Data Visualization with Python - PygWalker: Tableau in your Jupyter notebook - Slides

Sprints

Introduction to Data Visualization

Sprints

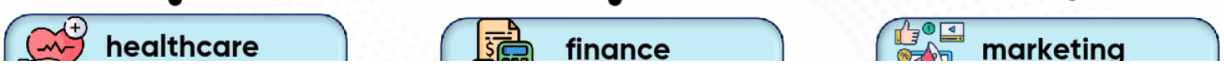
The Importance of Data Visualization

Data visualization plays a crucial role in data science by:

- transforming raw data into useful insights that aid decision-making.
- helps communicate complex information clearly and effectively.

Sprints

The Importance of Data Visualization

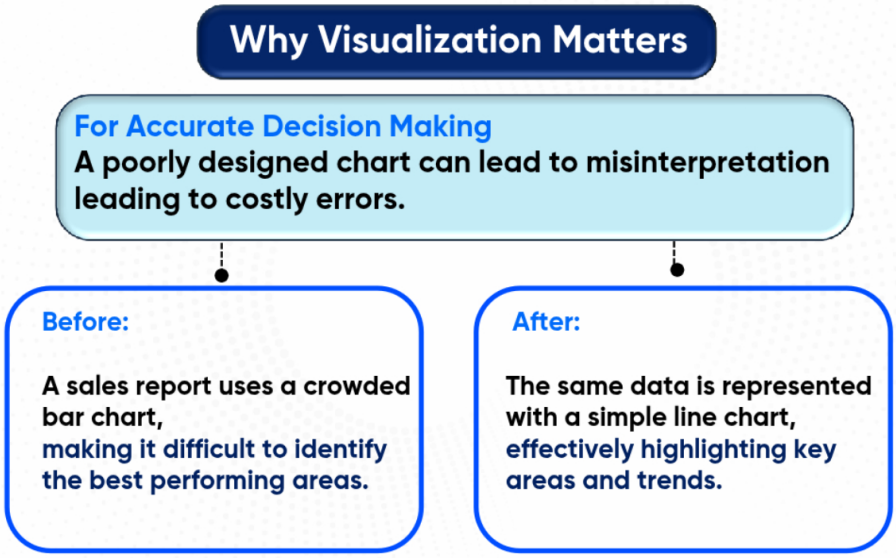


- visualization can help track patient outcomes
- monitor disease trends
- allocate resources efficiently.

- visualizing market data helps identify patterns
- make informed investment decisions.

- visualizations of customer data help identify consumer trends
- target demographics
- measure campaign effectiveness.

Sprints



Sprints

Visualization

Sprints

Key Principles of Effective Visualization

Simplicity
- Strip down visuals to their essential elements.
- Remove unnecessary elements that can confuse viewers.

Clarity
- Visuals should convey data in a straightforward manner.
- Labels, legends, and context are critical for a clear understanding.

Accuracy
- Avoid misleading scales, distortions, or selective representation that could mislead the audience.

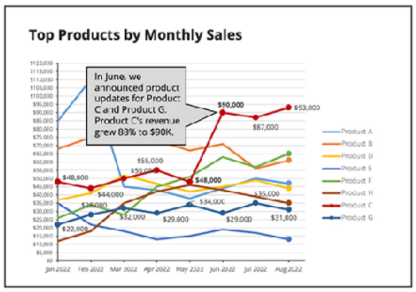
Efficiency
- Create visuals that allow viewers to grasp the message quickly.
- Use appropriate formats to reduce the time needed to interpret the data.

Sprints

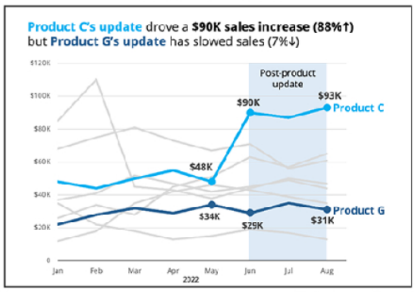
Avoid Clutter

Less is More
- **Avoid clutter by focusing on the essentials.**
- **be mindful of visual clutter such as unnecessary grid lines, excessive text, or colors that do not add value.**

With Clutter



Without Clutter



- simple bar chart that communicates the same information in a more accessible way.
- By keeping things simple, your audience can focus on the data story you want to tell.

