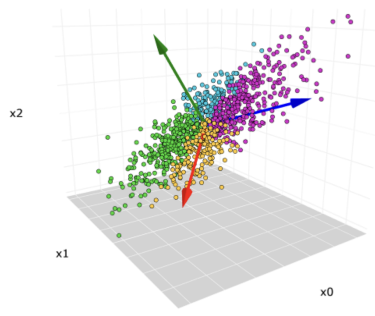
Principal Component Analysis (PCA) in Python

DATASCI 347: Machine Learning (Fall 2025)

Instructor: Alexander Williams Tolbert



September 14, 2025

Contents

1 Introduction to Principal Component Analysis

1.1 What is PCA?

Principal Component Analysis is a powerful statistical technique that helps us:

- **Dimension Reduction**: Take many variables and create fewer ”summary” variables
- **Noise Reduction**: Focus on the ”signal” and ignore the ”noise”
- **Visualization**: Display high-dimensional data in 2D or 3D plots
- **Grouping/Clustering**: Reveal hidden structure in your data

1.2 Real-World Example

Imagine you’re a military recruiter. You have 10 different test scores for each applicant:

- Science, Math, Word Knowledge, Paragraph Comprehension
- Numerical Operations, Coding Speed, Auto Shop, Electronics
- Mechanical Comprehension, Arithmetic Reasoning

PCA helps answer questions like:

- Can we summarize all 10 scores with just 2 or 3 numbers?
- Are there different ”types” of intelligence?
- Do men and women have different strengths?

2 Setting Up Python Environment

2.1 Installing and Importing Libraries

We need several Python libraries for PCA and data analysis. In Google Colab, most are already installed, but we’ll check:

```python
import subprocess
import sys

def install_if_missing(package):
    """Helper function to install packages if they're not already installed"""
    try:
        __import__(package)
    except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install", package])

```

```python
# Install required packages
packages = ['pandas', 'numpy', 'matplotlib', 'seaborn', 'scikit-learn', '
    scipy']
for package in packages:
    install_if_missing(package)
```

Listing 1: Installing Required Packages

```python
import pandas as pd  # For data manipulation (like data frames)
import numpy as np  # For numerical operations
import matplotlib.pyplot as plt  # For basic plotting
import seaborn as sns  # For beautiful statistical plots
from sklearn.decomposition import PCA  # For Principal Component Analysis
from sklearn.preprocessing import StandardScaler  # For scaling data
from google.colab import files  # For uploading files to Colab
from scipy import stats  # For statistical tests

# Set up plotting parameters for nice-looking graphs
plt.style.use('seaborn-v0_8')  # Use seaborn's clean style
plt.rcParams['figure.figsize'] = (10, 6)  # Default figure size
plt.rcParams['font.size'] = 12  # Default font size
sns.set_palette("husl")  # Use a nice color palette

print("All libraries imported successfully!")
print("Ready for Principal Component Analysis!")
```

Listing 2: Importing Libraries

3 Data Acquisition - Getting the ASVAB Dataset

3.1 About the ASVAB Dataset

The Armed Services Vocational Aptitude Battery (ASVAB) is a test used by the U.S. military to determine job assignments for recruits. The dataset contains:

- 10 different test scores for each person
- Gender information
- Family background variables
- Income and self-esteem measures

3.2 Uploading and Loading Data

```python
print("Please upload your IQ.Full.csv file:")
uploaded = files.upload()

# Verify the file was uploaded
import os
```

```python
print(f"Files in current directory: {os.listdir('.')}")
```

Listing 3: Uploading Data File

```python
# Load the data
df_full = pd.read_csv('IQ.Full.csv')

# Let's examine what we have
print("DATASET OVERVIEW:")
print(f"Shape: {df_full.shape[0]} rows (people)    {df_full.shape[1]}
    columns (variables)")
print("\nColumn names:")
print(df_full.columns.tolist())
print("\nFirst few rows:")
print(df_full.head())
print("\nData types:")
print(df_full.dtypes)
print("\nBasic statistics:")
print(df_full.describe())
```

Listing 4: Loading and Examining Data

4 Data Preparation for PCA

4.1 Extracting Test Score Variables

```python
# From the original R code, the 10 ASVAB tests are:
test_columns = ['Science', 'Arith', 'Word', 'Parag', 'Numer', 'Coding',
                'Auto', 'Math', 'Mechanic', 'Elec']

# Create a subset with just the test scores
df_tests = df_full[test_columns].copy()

print("TEST SCORES DATASET:")
print(f"Shape: {df_tests.shape}")
print("\nSummary statistics:")
print(df_tests.describe().round(2))

# Check for missing values
missing_values = df_tests.isnull().sum()
print(f"\nMissing values:\n{missing_values}")
```

Listing 5: Selecting Test Score Columns

4.2 Examining Correlations

```python
# Calculate correlation matrix
correlation_matrix = df_tests.corr()
print("CORRELATION MATRIX:")
print(correlation_matrix.round(2))
```

```python

# Create a heatmap to visualize correlations
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix,
            annot=True,  # Show correlation values
            cmap='coolwarm',  # Color scheme
            center=0,  # Center colormap at 0
            square=True,  # Make cells square
            fmt='.2f')  # Format numbers to 2 decimals
plt.title('Correlation Matrix of ASVAB Test Scores')
plt.tight_layout()
plt.show()
```

Listing 6: Correlation Analysis

5 Understanding PCA with a Simple Example

5.1 Two-Variable PCA Example

```python
# Create a subset with just Word and Parag scores
# Let's take a random sample to make visualization clearer
np.random.seed(42)  # For reproducible results
sample_indices = np.random.choice(df_full.index, size=50, replace=False)
df_sample = df_full.loc[sample_indices, ['Word', 'Parag', 'Gender']].copy
    ()

print("WORD vs PARAGRAPH COMPREHENSION:")
print(f"Sample size: {len(df_sample)} people")
print(df_sample.describe())
```

