PCA_Wine

May 29, 2026

1 PRINCIPAL COMPONENT ANALYSIS - WINE DATASET - SK LEARN

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
```

[2]:
```
data = load_wine()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name="target")

print("Shape:", X.shape)
print("Target classes:", np.unique(y))
print(X.head())
```

```
Shape: (178, 13)
Target classes: [0 1 2]
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
```

```
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline
0                          3.92   1065.0
1                          3.40   1050.0
2                          3.17   1185.0
3                          3.45   1480.0
4                          2.93    735.0
```

[3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[4]:
```
pca_full = PCA()
X_train_pca_full = pca_full.fit_transform(X_train_scaled)

explained = pca_full.explained_variance_ratio_
cum_explained = np.cumsum(explained)

print("Explained variance ratios:", np.round(explained, 4))
print("Cumulative variance:", np.round(cum_explained, 4))
print("Components needed for 95% variance:", np.argmax(cum_explained >= 0.95) +␣
↪1)
```

```
Explained variance ratios: [0.3579 0.1927 0.1102 0.0727 0.0672 0.0513 0.0438
0.025  0.0228 0.0188
 0.0178 0.0126 0.0072]
Cumulative variance: [0.3579 0.5506 0.6608 0.7335 0.8008 0.8521 0.8959 0.9209
0.9437 0.9624
 0.9803 0.9928 1.    ]
Components needed for 95% variance: 10
```

[5]:
```
pca_model = make_pipeline(
    StandardScaler(),
    PCA(n_components=0.95),
    LogisticRegression(max_iter=5000)
)

pca_model.fit(X_train, y_train)
y_pred = pca_model.predict(X_test)

print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred,␣
↪target_names=data.target_names))
```

```
pca_step = pca_model.named_steps["pca"]
print("PCA components retained:", pca_step.n_components_)
```

```
Accuracy: 0.9722
Confusion Matrix:
 [[12  0  0]
 [ 0 14  0]
 [ 0  1  9]]
Classification Report:
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        12
     class_1       0.93      1.00      0.97        14
     class_2       1.00      0.90      0.95        10

    accuracy                           0.97        36
   macro avg       0.98      0.97      0.97        36
weighted avg       0.97      0.97      0.97        36

PCA components retained: 10
```

[6]:
```
baseline_model = make_pipeline(
    StandardScaler(),
    LogisticRegression(max_iter=5000)
)

baseline_cv = cross_val_score(baseline_model, X, y, cv=5, scoring="accuracy").
 ↪mean()
pca_cv = cross_val_score(pca_model, X, y, cv=5, scoring="accuracy").mean()

print(f"Baseline (no PCA) mean CV accuracy: {baseline_cv:.4f}")
print(f"With PCA (95% variance) mean CV accuracy: {pca_cv:.4f}")
```

```
Baseline (no PCA) mean CV accuracy: 0.9832
With PCA (95% variance) mean CV accuracy: 0.9833
```

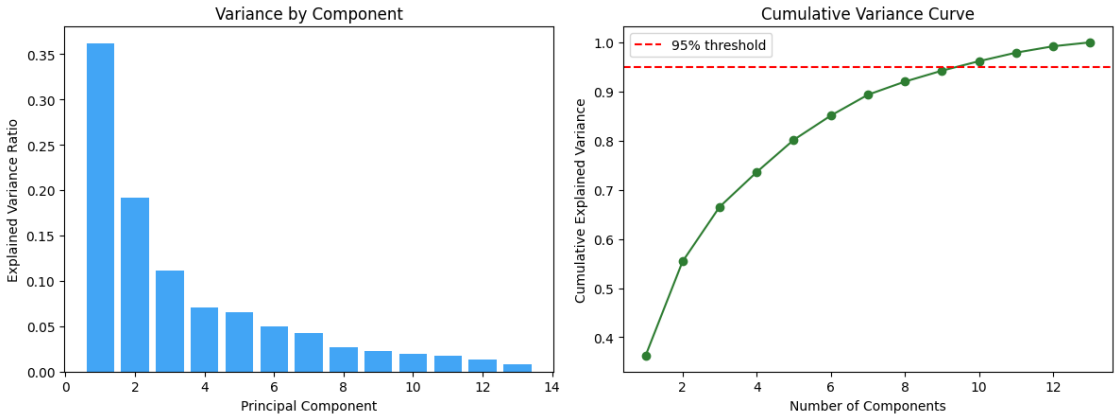
[7]:
```
pca_2 = PCA(n_components=2)
X_scaled = StandardScaler().fit_transform(X)
pca_2.fit(X_scaled)

loading_df = pd.DataFrame(
    pca_2.components_.T,
    columns=["PC1_loading", "PC2_loading"],
    index=X.columns
)