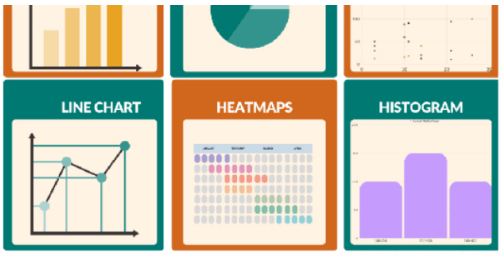
Sprints

Choosing Appropriate Chart Types

Selecting the appropriate chart type is crucial for accurate data communication:

Categorical Data:
Use bar charts or pie charts for comparison.

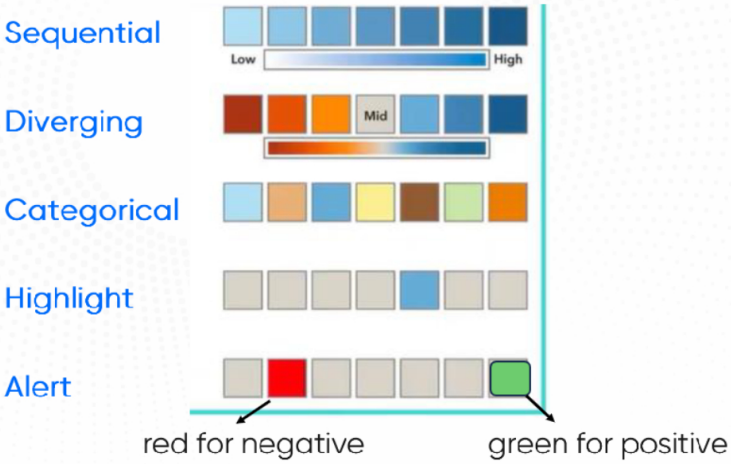
Continuous Data:
Use line charts for time series or scatter plots for showing relationships.



Sprints

Effective Use of Colors

Color Theory in Visualization



Sprints

Understanding Your Audience

Audience-Centered Visualization Design: Knowing your audience will influence the design of your visualization.

Executives high-level visuals showing only key takeaways

Analysts need more granular details.

Sprints

Overview of Matplotlib and Seaborn

Sprints

Types of libraries

Matplotlib

Seaborn

Sprints

Matplotlib

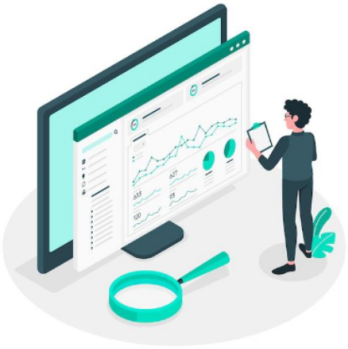
- Is a foundational Python library for creating static, animated, and interactive visualizations.
- It provides extensive control over plot elements, making it a popular choice for detailed, customized graphs.

Sprints

Matplotlib is well-suited for:

Static Reports:
Where consistency is crucial.

Highly Customized Plots:
If you need to adjust every aspect of the visual.



Sprints

Seaborn for Statistical Visualizations

- Seaborn is built on top of Matplotlib, offering a high-level interface to create visually appealing and informative statistical plots with minimal code.

Sprints

Benefits of Seaborn:

- Pre-set themes for aesthetically pleasing visuals.
- Built-in datasets for practice.
- Simple functions to create complex statistical plots, such as pair plots and heatmaps.

Sprints

Matplotlib vs. Seaborn

Feature	Matplotlib	Seaborn
Complexity	More control, but more coding	Easier to use, less detailed control
Plot Customization	High	Moderate
Suitable For	Custom plots, detailed tweaking	Statistical visualizations

Sprints

Overview of Plotly and PygWalker

Sprints

Types of libraries

Types of libraries

Plotly

PygWalker

Sprints

Interactive Visualization with Plotly

Plotly is an interactive graphing library that allows users to create dynamic visualizations.

It enables features like:

- zooming
- Hide and show

Sprints

Benifits of Plotly

- Ease of use
- Share with the user and the ability to modify the chart shape
- Hovering ability to see more information in the same chart
- The ability to create a dashboard

Sprints

PygWalker

PygWalker provides a quick and easy way to create exploratory visualizations within Jupyter notebooks.

It offers functionality similar to Tableau and Excel but directly within the Python ecosystem, enabling analysts to generate dashboards and insights with minimal coding.