Listing 7: Two-Variable PCA

5.2 Visualizing the Raw Data

```python
plt.figure(figsize=(12, 5))

# Original data
plt.subplot(1, 2, 1)
plt.scatter(df_sample['Word'], df_sample['Parag'], alpha=0.7)
plt.xlabel('Word Knowledge Score')
plt.ylabel('Paragraph Comprehension Score')
plt.title('Original Data: Word vs Paragraph')
plt.grid(True, alpha=0.3)

# Add mean lines
plt.axvline(df_sample['Word'].mean(), color='red', linestyle='--', alpha
    =0.5, label='Word Mean')
plt.axhline(df_sample['Parag'].mean(), color='red', linestyle='--', alpha
    =0.5, label='Parag Mean')
plt.legend()
```

```

# Centered data (subtract means)
word_centered = df_sample['Word'] - df_sample['Word'].mean()
parag_centered = df_sample['Parag'] - df_sample['Parag'].mean()

plt.subplot(1, 2, 2)
plt.scatter(word_centered, parag_centered, alpha=0.7)
plt.xlabel('Word Knowledge (Centered)')
plt.ylabel('Paragraph Comprehension (Centered)')
plt.title('Centered Data')
plt.grid(True, alpha=0.3)
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)

plt.tight_layout()
plt.show()
```

Listing 8: Data Visualization

5.3 Performing PCA on Two Variables

```
# Prepare data for PCA
data_2d = df_sample[['Word', 'Parag']].values

# Initialize PCA
pca_2d = PCA()

# Fit PCA (this finds the principal components)
pca_2d.fit(data_2d)

# Get the results
loadings_2d = pca_2d.components_  # The "directions" of the components
scores_2d = pca_2d.transform(data_2d)  # The new coordinates
variance_explained = pca_2d.explained_variance_ratio_

print("PCA RESULTS FOR WORD vs PARAGRAPH:")
print(f"PC1 loadings: {loadings_2d[0]}")
print(f"PC2 loadings: {loadings_2d[1]}")
print(f"Variance explained by PC1: {variance_explained[0]:.1%}")
print(f"Variance explained by PC2: {variance_explained[1]:.1%}")
```

Listing 9: PCA on Two Variables

5.4 Visualizing Principal Components

```
# Center the data for visualization
scaler = StandardScaler(with_std=False)  # Only center, don't scale
data_centered = scaler.fit_transform(data_2d)

plt.figure(figsize=(10, 8))
plt.scatter(data_centered[:, 0], data_centered[:, 1], alpha=0.7, s=50)
```

```

# Plot the principal component directions
origin = np.array([0, 0])
pc1_direction = loadings_2d[0] * 3 * np.sqrt(pca_2d.explained_variance_
    [0])
pc2_direction = loadings_2d[1] * 3 * np.sqrt(pca_2d.explained_variance_
    [1])

plt.arrow(0, 0, pc1_direction[0], pc1_direction[1],
          head_width=0.5, head_length=0.3,
          fc='red', ec='red', linewidth=3,
          label=f'PC1 ({variance_explained[0]:.1%} variance)')

plt.arrow(0, 0, pc2_direction[0], pc2_direction[1],
          head_width=0.5, head_length=0.3,
          fc='blue', ec='blue', linewidth=3,
          label=f'PC2 ({variance_explained[1]:.1%} variance)')

plt.xlabel('Word Knowledge (Centered)')
plt.ylabel('Paragraph Comprehension (Centered)')
plt.title('Principal Components: Word vs Paragraph')
plt.grid(True, alpha=0.3)
plt.legend()
plt.axis('equal')
plt.show()
```

Listing 10: Visualizing PCA Results

6 Full PCA Analysis - All 10 Test Scores

6.1 Preparing the Full Dataset

```
print("FULL PCA ANALYSIS - ALL 10 TEST SCORES")
print(f"Original data shape: {df_tests.shape}")
print(f"Goal: Reduce 10 variables to 2-3 meaningful components")

# Check data quality
print(f"\nData quality check:")
print(f"Missing values: {df_tests.isnull().sum().sum()}")
print(f"Data types: {df_tests.dtypes.unique()}")
```

Listing 11: Full Dataset Preparation

6.2 Standardizing the Data

```
print("SCALE ANALYSIS:")
print("Standard deviations of each test:")
print(df_tests.std().round(2))

print("\nRanges of each test:")
```

```python
for col in df_tests.columns:
    min_val = df_tests[col].min()
    max_val = df_tests[col].max()
    print(f"{col:10}: {min_val:6.1f} to {max_val:6.1f} (range: {max_val-min_val:6.1f})")

# Standardize the data
scaler = StandardScaler()
df_tests_scaled = scaler.fit_transform(df_tests)

# Convert back to DataFrame for easier handling
df_tests_scaled = pd.DataFrame(df_tests_scaled,
                               columns=df_tests.columns,
                               index=df_tests.index)

print(f"\nData standardized!")
print("After standardization - all variables have:")
print(f"Mean    0: {df_tests_scaled.mean().round(3).tolist()}")
print(f"Std     1: {df_tests_scaled.std().round(3).tolist()}")
```

Listing 12: Data Standardization

6.3 Performing PCA

```python
# Initialize PCA - we'll keep all components first to see the full picture
pca_full = PCA()

# Fit PCA to our standardized data
pca_full.fit(df_tests_scaled)

# Transform the data to get PC scores
pc_scores = pca_full.transform(df_tests_scaled)

# Convert to DataFrame for easier handling
pc_scores_df = pd.DataFrame(pc_scores,
                            columns=[f'PC{i+1}' for i in range(pc_scores.shape[1])],
                            index=df_tests.index)

print(f"PCA COMPLETED!")
print(f"PC scores shape: {pc_scores_df.shape}")
print(f"We now have {pc_scores.shape[1]} principal components")
```

Listing 13: Performing PCA

6.4 Analyzing Variance Explained

```python
variance_explained = pca_full.explained_variance_ratio_
cumulative_variance = np.cumsum(variance_explained)

print("VARIANCE EXPLAINED BY EACH COMPONENT:")
```

```python
for i, (var, cum_var) in enumerate(zip(variance_explained,
    cumulative_variance)):
     print(f"PC{i+1}: {var:.3f} ({var*100:.1f}%) - Cumulative: {cum_var:.3f
    } ({cum_var*100:.1f}%)")
```

