МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе № 2

по дисциплине «Машинное обучение»

Тема: Понижение размерности пространства признаков

Студенты гр. 6304 _____ Григорьев И.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы

Ознакомиться с методами понижения размерности данных из библиотеки *Scikit Learn*.

Ход работы

Загрузка данных

1. Датасет загружен в датафрейм. Данные разделены на описательные признаки и метки классов. Произведена нормировка данных к интервалу [0, 1].
2. Построены диаграммы рассеяния для пар признаков, которые приведены на рис. 1. Соответствие цвета на диаграмме и класса в датасете приведены на рис. 2.

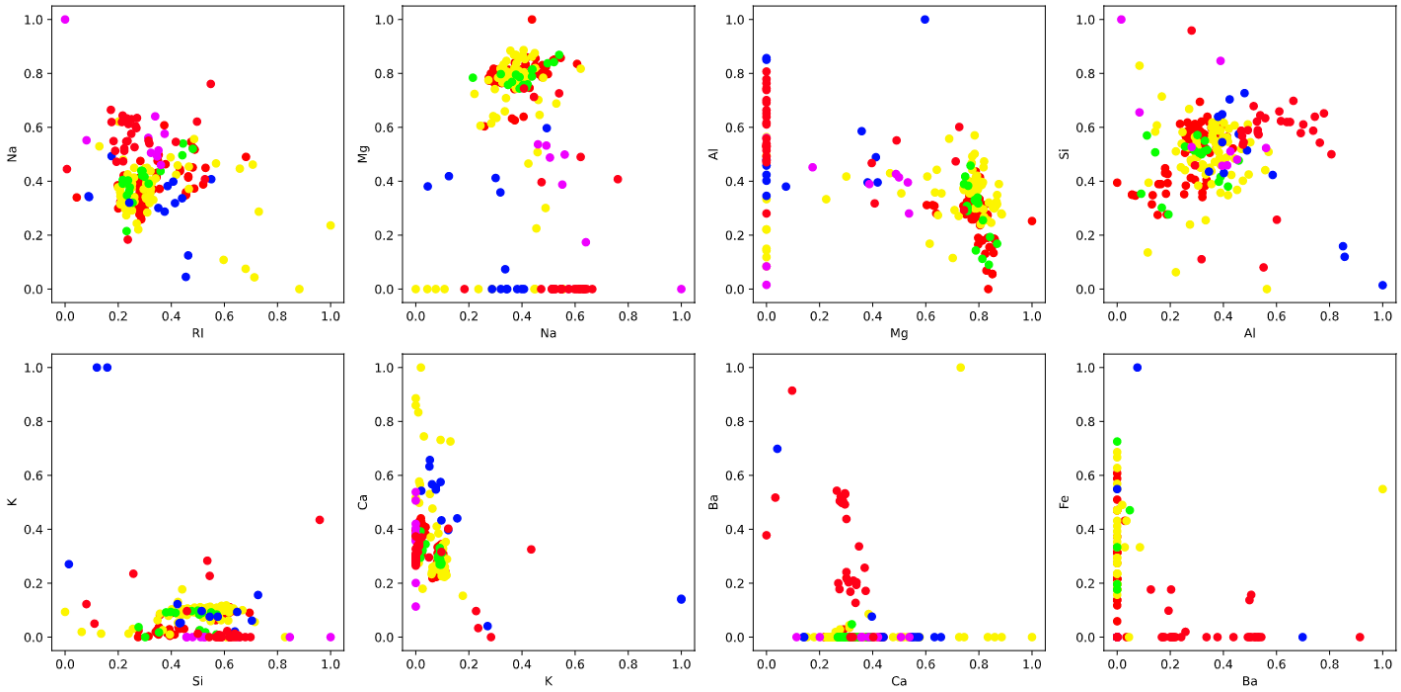

Рисунок 1 – Диаграммы рассеяния для пар признаков

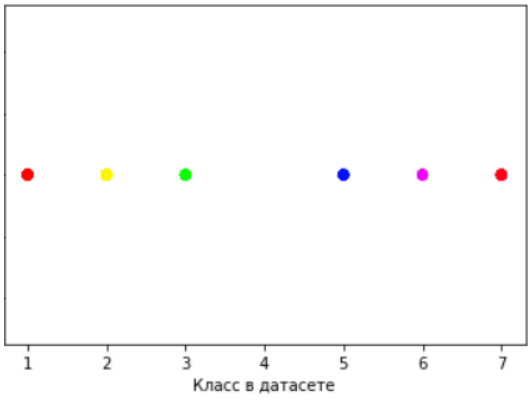

Рисунок 2 – Соответствие цвета и класса

Метод главных компонент

1. Произведено понижение размерности пространства данных с 9 до 2 с помощью PCA.
