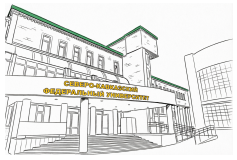

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Департамент цифровых, робототехнических систем и электроники

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ВИЗУАЛИЗАЦИИ ДАННЫХ

Методические указания
к выполнению лабораторных работ

Направление подготовки: 09.03.02 «Информационные системы и технологии»
Профиль: «Прикладное программирование и интернет-технологии»

Квалификация (степень) выпускника: бакалавр
Форма обучения: очная

Ставрополь – 2026

УДК 004.85:519.6
ББК 32.973.26-018.2

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ВИЗУАЛИЗАЦИИ ДАННЫХ

Методические указания к лабораторным работам

Аннотация: Методические указания содержат комплект из 9 лабораторных работ по дисциплине «Инструментальные средства визуализации данных». Каждая работа включает цель, задачи, теоретический минимум, пошаговый разбор примера, фрагменты программного кода для построения визуализаций, индивидуальное задание для самостоятельного выполнения, а также вопросы для самоконтроля. Материал охватывает ключевые методы представления данных: от базовых статических графиков до интерактивных дашбордов и визуального анализа многомерных структур. Рекомендуемый объем — 36 часов лабораторных занятий. Работы выполняются в среде Python с использованием библиотек Matplotlib, Seaborn, Plotly, Bokeh и Pandas.
Составитель: к.т.н. Д.Л. Винокурский
Рецензент: к.ф.м.н. Е.В. Крахоткина

Содержание

Введение

Дисциплина «Инструментальные средства визуализации данных» формирует у обучающихся компетенции в области эффективного представления, анализа и интерпретации информационных массивов. Умение выбирать адекватные визуальные формы, комбинировать методы отображения и создавать интерактивные интерфейсы для исследования данных является критически важным навыком современного специалиста в области информационных технологий.

Актуальность дисциплины обусловлена следующими факторами:

- Экспоненциальный рост объёмов данных требует новых подходов к их осмыслению и презентации;
- Визуализация снижает когнитивную нагрузку и ускоряет процесс принятия решений;
- Интерактивные дашборды становятся стандартом в бизнес-аналитике и научных исследованиях;
- Развитие веб-технологий открывает возможности для создания доступных и масштабируемых визуальных решений.

Цель методических указаний — предоставить обучающимся структурированный практический материал для освоения современных инструментов визуализации данных.

Задачи лабораторного практикума:

1. Сформировать навыки работы с библиотеками визуализации Python (Matplotlib, Seaborn, Plotly);
2. Научить выбирать тип графика в зависимости от природы данных и цели анализа;
3. Освоить техники кодирования визуальных переменных (цвет, размер, форма, позиция);
4. Развить способность критически оценивать качество и информативность визуализаций;
5. Подготовить к разработке интерактивных отчётов и дашбордов для прикладных задач.

Организация выполнения работ:

- Каждая лабораторная работа рассчитана на 2 академических часа;
- Перед началом работы студент обязан изучить теоретический материал и подготовить среду выполнения;
- Экспериментальная часть выполняется с использованием предоставленных вычислительных ресурсов;
- Отчёт по работе должен содержать: цель, описание данных, листинг кода, скриншоты визуализаций, выводы и ответы на контрольные вопросы;
- Защита работы включает демонстрацию работоспособности кода и обоснование выбранных визуальных решений.

Требования к программному обеспечению:

- Язык программирования: Python 3.8+;
- Библиотеки: NumPy, Pandas, Matplotlib, Seaborn, Plotly, Bokeh, Scikit-learn;
- Среда разработки: Jupyter Notebook или JupyterLab;
- Дополнительно: Dash/Streamlit для создания дашбордов, GeoPandas для геопространственных данных.

1 Введение в инструменты визуализации. Базовые принципы

Цель: Освоить фундаментальные принципы визуального восприятия и базовые возможности библиотек визуализации Python.

Задачи:

- Изучить этапы конвейера визуализации: данные → преобразование → визуальное кодирование → восприятие;
- Освоить синтаксис Matplotlib: фигуры, оси, художники (artists);
- Реализовать базовые типы графиков: линейный, столбчатый, точечный;
- Применить принципы эффективной визуализации: устранение «визуального шума», акцентирование важного.