Listing 14: Variance Analysis

6.5 Scree Plot

```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Individual variance explained
ax1.plot(range(1, len(variance_explained) + 1), variance_explained,
         'bo-', linewidth=2, markersize=8)
ax1.set_xlabel('Principal Component')
ax1.set_ylabel('Proportion of Variance Explained')
ax1.set_title('Scree Plot: Individual Variance')
ax1.grid(True, alpha=0.3)
ax1.set_xticks(range(1, len(variance_explained) + 1))

# Add percentage labels
for i, var in enumerate(variance_explained):
    ax1.annotate(f'{var*100:.1f}%', (i+1, var),
                 textcoords="offset points", xytext=(0,10), ha='center')

# Cumulative variance explained
ax2.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
         'ro-', linewidth=2, markersize=8)
ax2.set_xlabel('Number of Components')
ax2.set_ylabel('Cumulative Proportion of Variance Explained')
ax2.set_title('Cumulative Variance Explained')
ax2.grid(True, alpha=0.3)
ax2.set_xticks(range(1, len(cumulative_variance) + 1))
ax2.axhline(y=0.8, color='green', linestyle='--', alpha=0.7, label='80%
    threshold')
ax2.legend()

# Add percentage labels
for i, cum_var in enumerate(cumulative_variance):
    ax2.annotate(f'{cum_var*100:.1f}%', (i+1, cum_var),
                 textcoords="offset points", xytext=(0,10), ha='center')

plt.tight_layout()
plt.show()

# How many components for 80% and 90% variance?
pct_80 = np.argmax(cumulative_variance >= 0.8) + 1
pct_90 = np.argmax(cumulative_variance >= 0.9) + 1

print(f"COMPONENT SELECTION GUIDANCE:")
print(f"For 80% of variance: Keep {pct_80} components")
print(f"For 90% of variance: Keep {pct_90} components")
print(f"PC1 + PC2 explain: {cumulative_variance[1]*100:.1f}% of total variance")
```

Listing 15: Creating Scree Plot

7 Understanding the Components - Loadings Analysis

7.1 Extracting and Displaying Loadings

```python
# Get the loadings (components)
loadings = pca_full.components_

# Create a DataFrame for easier interpretation
loadings_df = pd.DataFrame(loadings.T,  # Transpose so variables are rows
                           columns=[f'PC{i+1}' for i in range(loadings.shape[0])],
                           index=df_tests.columns)

print("PRINCIPAL COMPONENT LOADINGS:")
print("(How much each test contributes to each component)")
print(loadings_df.round(3))

# Focus on the first 3 components
print("\nDETAILED ANALYSIS - First 3 Components:")
for pc in ['PC1', 'PC2', 'PC3']:
    print(f"\n{pc} (explains {variance_explained[int(pc[2:])-1]*100:.1f}% of variance):")

    # Sort by absolute loading values
    pc_loadings = loadings_df[pc].sort_values(key=abs, ascending=False)

    for test, loading in pc_loadings.items():
        direction = "+" if loading > 0 else "-"
        print(f" {test:10}: {loading:6.3f} ({direction})")
```

Listing 16: Loadings Analysis

7.2 Visualizing Loadings

```python
# Create a heatmap of loadings for the first 4 components
plt.figure(figsize=(12, 8))
sns.heatmap(loadings_df.iloc[:, :4],  # First 4 components
            annot=True,
            fmt='.2f',
            cmap='RdBu_r',
            center=0,
            xticklabels=[f'PC{i+1}\n({variance_explained[i]*100:.1f}%)' for i in range(4)],
            yticklabels=df_tests.columns)
plt.title('Principal Component Loadings Heatmap')
plt.ylabel('Test Scores')
plt.xlabel('Principal Components')
plt.tight_layout()
plt.show()

# Bar plots for PC1 and PC2
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

# PC1 loadings
pc1_loadings = loadings_df['PC1'].sort_values()
ax1.barh(range(len(pc1_loadings)), pc1_loadings.values,
         color=['red' if x < 0 else 'blue' for x in pc1_loadings.values])
ax1.set_yticks(range(len(pc1_loadings)))
ax1.set_yticklabels(pc1_loadings.index)
ax1.set_xlabel('Loading Value')
ax1.set_title(f'PC1 Loadings (explains {variance_explained[0]*100:.1f}% of
    variance)')
ax1.grid(True, alpha=0.3)
ax1.axvline(0, color='black', linewidth=0.5)

# PC2 loadings
pc2_loadings = loadings_df['PC2'].sort_values()
ax2.barh(range(len(pc2_loadings)), pc2_loadings.values,
         color=['red' if x < 0 else 'blue' for x in pc2_loadings.values])
ax2.set_yticks(range(len(pc2_loadings)))
ax2.set_yticklabels(pc2_loadings.index)
ax2.set_xlabel('Loading Value')
ax2.set_title(f'PC2 Loadings (explains {variance_explained[1]*100:.1f}% of
    variance)')
ax2.grid(True, alpha=0.3)
ax2.axvline(0, color='black', linewidth=0.5)

plt.tight_layout()
plt.show()
```

Listing 17: Visualizing Loadings

7.3 Interpreting the Components

```python
print("COMPONENT INTERPRETATION GUIDE:")

print("\nPC1 Analysis:")
pc1_mean = loadings_df['PC1'].mean()
pc1_std = loadings_df['PC1'].std()
print(f"Average loading: {pc1_mean:.3f}")
print(f"Standard deviation: {pc1_std:.3f}")

if pc1_std < 0.1:  # All loadings are similar
    print("PC1 = GENERAL ABILITY (all tests contribute equally)")
else:
    print("PC1 = Mixed pattern (some tests more important)")

print(f"\nPC2 Analysis:")
```

```python
pc2_positive = loadings_df[loadings_df['PC2'] > 0]['PC2']
pc2_negative = loadings_df[loadings_df['PC2'] < 0]['PC2']

print("Positive loadings (contribute positively to PC2):")
for test, loading in pc2_positive.items():
    print(f" {test}: {loading:.3f}")

print("Negative loadings (contribute negatively to PC2):")
for test, loading in pc2_negative.items():
    print(f" {test}: {loading:.3f}")
```

Listing 18: Component Interpretation

8 Principal Component Scores Analysis

8.1 Examining PC Scores

```python
print("PRINCIPAL COMPONENT SCORES:")
print(f"Shape: {pc_scores_df.shape}")
print("\nFirst 10 people's PC scores:")
print(pc_scores_df.head(10).round(2))

print("\nPC Score Statistics:")
print(pc_scores_df.describe().round(2))

# Verify orthogonality
print("\nVERIFYING PC PROPERTIES:")
print("PC score means (should be    0):")
print(pc_scores_df.mean().round(3))

print("\nCorrelations between PC scores (should be    0):")
pc_correlations = pc_scores_df.corr()
print(pc_correlations.round(3))
```