2. Значение объясненной дисперсии и собственные числа полученных компонентов представлены в табл. 1.

Таблица 1 – Собственные числа и объясненная дисперсия

	Первый компонент	Второй компонент
Объясненная дисперсия (доля от общей)	0.45429569	0.17990097
Собственные числа	5.1049308	3.21245688

Из табл. 1 видно, что такие данные имеют около 63.4% информации.

3. Диаграмма рассеяния для двухкомпонентных данных после применения PCA продемонстрирована на рис. 3.

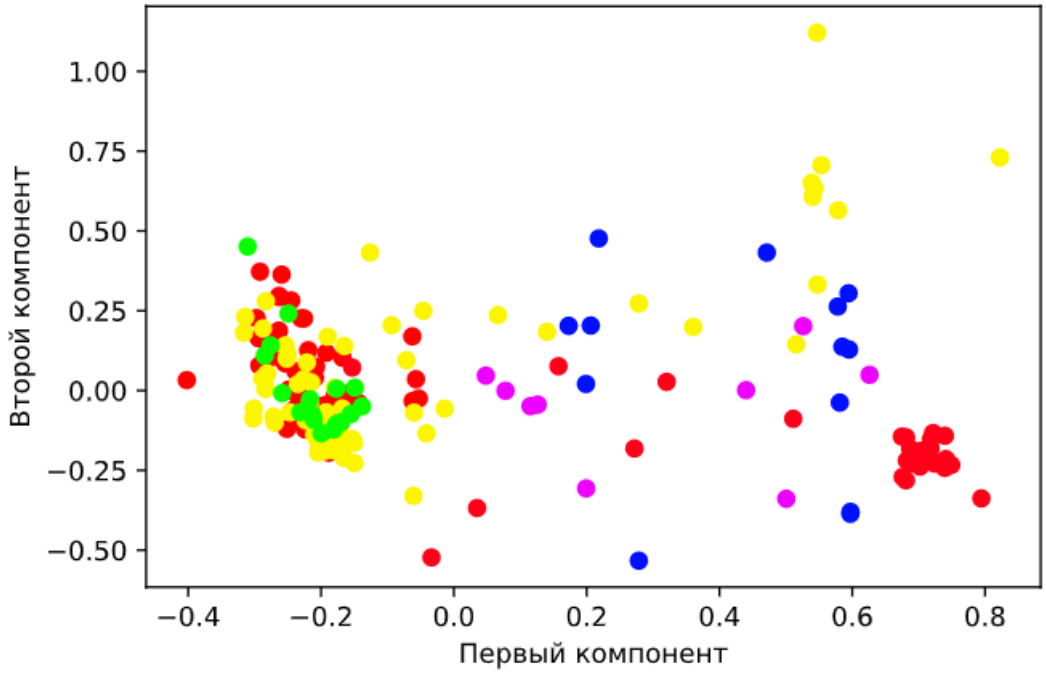

Рисунок 3 – Двухкомпонентная диаграмма рассеяния

Из рис. 3 можно заметить, что данные выходят за границы интервала [0,1].

4. В ходе применения PCA для разного количества компонент выявлено, что данные содержат не менее 85% информации при количестве компонент равным 4.
5. Данные восстановлены после PCA для 4 компонент. Диаграммы рассеяния для пар признаков восстановленных данных представлены на рис. 4.

