Data Visualization Techniques Demo

This notebook demonstrates a range of data visualization techniques using Matplotlib, Seaborn, and Plotly. Datasets used: `iris`, `tips`, and `flights` from seaborn.

Setup: installs and imports

Run the next cell to import libraries. If Plotly is not installed, the cell will install it programmatically.

```python
# Core imports and plotting defaults
import sys
import subprocess
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Set styles
%matplotlib inline
sns.set(style="darkgrid", context="notebook")
plt.rcParams['figure.figsize'] = (10, 5)
plt.rcParams['font.size'] = 12

# Ensure plotly is available
try:
    import plotly.express as px
except Exception:
    subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'plotly'])
    import plotly.express as px

# Ensure statsmodels is available for seaborn lowess/regression
try:
    import statsmodels.api as sm
except Exception:
    subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'statsmod
    import statsmodels.api as sm
print('libraries loaded')
```

Load example datasets

We'll load `iris`, `tips`, and `flights` from seaborn and inspect them briefly.

```python
import seaborn as sns
iris = sns.load_dataset('iris')
tips = sns.load_dataset('tips')
```

```
flights = sns.load_dataset('flights')

print('iris:', iris.shape)
display(iris.head())
print('tips:', tips.shape)
display(tips.head())
print('flights:', flights.shape)
display(flights.head())
```

Line plots & time series (Matplotlib / pandas)

Aggregate the `flights` dataset into a time series and demonstrate line plotting with rolling mean and shaded confidence band.

```
In [ ]:  # prepare a datetime index from flights dataset
         fl = flights.copy()
         fl['month_num'] = pd.to_datetime(fl['month'], format='%b').dt.month
         fl['date'] = pd.to_datetime(fl['year'].astype(str) + '-' + fl['month_num'].a
         ts = fl.set_index('date')['passengers'].sort_index()
         ts = ts.asfreq('MS')

         # plot with rolling mean and confidence band
         fig, ax = plt.subplots()
         ax.plot(ts.index, ts.values, label='passengers', color='C0')
         rolling = ts.rolling(window=12).mean()
         ax.plot(rolling.index, rolling.values, label='12-month rolling mean', color=
         # simple band: rolling +/- std
         rolling_std = ts.rolling(window=12).std()
         ax.fill_between(ts.index, (rolling - rolling_std).values, (rolling + rolling
         ax.set_title('Monthly airline passengers (flights)')
         ax.set_ylabel('passengers')
         ax.legend()
```

Scatter plots and regression overlays

Scatter plot for `iris` (sepal length vs sepal width) with Seaborn regression overlay.

```
In [ ]:  # scatter with seaborn and regression line
         fig, ax = plt.subplots()
         sns.scatterplot(data=iris, x='sepal_length', y='sepal_width', hue='species',
         sns.regplot(data=iris, x='sepal_length', y='sepal_width', scatter=False, ax=
         ax.set_title('Iris: sepal length vs sepal width (with LOWESS)')
         ax.legend(title='species')
```

Histograms and KDE

Histogram and KDE of `tips.total_bill` comparing density and bins.

```
In [ ]: fig, ax = plt.subplots()
        sns.histplot(tips['total_bill'], bins=30, kde=True, stat='density', ax=ax, c
        ax.set_title('Distribution of total bill (tips dataset)')
```

Boxplots and violin plots

Boxplot of total bill by day and violinplot with swarm overlay.

```
In [ ]: fig, axes = plt.subplots(ncols=2, figsize=(12,5))
        # Use `hue`=x and remove the legend to avoid the palette-without-hue depreca
        sns.boxplot(data=tips, x='day', y='total_bill', hue='day', dodge=False, ax=a
        if axes[0].get_legend() is not None:
            axes[0].get_legend().remove()
        axes[0].set_title('Boxplot: total_bill by day')
        # For violinplot we also pass hue and then remove the generated legend
        sns.violinplot(data=tips, x='day', y='total_bill', hue='day', split=False, i
        if axes[1].get_legend() is not None:
            axes[1].get_legend().remove()
        # overlay the raw points (swarm) as before
        sns.swarmplot(data=tips, x='day', y='total_bill', color='k', size=3, ax=axes
        axes[1].set_title('Violin + swarm: total_bill by day')
```