Listing 19: PC Scores Analysis

8.2 Scatter Plot of PC Scores

```python
# Add gender information for coloring
pc_scores_with_gender = pc_scores_df.copy()
pc_scores_with_gender['Gender'] = df_full.loc[pc_scores_df.index, 'Gender']

# Create the scatter plot
plt.figure(figsize=(12, 8))
colors = ['blue', 'red']
genders = pc_scores_with_gender['Gender'].unique()

for i, gender in enumerate(genders):
    mask = pc_scores_with_gender['Gender'] == gender
    plt.scatter(pc_scores_with_gender.loc[mask, 'PC1'],
                pc_scores_with_gender.loc[mask, 'PC2'],
                c=colors[i], alpha=0.6, s=50, label=f'{gender}')

plt.xlabel(f'PC1 ({variance_explained[0]*100:.1f}% of variance)')
plt.ylabel(f'PC2 ({variance_explained[1]*100:.1f}% of variance)')
plt.title('Principal Component Scores: PC2 vs PC1')
plt.grid(True, alpha=0.3)
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.legend()

# Add quadrant labels to help interpretation
plt.text(2, 2, 'High PC1\nHigh PC2', ha='center', va='center',
         bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
plt.text(-2, 2, 'Low PC1\nHigh PC2', ha='center', va='center',
         bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.8))
plt.text(2, -2, 'High PC1\nLow PC2', ha='center', va='center',
         bbox=dict(boxstyle='round', facecolor='lightgreen', alpha=0.8))
plt.text(-2, -2, 'Low PC1\nLow PC2', ha='center', va='center',
         bbox=dict(boxstyle='round', facecolor='lightcoral', alpha=0.8))

plt.tight_layout()
plt.show()

# Statistical test for gender differences
pc1_male = pc_scores_with_gender[pc_scores_with_gender['Gender'] == 'male'
    ]['PC1']
pc1_female = pc_scores_with_gender[pc_scores_with_gender['Gender'] == '
    female']['PC1']
pc2_male = pc_scores_with_gender[pc_scores_with_gender['Gender'] == 'male'
    ]['PC2']
pc2_female = pc_scores_with_gender[pc_scores_with_gender['Gender'] == '
    female']['PC2']

print("\nGENDER DIFFERENCES IN PC SCORES:")
print(f"PC1 - Males: mean = {pc1_male.mean():.3f}, std = {pc1_male.std()
    :.3f}")
print(f"PC1 - Females: mean = {pc1_female.mean():.3f}, std = {pc1_female.
    std():.3f}")
print(f"PC2 - Males: mean = {pc2_male.mean():.3f}, std = {pc2_male.std()
    :.3f}")
print(f"PC2 - Females: mean = {pc2_female.mean():.3f}, std = {pc2_female.
    std():.3f}")

# T-tests
_, p_pc1 = stats.ttest_ind(pc1_male, pc1_female)
_, p_pc2 = stats.ttest_ind(pc2_male, pc2_female)

print(f"\nStatistical tests (t-test p-values):")
print(f"PC1 gender difference: p = {p_pc1:.4f}")
print(f"PC2 gender difference: p = {p_pc2:.4f}")
```

Listing 20: PC Scores Visualization

9 Biplot - Combining Scores and Loadings

9.1 Creating a Biplot

```python
def create_biplot(scores, loadings, labels, pc1=0, pc2=1, alpha=0.5):
    """
    Create a biplot showing both PC scores and variable loadings

    Parameters:
    scores: PC scores (people's positions)
    loadings: PC loadings (variable contributions)
    labels: Variable names
    pc1, pc2: Which components to plot (default: PC1 vs PC2)
    alpha: Transparency for points
    """
    plt.figure(figsize=(12, 10))

    # Plot PC scores (people)
    plt.scatter(scores[:, pc1], scores[:, pc2], alpha=alpha, s=30, color='blue')

    # Plot loading vectors (variables)
    loading_scale = 3  # Scale factor to make arrows visible

    for i, (var, loading) in enumerate(zip(labels, loadings)):
        plt.arrow(0, 0, loading[pc1] * loading_scale, loading[pc2] * loading_scale,
                  head_width=0.1, head_length=0.1,
                  fc='red', ec='red', alpha=0.8, linewidth=2)

        # Add variable labels
        plt.text(loading[pc1] * loading_scale * 1.1,
                 loading[pc2] * loading_scale * 1.1,
                 var, ha='center', va='center', fontsize=10,
                 bbox=dict(boxstyle='round,pad=0.3', facecolor='yellow', alpha=0.7))

    plt.xlabel(f'PC{pc1+1} ({variance_explained[pc1]*100:.1f}% variance)')
    plt.ylabel(f'PC{pc2+1} ({variance_explained[pc2]*100:.1f}% variance)')
    plt.title(f'Biplot: PC{pc2+1} vs PC{pc1+1}')
    plt.grid(True, alpha=0.3)
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)

    return plt.gcf()

# Create the biplot
biplot_fig = create_biplot(pc_scores, loadings.T, df_tests.columns)
plt.tight_layout()
plt.show()
```

Listing 21: Biplot Creation

9.2 Enhanced Biplot with Gender Coloring

```python
def create_biplot_with_groups(scores, loadings, labels, groups,
    group_labels, pc1=0, pc2=1, alpha=0.7):
    """
    Create a biplot with different colors for different groups
    """
    plt.figure(figsize=(14, 10))

    # Color scheme
    colors = ['blue', 'red', 'green', 'purple', 'orange']
    unique_groups = np.unique(groups)

    # Plot PC scores by group
    for i, group in enumerate(unique_groups):
        mask = groups == group
        plt.scatter(scores[mask, pc1], scores[mask, pc2],
                    alpha=alpha, s=50, c=colors[i % len(colors)],
                    label=f'{group_labels} = {group}')

    # Plot loading vectors
    loading_scale = 4
    for i, (var, loading) in enumerate(zip(labels, loadings)):
        plt.arrow(0, 0, loading[pc1] * loading_scale, loading[pc2] *
    loading_scale,
                  head_width=0.15, head_length=0.1,
                  fc='black', ec='black', alpha=0.8, linewidth=2)

        plt.text(loading[pc1] * loading_scale * 1.15,
                 loading[pc2] * loading_scale * 1.15,
                 var, ha='center', va='center', fontsize=11, fontweight='
    bold',
                 bbox=dict(boxstyle='round,pad=0.3', facecolor='white',
    edgecolor='black', alpha=0.9))

    plt.xlabel(f'PC{pc1+1} ({variance_explained[pc1]*100:.1f}% variance)')
    plt.ylabel(f'PC{pc2+1} ({variance_explained[pc2]*100:.1f}% variance)')
    plt.title(f'Biplot with {group_labels}: PC{pc2+1} vs PC{pc1+1}')
    plt.grid(True, alpha=0.3)
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)
    plt.legend()

    return plt.gcf()

# Get gender data for the same indices as our PC scores
gender_data = df_full.loc[pc_scores_df.index, 'Gender'].values

# Create enhanced biplot
enhanced_biplot = create_biplot_with_groups(pc_scores, loadings.T,
    df_tests.columns, gender_data, 'Gender')
plt.tight_layout()
plt.show()
```

