МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙГОСУДАРСТВЕННЫЙТЕХНИЧЕСКИЙУНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №1

Специальность ИИ-23

Выполнил:

Глухарев Д.Е.

студент группы ИИ-23

Проверила:
К.В. Андренко,
«—» ———— 2025 г.

Брест 2025

Цель работы: научиться применять метод PCA для осуществления визуализации данных

Общее задание

1. Используя выборку по варианту, осуществить проецирование данных на плоскость первых двух и трех главных компонент (двумя способами: 1. вручную через использование numpy.linalg.eig для вычисления собственных значений и собственных векторов и 2. с помощью sklearn.decomposition.PCA для непосредственного применения метода PCA – два независимых варианта решения);
2. Выполнить визуализацию полученных главных компонент с использованием средств библиотеки matplotlib, обозначая экземпляры разных классов с использованием разных цветовых маркеров;
3. Используя собственные значения, рассчитанные на этапе 1, вычислить потери, связанные с преобразованием по методу PCA. Сделать выводы;
4. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Задание по варианту:

№ варианта	Выборка	Класс
5	wholesale+customers.zip	Region

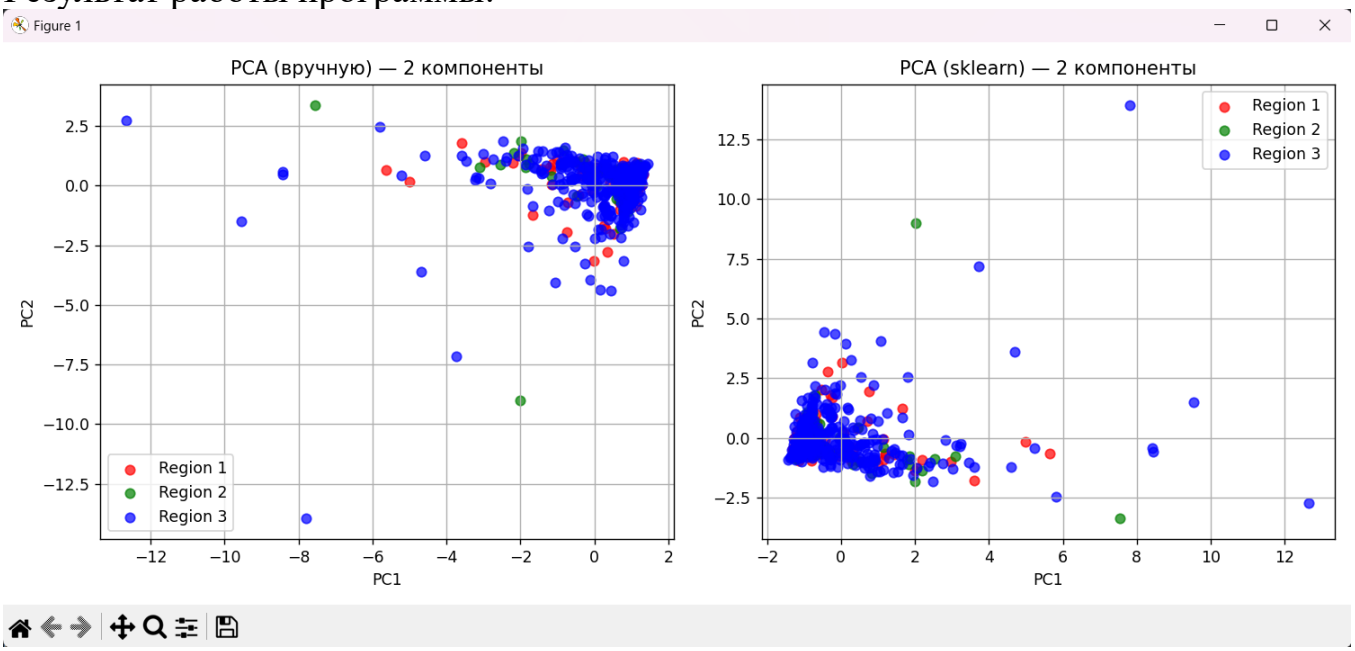
Ход работы:

