Programming Principles in Python (CSCI 503)

Visualization

Dr. David Koop
Exploring Data through Visualization
Exploring Data through Visualization
Why do we visualize data?

Figures are richer; provide more information with less clutter and in less space. Figures provide the gestalt effect: they give an overview; make structure more visible.

Figures are more accessible, easier to understand, faster to grasp, more comprehensible, more memorable, more fun, and less formal.

List adapted from: [Stasko et al. 1998]
Why Visual?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>10.0</td>
<td>8.04</td>
<td>10.0</td>
<td>9.14</td>
<td>10.0</td>
</tr>
<tr>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>8.14</td>
<td>8.0</td>
</tr>
<tr>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>8.74</td>
<td>13.0</td>
</tr>
<tr>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>8.77</td>
<td>9.0</td>
</tr>
<tr>
<td>11.0</td>
<td>8.33</td>
<td>11.0</td>
<td>9.26</td>
<td>11.0</td>
</tr>
<tr>
<td>14.0</td>
<td>9.96</td>
<td>14.0</td>
<td>8.10</td>
<td>14.0</td>
</tr>
<tr>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.13</td>
<td>6.0</td>
</tr>
<tr>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>3.10</td>
<td>4.0</td>
</tr>
<tr>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>9.13</td>
<td>12.0</td>
</tr>
<tr>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>7.26</td>
<td>7.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>4.74</td>
<td>5.0</td>
</tr>
</tbody>
</table>

[F. J. Anscombe]
Why Visual?

<table>
<thead>
<tr>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>10.0</td>
<td>8.04</td>
<td>10.0</td>
<td>9.14</td>
</tr>
<tr>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>8.14</td>
</tr>
<tr>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>8.74</td>
</tr>
<tr>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>8.77</td>
</tr>
<tr>
<td>11.0</td>
<td>8.33</td>
<td>11.0</td>
<td>9.26</td>
</tr>
<tr>
<td>14.0</td>
<td>9.96</td>
<td>14.0</td>
<td>8.10</td>
</tr>
<tr>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.13</td>
</tr>
<tr>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>3.10</td>
</tr>
<tr>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>9.13</td>
</tr>
<tr>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>7.26</td>
</tr>
<tr>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>4.74</td>
</tr>
</tbody>
</table>

Mean of x: 9  
Variance of x: 11  
Mean of y: 7.50  
Variance of y: 4.122  
Correlation: 0.816

[F. J. Anscombe]
Why Visual?

[F. J. Anscombe]
Why Visual?

**Mean of \( x \):** 9  
**Variance of \( x \):** 11  
**Mean of \( y \):** 7.50  
**Variance of \( y \):** 4.122  
**Correlation:** 0.816

— [F. J. Anscombe]
Visualization Goals

- "The purpose of visualization is insight, not pictures" – B. Schneiderman
- Identify patterns, trends
- Spot outliers
- Find similarities, correlation
The Python Visualization Landscape

- **javascript**
  - cufflinks
  - plotly
  - toyplot
  - holoviews
  - datashader
  - bokeh
  - bqplot
  - Vaex
  - mpld3
  - d3js
  - d3po
  - vega-lite
  - vega
  - altair
  - yt
  - matplotlib
  - basemap
  - networkx
  - yellowbrick
  - scikit-plot
  - ggpy
  - plotnine
  - glueviz
  - seaborn
  - pandas

- **OpenGL**
  - vispy
  - glumpy
  - visvis
  - galry
  - mayavi
  - pyglet
  - GR Framework

- **Lighting**
  - ipyvolume
  - ipyleaflet
  - pythreejs

- **Matplotlib**
  - d3po
  - vega
  - vega-lite
  - matplotlib
  - spitfire
  - matplotlib

- **Visualization Tools**
  - ipyparaview
  - pydeck

- **Other Tools**
  - glut
  - mayavi
  - galry
  - plotnine
  - glueviz
  - seaborn
  - pandas
  - matplotlib

- **Dependents**
  - pygal
  - chaco
  - PyQtGraph

D. Koop, CSCI 503, Spring 2021
matplotlib

• Strengths:
  - Designed like Matlab
  - Many rendering backends
  - Can reproduce almost any plot
  - Proven, well-tested