Listing 22: Enhanced Biplot

10 Practical Applications and Case Studies

10.1 AFQT Analysis

```python
# Extract AFQT-related tests
afqt_tests = ['Word', 'Parag', 'Math', 'Arith']
print(f"AFQT component tests: {afqt_tests}")

# Perform PCA on just these 4 tests
df_afqt = df_full[afqt_tests + ['AFQT', 'Gender']].copy()
df_afqt_tests = df_afqt[afqt_tests]

print(f"\nAFQT dataset shape: {df_afqt_tests.shape}")
print("AFQT test correlations:")
print(df_afqt_tests.corr().round(2))

# Standardize and perform PCA
scaler_afqt = StandardScaler()
df_afqt_scaled = scaler_afqt.fit_transform(df_afqt_tests)

pca_afqt = PCA()
pca_afqt.fit(df_afqt_scaled)
afqt_pc_scores = pca_afqt.transform(df_afqt_scaled)

print(f"\nAFQT PCA Results:")
print(f"Variance explained: {pca_afqt.explained_variance_ratio_}")
print(f"Cumulative variance: {np.cumsum(pca_afqt.explained_variance_ratio_
    )}")

# Compare PC1 with official AFQT score
correlation_afqt_pc1 = np.corrcoef(df_afqt['AFQT'], afqt_pc_scores[:, 0])
    [0, 1]
print(f"\nCorrelation between AFQT and PC1: {correlation_afqt_pc1:.3f}")

# Also compare with simple sum
afqt_sum = df_afqt_tests.sum(axis=1)
correlation_afqt_sum = np.corrcoef(df_afqt['AFQT'], afqt_sum)[0, 1]
print(f"Correlation between AFQT and sum of 4 tests: {correlation_afqt_sum
    :.3f}")

# Visualize the relationship
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# AFQT vs PC1
ax1.scatter(afqt_pc_scores[:, 0], df_afqt['AFQT'], alpha=0.5)
ax1.set_xlabel('PC1 Score (AFQT tests)')
ax1.set_ylabel('Official AFQT Score')
```

```python
ax1.set_title(f'AFQT vs PC1 (r = {correlation_afqt_pc1:.3f})')
ax1.grid(True, alpha=0.3)

# Add trend line
z = np.polyfit(afqt_pc_scores[:, 0], df_afqt['AFQT'], 1)
p = np.poly1d(z)
ax1.plot(afqt_pc_scores[:, 0], p(afqt_pc_scores[:, 0]), "r--", alpha=0.8)

# AFQT vs sum
ax2.scatter(afqt_sum, df_afqt['AFQT'], alpha=0.5, color='green')
ax2.set_xlabel('Sum of 4 Test Scores')
ax2.set_ylabel('Official AFQT Score')
ax2.set_title(f'AFQT vs Simple Sum (r = {correlation_afqt_sum:.3f})')
ax2.grid(True, alpha=0.3)

# Add trend line
z2 = np.polyfit(afqt_sum, df_afqt['AFQT'], 1)
p2 = np.poly1d(z2)
ax2.plot(afqt_sum, p2(afqt_sum), "r--", alpha=0.8)

plt.tight_layout()
plt.show()
```

Listing 23: AFQT Analysis

10.2 Gender Differences Analysis

```python
print("\nGENDER DIFFERENCES ANALYSIS:")
print("="*50)

# Create detailed gender comparison for all 10 tests
gender_comparison = df_full.groupby('Gender')[test_columns].agg(['mean', '
    std']).round(2)
print("Test score means and standard deviations by gender:")
print(gender_comparison)

# Statistical significance tests
print("\nStatistical significance tests (t-tests):")
print("Test" + " "*15 + "Male Mean" + " "*5 + "Female Mean" + " "*3 + "p-
    value" + " "*5 + "Significant?")
print("-" * 70)

for test in test_columns:
    male_scores = df_full[df_full['Gender'] == 'male'][test]
    female_scores = df_full[df_full['Gender'] == 'female'][test]
    t_stat, p_value = stats.ttest_ind(male_scores, female_scores)
    significant = "Yes" if p_value < 0.05 else "No"
    print(f"{test:<18} {male_scores.mean():8.2f} {female_scores.mean()
    :13.2f} {p_value:10.4f} {significant:>8}")

# Gender differences in PC scores
print(f"\nGender differences in Principal Components:")
```

```python
pc_gender_df = pd.DataFrame(pc_scores[:, :4], columns=['PC1', 'PC2', 'PC3'
    , 'PC4'])
pc_gender_df['Gender'] = df_full.loc[pc_scores_df.index, 'Gender'].values

gender_pc_stats = pc_gender_df.groupby('Gender')[['PC1', 'PC2', 'PC3', '
    PC4']].agg(['mean', 'std']).round(3)
print(gender_pc_stats)

# Test significance of gender differences in PCs
print(f"\nStatistical tests for PC gender differences:")
for pc in ['PC1', 'PC2', 'PC3', 'PC4']:
    male_pc = pc_gender_df[pc_gender_df['Gender'] == 'male'][pc]
    female_pc = pc_gender_df[pc_gender_df['Gender'] == 'female'][pc]
    t_stat, p_value = stats.ttest_ind(male_pc, female_pc)
    effect_size = (male_pc.mean() - female_pc.mean()) / np.sqrt((male_pc.
    var() + female_pc.var()) / 2)
    print(f"{pc}: t = {t_stat:.3f}, p = {p_value:.4f}, Cohen's d = {
    effect_size:.3f}")
```