Heatmap and correlation matrix

Show correlations for the `iris` dataset.

```
In [ ]: corr = iris.select_dtypes(include=np.number).corr()
        fig, ax = plt.subplots(figsize=(8,6))
        sns.heatmap(corr, annot=True, cmap='coolwarm', center=0, ax=ax)
        ax.set_title('Correlation matrix (iris)')
```

Pairplot for multivariate exploration

Pairwise relationships with hue by species. This can be slow on large datasets.

```
In [ ]: # Pairplot (can take a moment)
        sns.pairplot(iris, hue='species', diag_kind='kde', corner=False)
```

Interactive plots with Plotly Express

Interactive scatter (tips) and an interactive time series (flights).

```
In [ ]: import plotly.express as px
        from IPython.display import HTML, display
        # Create interactive scatter and attempt to show inline; fall back to HTML f
        fig = px.scatter(tips, x='total_bill', y='tip', color='day', hover_data=['se
        try:
```

```python
        fig.show()
except ValueError as _err:
    fname = 'interactive_tips.html'
    fig.write_html(fname, include_plotlyjs='cdn')
    display(HTML(f'<a href="{fname}" target="_blank">Open interactive_tips.h

    # interactive time series
ts_df = ts.reset_index().rename(columns={'index':'date', 'passengers':'passe
fig2 = px.line(ts_df, x='date', y='passengers', title='Monthly passengers (i
try:
    fig2.show()
except ValueError:
    fname2 = 'interactive_flights.html'
    fig2.write_html(fname2, include_plotlyjs='cdn')
    display(HTML(f'<a href="{fname2}" target="_blank">Open interactive_fligh
```

Saving figures and exporting

Examples of saving static Matplotlib figures and exporting Plotly to HTML.

```python
In [ ]:  # save last matplotlib figure (if any)
fig_static, ax = plt.subplots()
ax.plot([0,1,2], [10,20,15])
ax.set_title('Example save')
fig_static.savefig('example_plot.png', dpi=150, bbox_inches='tight')
print('saved example_plot.png')

# save plotly to HTML
fig.write_html('interactive_tips.html')
print('saved interactive_tips.html')
```

Notes on performance and large datasets

For very large point clouds consider hexbin, datashader, or downsampling. `plt.hexbin` and `sns.kde` are useful for dense plots.

Datashader example (for very large datasets)

This cell demonstrates how to use `datashader` to render large scatter clouds efficiently. If `datashader` is not installed in your environment the cell will fall back to a Matplotlib `hexbin` plot and print installation instructions.

```python
In [ ]:  # Datashader example: create a large synthetic point cloud by repeating `tip
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# build a larger dataset by repeating tips several times (adjust multiplier
mult = 2000  # result ~244 * 2000 ~ 488k rows; reduce if memory is constrain
```

```python
try:
    big = pd.concat([tips]*mult, ignore_index=True)
except Exception:
    big = tips.sample(frac=1, replace=True).reset_index(drop=True)
# add small jitter so points are not identical
np.random.seed(1)
big['x'] = big['total_bill'] + np.random.normal(0, 0.1, size=len(big))
big['y'] = big['tip'] + np.random.normal(0, 0.05, size=len(big))

# Try to use datashader, otherwise fallback to hexbin
try:
    import datashader as ds
    import datashader.transfer_functions as tf
    from datashader import reductions
    import colorcet
    from IPython.display import HTML, display
    # create canvas and aggregate
    cvs = ds.Canvas(plot_width=800, plot_height=400)
    agg = cvs.points(big, 'x', 'y', ds.count())
    img = tf.shade(agg, cmap=colorcet.m_fire, how='eq_hist')
    display(img)
except Exception as e:
    print('datashader not available or failed to render (', e, '). Falling b
    fig, ax = plt.subplots(figsize=(10,4))
    hb = ax.hexbin(big['x'], big['y'], gridsize=80, cmap='viridis')
    fig.colorbar(hb, ax=ax, label='counts')
    ax.set_xlabel('total_bill (jittered)')
    ax.set_ylabel('tip (jittered)')
    ax.set_title('Hexbin fallback (datashader not installed)')
    plt.show()
    print('To enable datashader: pip install datashader colorcet')
```

