Lab Exercise: Visual Exploration of Shape Features (Self-Contained)

Course: Data Visualization & Exploratory Analysis
Duration: 2 hours (lab) + 10 minutes (presentation)

Abstract

This lab is self-contained: students receive the PDF only. The PDF contains a data-generation script, full runnable code snippets to produce each required figure (histograms, heatmaps, rankings, trends), and step-by-step instructions so no external notebooks or data are required.

Prerequisites

Python (3.8+), pandas, seaborn, matplotlib, plotly, scikit-learn, ipywidgets

What to expect in this handout

- A small synthetic dataset generator (Python) to create `feature_vectors/*.csv` locally.
- Ready-to-run plotting scripts for: histograms, interactive comparison, correlation heatmaps, pairplots, clustered heatmap, ranking charts (including diverging and stacked), and trend analyses.
- AI prompts (for instructors) used to create the synthetic data.
- Deliverables, tasks, and graded questions.

1 Quick setup (5 minutes)

1. Create and activate your Python environment, then install required packages:

```
python -m pip install --upgrade pip
pip install pandas seaborn matplotlib plotly scikit-learn ipywidgets
```

2. Save the data-generator script below as `generate_synthetic_feature_vectors.py` and run it from the project root:

```
python generate_synthetic_feature_vectors.py
```

3. This will create a `feature_vectors/` folder with 5 CSV files (club, dining, lamp, mug, table). All remaining code snippets assume that folder exists in the current working directory.

2 Data generation script (save as `generate_synthetic_feature_vectors.py`)

```
"""
Creates synthetic feature CSVs for five categories: club, dining, lamp, mug, table.
Each CSV contains numeric features used in this lab and two metadata columns: source_file
    , category.
```

```
"""
import numpy as np
import pandas as pd
from pathlib import Path

np.random.seed(0)
out = Path('feature_vectors')
out.mkdir(exist_ok=True)

def make_cat(cat, n=200):
  # base ranges altered per category to create separability
  if cat == 'table':
    sa = np.random.normal(500, 80, n).clip(50, 2000)
    vol = np.random.normal(200, 40, n).clip(5, 1000)
  elif cat == 'mug':
    sa = np.random.normal(120, 30, n).clip(10, 800)
    vol = np.random.normal(45, 15, n).clip(1, 400)
  elif cat == 'lamp':
    sa = np.random.normal(180, 60, n).clip(10, 1200)
    vol = np.random.normal(60, 25, n).clip(1, 500)
  elif cat == 'club':
    sa = np.random.normal(90, 25, n).clip(10, 600)
    vol = np.random.normal(35, 12, n).clip(1, 300)
  else:  # dining
    sa = np.random.normal(300, 70, n).clip(20, 1500)
    vol = np.random.normal(120, 30, n).clip(1, 800)

  mean_curv = np.random.normal(25, 7, n).clip(0, 100) * (1 + (np.log1p(sa)/10 - 1))
  median_curv = mean_curv * (0.9 + 0.2*np.random.rand(n))
  silhouette_complexity = np.random.normal(10, 5, n).clip(0, 50)
  skeleton_complexity = np.random.normal(8, 4, n).clip(0, 50)
  aspect_ratio_y = np.random.normal(1.0 + (vol/500), 0.15, n).clip(0.2, 10)
  hollow_ratio = np.random.beta(2,5, n) * (1 + sa/2000)
  surface_to_volume_ratio = sa / (vol + 1e-6)

  df = pd.DataFrame({
    'mean_curvature': mean_curv,
    'median_curvature': median_curv,
    'surface_area': sa,
    'volume': vol,
    'silhouette_complexity': silhouette_complexity,
    'skeleton_complexity': skeleton_complexity,
    'aspect_ratio_y': aspect_ratio_y,
    'hollow_ratio': hollow_ratio,
    'surface_to_volume_ratio': surface_to_volume_ratio,
  })
  df['category'] = cat
  df['source_file'] = [f"{cat}_{i:04d}.obj" for i in range(len(df))]
  return df

cats = ['club','dining','lamp','mug','table']
for c in cats:
  df = make_cat(c, n=200)
  df.to_csv(out / f"{c}_features.csv", index=False)
print('Synthetic data written to', out)
```

How to run the analysis

Copy-paste these snippets into a .py file or run in Jupyter:

3 SECTION 1 — Histograms and basic stats

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pathlib import Path

root = Path('.')
fv_dir = root / 'feature_vectors'
csvs = sorted(fv_dir.glob('*_features.csv'))
dfs = [pd.read_csv(p).assign(category=p.stem.split('_')[0]) for p in csvs]
all_df = pd.concat(dfs, ignore_index=True)

def find_col(df, target):
  t = target.lower()
  for c in df.columns:
    if t in c.lower().replace('_',' '):
      return c
  return None

col = find_col(all_df, 'Surface-to-Volume Ratio')
vals = pd.to_numeric(all_df[all_df['category']=='mug'][col], errors='coerce').dropna()
plt.figure(figsize=(8,4))
sns.histplot(vals, bins=30, kde=True)
plt.title('Surface-to-Volume Ratio — mugs')
plt.show()
print('n =', len(vals))
print('mean =', vals.mean(), 'std =', vals.std())
```

4 SECTION 2 — Interactive comparison (overlay or violin/box)