Sprints

Comparison with Plotly/PygWalker and Matplotlib/Seaborn

Plotly and PygWalker:
Bring interactivity to visualization

Plotly:
Ideal for presentations where viewers need to interact with the data (e.g., dashboards).

PygWalker:
Quick, Tableau-like exploration in Jupyter.

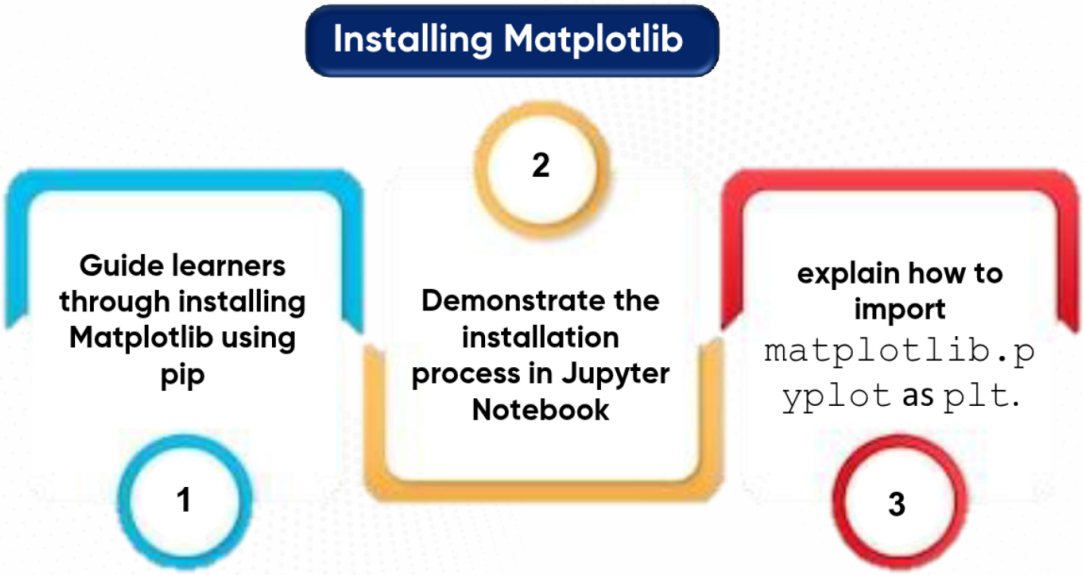
Matplotlib and Seaborn:
Doesn't bring interaction to the visualization.

Perfect for creating publication-quality static visuals.

Sprints

Matplotlib Fundamentals

Sprints



Sprints

1. Introduction to Matplotlib

Sprints

1.1 Line Plot and Scatter Plot

Example: Line Plot

Create a line plot to visualize sales over 12 months.

```
# Sample data
dates = pd.date_range(start='2023-01-01', periods=12, freq='M')
sales = [12, 18, 20, 25, 22, 30, 35, 40, 38, 32, 28, 36]
# Line Plot
plt.plot(dates, sales)
plt.show()
```

Exercise: Line Plot

- Modify the above line plot to show sales data for two different products on the same plot. Use different colors and markers to distinguish the products.

Sprints

1.1 Line Plot and Scatter Plot

Matplotlib is a powerful visualization library that allows for a high degree of customization.

Sprints

1.1 Line Plot and Scatter Plot

Example: Scatter Plot

Visualize the relationship between two variables, e.g., advertising budget and sales.

```
# Sample data
advertising_budget = np.random.randint(10, 50, 20)
sales = advertising_budget + np.random.normal(0, 5, 20)
advertising_budget, sales
# Scatter Plot
plt.scatter(advertising_budget, sales)
plt.show()
```

Exercise: Scatter Plot

- Create a scatter plot comparing two other numerical variables, such as social media spending and customer engagement. Customize the colors and add appropriate labels.

Sprints

1.2 Bar Plots and Histograms

Example: Bar Plot

```
# Sample data
categories = ['A', 'B', 'C', 'D']
values = [25, 40, 35, 50]
categories, values
# Bar Plot
plt.bar(categories, values)
plt.show()
```

Sprints

1.2 Bar Plots and Histograms

Example: Histogram

```
# Sample data
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=500)
data
# Histogram
plt.hist(data)
plt.show()
```

Exercise: Bar Plots and Histograms

- Create a bar plot for a different dataset, and create a histogram with different bin sizes to see how the distribution changes.