Animated plots (Matplotlib FuncAnimation)

This example creates a simple animated sine wave using Matplotlib's
FuncAnimation and displays it inline using HTML/JS. If your environment does not
support JS rendering, you can save the animation to MP4/GIF (may require
ffmpeg).

```python
In [ ]:  import numpy as np
         import matplotlib.pyplot as plt
         from matplotlib import animation
         from IPython.display import HTML

         fig, ax = plt.subplots()
         x = np.linspace(0, 2*np.pi, 400)
         line, = ax.plot(x, np.sin(x))
         ax.set_ylim(-1.5, 1.5)
         ax.set_title('Animated sine wave')

         def init():
             line.set_ydata(np.sin(x))
             return (line,)
```

```python
def animate(i):
    line.set_ydata(np.sin(x + i/10.0))
    return (line,)

anim = animation.FuncAnimation(fig, animate, init_func=init, frames=200, int
plt.close(fig)
# Display as JS animation in the notebook (will embed JS/CSS)
HTML(anim.to_jshtml())
```

In [ ]:
```python
# Export the previously-created `anim` to MP4 or GIF.
# This tries MP4 using ffmpeg first, then falls back to GIF via pillow.
from matplotlib import animation as anim_module
import os
out_mp4 = 'sine_animation.mp4'
out_gif = 'sine_animation.gif'
saved = False
# Try MP4 (requires ffmpeg available on PATH)
try:
    Writer = anim_module.writers['ffmpeg']
    writer = Writer(fps=30, metadata=dict(artist='auto'), bitrate=1800)
    anim.save(out_mp4, writer=writer)
    print(f'Saved animation as MP4: {out_mp4}')
    saved = True
except Exception as e:
    print('MP4 export failed or ffmpeg not available:', e)
# Fallback to GIF using pillow writer
if not saved:
    try:
        anim.save(out_gif, writer='pillow', fps=30)
        print(f'Saved animation as GIF: {out_gif}')
        saved = True
    except Exception as e:
        print('GIF export failed:', e)
# Final message with install hints if nothing worked
if not saved:
    print('Unable to save animation. To enable MP4 export install ffmpeg on
    print('For GIF export ensure pillow is installed: pip install pillow')
```

Requirements and how to run

Required Python packages: numpy, pandas, matplotlib, seaborn, plotly,
statsmodels. Install with: `pip install numpy pandas matplotlib seaborn`
`plotly statsmodels`

Recommended quick steps to run locally:

1. Create and activate a virtual environment (recommended): `python -m venv`
   `.venv` On Linux/macOS: `source .venv/bin/activate` On Windows
   (PowerShell): `./.venv/Scripts/Activate.ps1`
2. Install dependencies: `pip install -r requirements.txt` or the pip line
   above.

3. Start Jupyter: `jupyter lab` or `jupyter notebook` and open this file: `/home/sysadmin/Notebooks/data-visualization-demo.ipynb` .

Notes:

- Plotly inline rendering may raise an error if `nbformat` or notebook MIME renderers are missing; the notebook already falls back to writing interactive HTML files and showing links.
- To export animations to MP4/GIF you may need `ffmpeg` installed on your system.
- For very large datasets consider using datashader (optional) or downsampling before plotting.

Conclusion and next steps

This notebook provided quick examples across Matplotlib, Seaborn, and Plotly. Next steps: experiment with custom palettes, GridSpec layouts, animations, or datashader for very large datasets.

To run locally: in a terminal run `jupyter lab` or `jupyter notebook` and open this file: `/home/sysadmin/Notebooks/data-visualization-demo.ipynb` .

Exporting this notebook to PDF

This section demonstrates how to produce a PDF copy of this notebook programmatically using `nbconvert` . The cell below first attempts a direct PDF export (which commonly requires a LaTeX installation such as TeX Live). If that fails it falls back to exporting an HTML file which you can open in a browser and print/save as PDF.