Теоретическое описание: Визуализация данных — это процесс преобразования абстрактной информации в графическую форму, облегчающую когнитивную обработку. Ключевые принципы (по E. Tufte и C. Ware):

- **Максимум данных, минимум чернил**: каждый элемент графика должен нести информацию;
- **Иерархия восприятия**: важные элементы выделяются размером, цветом, позицией;
- **Контекст и сравнение**: данные должны быть представлены в сопоставимой форме;
- **Честность**: масштаб осей не должен искажать интерпретацию.

Библиотека Matplotlib реализует объектно-ориентированный API: `Figure` (контейнер), `Axes` (область построения), `Artist` (визуальные элементы).

Разобранный пример: 1. Загрузка данных: генерация синтетического набора (время, температура, влажность).
2. Построение: 2×1 сетка графиков (линейный + столбчатый).
3. Оформление: подписи осей, заголовок, легенда, сетка, экспорт в PNG/PDF.
4. Результат: профессионально оформленный график, готовый к публикации.

Программный код (базовая визуализация в Matplotlib):

```
import numpy as np
import matplotlib.pyplot as plt

#
np.random.seed(42)
time = np.linspace(0, 24, 100)  #
temp = 20 + 5*np.sin(2*np.pi*time/24) + np.random.normal(0, 1, 100)
humidity = 60 + 10*np.cos(2*np.pi*time/24) + np.random.normal(0, 2,
    100)

#
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

#              1:
ax1.plot(time, temp, 'r-', linewidth=2, label='
    ')
ax1.set_ylabel('                        ,  C ')
ax1.grid(True, alpha=0.3)
ax1.legend(loc='upper right')
```

```
ax1.set_title('
    ')

#                2:                        (
                                                                    )
hours = np.arange(0, 24)
hum_avg = [humidity[time//1==h].mean() for h in hours]
ax2.bar(hours, hum_avg, color='skyblue', edgecolor='navy', alpha=0.7)
ax2.set_xlabel('           ,    ')
ax2.set_ylabel('                    , %')
ax2.grid(True, axis='y', alpha=0.3)

plt.tight_layout()
plt.savefig('basic_plot.png', dpi=300, bbox_inches='tight')
print("                                  'basic_plot.png'")
plt.show()
```

Индивидуальное задание: Загрузите реальный датасет (например, `seaborn.load_dataset('tips')`). Постройте три различных типа графиков для одних и тех же данных (точечный, гистограмма, boxplot). Сравните информативность каждого представления.

Вопросы для самопроверки:

1. В чём разница между процедурным и объектно-ориентированным интерфейсом Matplotlib?
2. Почему важно указывать единицы измерения на осях графика?
3. Как цветовой код (sequential/diverging/categorical) влияет на восприятие данных?

2 Статические графики: одномерные и двумерные данные

Цель: Научиться выбирать и строить адекватные визуализации для одномерных и двумерных данных.

Задачи:

- Освоить визуализацию распределений: гистограммы, KDE, boxplot, violin plot;
- Реализовать двумерные графики: scatter plot, heatmap, contour plot;
- Применить фасетирование (small multiples) для сравнения подгрупп;
- Настроить параметры визуального кодирования: цвет, размер, прозрачность.

Теоретическое описание: Для **одномерных данных** ключевые задачи — показать распределение, центральную тенденцию и выбросы:

- `hist`: частотное распределение (выбор числа бинов по правилу Стерджеса/Фридмана);
- `kde`: оценка плотности распределения (ядерное сглаживание);
- `boxplot`: пятичисловая сводка (мин, Q1, медиана, Q3, макс) + выбросы;
- `violin`: комбинация boxplot и KDE для отображения формы распределения.

Для **двумерных данных** важны выявление корреляций и паттернов:

- `scatter`: точечная диаграмма с кодированием дополнительных переменных через цвет/размер;
- `hexbin`: агрегация точек в шестиугольные бины для плотных данных;
- `heatmap`: матрица значений с цветовой шкалой (корреляции, таблицы сопряжённости).

Разобранный пример: 1. Данные: `seaborn.penguins` (масса, длина клюва, пол, вид).
2. Визуализация: фасетированный scatter plot (масса vs длина клюва) по видам и полу.
3. Дополнительно: marginal histograms для отображения одномерных распределений.
4. Результат: информативный график, выявляющий межвидовые различия.