• Weaknesses:
  - API is imperative
  - Not originally designed for the web
  - Dated styles
Basic Example

- `import matplotlib.pyplot as plt
  plt.plot([1,5,2,7,3])`

- Default is line plot

- x-values are implicit (`range(5)`)  

- Can add x-values
  - `plt.plot([1,3,4,6,10],[1,5,2,7,3])`

- Can change type of plot
  - `plt.scatter([1,3,4,6,10],[1,5,2,7,3])`
  - `plt.plot([1,3,4,6,10],[1,5,2,7,3],'o')` # format string
Data is Encoded via Visual Channels

- **Position**
  - Horizontal
  - Vertical
  - Both

- **Color**

- **Shape**

- **Tilt**

- **Size**
  - Length
  - Area
  - Volume

[Munzner (ill. Maguire), 2014]
Many different types of charts
Anatomy of a Figure
Object-Based Plotting

• `fig, ax = plt.subplots()` # "constructor-like" method  
  `ax.scatter([1,3,4,6,10],[1,5,2,7,3])`

• Use getters/setters for labels and title
  - `ax.set_xlabel('Age')`
  - `ax.set_ylabel('Number of Jumps')`
  - `ax.set_title('Kangaroo Jumps Today')`

• We can also call methods on the figure:
  - `fig.tight_layout()` # reduce margins
pandas Integration

• Can call many of these methods directly from pandas
• Handled through kind kwarg or .plot accessor
• It will try to guess a reasonable visualization, but may fail:
  - fruit.plot()

• Instead, specify x and y and other parameters:
  - fruit.plot(kind='bar', x='name', y='price')
  - plt.bar(x='name', height='price', data=fruit)  # SIMILAR
  - fruit.plot.scatter(x='price', y='count', c='name')  # ERROR
  - colors = {'Apple': 'red', 'Orange': 'orange',
              'Banana': 'yellow', 'Pear': 'green'}
    fruit.plot.scatter(x='price', y='count',
                       c=fruit['name'].map(colors))
Assignment 8

• Back to Pokémon Data
• Calculate MaxCP in pandas and find highest per generation
• Analyze attack, defense, and speed by primary type and generation using visualizations created with matplotlib and altair
Final Exam

• Monday, April 26, 2:00-3:50pm, Online (Blackboard)
• **More** comprehensive than Test 2
• Expect questions from topics covered on Test 1 and 2
• Expect questions from the last three weeks of class (data, visualization, machine learning)
• Similar format
Test 2 Comments
Teaching Evaluations

- Online
- Check your email for instructions
History of Vega-Lite & Altair

- "Grammar of Graphics", L. Wilkinson
- "A Layered Grammar of Graphics", H. Wickham
- ggplot: plotting library for R
- Vega: similar idea for Javascript/JSON (U. Washington, A. Satyanarayan)
  - "Declarative language for creating, saving, and sharing interactive visualization designs"
  - More focus on interaction and reactive signals
  - Separation between specification and runtime
- Vega-Lite: higher-level language than Vega (U. Washington, D. Moritz)
  - uses carefully designed rules to default settings
History of Vega-Lite & Altair

- Altair: Python interface to Vega-Lite (J. VanderPlas)
  - "spend more time understanding your data and its meaning"
  - Specify the what, minimize the amount of code directing the how
  - Python can write JSON specification just as well as any other language
  - Bindings make it more Python-friendly, integrate with pandas, add support for Jupyter, etc.
Basic Example

- import altair as alt
  import pandas as pd
  data = pd.DataFrame({'x': [1, 3, 4, 6, 10], 'y': [1, 5, 2, 7, 3]})
  alt.Chart(data).mark_line().encode(x='x', y='y')

- Easiest to use data from a pandas data frame
  - Another option is a csv or json file
  - Can support geo_interface, too

- Chart is the basic unit
- mark_*() indicates the geometry created for each data item
- encode allows visual properties to be set to data attributes
Visual Marks

• **Marks** are the basic graphical elements in a visualization
• Marks classified by dimensionality:
  - Points
  - Lines
  - Areas
• Also can have surfaces, volumes
• Think of marks as a mathematical definition, or if familiar with tools like Adobe Illustrator or Inkscape, the path & point definitions
• Altair: area, bar, circle, geoshape, image, line, point, rect, rule, square, text, tick
  - Also compound marks: boxplot, errorband, errorbar
Encode via Visual Channels