Note: the code assumes this notebook file is named `data-visualization-demo.ipynb` and is in the current working directory. If your notebook has a different name or path, update the `nb` variable in the code cell.

```python
In [ ]:  # Programmatic notebook -> PDF example with fallback to HTML
         from pathlib import Path
         import subprocess, sys, shutil

         # Assumption: notebook file is in the current working directory with this na
         nb = Path('data-visualization-demo.ipynb')
         if not nb.exists():
             print(f'Notebook file not found at {nb.resolve()}. Update the `nb` varia
         else:
             out_pdf = nb.with_suffix('.pdf')

             def run_cmd(cmd):
```

```python
        print('Running:', ' '.join(cmd))
        res = subprocess.run(cmd, stdout=subprocess.PIPE, stderr=subprocess.
        print(res.stdout)
        return res.returncode == 0

    success = False
    # Try jupyter nbconvert (preferred). Note: PDF conversion usually requir
    jupyter = shutil.which('jupyter')
    if jupyter:
        success = run_cmd([jupyter, 'nbconvert', '--to', 'pdf', str(nb)])

    # If direct PDF failed, export to HTML and instruct the user to print to
    if not success:
        print('Direct PDF export failed or is not available. Falling back to
        html_file = nb.with_suffix('.html')
        # Try nbconvert to HTML (more widely supported)
        if jupyter:
            ok = run_cmd([jupyter, 'nbconvert', '--to', 'html', str(nb)])
        else:
            ok = run_cmd([sys.executable, '-m', 'nbconvert', '--to', 'html',
        if ok:
            print(f'HTML exported to {html_file}. You can open it in a brows
        else:
            print('HTML export also failed. You may need to install jupyter/

    else:
        print(f'PDF exported to: {out_pdf.resolve()}')
```

Notes and alternatives

- Direct PDF export via `nbconvert --to pdf` typically uses a LaTeX intermediate; installing a TeX distribution (TeX Live on Linux) is often required: e.g., `sudo apt install texlive-xetex texlive-fonts-recommended texlive-latex-recommended`.
- If you prefer not to install LaTeX, export to HTML (`nbconvert --to html`) and then: open in a browser and Print -> Save as PDF, or use a headless browser (Chromium/Chrome) to print-to-pdf programmatically (e.g., Puppeteer/Playwright).
- Example command-line alternative (if you have chromium): `chromium --headless --disable-gpu --print-to-pdf=output.pdf data-visualization-demo.html`
- If you want, I can also add an automated Chromium-based HTML->PDF step (requires installing Chromium) or a Playwright/Pyppeteer example—tell me which fallback you'd prefer.

Automated Chromium HTML -> PDF (optional)

The cell below will look for a Chromium/Chrome binary on your PATH and, if found, will run it in headless mode to convert the previously-exported HTML

(`data-visualization-demo.html`) to a PDF. This is useful when LaTeX is not
installed and you want a fully automated HTML->PDF step.

If you don't have Chromium/Chrome installed, the cell will print install
suggestions for common Linux distributions.

```
In [ ]:   # Automated Chromium/Chrome HTML -> PDF converter
          from pathlib import Path
          import shutil, subprocess, sys

          nb = Path('data-visualization-demo.ipynb')
          html = nb.with_suffix('.html')
          out_pdf = nb.with_suffix('.chromium.pdf')

          if not html.exists():
              print(f'HTML file not found at {html.resolve()}. Run the HTML export ste
          else:
              # Common executable names for chromium/google-chrome
              candidates = ['chromium', 'chromium-browser', 'chromium-stable', 'google
              exe = None
              for c in candidates:
                  path = shutil.which(c)
                  if path:
                      exe = path
                      break

              if not exe:
                  print('No Chromium/Chrome executable found on PATH.')
                  print('Install suggestions (Debian/Ubuntu): sudo apt install chromiu
                  print('Or on Debian/Ubuntu to get a recent Chrome: install google-ch
              else:
                  # Use an absolute path to the HTML file
                  html_path = str(html.resolve())
                  pdf_path = str(out_pdf.resolve())
                  cmd = [exe, '--headless', '--no-sandbox', '--disable-gpu', f'--print
                  print('Running:', ' '.join(cmd))
                  res = subprocess.run(cmd, stdout=subprocess.PIPE, stderr=subprocess.
                  print(res.stdout)
                  if res.returncode == 0 and Path(pdf_path).exists():
                      print(f'Chromium produced PDF: {pdf_path}')
                  else:
                      print('Chromium print-to-pdf failed. Review the output above. As
```