Программный код (двумерная визуализация с Seaborn):

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

#
penguins = sns.load_dataset('penguins').dropna()

# Faceted scatter plot

g = sns.JointGrid(data=penguins, x='bill_length_mm', y='body_mass_g',
                  hue='species', height=8, marginal_ticks=True)

#                              : scatter

g.plot_joint(sns.scatterplot, alpha=0.6, s=60)

#                                       : KDE

for species in penguins['species'].unique():
    subset = penguins[penguins['species'] == species]
    g.plot_marginals(sns.kdeplot, data=subset, x='bill_length_mm',
```

```
                        label=species, linewidth=2)
    g.plot_marginals(sns.kdeplot, data=subset, y='body_mass_g',
                        label=species, linewidth=2)

g.add_legend()
g.set_axis_labels('                          ,      ', '
   ,    ')
g.fig.suptitle('
              ', y=1.02, fontsize=14)
plt.tight_layout()
plt.savefig('penguins_scatter.png', dpi=300, bbox_inches='tight')
plt.show()

#                                              :

corr = penguins.select_dtypes(include='number').corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, fmt='.2f', cmap='coolwarm', center=0)
plt.title('
                        ')
plt.tight_layout()
plt.show()
```

Индивидуальное задание: Постройте матрицу попарных графиков (`pairplot`) для датасета `iris`. Добавьте цветовое кодирование по виду и настройте отображение только нижней треугольной матрицы для экономии места.

Вопросы для самопроверки:

1. Когда целесообразно использовать violin plot вместо boxplot?
2. Как интерпретировать перекос (skew) распределения по форме гистограммы?
3. Почему для плотных точечных диаграмм предпочтительнее hexbin, чем обычный scatter?

3 Статистическая визуализация и анализ распределений

Цель: Освоить методы визуальной оценки статистических гипотез и параметров распределений.

Задачи:

- Построить Q-Q plot для проверки нормальности распределения;
- Визуализировать доверительные интервалы и результаты статистических тестов;
- Реализовать визуальное сравнение распределений нескольких групп;
- Применить бутстрэп для оценки неопределённости статистик.

Теоретическое описание: Q-Q plot (quantile-quantile) сравнивает квантили эмпирического распределения с квантилями теоретического (например, нормального). Если точки лежат близко к диагонали — распределение соответствует модели.
Доверительные интервалы визуализируются через error bars, заштрихованные области или «усы» boxplot.
Сравнение групп требует учёта масштаба: использование одинаковых осей, фасетирование или наложение с прозрачностью.
Бутстрэп — ресэмплинг с возвращением для оценки распределения статистики; результаты отображаются как гистограмма бутстрэп-оценок.

Разобранный пример: 1. Данные: две выборки (контрольная и экспериментальная группы).
2. Анализ: Q-Q plot для каждой группы, сравнение медиан с 95% ДИ.
3. Бутстрэп: 1000 ресэмплингов для оценки ДИ разницы медиан.
4. Результат: визуальное подтверждение/опровержение гипотезы о сдвиге распределений.

Программный код (статистическая визуализация):

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.utils import resample

#
np.random.seed(42)
control = np.random.normal(loc=100, scale=15, size=200)
experiment = np.random.normal(loc=108, scale=18, size=200)

# 1. Q-Q plot
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
stats.probplot(control, dist="norm", plot=axes[0])
axes[0].set_title('                                    ')
stats.probplot(experiment, dist="norm", plot=axes[1])
axes[1].set_title('                                      ')
plt.suptitle('Q-Q plot:                                 ')
plt.tight_layout()
plt.show()

# 2.

def bootstrap_ci(data, stat_func, n_boot=1000, alpha=0.05):
```

```
    """                    -
                             """
    boot_stats = [stat_func(resample(data)) for _ in range(n_boot)]
    return np.percentile(boot_stats, [100*alpha/2, 100*(1-alpha/2)])

medians = [np.median(control), np.median(experiment)]
ci_control = bootstrap_ci(control, np.median)
ci_experiment = bootstrap_ci(experiment, np.median)

plt.figure(figsize=(8, 5))
sns.violinplot(data=[control, experiment], inner='box', palette='Set2
   ')
plt.xticks([0, 1], ['                ', '                      '])
plt.ylabel('               ')
#

plt.errorbar([0, 1], medians,
             yerr=[[medians[0]-ci_control[0], medians[1]-
                ci_experiment[0]],
                   [ci_control[1]-medians[0], ci_experiment[1]-
                      medians[1]]],
             fmt='o', color='red', capsize=5, label='
                   95%     ')
plt.legend()
plt.title('
                   -    ')
plt.grid(axis='y', alpha=0.3)
plt.show()

print(f"                          : {medians[1] - medians[0]:.2f}")
print(f"95%                     (               ): {bootstrap_ci(
   experiment - control, np.median)}")
```

Индивидуальное задание: Загрузите датасет `seaborn.tips`. Визуализируйте распределение чаевых (`tip`) для дней недели с помощью фасетированных violin plot. Добавьте статистическую аннотацию: результаты теста Крускала-Уоллиса для проверки гипотезы о различии распределений.