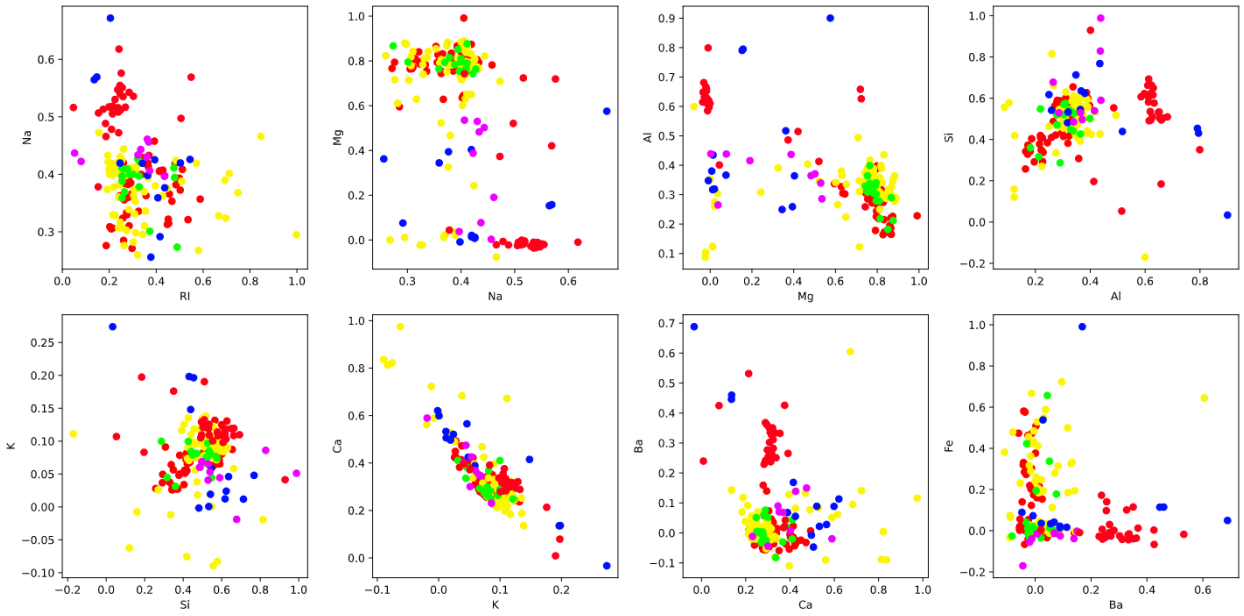

Рисунок 4 – Диаграммы рассеяния для пар признаков (восстановленные данные)

Из-за потери информации в приблизительно 15% восстановленные данные отличаются от исходных, однако имеют похожую форму (среднее не изменилось, но дисперсия искажена).

6. PCA исследован для различных параметров svd_solver (full, arpack, randomized). Различные методы решения SVD дают один и тот же результат для имеющегося набора данных.

Модификации метода главных компонент

1. KernelPCA позволяет нелинейно уменьшать размерность с помощью использования ядерных функций, представленных в табл. 2.

Таблица 2 – Ядерные функции KernelPCA

kernel (параметр KernelPCA)	Ядерная функция
linear	$k(x, y) = x^T y$

poly	$k(x,y) = (\gamma x^T y + c_0)^d$
rbf	$k(x,y) = \exp(-\gamma \|x-y\|^2)$
sigmoid	$k(x,y) = \tanh(\gamma x^T y + c_0)$
cosine	$k(x,y) = \frac{xy^T}{\|x\|\|y\|}$
precomputed	Для возможности задать любое ядро

В ходе применения различных ядерных функций выяснено, что собственные значения ядерных матриц различаются, однако обобщенная дисперсия различается незначительно. Диаграммы для каждой ядерной функции приведены на рис. 5-9.