```
# Overlay two categories
c1, c2 = 'mug','table'
f = 'surface_to_volume_ratio'
v1 = pd.to_numeric(all_df[all_df['category']==c1][f], errors='coerce').dropna()
v2 = pd.to_numeric(all_df[all_df['category']==c2][f], errors='coerce').dropna()
plt.figure(figsize=(8,4))
sns.histplot(v1, color='tab:blue', alpha=0.5, bins=30, label=c1)
sns.histplot(v2, color='tab:orange', alpha=0.5, bins=30, label=c2)
plt.legend()
plt.title(f'Overlay: {f} — {c1} vs {c2}')
plt.show()

# Normalized (z-score)
import numpy as np
combined = pd.concat([v1, v2])
mu, sigma = combined.mean(), combined.std()
v1n = (v1 - mu)/sigma
v2n = (v2 - mu)/sigma
plt.figure(figsize=(8,4))
sns.histplot(v1n, color='tab:blue', alpha=0.5, bins=30, label=c1)
sns.histplot(v2n, color='tab:orange', alpha=0.5, bins=30, label=c2)
plt.legend()
plt.title('Normalized overlay (z-score)')
plt.show()
```

5 SECTION 3 — Heatmaps and pairplot

```
import numpy as np
num_cols = all_df.select_dtypes(include='number').columns.tolist()
cols = num_cols[:20]
```

```
corr = all_df[cols].corr()
plt.figure(figsize=(10,8))
mask = np.triu(np.ones_like(corr, dtype=bool))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, annot=True, fmt='.2f')
plt.title('Correlation heatmap')
plt.show()

# Pairplot (choose -46 features)
top = all_df[cols].var().sort_values(ascending=False).index[:6].tolist()
sns.pairplot(all_df[top + ['category']].dropna(), hue='category', diag_kind='kde', corner
    =True)
plt.show()
```

6 SECTION 4 — Rankings (bar charts, error bars, stacked, diverging, lollipop)

```
rank_df = pd.DataFrame({'feature': cols, 'importance': np.linspace(1, len(cols), len(cols
    ))[::-1]})
# classic horizontal bar
top = rank_df.sort_values('importance', ascending=True).tail(15)
plt.figure(figsize=(8,6))
plt.barh(top['feature'], top['importance'], color='tab:blue')
plt.title('Top features — horizontal ranking (synthetic)')
plt.show()

# error bars using means/std across shapes
top_feats = rank_df['feature'].tolist()[:10]
means = all_df[top_feats].mean()
stds = all_df[top_feats].std()
order = means.sort_values(ascending=True).index
plt.figure(figsize=(8,6))
plt.barh(order, means[order], xerr=stds[order], color='tab:green')
plt.title('Means ± std (top features)')
plt.show()

# stacked bar: mean per category for a small set
feats = top_feats[:6]
pivot = all_df.groupby('category')[feats].mean()
pivot.plot(kind='bar', stacked=True)
plt.title('Stacked: mean per category')
plt.show()

# diverging: difference between two categories
c1, c2 = sorted(all_df['category'].unique())[:2]
mean1 = all_df[all_df['category']==c1][feats].mean()
mean2 = all_df[all_df['category']==c2][feats].mean()
diff = (mean1 - mean2).sort_values()
plt.figure(figsize=(8,4))
colors = ['tab:blue' if v>0 else 'tab:orange' for v in diff.values]
plt.barh(diff.index, diff.values, color=colors)
plt.axvline(0, color='k', linewidth=0.6)
plt.title(f'Diverging: {c1} - {c2}')
plt.show()

# lollipop
dfp = rank_df.sort_values('importance', ascending=True).tail(15)
plt.figure(figsize=(8,6))
plt.hlines(y=dfp['feature'], xmin=0, xmax=dfp['importance'], color='gray')
plt.plot(dfp['importance'], dfp['feature'], 'o', color='tab:red')
plt.title('Lollipop chart')
plt.show()
```

7 SECTION 5 — Trends (rolling mean, binned, quantiles)

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# compute a proxy target (if no real score available)
num = all_df.select_dtypes(include='number').drop(columns=['source_file'], errors='ignore')
drop_ids = [c for c in num.columns if 'id' in c.lower() or c.lower().startswith('shape')]
num = num.drop(columns=drop_ids, errors='ignore')
X = StandardScaler().fit_transform(num.fillna(num.median()))
pc1 = PCA(n_components=1, random_state=0).fit_transform(X).ravel()
all_df['aesthetic_proxy'] = pc1

# rolling trend example
f = 'mean_curvature'
target = 'aesthetic_proxy'
df = all_df[[f, target]].dropna().sort_values(f)
df['y_roll'] = df[target].rolling(window=max(5, int(len(df)*0.12)), center=True).mean()
plt.figure(figsize=(8,4))
plt.scatter(df[f], df[target], s=10, alpha=0.3)
plt.plot(df[f], df['y_roll'], color='tab:red')
plt.title('Scatter + rolling mean')
plt.show()
```

8 Deliverables (submit PDF)

- A short summary (1 paragraph) of the lab and methods used.
- Three figures (a histogram, a heatmap/pairplot, and a ranking chart) with concise captions and interpretation (2–3 sentences each).
- Answer: ”Which single feature best separates mugs from tables?” Provide a figure and justification.

9 Tips and grading

- Figures: 40%
- Written answers: 40%
- Clean runnable scripts/notebook: 15%
- Extra credit: 5%

10 Academic integrity

Work in pairs at most. Cite any AI or external code used.