Вопросы для самопроверки:

1. Почему отклонение от диагонали в хвостах Q-Q plot указывает на тяжёлые хвосты распределения?
2. Как интерпретировать перекрывающиеся доверительные интервалы при сравнении групп?
3. В чём преимущество бутстрэпа перед параметрическими методами оценки ДИ?

4 Интерактивная визуализация в веб-среде

Цель: Освоить создание интерактивных графиков с использованием Plotly и Bokeh для веб-развёртывания.

Задачи:

- Изучить различия между статическими (Matplotlib) и интерактивными (Plotly/Bokeh) библиотеками;
- Реализовать базовые интерактивные элементы: тултипы, зум, фильтрация, выделение;
- Создать связные (linked) графики для координированного исследования данных;
- Экспортировать визуализацию в HTML для встраивания в веб-страницу.

Теоретическое описание: Интерактивная визуализация расширяет аналитические возможности за счёт:

- **Детализация по требованию**: тултипы показывают точные значения при наведении;
- **Фокусировка**: зум и панорамирование позволяют исследовать области интереса;
- **Фильтрация**: динамическое исключение/включение подмножеств данных;
- **Связность**: выделение точки на одном графике подсвечивает соответствующие на других.

Plotly использует декларативный синтаксис на основе JSON и рендеринг через WebGL/Canvas. **Bokeh** ориентирован на потоковые данные и серверные приложения.

Разобранный пример: 1. Данные: мировые показатели (ВВП, население, продолжительность жизни).
2. Визуализация: интерактивный scatter plot с анимацией по годам.
3. Интерактив: тултипы с деталями страны, слайдер времени, легенда с фильтрацией.
4. Экспорт: сохранение в standalone HTML-файл.

Программный код (интерактивная визуализация в Plotly):

```
import plotly.express as px
import pandas as pd

#                                    (Gapminder)
df = px.data.gapminder().query("year >= 1990")

#                                scatter plot
fig = px.scatter(df,
                 x="gdpPercap", y="lifeExp",
                 size="pop", color="continent",
                 hover_name="country",
                 animation_frame="year",
                 animation_group="country",
                 log_x=True,  #

                 size_max=60,
                 title="                                  :
                                                                ",
                 labels={"gdpPercap": "
                                          ($)",
                         "lifeExp": "
```

```
                                                    (       )"})
19
20  #
21  fig.update_traces(marker=dict(opacity=0.7, line=dict(width=0.5, color
        ='DarkSlateGray')))
22  fig.update_layout(
23      hovermode="closest",
24      xaxis_title="                                                (       .
                    )",
25      yaxis_title="                                                    ,
                ",
26      legend_title="                      ",
27      height=600
28  )
29
30  #                              HTML
31  fig.write_html("interactive_gapminder.html", include_plotlyjs='cdn')
32  print("                                                                 '
        interactive_gapminder.html'")
33
34  #                             (    Jupyter)
35  fig.show()
```

Индивидуальное задание: Создайте дашборд из двух связанных графиков: (1) scatter plot «доход/образование», (2) гистограмма распределения дохода. Реализуйте кросс-фильтрацию: выделение региона на первом графике обновляет второй. Используйте `plotly.subplots` или `Dash` для компоновки.

Вопросы для самопроверки:

1. В каких случаях интерактивная визуализация предпочтительнее статической?
2. Как оптимизировать производительность при визуализации больших датасетов в Plotly?
3. Почему важно предоставлять альтернативный статический экспорт для интерактивных графиков?

5 Визуализация многомерных данных

Цель: Освоить техники отображения данных с числом признаков $p > 3$.

Задачи:

- Реализовать параллельные координаты и матрицу рассеяния с цветовым кодированием;
- Применить техники визуального уменьшения размерности: glyph plots, radar charts;
- Использовать интерактивное связывание для исследования многомерных паттернов;
- Оценить информативность различных представлений для конкретной задачи.