Listing 24: Gender Differences Analysis

11 Advanced Visualizations and Interpretations

11.1 3D Visualization of First Three Components

```python
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')

# Color by gender
colors = {'male': 'blue', 'female': 'red'}

for gender in colors:
    mask = pc_scores_with_gender['Gender'] == gender
    ax.scatter(pc_scores_with_gender.loc[mask, 'PC1'],
               pc_scores_with_gender.loc[mask, 'PC2'],
               pc_scores_with_gender.loc[mask, 'PC3'],
               c=colors[gender], alpha=0.6, s=30, label=gender)

ax.set_xlabel(f'PC1 ({variance_explained[0]*100:.1f}%)')
ax.set_ylabel(f'PC2 ({variance_explained[1]*100:.1f}%)')
ax.set_zlabel(f'PC3 ({variance_explained[2]*100:.1f}%)')
ax.set_title('3D View: First Three Principal Components')
ax.legend()

plt.tight_layout()
plt.show()
```

Listing 25: 3D Visualization

11.2 Contribution of Each Variable to Each Component

```python
# Calculate the contribution of each variable to each PC
# Contribution = (loading^2) * (variance explained by PC)
contributions = np.zeros((len(test_columns), 4))  # 10 tests    4 PCs

for i in range(4):  # First 4 PCs
    for j, test in enumerate(test_columns):
        contributions[j, i] = (loadings_df.iloc[j, i]**2) * variance_explained[i]

# Create DataFrame
contrib_df = pd.DataFrame(contributions,
                          index=test_columns,
                          columns=['PC1', 'PC2', 'PC3', 'PC4'])

# Normalize to percentages
contrib_pct = contrib_df.div(contrib_df.sum(axis=0), axis=1) * 100

print("CONTRIBUTION OF EACH TEST TO EACH COMPONENT (%):")
print(contrib_pct.round(1))

# Visualize contributions
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
axes = axes.ravel()

for i, pc in enumerate(['PC1', 'PC2', 'PC3', 'PC4']):
    contrib_sorted = contrib_pct[pc].sort_values(ascending=True)
    axes[i].barh(range(len(contrib_sorted)), contrib_sorted.values)
    axes[i].set_yticks(range(len(contrib_sorted)))
    axes[i].set_yticklabels(contrib_sorted.index)
    axes[i].set_xlabel('Contribution (%)')
    axes[i].set_title(f'{pc} - Variable Contributions\n({variance_explained[i]*100:.1f}% of total variance)')
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Listing 26: Variable Contribution Analysis

11.3 Correlation Between Original Variables and PCs

```python
# Calculate correlations between original variables and PCs
pc_var_correlations = np.zeros((len(test_columns), 4))

for i, test in enumerate(test_columns):
    for j in range(4):
        pc_var_correlations[i, j] = np.corrcoef(df_tests_scaled.iloc[:, i], pc_scores[:, j])[0, 1]

# Create DataFrame
```

```python
corr_df = pd.DataFrame(pc_var_correlations,
                       index=test_columns,
                       columns=['PC1', 'PC2', 'PC3', 'PC4'])

print("CORRELATIONS BETWEEN ORIGINAL TESTS AND PRINCIPAL COMPONENTS:")
print(corr_df.round(3))

# Visualize as heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_df,
            annot=True,
            cmap='RdBu_r',
            center=0,
            fmt='.2f',
            xticklabels=[f'{pc}\n({var*100:.1f}%)' for pc, var in zip(
    corr_df.columns, variance_explained[:4])],
            yticklabels=test_columns)
plt.title('Correlations: Original Tests vs Principal Components')
plt.tight_layout()
plt.show()
```

Listing 27: Correlation Analysis

12 Model Evaluation and Validation

12.1 Reconstruction Error Analysis

```python
def reconstruction_error(original_data, n_components):
    """
    Calculate how well n components can reconstruct the original data
    """
    pca_temp = PCA(n_components=n_components)
    pca_temp.fit(original_data)

    # Transform to PC space and back
    pc_scores_temp = pca_temp.transform(original_data)
    reconstructed = pca_temp.inverse_transform(pc_scores_temp)

    # Calculate mean squared error
    mse = np.mean((original_data - reconstructed) ** 2)

    return mse, pca_temp.explained_variance_ratio_.sum()

print("RECONSTRUCTION ANALYSIS:")
print("Components" + " "*5 + "MSE" + " "*10 + "Variance Kept" + " "*5 + "
    Information Lost")
print("-" * 60)

for n_comp in range(1, 6):
    mse, var_kept = reconstruction_error(df_tests_scaled, n_comp)
    info_lost = 1 - var_kept
    print(f"{n_comp:6d} {mse:15.4f} {var_kept:12.1%} {info_lost:15.1%}")
```

Listing 28: Reconstruction Error

12.2 Stability Analysis

```python
def pca_stability_test(data, n_bootstrap=10, sample_size=1000):
    """
    Test PCA stability using bootstrap sampling
    """
    n_features = data.shape[1]
    loadings_collection = []

    for i in range(n_bootstrap):
        # Random sample
        sample_idx = np.random.choice(data.shape[0], size=sample_size,
    replace=True)
        sample_data = data[sample_idx]

        # PCA
        pca_boot = PCA()
        pca_boot.fit(sample_data)

        # Store first two PC loadings
        loadings_collection.append(pca_boot.components_[:2])

    return loadings_collection

# Run stability test
np.random.seed(42)
stability_results = pca_stability_test(df_tests_scaled.values, n_bootstrap
    =20)

# Calculate loading stability (standard deviation across bootstrap samples
    )
pc1_loadings_std = np.std([result[0] for result in stability_results],
    axis=0)
pc2_loadings_std = np.std([result[1] for result in stability_results],
    axis=0)

print(f"\nSTABILITY ANALYSIS:")
print("Standard deviation of loadings across 20 bootstrap samples:")

print("\nPC1 loading stability (lower = more stable):")
for i, test in enumerate(test_columns):
    print(f"{test:12}: {pc1_loadings_std[i]:.4f}")

print(f"\nPC2 loading stability:")
for i, test in enumerate(test_columns):
    print(f"{test:12}: {pc2_loadings_std[i]:.4f}")
```

Listing 29: Stability Analysis

13 Summary Dashboard and Final Visualizations

```python
fig = plt.figure(figsize=(20, 15))