print("Top absolute contributors to PC1:")
```

```
print(loading_df["PC1_loading"].abs().sort_values(ascending=False).head(8))
```

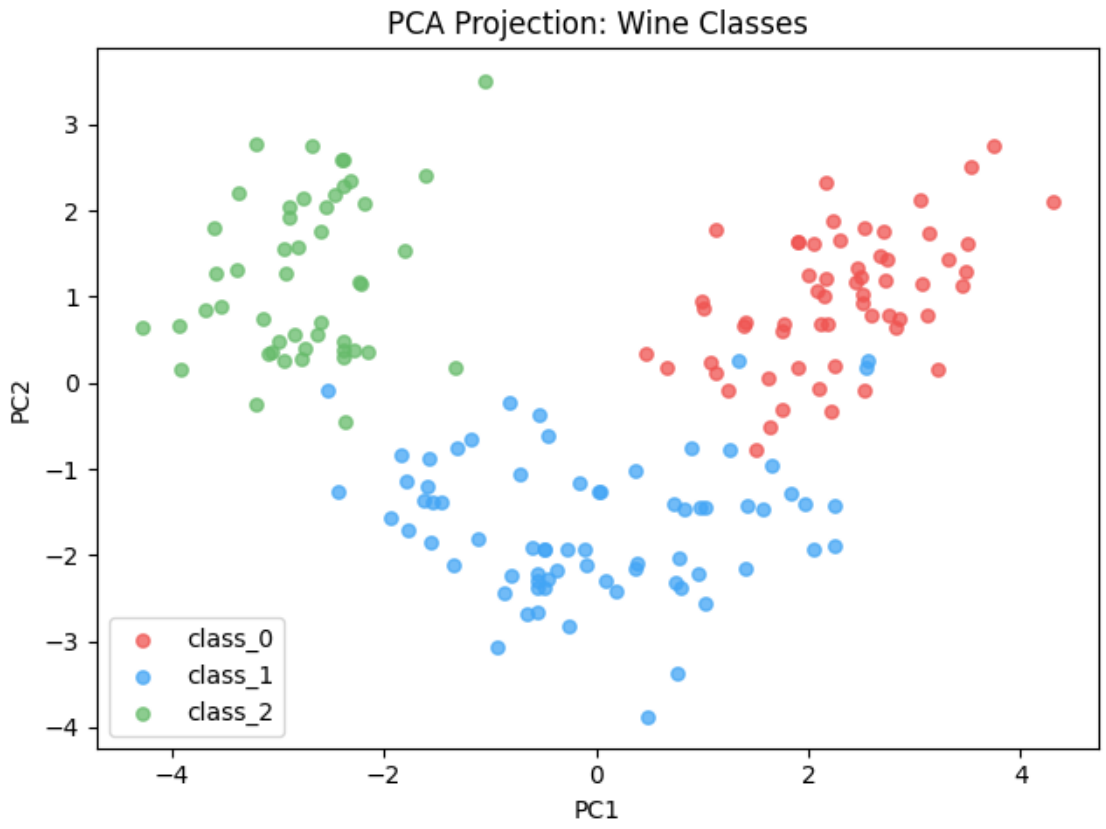
```
Top absolute contributors to PC1:
flavanoids                      0.422934
total_phenols                   0.394661
od280/od315_of_diluted_wines    0.376167
proanthocyanins                 0.313429
nonflavanoid_phenols            0.298533
hue                             0.296715
proline                         0.286752
malic_acid                      0.245188
Name: PC1_loading, dtype: float64
```

2 GRAPHS

[8]:
```
pca_vis = PCA()
X_scaled = StandardScaler().fit_transform(X)
pca_vis.fit(X_scaled)

exp = pca_vis.explained_variance_ratio_
cum = np.cumsum(exp)

fig, axes = plt.subplots(1, 2, figsize=(12, 4.6))

axes[0].bar(range(1, len(exp) + 1), exp, color="#42a5f5")
axes[0].set_xlabel("Principal Component")
axes[0].set_ylabel("Explained Variance Ratio")
axes[0].set_title("Variance by Component")

axes[1].plot(range(1, len(cum) + 1), cum, marker="o", linewidth=1.5,␣
 ↪color="#2e7d32")
axes[1].axhline(0.95, color="red", linestyle="--", label="95% threshold")
axes[1].set_xlabel("Number of Components")
axes[1].set_ylabel("Cumulative Explained Variance")
axes[1].set_title("Cumulative Variance Curve")
axes[1].legend()

