Day 3, Session 1: Data Visualization

Three-Day Data Analysis with Python Course

Session Overview

1. **Visualization Principles**
2. **DataFrame-based Plotting**
3. **Plotly Express** (Interactive visualizations)
4. **Seaborn** (Statistical graphics)
5. **Interactive Tools** (marimo UI elements)

Part 1: Visualization Principles

Why Visualize Data?

"A picture is worth a thousand rows"

- Human brains process visuals faster than tables
- Patterns and outliers become obvious
- Tells a story to stakeholders
- Drives decision-making

Data without visualization = answers you can't see

Choosing the Right Chart Type

Question	Chart Type
How has X changed over time?	Line chart
What are the top/bottom categories?	Bar chart
What's the distribution of values?	Histogram, box plot, KDE plot
Is there a relationship between X and Y?	Scatter plot
What's the composition of the whole?	Pie chart, stacked bar plot
How do values compare across groups?	Box plot, violin plot

Rule: Chart type depends on your data and question

Best Practices for Effective Visualizations

DO:

✓ Label your axes clearly
✓ Add informative titles
✓ Use color purposefully (not just decoration)
✓ Keep it simple (remove chart junk)
✓ Sort categories meaningfully (by value, not alphabetically)

Best Practices for Effective Visualizations

DON'T:

✗ Use 3D charts (they distort perception)

✗ Truncate the y-axis to exaggerate differences

✗ Use too many colors (max 5-7 distinct colors)

✗ Use pie charts

Static Visualizations

This means the goal is to export images as PNG or PDF files.

- Good for: Reports, presentations, publications
- Tools: Matplotlib, **Seaborn**
- Pro: Simple, lightweight, print-friendly
- Con: Fixed view, can't explore

Interactive vs Static Visualizations

The goal is to produce a visualization for the web.

- Good for: Dashboards, exploratory analysis, web apps
- Tools: **Plotly**, Altair, Bokeh
- Pro: Zoom, pan, hover tooltips, filters
- Con: Requires browser, larger file sizes

Part 2: DataFrame-based Plotting

The DataFrame Plotting Paradigm

Key idea: Pass the entire DataFrame, specify columns by name

```
# Traditional matplotlib (low-level)
plt.scatter(df['column_x'], df['column_y'])
plt.xlabel('X Label')
plt.ylabel('Y Label')
```

```
# DataFrame-based (high-level)
px.scatter(df, x='column_x', y='column_y')
```

Why This Matters

You already know pandas DataFrames!

Same mental model:

- `df.groupby('category')` → group data
- `px.bar(df, x='category', y='value')` → visualize grouped data

Part 3: Plotly Express

Interactive Visualizations

What is Plotly Express?

Plotly Express is a high-level plotting library:

- Built on top of Plotly
- Creates interactive visualizations in one line
- Perfect for data exploration

```
import plotly.express as px

fig = px.scatter(df, x='age', y='salary')
fig  # Shows interactive plot in marimo!
```

In marimo: Just return the figure object, it renders automatically!

Common Plot Types in Plotly Express

Function	Plot type	Use Case
`px.scatter()`	Scatter plot	Relationships between two variables
`px.line()`	Line chart	Trends over time
`px.bar()`	Bar plot	Compare categories
`px.histogram()`	Histogram	Distribution of single variable
`px.box()`	Box plot	Distribution + outliers
`px.violin()`	Violin plot	Distribution shape
`px.choropleth()`	Map plot	Geographic data

Example: Scatter Plot

```
import plotly.express as px

# Simple scatter plot
fig = px.scatter(df,
                 x='total_purchases',
                 y='total_spent')
fig
```

Interactive features (automatic!):