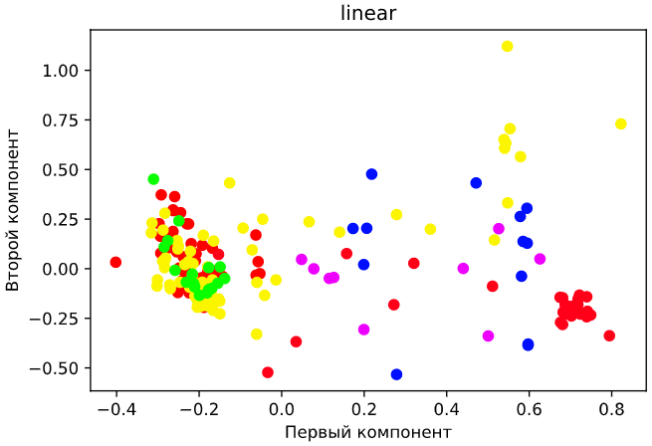

Рисунок 5 – KernelPCA для линейного ядра

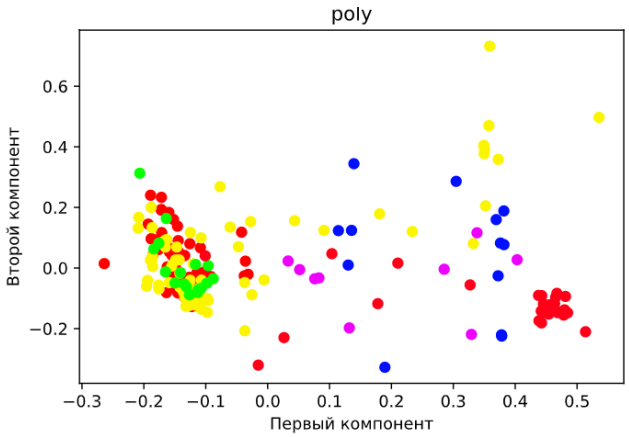

Рисунок 6 – KernelPCA для полиномиального ядра

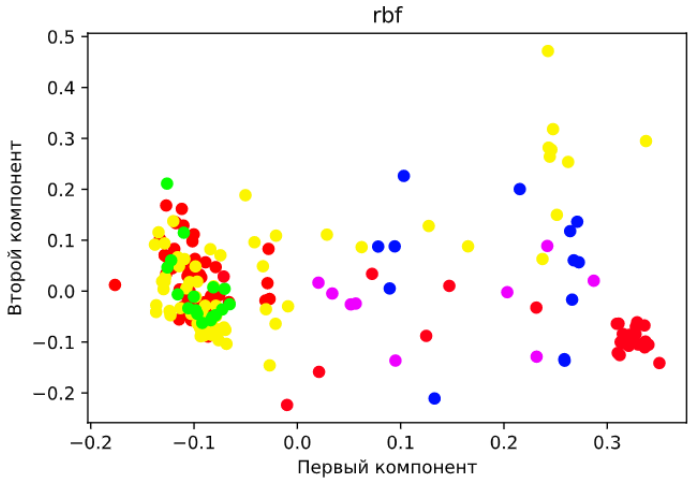

Рисунок 7 – KernelPCA для ядра с радиально-базисной функцией

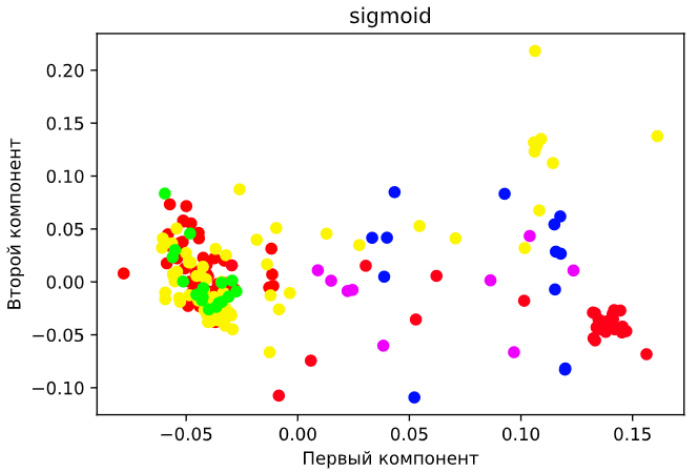

Рисунок 8 – KernelPCA для ядра с функцией сигмоиды

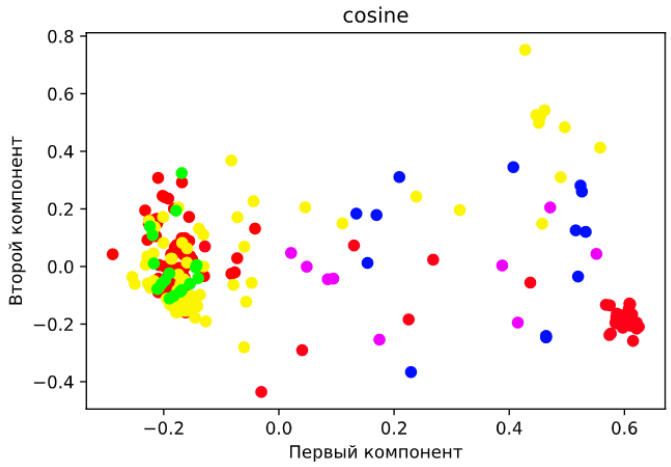

Рисунок 9 – KernelPCA для ядра с функцией косинуса

2. KernelPCA работает аналогично PCA при линейной ядерной функции (*kernel=”linear”*).
3. SparsePCA извлекает набор разреженных компонентов, которые лучше всего восстанавливают данные.