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



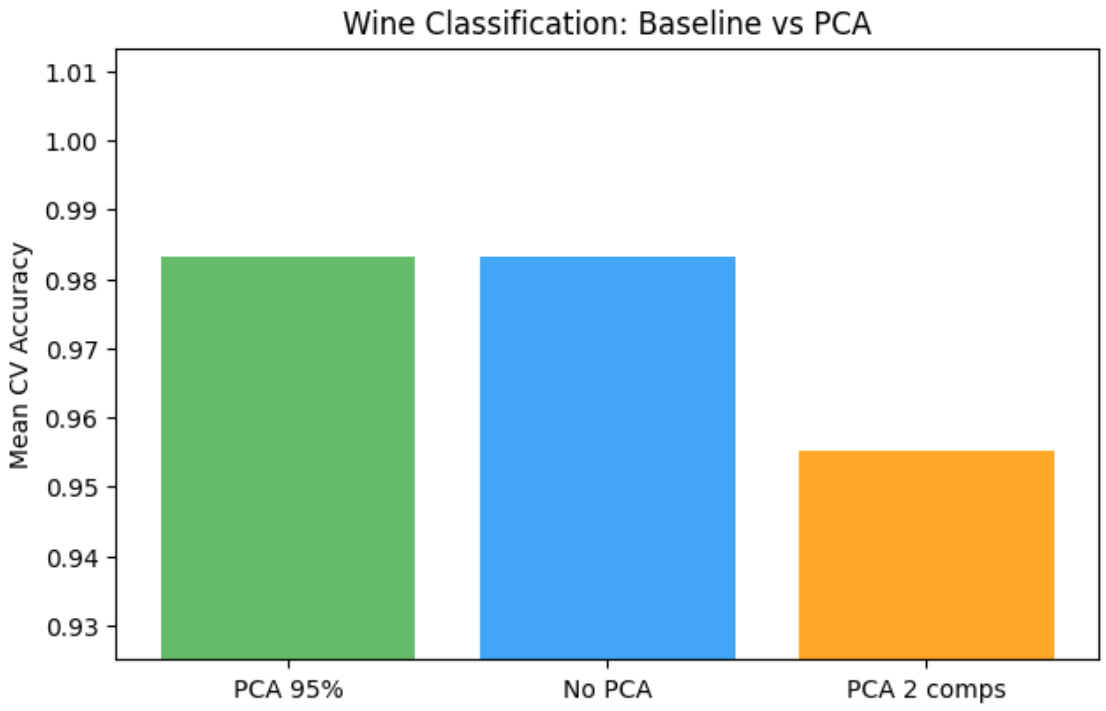
[9]:
```
X_scaled = StandardScaler().fit_transform(X)
X_2d = PCA(n_components=2).fit_transform(X_scaled)

plt.figure(figsize=(7.4, 5.2))
colors = ["#ef5350", "#42a5f5", "#66bb6a"]
for cls, color, name in zip(sorted(np.unique(y)), colors, data.target_names):
    mask = (y == cls)
    plt.scatter(X_2d[mask, 0], X_2d[mask, 1], s=28, alpha=0.75, label=name,␣
 ↪color=color)

plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("PCA Projection: Wine Classes")
plt.legend()
plt.show()
```



[10]:
```
models = {
    "No PCA": make_pipeline(StandardScaler(),␣
 ↪LogisticRegression(max_iter=5000)),
    "PCA 95%": make_pipeline(StandardScaler(), PCA(n_components=0.95),␣
 ↪LogisticRegression(max_iter=5000)),
    "PCA 2 comps": make_pipeline(StandardScaler(), PCA(n_components=2),␣
 ↪LogisticRegression(max_iter=5000))
}

rows = []
for name, model in models.items():
    score = cross_val_score(model, X, y, cv=5, scoring="accuracy").mean()
    rows.append((name, score))

comp = pd.DataFrame(rows, columns=["model", "cv_accuracy"]).
 ↪sort_values("cv_accuracy", ascending=False)
print(comp)

plt.figure(figsize=(7.2, 4.5))
```

```
plt.bar(comp["model"], comp["cv_accuracy"], color=["#66bb6a", "#42a5f5",␣
  ↪"#ffa726"])
plt.ylabel("Mean CV Accuracy")
plt.title("Wine Classification: Baseline vs PCA")
plt.ylim(comp["cv_accuracy"].min() - 0.03, comp["cv_accuracy"].max() + 0.03)
plt.show()
```

```
         model  cv_accuracy
1      PCA 95%     0.983333
0       No PCA     0.983175
2  PCA 2 comps     0.955079
```



[11]:
```
pca_2 = PCA(n_components=2)
X_scaled = StandardScaler().fit_transform(X)
pca_2.fit(X_scaled)

load = pd.DataFrame({
    "feature": X.columns,
    "pc1_loading": pca_2.components_[0]
})
load["abs_loading"] = load["pc1_loading"].abs()
top = load.sort_values("abs_loading", ascending=False).head(10).
  ↪sort_values("pc1_loading")
```

```
colors = ["#ef5350" if v < 0 else "#66bb6a" for v in top["pc1_loading"]]

plt.figure(figsize=(8.4, 5))
plt.barh(top["feature"], top["pc1_loading"], color=colors)
plt.axvline(0, color="black", linewidth=1)
plt.xlabel("PC1 Loading")
plt.title("Top PC1 Contributors (Wine)")
plt.tight_layout()
plt.show()
```