- Hover to see data points
- Zoom and pan
- Toggle legend items
- Save as PNG

Customizing Plotly Plots

```
fig = px.scatter(
    df,
    x='total_purchases',
    y='total_spent',
    color='age_group',           # Color points by category
    size='avg_purchase_value',   # Size by value
    hover_data=['name', 'city'], # Show extra info on hover
    title='Customer Spending Analysis',
    labels={                      # Rename axis labels
        'total_purchases': 'Number of Purchases',
        'total_spent': 'Total Spent (£)'
    }
)
fig
```

Adding Colors and Sizes

Color by category:

```
px.scatter(df, x='x', y='y', color='category')
```

Size by value:

```
px.scatter(df, x='x', y='y', size='value')
```

Continuous color scale:

```
px.scatter(df, x='x', y='y', color='value',
           color_continuous_scale='Viridis')
```

Plotly automatically creates legends and color bars!

Line Charts for Time Series

```
# Revenue over time
monthly_sales = df.groupby('month')['sales'].sum().reset_index()

fig = px.line(
    monthly_sales,
    x='month',
    y='sales',
    title='Monthly Sales Trend',
    markers=True  # Add dots on the line
)
fig
```

Great for:

- Revenue trends
- User growth
- Performance metrics over time

Bar Charts for Comparisons

```
# Top 10 products by revenue
top_products = df.groupby('product')['revenue'].sum() \
    .sort_values(ascending=False) \
    .head(10) \
    .reset_index()

fig = px.bar(
    top_products,
    x='revenue',
    y='product',
    orientation='h',  # Horizontal bars
    title='Top 10 Products by Revenue'
)
fig
```

Tip: Horizontal bars work better for long category names!

Histograms for Distributions

```
# Distribution of customer ages
fig = px.histogram(
    df,
    x='age',
    nbins=20,           # Number of bins
    title='Age Distribution',
    labels={'age': 'Customer Age'}
)
fig
```

Shows:

- How values are distributed
- Most common ranges
- Outliers

Faceting: Multiple Subplots

Facet by column (vertical):

```
px.scatter(df, x='x', y='y', facet_col='region')
```

Facet by row (horizontal):

```
px.scatter(df, x='x', y='y', facet_row='year')
```

Facet by both:

```
px.scatter(df, x='x', y='y',
           facet_row='region',
           facet_col='year')
```

Use case: Compare patterns across groups side-by-side

Faceting Example

```
# Sales by region over time
fig = px.line(
    df,
    x='month',
    y='sales',
    color='product',
    facet_col='region',  # One subplot per region
    title='Sales by Region and Product'
)
fig
```

Result: 4 subplots (one per region), each showing all products

Great for comparing trends across segments!

Part 4: Seaborn

Statistical Graphics

What is Seaborn?

Seaborn is a statistical visualization library:

- Built on top of Matplotlib
- Specializes in statistical plots
- Beautiful default styling
- Also uses DataFrame-based syntax!

```
import seaborn as sns

sns.scatterplot(data=df, x='age', y='salary')
```

When to use Seaborn:

- Statistical relationships
- Distribution analysis

Seaborn vs Plotly Express

Feature	Plotly Express	Seaborn
Interactivity	✓ Interactive	✗ Static
Use case	Dashboards, exploration	Reports, publications
Statistical plots	Basic	Advanced
Styling	Modern, customizable	Beautiful defaults
Speed	Slower (web rendering)	Faster (matplotlib)

Both use DataFrame-based syntax!

In practice: Use both! Plotly for exploration, Seaborn for reports

Distribution Plots in Seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt

# Histogram with KDE (kernel density estimate)
sns.histplot(data=df, x='age', kde=True)
plt.title('Age Distribution')
plt.show()
```

KDE: Smoothed version of histogram, shows distribution shape

In marimo: Use `plt.gca()` to return the plot

Box Plots and Violin Plots

Box plot:

```
sns.boxplot(data=df, x='region', y='sales')
```

Violin plot (box + distribution shape):