Теоретическое описание: Проблема многомерной визуализации — ограничение человеческого восприятия 2–3 измерениями. Основные подходы:

- **Проекция**: отображение p мер в 2D/3D с сохранением структуры (PCA, t-SNE — см. следующая работа);
- **Кодирование**: использование визуальных переменных (цвет, размер, форма, текстура) для дополнительных измерений;
- **Фасетирование**: разбиение данных на подгруппы с отображением в отдельных панелях;
- **Интерактивность**: динамическое изменение отображаемых измерений.

Параллельные координаты: каждая ось — признак, каждая линия — наблюдение. Позволяет выявлять кластеры и корреляции, но страдает от загромождения при большом n.

Разобранный пример: 1. Данные: `seaborn.penguins` (6 признаков: 3 количественных + 3 категориальных).
2. Визуализация: параллельные координаты с цветовым кодированием по виду.
3. Дополнительно: интерактивная матрица рассеяния с возможностью выбора признаков.
4. Результат: выявление морфологических паттернов, характерных для каждого вида.

Программный код (многомерная визуализация):

```
import pandas as pd
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt

#
penguins = sns.load_dataset('penguins').dropna()
#

penguins['sex_code'] = penguins['sex'].map({'Male': 1, 'Female': 0}).
    fillna(-1)
penguins['island_code'] = penguins['island'].astype('category').cat.
    codes

# 1.                                                    (Plotly)
fig_parcoords = px.parallel_coordinates(
    penguins,
    color="species",
```

```
    dimensions=["bill_length_mm", "bill_depth_mm", "flipper_length_mm
        ", "body_mass_g"],
    labels={
        "bill_length_mm": "                        ",
        "bill_depth_mm": "                           ",
        "flipper_length_mm": "                        ",
        "body_mass_g": "                      "
    },
    title="                                          :
                                                      "
)
fig_parcoords.update_layout(height=600)
fig_parcoords.write_html("parcoords_penguins.html")
fig_parcoords.show()

# 2.

numeric_cols = ["bill_length_mm", "bill_depth_mm", "flipper_length_mm
    ", "body_mass_g"]
fig_scatter = px.scatter_matrix(
    penguins,
    dimensions=numeric_cols,
    color="species",
    title="
                                                            ",
    height=800
)
fig_scatter.update_traces(diagonal_visible=False)  #

fig_scatter.write_html("scatter_matrix_penguins.html")
fig_scatter.show()

# 3.                                                 : pairplot
    Seaborn
sns.pairplot(penguins, hue="species", vars=numeric_cols[:3],
             plot_kws={"alpha": 0.6, "s": 30}, diag_kind="kde")
plt.suptitle("Pairplot:                                          ", y
    =1.02)
plt.savefig("pairplot_static.png", dpi=300, bbox_inches="tight")
plt.show()
```

Индивидуальное задание: Визуализируйте датасет `seaborn.cars` (или аналогичный с $p \geq 5$ признаков) с помощью radar chart (лепестковой диаграммы) для сравнения нескольких наблюдений. Реализуйте интерактивный выбор сравниваемых объектов.

Вопросы для самопроверки:

1. Почему параллельные координаты чувствительны к порядку осей?
2. Как выбрать оптимальный набор визуальных переменных для кодирования категориальных признаков?
3. В чём ограничение лепестковых диаграмм при сравнении более 5–6 объектов?

6 Визуализация результатов снижения размерности

Цель: Научиться интерпретировать и визуализировать результаты методов уменьшения размерности (PCA, t-SNE, UMAP).

Задачи:

- Реализовать PCA и визуализировать объяснённую дисперсию (scree plot, biplot);
- Построить 2D-проекции t-SNE/UMAP с цветовым кодированием по целевой переменной;
- Оценить сохранение локальной/глобальной структуры при разных гиперпараметрах;
- Сравнить интерпретируемость результатов различных методов.

Теоретическое описание: PCA (Principal Component Analysis) — линейный метод, находящий ортогональные направления максимальной дисперсии. Результаты визуализируются через:

- **Scree plot**: доля объяснённой дисперсии в зависимости от номера компоненты;
- **Biplot**: проекция наблюдений + векторы исходных признаков в пространстве первых двух ПК.

t-SNE (t-distributed Stochastic Neighbor Embedding) — нелинейный метод, сохраняющий локальные соседства. Ключевые параметры: `perplexity` (баланс локаль/глобаль), `learning_rate`.
UMAP (Uniform Manifold Approximation and Projection) — сохраняет как локальную, так и глобальную структуру, работает быстрее t-SNE.

