Data Visualization — Topper-Style Notes

- **Aim**: Communicate insights clearly and convincingly.

Principles

- Clarity, minimalism, truthful scales, readable legends/labels.
- Use preattentive attributes (position, length, color) wisely.

Chart Selection

- Comparison: bar/line; Distribution: histogram/box; Part-to-whole: stacked; Relationship: scatter.
- Maps for spatial data; networks for relationships.

Question	Good Charts	Notes
Compare categories	Grouped/stacked bar	Show totals or share?
Distribution	Hist, KDE, box, violin	Outliers and skew
Relationship	Scatter, bubble	Correlation ≠ causation
Composition	Stacked bar/area	Avoid 3D pies
Time trend	Line, area	Uniform time steps

Python: Seaborn quick API

```
 import seaborn as sns, pandas as pd
sns.set_theme(style='whitegrid', context='talk')
# Distributions
sns.displot(df['x'], bins=30)
sns.kdeplot(df['x'])
# Relationships
sns.jointplot(data=df, x='x', y='y', kind='reg')
sns.pairplot(df.select_dtypes('number'))
# Categorical
sns.barplot(data=df, x='cat', y='val', estimator=np.median)
sns.boxplot(data=df, x='cat', y='val')
sns.violinplot(data=df, x='cat', y='val', split=True)
sns.stripplot(data=df, x='cat', y='val', jitter=True, dodge=True)
# Matrix
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='Blues')
# Grids
g = sns.FacetGrid(df, col='segment', row='region'); g.map_dataframe(sns.sca
```

RAWGraphs (no-code complex charts)

- **Purpose**: Open-source data visualization tool for complex, non-traditional charts.
- **Why**: Simplifies visualization of complex data for everyone; built on d3.js.
- **Workflow**: Paste data → choose chart → map fields to encodings → customize → export SVG/PNG.
- **Data Import**: Copy-paste from any spreadsheet application.
- **Chart Types**: Alluvial/Sankey diagrams, beeswarm plots, bump charts, circle packing, treemap, streamgraph, sunburst, voronoi diagrams, hexagonal binning.
- **Mapping Process**:
  - Rating → horizontal axis
  - Production budget → vertical axis
  - Box office → bubble areas
  - Genre → color coding
  - Movie names → labels
- **Export Options**: Vector (SVG) or raster images for further editing.
- **Advanced Features**: Fine-tuning with graphic editors post-export.
- **Use Cases**: Academic research, journalism, business presentations.
- **Pitfalls**: Limited interactivity; requires external editing for polish.
- **Checklist**: Validate data format; choose appropriate chart type; test readability.

Pipeline

- Prepare tidy data → choose chart → annotate → test with an audience.

Video takeaways

- Seaborn: single-line complex plots; grids for faceting; style contexts for presentation.
- RAWGraphs: field-to-encoding mapping workflow; export vector graphics for editing.

Checks and tips

- Pick palettes by data type (sequential/diverging/qualitative) and ensure colorblind safety.
- Use small multiples over overloaded legends when categories > ~4.
- Control overplotting with alpha/jitter/binning/hex; sample when needed.
- Explain heatmap scaling and clustering choices in captions.

Deep dive details

Palette choice:

Data type	Palette	Example
Sequential (0→high)	`viridis`, `magma`	temperatures
Diverging (−→+)	`RdBu`, `coolwarm`	z-scores
Qualitative (categories)	`tab10`, `Set2`	product lines

Overplotting tactics:

```
 sns.scatterplot(data=df, x='x', y='y', alpha=0.3)
# Binning
sns.histplot(data=df, x='x', y='y', bins=40, pmax=0.95, cbar=True)
# Hexbin
plt.hexbin(df['x'], df['y'], gridsize=40, cmap='viridis')
```

Facet vs stack guidance:

- Facet (small multiples) when categories are many or patterns differ.
- Stack when comparing parts of a whole over time, but watch readability.