```
sns.violinplot(data=df, x='region', y='sales')
```

Shows:

- Median (middle line)
- Quartiles (box)
- Outliers (dots)
- Full distribution shape (violin)

Categorical Plots

Count plot (bar chart of frequencies):

```
sns.countplot(data=df, x='category')
```

Bar plot (with aggregation):

```
sns.barplot(data=df, x='category', y='value',
            estimator='mean')  # Shows mean by default
```

Point plot (mean + confidence interval):

```
sns.pointplot(data=df, x='month', y='sales')
```

Relationship Plots

Scatter with regression line:

```
sns.regplot(data=df, x='age', y='salary')
```

Multiple relationships:

```
sns.pairplot(df[['age', 'salary', 'purchases']])
```

Result: Matrix of scatter plots for all column pairs

Great for exploring correlations!

Heatmaps for Correlations

```
# Calculate correlation matrix
corr = df[['age', 'salary', 'purchases', 'spending']].corr()

# Visualize as heatmap
sns.heatmap(corr,
            annot=True,       # Show values
            cmap='coolwarm', # Color scheme
            center=0,         # Center colormap at 0
            vmin=-1, vmax=1) # Correlation range
plt.title('Feature Correlations')
plt.show()
```

Perfect for: Identifying relationships between variables

Seaborn Styling

Seaborn has beautiful built-in themes:

```
# Set theme for all plots
sns.set_theme(style='darkgrid')  # or 'whitegrid', 'dark', 'white', 'ticks'

# Set color palette
sns.set_palette('husl')  # or 'deep', 'muted', 'bright', 'pastel', 'dark'

# Now all plots use this style
sns.scatterplot(data=df, x='x', y='y')
```

Makes your plots look professional with zero effort!

Part 5: Interactive Tools

Marimo UI Elements

Make your visualizations interactive with marimo:

```
import marimo as mo

# Dropdown to filter data
country_selector = mo.ui.dropdown(
    options=df['Country'].unique().tolist(),
    value='United Kingdom'
)

# Filter data based on selection
filtered_df = df[df['Country'] == country_selector.value]

# Plot updates automatically when selection changes!
px.scatter(filtered_df, x='Quantity', y='UnitPrice')
```

Reactive: Charts update automatically when inputs change!

Common Marimo UI Elements

```
# Dropdown
mo.ui.dropdown(options=['A', 'B', 'C'])

# Slider
mo.ui.slider(start=0, stop=100, value=50)

# Date range
mo.ui.date(value='2024-01-01')

# Text input
mo.ui.text(value='Enter text')

# Checkbox
mo.ui.checkbox(label='Show outliers')

# Radio buttons
mo.ui.radio(options=['Option 1', 'Option 2'])
```


Use case: Build interactive dashboards without web frameworks!

Other Visualization Libraries

Altair (declarative grammar):

- Excellent marimo integration
- Grammar of Graphics approach
- Great for complex interactions
- More verbose than Plotly Express

Automated EDA Tools:

- `ydata-profiling` (formerly pandas-profiling)
- Generates comprehensive HTML report
- One line: `df.profile_report()`
- Great for quick exploratory analysis

We won't cover these in detail, but worth exploring!

Session 1 Summary

- ✓ **Principles**: Right chart for the question, clear labels, purposeful design
- ✓ **DataFrame plotting**: Pass df, specify columns by name
- ✓ **Plotly Express**: Interactive exploration, one-line charts
- ✓ **Seaborn**: Statistical plots, beautiful defaults
- ✓ **Marimo UI**: Reactive dashboards

Key Takeaways

1. **Visualizations communicate insights** - choose the right chart type
2. **DataFrame-based plotting** makes code clean and readable
3. **Plotly Express** for interactive exploration and dashboards
4. **Seaborn** for statistical analysis and publication-quality plots
5. **Both libraries** use similar syntax - easy to switch!

Next: Practice session - you'll create all these chart types!

Questions?

Break time! ☕

Then: Session 2 - Advanced Pandas & Funnel Analysis