Разобранный пример: 1. Данные: `sklearn.datasets.load`$_d igits(64 \rightarrow$ `10 классов`).
`2. PCA: scree plot показывает, что 2 компоненты объясняют` $\sim 30\%$ `дисперсии.`
`3. t-SNE/UMAP: 2D-проекции чётко разделяют классы цифр.`
`4. Сравнение: UMAP лучше сохраняет глобальную структуру (кластеры цифр 0,1,2... расположены логично).`

`Программный код (визуализация снижения размерности):`

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE, UMAP
from sklearn.preprocessing import StandardScaler

#
digits = load_digits()
X, y = digits.data, digits.target
X_scaled = StandardScaler().fit_transform(X)

# 1. PCA: scree plot    biplot
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
explained = np.cumsum(pca.explained_variance_ratio_)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(explained)+1), explained, 'bo-')
```

```
plt.axhline(0.9, color='r', linestyle='--', label='90%
                    ')
plt.xlabel('                                ');  plt.ylabel('
                                                              ')
plt.title('Scree plot PCA');  plt.grid(True, alpha=0.3);  plt.legend()

# Biplot
plt.subplot(1, 2, 2)
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='tab10',
                      s=20, alpha=0.6, edgecolors='w')
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]*100:.1f}%)')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]*100:.1f}%)')
plt.title('PCA:                              ')
plt.colorbar(scatter, label='          ')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('pca_digits.png', dpi=300)
plt.show()

# 2.                    t-SNE    UMAP
methods = {
    't-SNE': TSNE(n_components=2, perplexity=30, random_state=42,
       n_iter=1000),
    'UMAP': UMAP(n_components=2, random_state=42, n_neighbors=15)
}

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for ax, (name, model) in zip(axes, methods.items()):
    X_emb = model.fit_transform(X_scaled)
    scatter = ax.scatter(X_emb[:, 0], X_emb[:, 1], c=y, cmap='tab10',
                         s=15, alpha=0.7, edgecolors='w')
    ax.set_title(f'{name}:                          ')
    ax.set_xlabel('Dim 1');  ax.set_ylabel('Dim 2')
    ax.grid(True, alpha=0.2)
plt.colorbar(scatter, ax=axes.ravel().tolist(), label='          ')
plt.tight_layout()
plt.savefig('tsne_umap_comparison.png', dpi=300)
plt.show()
```

Индивидуальное задание: Исследуйте влияние параметра perplexity в t-SNE на качество визуализации датасета $\text{load}_digits.2D - \text{perplexity} = [5, 30, 50, 100],.$

Вопросы для самопроверки:

1. Почему PCA может быть недостаточен для визуализации нелинейных многообразий?
2. Как интерпретировать направление векторов в biplot относительно исходных признаков?
3. Почему результаты t-SNE/UMAP не следует использовать для количественных выводов о расстояниях между кластерами?

7 Визуализация временных рядов и геопространственных данных

Цель: Освоить специализированные техники для отображения данных с временной и пространственной привязкой.

Задачи:

- Реализовать визуализацию временных рядов: линии, области, сезонная декомпозиция;
- Применить аннотации для выделения событий и трендов;
- Построить интерактивные карты с использованием GeoPandas и Plotly;
- Визуализировать пространственно-временные паттерны (анимация, small multiples).

Теоретическое описание: Временные ряды требуют учёта порядка наблюдений и возможных сезонных паттернов:

- Линейные графики: основной способ, важно избегать пересечения линий при множестве серий;
- Области (area): показывают вклад компонентов в сумму (stacked) или неопределённость (confidence band);
- Сезонная декомпозиция: разложение на тренд, сезонность, остаток (STL, X-11).

Геопространственные данные визуализируются через:

- Хороплеты: заливка регионов цветом по значению переменной;
- Точечные карты: маркеры с кодированием через размер/цвет;
- Потоковые карты: линии с направлением и толщиной, пропорциональной интенсивности.

Библиотека GeoPandas расширяет Pandas поддержкой геометрий, а Plotly Express предоставляет высокоуровневый интерфейс для интерактивных карт.

Разобранный пример: 1. Временной ряд: ежедневные продажи с сезонностью.
2. Визуализация: линия + скользящее среднее + аннотация пиков.
3. Гео-данные: распределение населения по регионам РФ.
4. Результат: интерактивная хороплет-карта с тултипами и зумом.

Программный код (временные ряды и гео-визуализация):

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
from statsmodels.tsa.seasonal import seasonal_decompose

# =====             1:                              =====
#

dates = pd.date_range('2023-01-01', periods=365, freq='D')
trend = np.linspace(100, 150, 365)
seasonal = 20 * np.sin(2 * np.pi * np.arange(365) / 365 * 4)  # 4

noise = np.random.normal(0, 5, 365)
sales = trend + seasonal + noise