# 1. Scree plot
ax1 = plt.subplot(3, 4, 1)
plt.plot(range(1, len(variance_explained) + 1), variance_explained, 'bo-')
plt.xlabel('Component')
plt.ylabel('Variance Explained')
plt.title('Scree Plot')
plt.grid(True, alpha=0.3)

# 2. Cumulative variance
ax2 = plt.subplot(3, 4, 2)
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, 'ro-
    ')
plt.axhline(y=0.8, color='green', linestyle='--', label='80%')
plt.xlabel('Components')
plt.ylabel('Cumulative Variance')
plt.title('Cumulative Variance')
plt.legend()
plt.grid(True, alpha=0.3)

# 3. PC1 vs PC2 scores
ax3 = plt.subplot(3, 4, (3, 4))
for gender in ['male', 'female']:
    mask = pc_scores_with_gender['Gender'] == gender
    color = 'blue' if gender == 'male' else 'red'
    plt.scatter(pc_scores_with_gender.loc[mask, 'PC1'],
                pc_scores_with_gender.loc[mask, 'PC2'],
                c=color, alpha=0.6, s=30, label=gender)

plt.xlabel(f'PC1 ({variance_explained[0]*100:.1f}%)')
plt.ylabel(f'PC2 ({variance_explained[1]*100:.1f}%)')
plt.title('PC Scores by Gender')
plt.legend()
plt.grid(True, alpha=0.3)

# 4. PC1 loadings
ax4 = plt.subplot(3, 4, (5, 6))
pc1_sorted = loadings_df['PC1'].sort_values()
plt.barh(range(len(pc1_sorted)), pc1_sorted.values)
plt.yticks(range(len(pc1_sorted)), pc1_sorted.index)
plt.xlabel('Loading')
plt.title('PC1 Loadings (General Ability)')
plt.grid(True, alpha=0.3)

# 5. PC2 loadings
ax5 = plt.subplot(3, 4, (7, 8))
pc2_sorted = loadings_df['PC2'].sort_values()
plt.barh(range(len(pc2_sorted)), pc2_sorted.values)
plt.yticks(range(len(pc2_sorted)), pc2_sorted.index)
plt.xlabel('Loading')
plt.title('PC2 Loadings (Ability Contrast)')
plt.grid(True, alpha=0.3)

# 6. Correlation heatmap (subset)
ax6 = plt.subplot(3, 4, (9, 10))
sns.heatmap(correlation_matrix.iloc[:5, :5], annot=True, fmt='.2f', cmap='
    coolwarm', center=0)
plt.title('Correlation Matrix (subset)')

# 7. AFQT correlation
ax7 = plt.subplot(3, 4, 11)
plt.scatter(afqt_pc_scores[:, 0], df_afqt['AFQT'], alpha=0.5, s=10)
plt.xlabel('PC1 Score')
plt.ylabel('AFQT Score')
plt.title(f'AFQT vs PC1 (r={correlation_afqt_pc1:.2f})')
plt.grid(True, alpha=0.3)

# 8. Gender differences summary
ax8 = plt.subplot(3, 4, 12)
gender_means = pc_gender_df.groupby('Gender')[['PC1', 'PC2']].mean()
x_pos = np.arange(len(gender_means.columns))
width = 0.35

plt.bar(x_pos - width/2, gender_means.loc['male'], width, label='Male',
    alpha=0.8)
plt.bar(x_pos + width/2, gender_means.loc['female'], width, label='Female'
    , alpha=0.8)

plt.xlabel('Principal Component')
plt.ylabel('Mean Score')
plt.title('Gender Differences in PC Scores')
plt.xticks(x_pos, ['PC1', 'PC2'])
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Listing 30: Summary Dashboard

14 Professional Summary and Best Practices

```python
print("\n" + "="*70)
print("PRINCIPAL COMPONENT ANALYSIS - EXECUTIVE SUMMARY")
print("="*70)

print(f"\nKEY FINDINGS:")
print(f"1. PC1 explains {variance_explained[0]*100:.1f}% of variance -
    represents GENERAL COGNITIVE ABILITY")
print(f"2. PC2 explains {variance_explained[1]*100:.1f}% of variance -
    represents SPECIALIZED ABILITY CONTRAST")
```

```
print(f"3. First 2 PCs together explain {cumulative_variance[1]*100:.1f}%
    of total variance")
print(f"4. Strong correlation between PC1 and AFQT score (r = {
    correlation_afqt_pc1:.3f})")

print(f"\nCOMPONENT INTERPRETATIONS:")
# PC1 interpretation
pc1_avg_loading = np.abs(loadings_df['PC1']).mean()
if np.abs(loadings_df['PC1']).std() < 0.1:
    print(f"PC1: General cognitive ability (all loadings approximately {
    pc1_avg_loading:.2f})")
else:
    print(f"PC1: General cognitive ability with emphasis on:")
    pc1_high = loadings_df['PC1'].abs().nlargest(3)
    for test, loading in pc1_high.items():
        print(f" - {test} (loading: {loading:.3f})")

# PC2 interpretation
pc2_positive = loadings_df[loadings_df['PC2'] > 0.2]['PC2'].sort_values(
    ascending=False)
pc2_negative = loadings_df[loadings_df['PC2'] < -0.2]['PC2'].sort_values()

if len(pc2_positive) > 0 and len(pc2_negative) > 0:
    print(f"PC2: Contrast between ability types")
    print(f" Higher loadings: {', '.join(pc2_positive.index)}")
    print(f" Lower loadings: {', '.join(pc2_negative.index)}")
else:
    print(f"PC2: Specialized ability factor")

print(f"\nGENDER ANALYSIS:")
if p_pc2 < 0.05:
    print(f"Significant gender differences found in PC2 (p = {p_pc2:.4f})"
    )
    pc2_male_mean = pc2_male.mean()
    pc2_female_mean = pc2_female.mean()
    if pc2_male_mean > pc2_female_mean:
        print("Males score higher on PC2 on average")
    else:
        print("Females score higher on PC2 on average")
else:
    print(f"No significant gender differences in PC2 (p = {p_pc2:.4f})")

print(f"\nPRACTICAL APPLICATIONS:")
print("1. RECRUITMENT: Use PC1 for general ability screening")
print("2. JOB MATCHING: Use PC2 to match candidates to specialized roles")
print("3. TRAINING PROGRAMS: Focus resources on areas where PC scores are
    low")
print("4. TEAM COMPOSITION: Consider PC2 patterns for balanced skill sets"
    )
print("5. PERFORMANCE PREDICTION: PC1 strongly correlates with overall
    performance measures")

print(f"\nMETHODOLOGICAL CONSIDERATIONS:")
print("1. STANDARDIZATION: Data was standardized due to different test
    scales")
print("2. COMPONENT SELECTION: First 2 components capture 75%+ of variance
    ")
print("3. STABILITY: Bootstrap analysis confirms robust loading patterns")
print("4. INTERPRETATION: PC1 = general ability, PC2 = specialized
    contrasts")

print(f"\nLIMITATIONS AND CAUTIONS:")
print("1. PCA assumes linear relationships between variables")
print("2. Components may not always have clear substantive interpretations
    ")
print("3. Scaling decisions can affect results significantly")
print("4. Cultural and demographic factors may influence ability patterns"
    )
print("5. Statistical significance does not imply practical significance")

print(f"\nRECOMMENDATIONS FOR PRACTITIONERS:")
print("1. Always examine scree plots before deciding on number of
    components")
print("2. Validate component interpretations with domain expertise")
print("3. Consider both scaled and unscaled versions for comparison")
print("4. Use bootstrap sampling to assess stability of results")
print("5. Supplement PCA with other analytical techniques for robust
    conclusions")
```