Sprints

1.3 Box Plots

Example: Box Plot

```
# Sample data
```

```
data = pd.DataFrame({
    'Category': ['A', 'B', 'C', 'D'] * 20,
    'Values': np.random.randint(10, 50, 80)
})
data
# Box Plot
plt.boxplot([data[data['Category'] == cat]['Values'] for cat in
data['Category'].unique()], labels=data['Category'].unique())
plt.show()
```

Sprints

Matplotlib Advanced

Sprints

2.1 Customizing Colors, Markers, Titles, Labels, and Legends

Example: Customizing Plot Appearance

```
# Sample data
dates = pd.date_range(start='2023-01-01', periods=12, freq='M')
sales_product_a = [12, 18, 20, 25, 22, 30, 35, 40, 38, 32, 28, 36]
sales_product_b = [15, 17, 23, 20, 26, 29, 33, 37, 39, 30, 27, 34]
# Custom Line Plot
plt.figure(figsize=(12, 6))
plt.plot(dates, sales_product_a)
plt.show()
```

Sprints

2.1 Customizing Colors, Markers, Titles, Labels, and Legends

Example: Customizing Plot Appearance

```
# Custom Line Plot
plt.figure(figsize=(12, 6))
plt.plot(dates, sales_product_a)
plt.title('Monthly Sales Comparison', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Sales', fontsize=14)
plt.show()
```

Sprints

2.1 Customizing Colors, Markers, Titles, Labels, and Legends

Example: Customizing Plot Appearance

```
# Custom Line Plot
plt.figure(figsize=(12, 6))
plt.plot(dates, sales_product_a, label='Product A')
plt.plot(dates, sales_product_b, label='Product B')
plt.title('Monthly Sales Comparison', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Sales', fontsize=14)
plt.grid(True)
plt.show()
```

Sprints

2.1 Customizing Colors, Markers, Titles, Labels, and Legends

Example: Customizing Plot Appearance

```
# Custom Line Plot
plt.figure(figsize=(12, 6))
plt.plot(dates, sales_product_a, marker='o', color='b', linestyle='-', linewidth=2,
label='Product A')
plt.plot(dates, sales_product_b, marker='s', color='g', linestyle='--', linewidth=2,
label='Product B')
```

```
plt.title('Monthly Sales Comparison', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Sales', fontsize=14)
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```

Sprints

2.1 Customizing Colors, Markers, Titles, Labels, and Legends

Example: Annotating Plots

```
# Annotate highest sales month
plt.figure(figsize=(12, 6))
plt.plot(dates, sales_product_a, marker='o', color='b', linestyle='-', linewidth=2,
label='Product A')
plt.title('Monthly Sales with Annotation')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.annotate('Highest Sales', xy=(dates[6], sales_product_a[6]), xytext=(dates[4],
sales_product_a[6] + 5),
             arrowprops=dict(facecolor='black', arrowstyle='->'))
plt.grid(True)
plt.show()
```

Sprints

2.2 Creating Subplots

Example: Subplots

```
# Sample data
x = np.linspace(0, 10, 100)

# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs[0, 0].plot(x, np.sin(x), 'r')
axs[0, 0].set_title('Sine Wave')

axs[0, 1].plot(x, np.cos(x), 'b')
axs[0, 1].set_title('Cosine Wave')
```

Sprints

2.2 Creating Subplots

Example: Subplots - Contintue

```
axs[1, 0].plot(x, np.tan(x), 'g')
axs[1, 0].set_title('Tangent Wave')

axs[1, 1].plot(x, -np.sin(x), 'm')
axs[1, 1].set_title('Negative Sine Wave')

for ax in axs.flat:
    ax.set(xlabel='X Axis', ylabel='Y Axis')
    ax.grid(True)

plt.tight_layout()
plt.show()
```

Sprints

2.3 Advanced Layout Techniques

Example: Complex Layout

```
# Sample data
dates = pd.date_range(start='2023-01-01', periods=12, freq='M')
sales = [12, 18, 20, 25, 22, 30, 35, 40, 38, 32, 28, 36]
ad_budget = [5, 7, 10, 9, 11, 14, 16, 18, 17, 15, 13, 19]