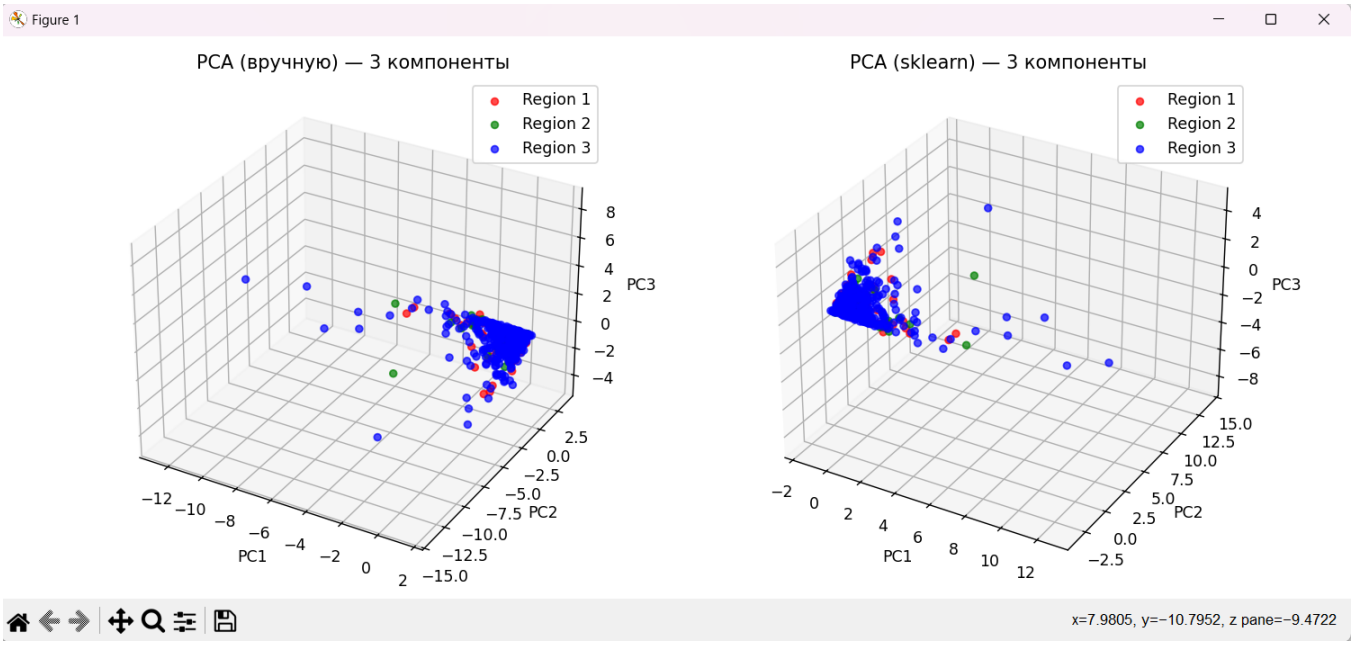
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D

df = pd.read_csv('wholesale.csv')

feature_columns = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']
X = df[feature_columns].values
y = df['Region'].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

cov_matrix = np.cov(X_scaled.T)

eigenvals, eigenvecs = np.linalg.eig(cov_matrix)

eigenvals = np.real(eigenvals)
eigenvecs = np.real(eigenvecs)

sorted_indices = np.argsort(eigenvals)[::-1]
eigenvals_sorted = eigenvals[sorted_indices]
eigenvecs_sorted = eigenvecs[:, sorted_indices]

X_pca_manual_2d = X_scaled @ eigenvecs_sorted[:, :2]
X_pca_manual_3d = X_scaled @ eigenvecs_sorted[:, :3]
```