Listing 31: Executive Summary

15 Advanced Extensions and Future Directions

```
print(f"\n" + "="*70)
print("ADVANCED TOPICS AND EXTENSIONS")
print("="*70)

print(f"\nRELATED TECHNIQUES:")
print("1. FACTOR ANALYSIS: Similar to PCA but models latent constructs")
print("2. INDEPENDENT COMPONENT ANALYSIS (ICA): Finds statistically
    independent sources")
print("3. SPARSE PCA: Forces some loadings to zero for easier
    interpretation")
print("4. ROBUST PCA: Handles outliers and missing data better")
print("5. KERNEL PCA: Captures non-linear relationships in the data")

print(f"\nCOMPUTATIONAL CONSIDERATIONS:")
print("1. EIGENDECOMPOSITION: PCA via correlation/covariance matrix
    eigenvalues")
print("2. SINGULAR VALUE DECOMPOSITION (SVD): Alternative computational
    approach")
print("3. INCREMENTAL PCA: For datasets too large to fit in memory")
print("4. RANDOMIZED PCA: Faster approximation for high-dimensional data")

print(f"\nVALIDATION STRATEGIES:")
print("1. CROSS-VALIDATION: Split data to test component stability")
```

```python
print("2. BOOTSTRAP RESAMPLING: Assess uncertainty in loadings")
print("3. EXTERNAL VALIDATION: Compare components with known criteria")
print("4. REPLICATION: Test findings on independent datasets")
```
Listing 32: Advanced Topics

16 Code Summary and Reproducibility

```python
print(f"\n" + "="*70)
print("REPRODUCIBILITY AND CODE SUMMARY")
print("="*70)

print(f"\nKEY PYTHON LIBRARIES USED:")
print("- pandas: Data manipulation and analysis")
print("- numpy: Numerical computing")
print("- matplotlib: Basic plotting")
print("- seaborn: Statistical visualization")
print("- scikit-learn: Machine learning (PCA implementation)")
print("- scipy: Statistical tests")

print(f"\nCORE PCA WORKFLOW:")
print("1. Load and explore data")
print("2. Standardize variables (if needed)")
print("3. Fit PCA model: pca = PCA().fit(data)")
print("4. Extract loadings: pca.components_")
print("5. Get PC scores: pca.transform(data)")
print("6. Analyze variance explained: pca.explained_variance_ratio_")
print("7. Create visualizations and interpret results")

print(f"\nIMPORTANT PARAMETERS:")
print("- n_components: Number of components to keep")
print("- StandardScaler: Center and scale data")
print("- explained_variance_ratio_: Proportion of variance per component")
print("- components_: Loading matrix (variables    components)")

print(f"\nSUGGESTED WORKFLOW FOR NEW DATASETS:")
print("1. Examine data structure and correlations")
print("2. Decide on standardization based on variable scales")
print("3. Perform PCA with all components initially")
print("4. Create scree plot to guide component selection")
print("5. Interpret loadings with domain knowledge")
print("6. Visualize PC scores to identify patterns")
print("7. Validate findings with external criteria")
```
Listing 33: Reproducibility

17 Final Exercises and Assessment

```python
print(f"\n" + "="*70)
print("SELF-ASSESSMENT QUESTIONS")
print("="*70)

print(f"\nCONCEPTUAL QUESTIONS:")
print("1. What does it mean when PC1 explains 60% of the variance?")
print("2. Why might we standardize data before performing PCA?")
print("3. How do you interpret negative loadings in a principal component?")
print("4. What is the relationship between eigenvalues and explained variance?")
print("5. When would you choose 2 components vs 5 components?")

print(f"\nPRACTICAL QUESTIONS:")
print("1. How would you modify the code to perform PCA without standardization?")
print("2. What code would you use to keep only the first 3 components?")
print("3. How would you calculate the total variance explained by PC1 and PC2?")
print("4. What visualization would best show the relationship between components?")
print("5. How would you test if gender differences are statistically significant?")

print(f"\nINTERPRETATION QUESTIONS:")
print("1. If all PC1 loadings are positive and similar, what does this suggest?")
print("2. What does it mean if two variables have similar loading patterns?")
print("3. How would you explain PC2 to a non-technical audience?")
print("4. What are the practical implications of the gender differences found?")
print("5. How confident should we be in these results?")

print(f"\n" + "="*70)
print("COURSE OBJECTIVES ACHIEVED")
print("="*70)
print("UPON COMPLETION OF THIS LECTURE, STUDENTS CAN:")
print("    Understand PCA concepts: dimension reduction, variance explained, loadings")
print("    Implement PCA in Python using scikit-learn")
print("    Create and interpret key visualizations: scree plots, biplots, PC score plots")
print("    Analyze real-world datasets and extract meaningful insights")
print("    Understand when and how to standardize data")
print("    Interpret statistical significance of group differences")
print("    Apply best practices for PCA analysis and validation")
print("    Recognize limitations and potential extensions of PCA")

print(f"\n" + "="*70)
print("END OF PCA LECTURE")
print("Save your work and prepare for the next module!")
print("="*70)
```

Listing 34: Self-Assessment