# Create subplots with different sizes
fig = plt.figure(constrained_layout=True, figsize=(14, 10))
gs = fig.add_gridspec(3, 3)

ax1 = fig.add_subplot(gs[0, :2])
ax1.plot(dates, sales, marker='o', color='b')
ax1.set_title('Sales Over Time')
```

Sprints

2.3 Advanced Layout Techniques

Example: Complex Layout - Continue

```
ax2 = fig.add_subplot(gs[0, 2])
ax2.bar(['A', 'B', 'C', 'D'], [20, 35, 30, 35], color='c')
ax2.set_title('Category Comparison')

ax3 = fig.add_subplot(gs[1:, 1:])
ax3.scatter(ad_budget, sales, color='r')
ax3.set_title('Advertising Budget vs Sales')

ax4 = fig.add_subplot(gs[1:, 0])
ax4.plot(dates, ad_budget, marker='s', linestyle='--', color='g')
ax4.set_title('Advertising Budget Over Time')
```

```
plt.show()
```

Sprints

2.4 Introduction to 3D Plotting

Example: 3D Plot

```
from mpl_toolkits.mplot3d import Axes3D

# Sample data
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y)
z = np.sin(np.sqrt(x**2 + y**2))
```

Sprints

2.4 Introduction to 3D Plotting

Example: 3D Plot

```
# 3D Surface Plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, z, cmap='viridis')
ax.set_title('3D Surface Plot')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')
plt.show()
```

Sprints

2.4 Introduction to 3D Plotting

Example: Customizing 3D Plots

```
# Customizing 3D Plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, z, cmap='plasma', edgecolor='k', alpha=0.7)
```

```
ax.view_init(elev=30, azim=45)  # Change viewing angle
ax.set_title('Customized 3D Surface Plot')
plt.show()
```

Sprints

2.5 Saving Plots

Example: Saving Plots

```
# Save plot to file
plt.figure(figsize=(10, 6))
plt.plot(dates, sales, marker='o', color='b')
plt.title('Monthly Sales Over a Year')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.grid(True)
plt.savefig('monthly_sales.png', format='png', dpi=300)
plt.show()
```

Sprints

3. Exploring Seaborn

Sprints

3. Exploring Seaborn

- **Seaborn is a high-level interface built on top of Matplotlib that is specifically designed for statistical visualizations.**
- **It allows for easy creation of complex visualizations with minimal code, making it ideal for quick exploratory data analysis.**

Sprints

3.1 Distribution Plots

Example: Histogram & KDE Plot

```
# Sample data
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=500)
# Histogram and KDE
plt.figure(figsize=(10, 6))
sns.histplot(data, kde=True, color='g', bins=20)
plt.title('Distribution of Random Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

Sprints

3.2 Box Plot, Violin Plot, Swarm Plot, and Count Plot

Example: Box Plot

```
# Sample data
data = pd.DataFrame({
    'Category': ['A', 'B', 'C', 'D'] * 20,
    'Values': np.random.randint(10, 50, 80)
})
# Box Plot
plt.figure(figsize=(10, 6))
sns.boxplot(x='Category', y='Values', data=data, palette='Set2')
plt.title('Box Plot of Values by Category')
```

```
plt.title('Box Plot of Values by Category')
plt.xlabel('Category')
plt.ylabel('Values')
plt.grid(True)
plt.show()
```

Sprints

3.2 Box Plot, Violin Plot, Swarm Plot, and Count Plot

Example: Violin Plot

```
# Violin Plot
plt.figure(figsize=(10, 6))
sns.violinplot(x='Category', y='Values', data=data, palette='Set3')
plt.title('Violin Plot of Values by Category')
plt.xlabel('Category')
plt.ylabel('Values')
plt.grid(True)
plt.show()
```

Sprints

3.2 Box Plot, Violin Plot, Swarm Plot, and Count Plot

Example: Swarm Plot

```
# Swarm Plot
plt.figure(figsize=(10, 6))
sns.swarmplot(x='Category', y='Values', data=data, palette='Dark2')
plt.title('Swarm Plot of Values by Category')
plt.xlabel('Category')
plt.ylabel('Values')
plt.grid(True)
plt.show()
```

Sprints

3.2 Box Plot, Violin Plot, Swarm Plot, and Count Plot

Example: Count Plot

```
# Count Plot
plt.figure(figsize=(10, 6))
```

```
plt.figure(figsize=(10, 6))
sns.countplot(x='Category', data=data, palette='Set1')
plt.title('Count Plot of Categories')
plt.xlabel('Category')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```

Exercise: Box, Violin, Swarm, and Count Plots

- Create a combination of a box plot and swarm plot to compare categorical data with added individual data points for better context.

Sprints

3.3 Correlation Heatmap

Example: Correlation Heatmap

```
# Sample data
data = pd.DataFrame({
    'A': np.random.rand(50),
    'B': np.random.rand(50),
    'C': np.random.rand(50),
    'D': np.random.rand(50)})
# Correlation Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

Exercise: Correlation Heatmap: Create a heatmap for a larger dataset, such as the Titanic dataset, to explore the correlations between various features.

