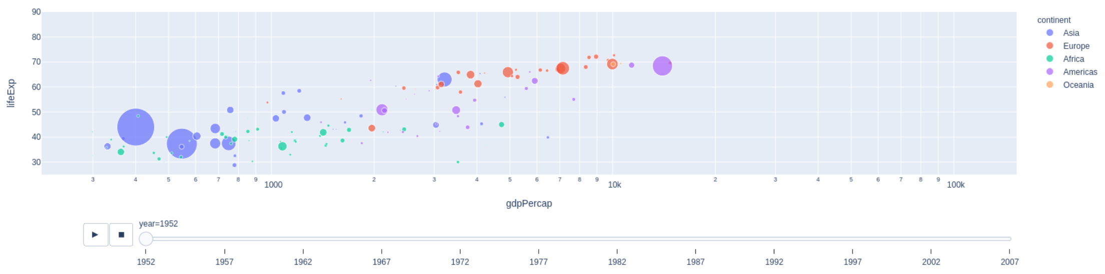
Data can be visualised an infinite number of ways. See One Dataset, Visualized 25 Ways

Careful use of visual encodings

- position
- color
- shape
- time

[19]:
```python
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x="gdpPercap", y="lifeExp", size="pop", color="continent",
           hover_name="country", log_x=True, size_max=60,␣
 ↪animation_frame="year", range_y=[25,90])
```

```
fig.show()
```



### Good for Ordinal Data

- size
- orientation
- colour saturation

### Good for Categoical Data

- colour hue
- shape
- texture

### 1.9.1 Useful tips

- Select an accessable colour palette (use a tool: eg Data Color Picker)
- Always title your plots
- Always label your axes
- Don't truncate your y-axis
- Don't use word clouds for preenting information (use a bar chart instead)
- Don't use pie charts for comparing many/small categories

## 1.10 Making Interactive Dashboards

Contexts: * A dashboard for decision support (eg business intelligence) * Custom reports * Making an interactive notebook available for other people to explore * Custom tool for use in data project

Libraries: * Voila * Panel * Plotly Dash * Streamlit

Useful Talk Surveying these: *Analytic Web Apps in Python* * Video * Slides

## 1.11 Assorted Links

- The Python Graph Gallery
- The Pudding
- Flowing Data
- WTF Visualizations

- Story Telling With Data podcast
- Data Stories podcast

### 1.11.1 Wrapping Up

- Python has a *lot* of visualisation tools to choose from
- Can be daunting to wade through them
- Think about the following questions:
- What kind of data will I be working with?
- What is the purpose of the visualisations?
- Who (if any) is the audience?
- How will they consume the visualisations?

Once you answer these questions, you will have narrowed it down to a much smaller number. Try all the candidates to see what works for you.

No one tool is going to meet all your visualisation needs, so you will need to mix it up across projects, and frequently within the one project.

Happy plotting!