Heatmap normalization:

```
 mat = df.pivot_table(index='row', columns='col', values='val', aggfunc='me
mat_norm = (mat - mat.min().min()) / (mat.max().max() - mat.min().min())
sns.heatmap(mat_norm, cmap='Blues')
```

Python plotting

- Purpose: fast EDA and publication-ready charts.
- Why: code-based, reproducible visuals.
- Core: pairplot/jointplot/FacetGrid, heatmaps with correlations/pivots, style/context control.
- Examples:

```
 sns.pairplot(df.select_dtypes('number'))
sns.jointplot(data=df, x='x', y='y', kind='hex')
g = sns.FacetGrid(df, col='segment'); g.map_dataframe(sns.lineplot, x='t',
```

- Pitfalls: default themes that obscure patterns; misleading color scales.
- Checklist: label axes/units; choose palette by data type; annotate takeaways.

Web/presentation tools

- Purpose: share interactive or animated visuals quickly.
- Why: low friction for non-coders.
- Core: template selection, data upload, simple transforms, export embeds/PDFs.
- Pitfalls: heavy embeds; inconsistent branding; privacy of uploaded data.
- Checklist: compress assets; consistent styles; verify sharing permissions.

Office animation

- Purpose: narrate processes and changes over time.
- Why: audiences grasp motion better than static deltas.
- Core: motion paths, timings, emphasis; export to video for distribution.
- Pitfalls: excessive motion; unreadable text during movement.
- Checklist: one movement per beat; large fonts; voiceover or captions.

Google ecosystem dashboards

- Purpose: light dashboards and quick prototypes.
- Why: easy embedding; broad familiarity.
- Core: connectors, calculated fields, filters; watch quotas.
- Pitfalls: stale caches; quota overruns; slow queries.
- Checklist: pre-aggregate; cache; keep views simple.

Data Storytelling Fundamentals

- **Purpose**: Make insights memorable and actionable through narrative.
- **Why**: Decisions follow stories; data alone doesn't drive action.
- **Four Storytelling Methods**:
    1. **Numbers**: Raw data with conditional formatting (correlation matrices)
    2. **Visuals**: Charts that tell stories (declining COVID cases over time)
    3. **Text**: Plain language reports explaining performance vs targets
    4. **Illustrations**: Comic-style narratives (Tonga obesity crisis)
- **Integration Approach**: Combine all methods for maximum impact.
- **Core Elements**: Headline + supporting visual, before/after contrasts, annotations.
- **Communication Channels**: Text and illustrations as strong support to numbers/visuals.
- **Future Trend**: Bulk of data communication moving toward integrated storytelling.
- **Applications**: Business reports, academic presentations, journalism, policy communication.
- **Pitfalls**: Burying the lede; chart junk; unclear labels; single-method limitation.
- **Checklist**: One message per visual; annotate key insights; clarify uncertainty; combine methods strategically.

Actor Network Analysis

- **Purpose**: Analyze social networks and find connection patterns (e.g., shortest path between actors).

- **Why**: Understand collaboration patterns, influence networks, degrees of separation.
- **Data Sources**: IMDb datasets, social media connections, collaboration records.
- **Core Process**:
    - **Data Acquisition**: Download large TSV files (IMDb non-commercial datasets)
    - **Data Filtering**: Focus on specific categories (actors/actresses), set thresholds
    - **Actor Pairing**: Identify significant collaborations (minimum films/co-appearances)
    - **Network Creation**: Build graph from collaboration data
- **Technical Implementation**:

```
 # Libraries: networkx, scikit-network
import networkx as nx
G = nx.Graph()
G.add_edges_from(actor_pairs)
shortest_path = nx.shortest_path(G, 'Govinda', 'Angelina Jolie')
```

- **Analysis Techniques**:
    - Shortest path algorithms
    - Centrality metrics (betweenness, closeness, degree)
    - Community detection
    - Clustering coefficients
- **Data Challenges**: Name variations, data inconsistencies, large dataset management.
- **Applications**: Six degrees of separation, influence mapping, collaboration analysis.
- **Pitfalls**: Hairball graphs; unlabeled nodes; hiding degree distribution; data quality issues.
- **Checklist**: Filter meaningfully; validate data consistency; explain network metrics; document methodology.

