✏️

Chapter 3: Quantitative comparisons and statistical visualizations

Created	@May 7, 2024 5:00 PM
Class	Introduction to Data Visualization with Matplotlib

Creating Bar Charts

Shows the quantitative comparision between parts of the data

Olympic Medals Dataset

- Number of medals won by countries in 2016 Olympics

Loading Data

```
medals = pd.read_csv('medals.csv', index_col=0)
```

- Load data into DataFrame with country names as index

Bar Chart for Gold Medals

```
fig, ax = plt.subplots()
ax.bar(medals.index, medals['Gold'])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel('Number of medals')
```

- `ax.bar()` creates bar for each row
- Rotate x-tick labels to avoid overlap
- Label y-axis

Stacked Bar Chart

```
fig, ax = plt.subplots()
ax.bar(medals.index, medals['Gold'], label='Gold')
ax.bar(medals.index, medals['Silver'], bottom=medals['Gold'],
label='Silver')
ax.bar(medals.index, medals['Bronze'], bottom=medals['Gold']+
medals['Silver'], label='Bronze')
ax.legend()
```

- Stack bars by setting `bottom` to previous bars
- Use `label` to identify bars in legend
- Call `ax.legend()` to display legend

Improving Visualization

- Use colors to distinguish bar types
- Add title, adjust spacing/sizing as needed

Histograms

Shows us entire distribution/ full distribution of values with a variable

Data

- Heights of medal winners in men's rowing and gymnastics (2016 Olympics)

Simple Bar Chart

```
fig, ax = plt.subplots()
ax.bar('Rowing', rowers['Height'].mean())
ax.bar('Gymnastics', gymnasts['Height'].mean())
ax.set_ylabel('Height (cm)')
```

- Bar chart shows mean heights
- Lacks distribution details

Histogram

```
fig, ax = plt.subplots()
ax.hist(rowers['Height'], label='Rowing', bins=10)
ax.hist(gymnasts['Height'], label='Gymnastics', bins=10)
ax.legend()
```

- `ax.hist()` plots histogram
- Bins split data into bars
- Default is 10 bins

Histogram Options

```
ax.hist(rowers['Height'], bins=20)
```

- `bins` sets number of bins

```
bins = [150, 160, 170, 180, 190, 200]
ax.hist(rowers['Height'], bins=bins)
```

- Bins can be a sequence to specify boundaries

Histogram Types

```
ax.hist(rowers['Height'], histtype='step')
ax.hist(gymnasts['Height'], histtype='step')
```

- `histtype='step'` uses lines instead of bars
- Prevents occlusion between histograms

Statistical Visualization Techniques

Key Details

- Using visualizations to make quantitative comparisons between data
- Two main techniques covered: error bars and box plots

Error Bars

- Additional markers showing distribution summary (e.g. standard deviation)

Bar Chart with Error Bars

```
ax.bar(x, heights.mean(), yerr=heights.std())
```

- `yerr` specifies data for error bar size (std dev)

Line Plot with Error Bars

```
ax.errorbar(x, y, yerr=y_err)
```

- `ax.errorbar()` plots line with error bars
- `yerr` specifies error bar data

Box Plots

- Visualization showing key landmarks of distributions
- Invented by John Tukey

```
ax.boxplot([data1, data2], labels=['Label1', 'Label2'])
# for labels
# ax.set_xticklabels=(['Label1', 'Label2'])
ax.set_ylabel('Value')
```

Box Plot Elements

- Red line = median
- Box = inter-quartile range (25th-75th percentile)
- Whiskers = 1.5 x inter-quartile range
- Encompassess roughly 99% of the distribution if the data is Gaussian or Normal
- Points beyond whiskers = outliers

Key Takeaways

- Error bars summarize distributions
- Box plots show full distribution characteristics visually
- Useful for quantitative comparisons between datasets

Scatter Plots for Bivariate Comparisons

Bivariate Comparison: Comparision fo two different variables

Key Details

- Scatter plots visualize relationship between two variables
- Each point represents an observation's values for both variables

Example: Climate Data

- Variables: CO2 levels and relative temperatures

```
fig, ax = plt.subplots()
ax.scatter(data['co2'], data['relative_temp'])
ax.set_xlabel('CO2 (ppm)')
ax.set_ylabel('Relative Temperature')
plt.show()
```

Customizing Scatter Plots

Multiple Datasets

- Use different colors/labels to distinguish datasets
- Can add legend to explain colors

```
fig, ax = plt.subplots()
ax.scatter(eighties['co2'], eighties['relative_temp'], color
='red', label='1980s')
ax.scatter(nineties['co2'], nineties['relative_temp'], color
='blue', label='1990s')
ax.legend()
```

Encoding Third Variable

- Use `c` parameter to encode a third variable with color

```
ax.scatter(data['co2'], data['relative_temp'], c=data.index)
```

- Here color represents time variable
- The variable is encoded as the brightness of the color

Key Takeaways

- Scatter plots show bivariate relationships
- Can plot multiple datasets together

- Color can distinguish datasets or encode third variable
- Useful for exploring relationships between variables