В качестве решения имеется возможность использовать один из следующих методов: наименьшая угловая регрессия (least angle regression) и координатный спуск (coordinate descent). Метод lars быстрее, если компоненты разрежены. При прочих равных результаты применения методов не различаются. Результат продемонстрирован на рис. 10.

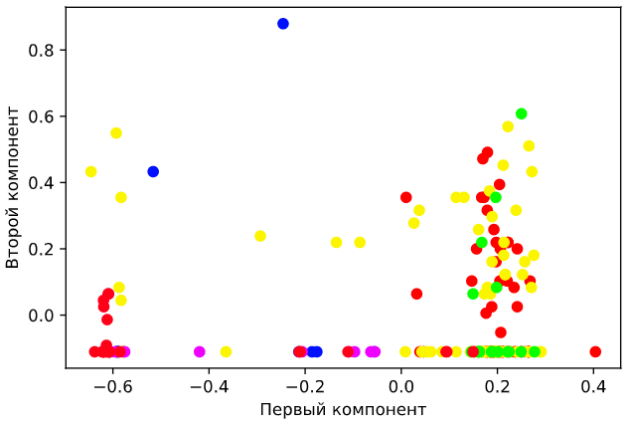

Рисунок 10 – Результат SparsePCA

SparsePCA вычисляет различные результаты при изменении параметра, контролирующего разреженность (*alpha*). Чем выше значение, тем компоненты более разреженные. Результат представлен на рис. 11.