```

```
df_ts = pd.DataFrame({'date': dates, 'sales': sales})
df_ts.set_index('date', inplace=True)

#
decomp = seasonal_decompose(df_ts['sales'], model='additive', period
    =91)
fig, axes = plt.subplots(4, 1, figsize=(12, 10), sharex=True)
df_ts['sales'].plot(ax=axes[0], title='                          ')
decomp.trend.plot(ax=axes[1], title='          ', color='orange')
decomp.seasonal.plot(ax=axes[2], title='                    ', color=
    'green')
decomp.resid.plot(ax=axes[3], title='              ', color='red')
for ax in axes: ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('time_series_decomposition.png', dpi=300)
plt.show()

# =====             2:
                                =====
#             :                                          (
                                        )
regions = pd.DataFrame({
    'region': ['             ', '      ', '                    ', '
                                ', '            '],
    'population': [12.5, 5.4, 1.6, 1.5, 1.3],  #               .
    'lat': [55.75, 59.93, 55.00, 56.83, 55.79],
    'lon': [37.62, 30.36, 82.93, 60.61, 49.12]
})

#                                          (Plotly)
fig_map = px.scatter_mapbox(regions,
                            lat="lat", lon="lon",
                            size="population", color="population",
                            hover_name="region",
                            zoom=3, height=600,
                            title="
                                                    ",
                            color_continuous_scale=px.colors.
                               sequential.Viridis)
fig_map.update_layout(mapbox_style="open-street-map")
fig_map.write_html("geo_map_russia.html")
fig_map.show()

#                                                  :                  (
                        GeoJSON                    )
# (                                                                GeoJSON
              gpd.read_file())
print("      -                                                    '
    geo_map_russia.html'")
```

Индивидуальное задание: Загрузите датасет с временными рядами (например, seaborn.flights или данные о котировках акций). Постройте анимированную визуализацию изменения показателя во времени с использованием Plotly Express

(animation_frame). Добавьте возможность фильтрации по категории.

Вопросы для самопроверки:

1. Почему при визуализации временных рядов важно сохранять пропорции временной оси?
2. Как выбрать между хороплетом и точечной картой для геопространственных данных?
3. В чём преимущество интерактивных карт перед статическими для исследовательского анализа?

8 Разработка дашбордов и интерактивных отчётов

Цель: Освоить создание комплексных интерактивных отчётов с использованием Dash и Streamlit.

Задачи:

- Изучить архитектуру веб-приложений для визуализации данных;
- Реализовать базовый дашборд с фильтрами, графиками и таблицами;
- Настроить реактивность: обновление визуализаций при изменении параметров;
- Обеспечить адаптивность и удобство использования интерфейса.

Теоретическое описание: Дашборд - это единая панель управления, объединяющая ключевые метрики и визуализации для мониторинга и анализа. Принципы эффективного дашборда:

- Иерархия: важные метрики - вверху/слева (паттерн чтения «F»);
- Контекст: каждый график должен иметь заголовок, подписи, легенду;
- Интерактивность: фильтры и детализация по требованию, но без перегрузки;
- Производительность: предзагрузка данных, кэширование, ленивая отрисовка.

Dash (Plotly) - фреймворк на основе React/Flask для создания аналитических веб-приложений на Python. Streamlit - более простой инструмент для быстрого прототипирования с декларативным синтаксисом.

Разобранный пример: 1. Данные: продажи по категориям, регионам, времени.
2. Компоненты: KPI-карточки, фильтр по дате/региону, линейный график, таблица.
3. Реактивность: выбор региона в выпадающем списке обновляет все графики.
4. Развёртывание: запуск локально или деплой на Heroku/Streamlit Cloud.