HTML Presentations (RevealJS)

- Purpose: interactive web-based slideshows for technical content.

- Why: live demos, code highlighting, math equations, remote-friendly.
- Core features:
  - Markdown support for rapid content creation
  - Interactive charts and live data integration
  - Fragment animations for progressive disclosure
  - Speaker notes and presenter mode
- Examples:

```
 <!-- Code with line highlighting -->
<pre><code class="python" data-line-numbers="1-2|4,6-7">
import pandas as pd
data = pd.read_csv('data.csv')
sns.scatterplot(data=data, x='x', y='y')
</code></pre>

<!-- Interactive chart integration -->
<section>
  <div id="chart"></div>
  <script>
    Reveal.addEventListener('slidechanged', function(event) {
      if (event.currentSlide.id === 'chart-slide') {
        updateChart(data);
      }
    });
  </script>
</section>
```

- Advanced: MathJax equations, Mermaid diagrams, D3 integration.
- Pitfalls: performance with heavy visualizations; responsive design challenges.
- Checklist: test on multiple devices; optimize loading; provide fallbacks.

Interactive Notebooks (Marimo)

- Purpose: reactive notebooks that update automatically like spreadsheets.
- Why: reproducible, debuggable, interactive web apps.
- Key differences from Jupyter:
  - Cells can’t run out of order (enforced dependency graph)
  - Automatic reactivity when cells change
  - Native Python files (version control friendly)
- Core operations:

```
 # Interactive widgets
slider = mo.ui.slider(1, 100)
mo.md(f"Value: {slider.value} {'🟢' * slider.value}")

# Reactive updates
filtered_data = data[data['value'] > slider.value]
mo.ui.table(filtered_data)
```

- Applications: data exploration, interactive reports, web app prototypes.
- Pitfalls: mental shift from Jupyter workflow; learning reactive paradigm.
- Checklist: keep cells atomic; document dependencies; test reactivity.

Excel Time-Series Visualization

- Purpose: explore financial/temporal data with built-in tools.
- Why: accessible, collaborative, powerful statistical functions.
- **Sparklines**: In-cell trend visualization
  - Line sparklines for trends over time
  - Column sparklines for period comparisons
  - Win/Loss sparklines for binary outcomes
- **Advanced Analysis**:

```
 # Forecasting with GROWTH function
=GROWTH(known_y_values, known_x_values, new_x_value)

# Correlation matrix with Data Analysis ToolPak
Data > Analysis > Correlation

# Volatility measurement
=STDEV(data_range) / AVERAGE(data_range)  # Coefficient of variation
```

- **Correlation Analysis**: Scatter plots with R-squared, correlation matrices.
- Applications: financial analysis, trend identification, relationship discovery.
- Pitfalls: manual effort for complex analysis; limited statistical functions.
- Checklist: validate formulas; use conditional formatting; document assumptions.

Animated Data Visualization

- Purpose: show change over time and engage audiences.
- Why: motion reveals patterns better than static comparisons.
- **PowerPoint Approach**:
  - Morph transition for smooth animations
  - Manual bar creation with precise sizing
  - Voiceover synchronization with slide timing
  - Export to video format
- **Flourish Approach**:
  - Template-based race charts (bar, line)
  - Morphing between chart types
  - Scatter plot animations with stagger effects
  - Object constancy for data storytelling
- Examples:

```
# PowerPoint: Size bars by data values
Shape Format > Width: 6.5" (for $6.5B value)

# Flourish: Animation controls
- Duration: Timeline speed
- Transition: Between-state smoothing
- Stagger: Cascading element animations
```

- Applications: data stories, social media content, presentation engagement.
- Pitfalls: excessive motion; unclear during transitions; accessibility issues.
- Checklist: one movement per message; readable text; provide static alternatives.