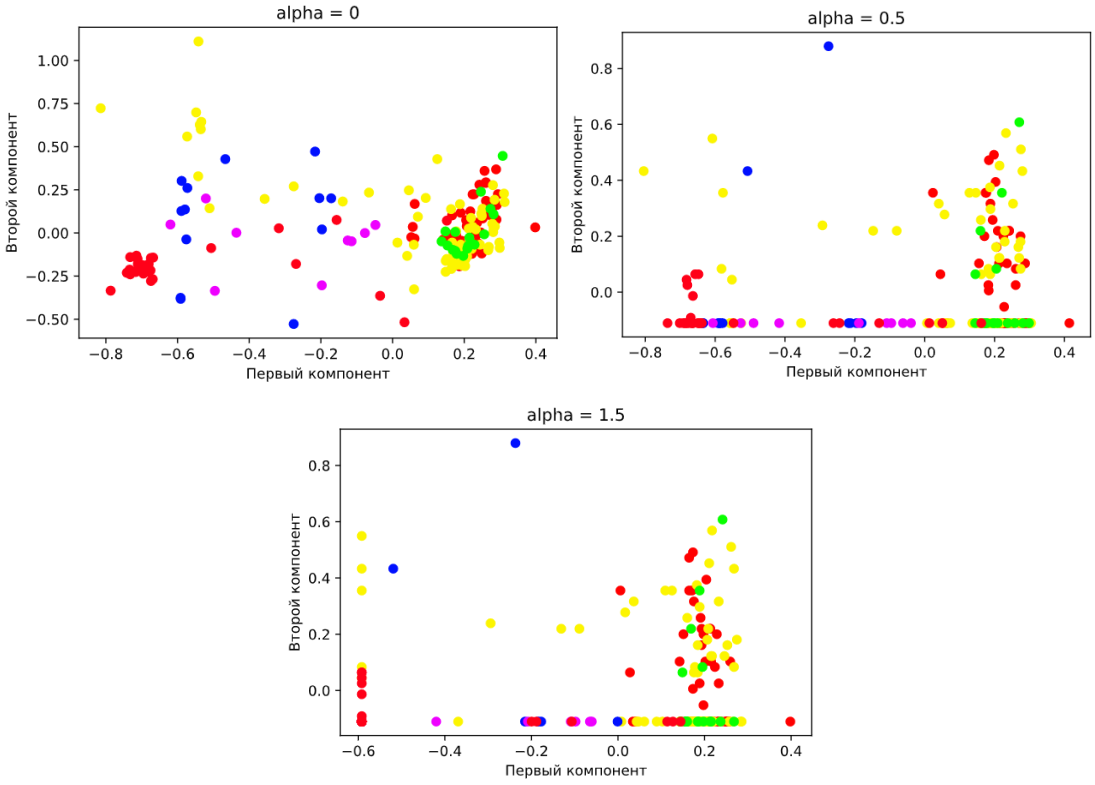

Рисунок 11 – SparsePCA для различных *alpha*

Факторный анализ

Результат факторного анализа продемонстрирован на рис. 12. Имеются различия с результатом PCA: видны 2 фактора с отрицательной корреляцией.

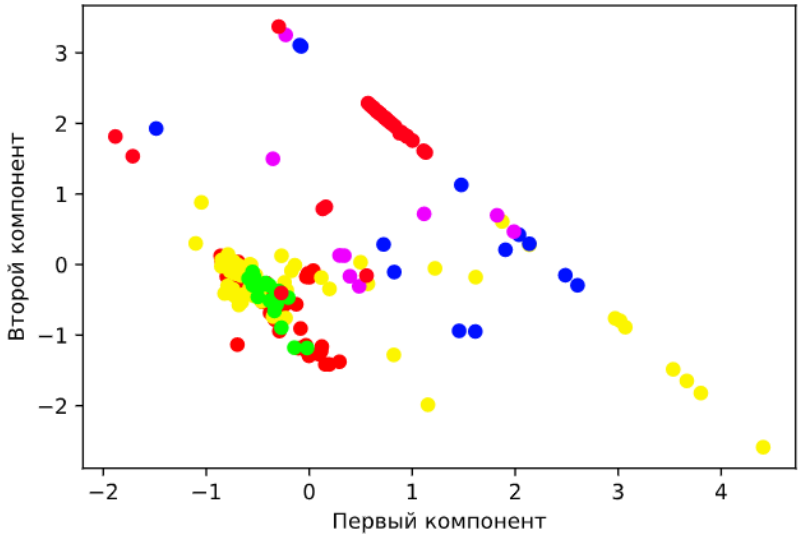

Рисунок 12 – Результат факторного анализа

Основные различия PCA и FA:

1. В основном PCA используется для уменьшения размерности данных, а FA для поиска скрытых факторов.
2. FA объясняет ковариацию или корреляцию между переменными, а PCA объясняет большую часть общей дисперсии.
3. В PCA компоненты вычисляются как линейные комбинации переменных. В FA исходные переменные определены как линейные комбинации факторов.

Выводы

В ходе лабораторной работы изучены методы понижения размерности данных из библиотеки *Scikit Learn*: метод главных компонент PCA и его модификации KernelPCA и SparsePCA. Также для понижения размерности был использован факторный анализ FA. Приведены основные различия FA и PCA.

Приложение А

Код программы на python

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %% [markdown]
# # Лабораторная работа №2. Понижение размерности пространства признаков
# ## Загрузка данных

# %%
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA, KernelPCA, SparsePCA, FactorAnalysis

# %%
df = pd.read_csv('glass.csv')

var_names = list(df.columns) #получение имен признаков

labels = df.to_numpy('int')[:,-1] #метки классов
data = df.to_numpy('float')[:,:-1] #описательные признаки

# %%
data = preprocessing.minmax_scale(data)

# %%
def show_scatters(data):
    fig, axs = plt.subplots(2, 4, figsize=(16, 8))
    for i in range(data.shape[1] - 1):
        axs[i // 4, i % 4].scatter(data[:,i], data[:,(i+1)], c=labels, cmap='hsv')
        axs[i // 4, i % 4].set_xlabel(var_names[i])
        axs[i // 4, i % 4].set_ylabel(var_names[i + 1])

    fig.tight_layout()
    plt.show()