Sprints

3.4 Regression Plots

Example: Regression Plot

```
# Sample data
np.random.seed(42)
x = np.random.rand(100) * 100
y = 2 * x + np.random.normal(0, 25, 100)
# Regression Plot
plt.figure(figsize=(10, 6))
sns.regplot(x=x, y=y, color='b')
plt.title('Regression Plot of X vs Y')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
plt.show()
```

Sprints

3.5 Pair Plots

Example: Pair Plot

```
# Sample data
data = sns.load_dataset('iris')

# Pair Plot
sns.pairplot(data, hue='species', palette='Set1')
plt.show()
```

Exercise: Pair Plot

- Use a different dataset, such as the Titanic dataset, and create a pair plot to visualize relationships between multiple features. Set `hue` to a categorical variable like 'sex' or 'class'.

Sprints

Comparison between Seaborn and Matplotlib

Sprints

Seaborn	Matplotlib
Ease of Use: Provides a high-level interface for creating attractive and informative statistical graphics with fewer lines of code	**Customization:** Offers more control over the customization of plots, making it more flexible but often more verbose.
customization: Default styles and color palettes are more aesthetically pleasing	

Statistical Visualizations: It simplifies the process of creating complex visualizations such as Box Plots, violin plots and pair plots,.

Sprints

4. Interactive Visualization with Plotly

Sprints

4. Interactive Visualization with Plotly

- Plotly is a visualization library that allows for the creation of interactive, web-based visualizations.
- Its unique capability to create interactive charts makes it ideal for presentations and data exploration.

Sprints

4.1 Interactive Scatter Plot

Example: Interactive Scatter Plot

Example: Interactive Scatter Plot

```
# Sample data
advertising_budget = np.random.randint(10, 50, 50)
sales = advertising_budget + np.random.normal(0, 5, 50)

# Interactive Scatter Plot
fig = px.scatter(x=advertising_budget, y=sales, title='Advertising Budget vs Sales',
labels={'x': 'Advertising Budget ($k)', 'y': 'Sales ($k)'})
fig.update_traces(marker=dict(size=10, color='blue'))
fig.update_layout(title_font_size=18)
fig.show()
```

Exercise: Interactive Scatter Plot

- Modify the scatter plot to include another variable, such as product category, as the color dimension. This will allow you to see the differences between categories interactively.

Sprints

4.2 Interactive Line Plot

Example: Interactive Line Plot

```
# Sample data
fig = px.line(x=dates, y=sales_product_a, title='Interactive Sales Over Time',
labels={'x': 'Month', 'y': 'Sales'})
fig.update_traces(line=dict(width=3, color='green'))
fig.update_layout(title_font_size=18)
fig.show()
```

Exercise: Interactive Line Plot

- Add multiple lines to the interactive line plot, representing sales data for different products. Use different colors for each line.

Sprints

4.3 Creating Dashboards with Plotly

Example: Creating a Simple Dashboard

```
# Creating Subplots
fig = make_subplots(rows=1, cols=3, subplot_titles=('Line Plot', 'Scatter Plot', 'Bar
Plot'))

# Line Plot
fig.add_trace(go.Scatter(x=dates, y=sales, mode='lines+markers', name='Sales'), row=1,
col=1)

# Scatter Plot
fig.add_trace(go.Scatter(x=advertising_budget, y=sales, mode='markers', name='Ad
Budget vs Sales'), row=1, col=2)
```

Sprints

4.3 Creating Dashboards with Plotly

Example: Creating a Simple Dashboard

```
# Bar Plot
categories = ['A', 'B', 'C', 'D']
sales_per_category = [150, 200, 250, 300]
fig.add_trace(go.Bar(x=categories, y=sales_per_category, name='Sales per Category'),
row=1, col=3)

fig.update_layout(title_text='Interactive Dashboard Example', height=500)
fig.show()
```

Exercise: Creating a Dashboard

- Add more plots to the dashboard, such as pie charts or area plots, to provide a more comprehensive view of the data.