```
pca_sklearn_2 = PCA(n_components=2)
X_pca_sklearn_2d = pca_sklearn_2.fit_transform(X_scaled)

pca_sklearn_3 = PCA(n_components=3)
X_pca_sklearn_3d = pca_sklearn_3.fit_transform(X_scaled)

print("Совпадение 2D (ручной vs sklearn):", np.allclose(np.abs(X_pca_manual_2d),
np.abs(X_pca_sklearn_2d), atol=1e-6))
print("Совпадение 3D (ручной vs sklearn):", np.allclose(np.abs(X_pca_manual_3d),
np.abs(X_pca_sklearn_3d), atol=1e-6))

regions = np.unique(y)
colors = ['red', 'green', 'blue']

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

plt.subplot(1, 2, 1)
for i, region in enumerate(regions):
    mask = (y == region)
    plt.scatter(X_pca_manual_2d[mask, 0], X_pca_manual_2d[mask, 1],
                label=f'Region {region}', color=colors[i], alpha=0.7)
plt.title('PCA (вручную) — 2 компоненты')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
for i, region in enumerate(regions):
    mask = (y == region)
    plt.scatter(X_pca_sklearn_2d[mask, 0], X_pca_sklearn_2d[mask, 1],
                label=f'Region {region}', color=colors[i], alpha=0.7)
plt.title('PCA (sklearn) — 2 компоненты')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

fig = plt.figure(figsize=(12, 5))

ax1 = fig.add_subplot(121, projection='3d')
for i, region in enumerate(regions):
    mask = (y == region)
    ax1.scatter(X_pca_manual_3d[mask, 0], X_pca_manual_3d[mask, 1], X_pca_manual_3d[mask, 2],
                label=f'Region {region}', color=colors[i], alpha=0.7)
ax1.set_title('PCA (вручную) — 3 компоненты')
ax1.set_xlabel('PC1')
ax1.set_ylabel('PC2')
ax1.set_zlabel('PC3')
ax1.legend()

ax2 = fig.add_subplot(122, projection='3d')
for i, region in enumerate(regions):
    mask = (y == region)
    ax2.scatter(X_pca_sklearn_3d[mask, 0], X_pca_sklearn_3d[mask, 1], X_pca_sklearn_3d[mask, 2],
                label=f'Region {region}', color=colors[i], alpha=0.7)
ax2.set_title('PCA (sklearn) — 3 компоненты')
ax2.set_xlabel('PC1')
ax2.set_ylabel('PC2')
ax2.set_zlabel('PC3')
ax2.legend()
```

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

total_variance = np.sum(eigenvals_sorted)
explained_2 = np.sum(eigenvals_sorted[:2])
explained_3 = np.sum(eigenvals_sorted[:3])

loss_2d = 1 - explained_2 / total_variance
loss_3d = 1 - explained_3 / total_variance

print(f"\n--- Потери при PCA ---")
print(f"Потери при проекции на 2 компоненты: {loss_2d:.4f} ({loss_2d*100:.2f}%)")
print(f"Потери при проекции на 3 компоненты: {loss_3d:.4f} ({loss_3d*100:.2f}%)")

print(f"\nДоля объяснённой дисперсии (sklearn):")
print(f"PC1: {pca_sklearn_2.explained_variance_ratio_[0]:.4f}")
print(f"PC2: {pca_sklearn_2.explained_variance_ratio_[1]:.4f}")
print(f"Сумма первых 2: {np.sum(pca_sklearn_2.explained_variance_ratio_):.4f}")
print(f"Сумма первых 3: {np.sum(pca_sklearn_3.explained_variance_ratio_):.4f}")

print("\n--- Выводы ---")
print("1. PCA успешно выполнен двумя способами — результаты совпадают (с точностью до знака).")
print("2. Визуализация показывает, что классы (Region) не сильно разделимы в пространстве первых 2–3 главных компонент.")
print(f"3. При использовании 2 компонент теряется ~{loss_2d*100:.1f}% информации.")
print(f"   При использовании 3 компонент — ~{loss_3d*100:.1f}%.")
print("4. Для сохранения >90% дисперсии, вероятно, потребуется больше 3 компонент.")
```

Результат работы программы:





Вывод: научился применять метод PCA для осуществления визуализации данных