# %%
show_scatters(data)

# %%
plt.scatter(labels, [0]*len(labels), c=labels, cmap='hsv')
plt.xlabel('Класс в датасете')
plt.show()

# %% [markdown]
# ## Метод главных компонент

# %%
pca = PCA(n_components = 4)
pca_data = pca.fit_transform(data)

# %%
pca.explained_variance_ratio_

# %%
sum(pca.explained_variance_ratio_)

# %%
pca.singular_values_
```

```
# %%
np.sum(pca.singular_values_**2) / np.sum(PCA(n_components = data.shape[1]).fit(data).
singular_values_**2)

# %%
def show_scatter(data):
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='hsv')
    plt.xlabel('Первый компонент')
    plt.ylabel('Второй компонент')
    plt.show()

# %%
show_scatter(pca_data)

# %%
recovered_data = pca.inverse_transform(pca_data)
print('Mean', np.mean(data, axis=0))
print('Recovered mean', np.mean(recovered_data, axis=0), '\n')
print('Var', np.var(data, axis=0))
print('Recovered var', np.var(recovered_data, axis=0))

# %%
show_scatters(recovered_data)

# %%
pca_full = PCA(n_components = 2, svd_solver='full')
pca_full_svd_data = pca_full.fit_transform(data)

pca_arpack = PCA(n_components = 2, svd_solver='arpack')
pca_arpack_svd_data = pca_arpack.fit_transform(data)

pca_randomized = PCA(n_components = 2, svd_solver='randomized')
pca_randomized_svd_data = pca_randomized.fit_transform(data)

# %%
def print_pca(pca):
    print(pca.explained_variance_ratio_, sum(pca.explained_variance_ratio_))
    print(pca.singular_values_)
    print('\n')

print_pca(pca)
print_pca(pca_full)
print_pca(pca_arpack)
print_pca(pca_randomized)

# %%
show_scatter(pca_full_svd_data)

# %%
show_scatter(pca_arpack_svd_data)

# %%
show_scatter(pca_randomized_svd_data)

# %% [markdown]
# ## Модификации метода главных компонент

# %%
def get_kernel_pca_data(kernel='linear'):
    kernel_pca = KernelPCA(n_components=4, kernel=kernel)
    kernel_pca_all = KernelPCA(n_components=data.shape[1], kernel=kernel).fit(data)
```

```
    kernel_pca_data = kernel_pca.fit_transform(data)
    print('eigenvals', kernel_pca.lambdas_)
    print('var', np.sum(kernel_pca.lambdas_) / np.sum(kernel_pca_all.lambdas_))
    plt.title(kernel)
    show_scatter(kernel_pca_data)

# %%
get_kernel_pca_data(kernel='linear')

# %%
get_kernel_pca_data(kernel='poly')

# %%
get_kernel_pca_data(kernel='rbf')

# %%
get_kernel_pca_data(kernel='sigmoid')

# %%
get_kernel_pca_data(kernel='cosine')

# %%
sparse_pca = SparsePCA(n_components=2, method='lars')
sparse_pca_data = sparse_pca.fit_transform(data)
show_scatter(sparse_pca_data)

# %%
sparse_pca = SparsePCA(n_components=2, method='cd')
sparse_pca_data = sparse_pca.fit_transform(data)
show_scatter(sparse_pca_data)

# %%
sparse_pca = SparsePCA(n_components=2, method='lars', alpha=0)
sparse_pca_data = sparse_pca.fit_transform(data)
plt.title('alpha = 0')
show_scatter(sparse_pca_data)

# %%
sparse_pca = SparsePCA(n_components=2, method='lars', alpha=0.5)
sparse_pca_data = sparse_pca.fit_transform(data)
plt.title('alpha = 0.5')
show_scatter(sparse_pca_data)

# %%
sparse_pca = SparsePCA(n_components=2, alpha=1.5)
sparse_pca_data = sparse_pca.fit_transform(data)
plt.title('alpha = 1.5')
show_scatter(sparse_pca_data)

# %%
fa = FactorAnalysis(n_components=2)
fa_data = fa.fit_transform(data)
show_scatter(fa_data)
```