Sprints

4.4 Adding Animations to Plots

Example: Animated Scatter Plot

```
# Sample data
df = px.data.gapminder()

# Animated Scatter Plot
fig = px.scatter(df, x='gdpPercap', y='lifeExp', animation_frame='year',
animation_group='country',
                 size='pop', color='continent', hover_name='country', log_x=True,
size_max=55,
                 title='Life Expectancy vs GDP per Capita Over Time')
fig.update_layout(title_font_size=18)
fig.show()
```

Exercise: Animated Plot

- Create an animated bar chart showing population growth by continent over time.

Sprints

4.5 Interactive Time Series Plot

Example: Interactive Time Series Plot

```
# Sample time series data
time_series_dates = pd.date_range(start='2020-01-01', periods=100, freq='D')
time_series_values = np.cumsum(np.random.randn(100))
```

```
# Interactive Time Series Plot
fig = px.line(x=time_series_dates, y=time_series_values, title='Interactive Time
Series Data', labels={'x': 'Date', 'y': 'Value'})
fig.update_traces(line=dict(width=3, color='blue'))
fig.update_layout(title_font_size=18)
fig.show()
```

Exercise: Interactive Time Series Plot

- Create a time series plot for multiple variables (e.g., stock prices of different companies). Use different colors and add interactive elements to visualize the trends over time.

4.6 Saving and Exporting Plotly Visualizations

Example: Saving Plotly Visualizations

```
# Save interactive plot as HTML
fig.write_html('interactive_scatter.html')

# Save as static image (requires kaleido)
fig.write_image('interactive_scatter.png')
```


5. Plotly Geo-Visualization

5.1 Introduction to Geographical

- Plotly allows for the creation of interactive geographical visualizations.
- which are particularly useful for visualizing data across different regions.

Sprints

5.1 Introduction to Geographical Plotting

Example: Choropleth Map

```
# Sample data
df = px.data.gapminder()

# Choropleth Map
fig = px.choropleth(df, locations='iso_alpha', color='lifeExp', hover_name='country',
                    animation_frame='year', title='Life Expectancy by Country')
fig.update_layout(title_font_size=18)
fig.show()
```

Sprints

5.2 Creating Simple Maps

Example: Simple Geographical Map

```
# Sample data
df = px.data.gapminder().query('year == 2007')

# Simple Geographical Map
fig = px.scatter_geo(df, locations='iso_alpha', color='continent',
hover_name='country',
                     size='pop', projection='natural earth', title='Global Population
in 2007')
fig.update_layout(title_font_size=18)
```

```
fig.show()
```

Sprints

5.3 Advanced Geographical Visualizations

Example: Layered Geospatial Visualizations

```
# Sample data
df = px.data.gapminder().query('year == 2007')

# Layered Geospatial Visualization
fig = px.scatter_geo(df, locations='iso_alpha', color='continent',
hover_name='country',
                     size='pop', projection='orthographic', title='Orthographic View
of Global Population')
fig.update_layout(title_font_size=18)
fig.show()
```

Sprints

6. Using PygWalker for Rapid Insights

Sprints

6. Using PygWalker for Rapid Insights

PygWalker is a tool for rapid, interactive data exploration,

similar to Tableau, but integrated within Jupyter Notebooks.

Sprints

6.1 Quick Data Exploration

Example: PygWalker Integration

Make sure you have installed PygWalker:

```
pip install pygwalker
```

Then, use it with a Pandas DataFrame:

```
# Sample data
df = px.data.gapminder()

# PygWalker for Quick Insights
pyg.walk(df)
```

This will open a Tableau-like interface to quickly explore the dataset, create charts, and gain insights without writing code for each visualization.

Exercise: PygWalker Exploration

- Use PygWalker to explore a different dataset, such as one with more numerical and categorical variables. Create different types of charts and discuss the insights you gain from each.

Sprints

6.2 Customizing PygWalker Outputs

Example: Customizing Outputs

- Customize visualizations generated by PygWalker by selecting different chart types, adjusting axes, and changing colors to better represent the data.

Sprints

THANK YOU

www.sprints.ai

Feedback