- **Position**
  - Horizontal
  - Vertical
  - Both

- **Color**

- **Shape**

- **Tilt**

- **Size**
  - Length
  - Area
  - Volume

[Munzner (ill. Maguire), 2014]
Easily Explore Different Encodings

- `data = pd.DataFrame({
  'age': [1, 3, 4, 6, 10],
  'weight': [20, 50, 25, 55, 125],
  'zoo_area': [1, 3, 3, 1, 2],
  'num_scoops': [3, 2, 4, 2, 3]
})
alt.Chart(data).mark_point(
    filled=True, size=50,
    stroke='black', strokeWidth=1
).encode(
    x='age',
    y='weight',
    color='zoo_area'
)
Problem: zoo_area is not a continuous value, nor is it ordered in any way!
Data Attributes and Altair Types

- Categorical
- Ordered

- Ordinal
- Quantitative

[Muñzner (ill. Maguire), 2014]
Data Attributes and Altair Types

- Categorical data = Nominal (N)
- Ordinal data = Ordinal (O)
- Quantitative data = Quantitative (Q)
- Temporal data = Temporal (T)

[Munzner (ill. Maguire), 2014]
Specifying the Type

zoo_area: 0

zoo_area: N
Different Channels for Different Attribute Types

**Magnitude Channels: Ordered Attributes**
- Position on common scale
- Position on unaligned scale
- Length (1D size)
- Tilt/angle
- Area (2D size)
- Depth (3D position)
- Color luminance
- Color saturation
- Curvature
- Volume (3D size)

**Identity Channels: Categorical Attributes**
- Spatial region
- Color hue
- Motion
- Shape

Altair will use its rules to pick whether to use color hue or saturation based on the type

[Munzner (ill. Maguire), 2014]
Multiple Views in Visualization

The image shows a line graph depicting the stock prices of different companies over a period from 2000 to 2010. The x-axis represents the date, while the y-axis represents the price. The graph includes lines for companies such as AAPL, AMZN, GOOG, IBM, and MSFT, each represented by a different color and line style.
Multiple Views in Visualization

[Improvise, Weaver, 2004]
Multiple Views in Visualization

[Diagram showing scatter plots of sepal length vs sepal width, and petal length vs petal width in various colors and scales.]

[M. Bostock]
Altair Supports Concatenation, Layering, & Repetition

• Layering:
  - + Operator

• Concatenation:
  - Horizontal: | operator
  - Vertical: & operator

• Repetition
  - Use of .repeat for layout
  - Reference repeated variables in the encoding
[Rock 'N' Roll is Here to Pay, R. Garofalo, 1977 (via Tufte)]
Also Visualization, but with Interaction
Interaction

- Grammar of Graphics, why not Grammar of Interaction?
- Vega-Lite/Altair is about interactive graphics
- Types of Interactions:
  - Selection
  - Zoom
  - Brushing
Selection

• Selection is often used to initiate other changes
• User needs to select something to drive the next change
• What can be a selection target?
  - Items, links, attributes, (views)
• How?
  - mouse click, mouse hover, touch
  - keyboard modifiers, right/left mouse click, force
• Selection modes:
  - Single, multiple
  - Contiguous?
Highlighting

• Selection is the user action
• Feedback is important!
• How? Change selected item's visual encoding
  - Change color: want to achieve visual popout
  - Add outline mark: allows original color to be preserved
  - Change size (line width)
  - Add motion: marching ants
Highlighting

• Selection is the user action
• Feedback is important!
• How? Change selected item's visual encoding
  - Change color: want to achieve visual popout
  - Add outline mark: allows original color to be preserved
  - Change size (line width)
  - Add motion: marching ants
Interaction Overview

- **Change over Time**
- **Navigate**
  - **Item Reduction**
    - **Zoom**
    - **Geometric or Semantic**
  - **Pan/Translate**
  - **Constrained**
- **Attribute Reduction**
  - **Slice**
  - **Cut**
  - **Project**

[Munzner (ill. Maguire), 2014]
Altair's Interactive Charts