Network Visualization (Kumu)

- Purpose: visualize complex relationship networks and communities.
- Why: understand social structures, collaboration patterns, influence flows.
- **Data Preparation**:
  - Actor collaboration matrices via matrix multiplication
  - Sparse matrix optimization (CSR format)
  - “From-to” node format with connection strength
- **Analysis Features**:
  - Community detection and clustering
  - Direct/indirect connection exploration

  - Filtering by attributes (year, region, frequency)
- **Visualization Capabilities**:

```
# Create collaboration matrix
actor_matrix = movie_actor_matrix @ movie_actor_matrix.T

# Convert to network format
connections = [(actor1, actor2, strength)
               for actor1, actor2, strength in sparse_matrix_entries
               if strength > threshold]
```

- Applications: social network analysis, collaboration mapping, influence identification.
- Pitfalls: overwhelming complexity; sparse data interpretation; scalability limits.
- Checklist: filter meaningfully; explain network metrics; highlight key insights.

Data Visualization with ChatGPT

- **Purpose**: Leverage LLMs for end-to-end visualization creation and enhancement.
- **Why**: Accelerates dataset discovery, analysis, and visualization creation.
- **Prerequisites**: ChatGPT Plus ($20/month), Gemini (free), GitHub account, basic HTML/CSS/JS.
- **Optional Tools**: Claude ($17/month), command-line tools (Claude Code, Gemini CLI).
- **Dataset Discovery Process**:

```
Prompt: "I need an interesting dataset for data visualization that:
- Has 10,000-100,000 rows
- Includes various column types (text, numbers, categories)
- Could tell an engaging story for a general audience
- Ideally covers [your preferred theme/domain]"
```

- **Story Ideation**: Request dozen potential stories with target audiences and analysis approaches.
- **Analysis Instructions**: Statistical tests, significance filtering, aesthetic considerations, outlier handling.

- **Web Deployment**: GitHub Pages optimization, 2MB payload limits, modern JavaScript practices.
- **Design Enhancement**: Professional typography, color schemes, NYT-style layouts, proper annotations.
- **Best Practices**:
  - Iterate with LLM feedback
  - Be specific about requirements
  - Consider performance constraints
  - Request maintainable, accessible code
  - Include comprehensive documentation
- **Example Projects**: Books visualization, coffee reviews, LLM data exploration.
- **Pitfalls**: First-try expectations; vague requirements; ignoring technical constraints.
- **Checklist**: Define clear requirements; iterate on outputs; test across devices; document thoroughly.

Data Storytelling with LLMs

- **Purpose**: Complete data-to-story pipeline using AI assistance.
- **Why**: LLMs excel at each stage: engineering, analysis, visualization.
- **Three-Stage Process**:
  1. **Data Engineering**: Web scraping, data cleaning, format standardization
  2. **Data Analysis**: Pattern discovery, topic modeling, statistical analysis
  3. **Data Visualization**: Chart creation, narrative development, presentation
- **Web Scraping with LLMs**:

```
 // Example: WhatsApp message extraction
// LLM writes DevTools console code for:
// 1. HTML copying with length management
// 2. Clipboard integration
// 3. JSON parsing with message details
```

- **Data Cleaning Approach**: Identify issues, suggest strategies, write cleaning code, handle edge cases.

- **Topic Modeling Pipeline**:
  - Calculate text embeddings
  - K-means clustering
  - GPT-powered cluster naming
  - Label integration
- **Analysis Strategy**: Request 10 diverse angles (obvious + quirky), code generation, result interpretation.
- **Key Principles**:
  - **Delegate Generously**: Try LLMs on complex tasks
  - **Write Code**: More reliable than direct processing
  - **Start Fresh**: Often easier than fixing broken attempts
  - **Track Impossibilities**: Monitor capability evolution
- **Workflow Example**: Collection → Cleaning → Analysis → Visualization → Storytelling.
- **Advanced Tips**: Multiple options, specific requirements, quality iteration, graceful failure handling.
- **Applications**: Social media analysis, news monitoring, business intelligence, research projects.
- **Pitfalls**: Over-reliance on first attempts; unclear specifications; ignoring technical limitations.
- **Checklist**: Break complex tasks down; specify target audience; iterate on quality; handle failures gracefully.