Программный код (простой дашборд на Streamlit):

```
# dashboard.py --              : streamlit run dashboard.py
import streamlit as st
import pandas as pd
import plotly.express as px
import numpy as np

#
st.set_page_config(page_title="                                    ",
    layout="wide")

#
st.title("                                                          ")
st.markdown("
                        ")

#
@st.cache_data
def load_data():
    dates = pd.date_range('2023-01-01', periods=100, freq='D')
    regions = ['          ', '    ', '             ', '          ']
```

```
    categories = ['                    ', '              ', '
                    ']
    data = []
    for _ in range(500):
        data.append({
            'date': np.random.choice(dates),
            'region': np.random.choice(regions),
            'category': np.random.choice(categories),
            'revenue': np.random.exponential(500) + 100,
            'units': np.random.randint(1, 20)
        })
    return pd.DataFrame(data)

df = load_data()

#
st.sidebar.header("              ")
selected_regions = st.sidebar.multiselect(
    "            ", options=df['region'].unique(), default=df['region
        '].unique()
)
selected_categories = st.sidebar.multiselect(
    "                ", options=df['category'].unique(), default=df
        ['category'].unique()
)
date_range = st.sidebar.date_input(
    "            ", value=(df['date'].min(), df['date'].max()),
    min_value=df['date'].min(), max_value=df['date'].max()
)

#
mask = (df['region'].isin(selected_regions) &
        df['category'].isin(selected_categories) &
        df['date'].between(*date_range))
df_filtered = df[mask]

# KPI-
col1, col2, col3 = st.columns(3)
with col1:
    st.metric("                        ", f"{df_filtered['revenue'].
        sum():.0f}    ")
with col2:
    st.metric("                          ", f"{df_filtered['units'].
        sum()}")
with col3:
    avg_check = df_filtered['revenue'].sum() / df_filtered['units'].
        sum()
    st.metric("                    ", f"{avg_check:.0f}    ")

#
col_chart1, col_chart2 = st.columns(2)

with col_chart1:
```

```
    st.subheader("                                                    ")
    fig1 = px.bar(df_filtered.groupby('category')['revenue'].sum().
        reset_index(),
                  x='category', y='revenue', color='category')
    st.plotly_chart(fig1, use_container_width=True)

with col_chart2:
    st.subheader("                                      ")
    daily = df_filtered.groupby('date')['revenue'].sum().reset_index
        ()
    fig2 = px.line(daily, x='date', y='revenue', markers=True)
    st.plotly_chart(fig2, use_container_width=True)

#
with st.expander("                                                ):
    st.dataframe(df_filtered.sort_values('date', ascending=False),
        use_container_width=True)

#
st.markdown("---")
st.caption("
                                  Streamlit    Plotly")
```

Индивидуальное задание: Расширьте дашборд: добавьте карту регионов с хороплетом, гистограмму распределения чеков и возможность экспорта отфильтрованных данных в CSV. Реализуйте кэширование тяжёлых вычислений через @st.cache_data.

Вопросы для самопроверки:

1. Почему важно ограничивать количество интерактивных элементов на дашборде?
2. Как обеспечить воспроизводимость дашборда при обновлении исходных данных?
3. В чём преимущества Streamlit перед Dash для быстрого прототипирования?

9 Комплексный проект: визуальный анализ реального датасета

Цель: Интегрировать полученные знания для решения прикладной задачи визуального анализа данных.

Задачи:

- Выбрать и загрузить реальный датасет (открытые данные, Kaggle, корпоративные данные);
- Провести разведочный анализ (EDA) с визуализацией распределений, пропусков, выбросов;
- Сформулировать аналитические вопросы и подобрать адекватные визуальные методы;
- Создать интерактивный отчёт/дашборд с выводами и рекомендациями.

Пример постановки задачи: *Анализ данных о качестве воздуха в крупных городах.*
Цели:

- Выявить сезонные паттерны загрязнения;
- Сравнить уровень PM2.5 между городами;
- Оценить корреляцию с метеопараметрами (температура, ветер);
- Предложить визуальные инсайты для экологического мониторинга.

Данные: OpenAQ API или датасет Kaggle «Air Quality Data».

Структура отчёта (Jupyter Notebook):

1. Титульный блок: название, автор, дата, источник данных;
2. Описание данных: переменные, типы, объём, потенциальные проблемы;
3. Разведочный анализ: визуализация пропусков, распределений, корреляций;
4. Углублённый анализ: ответ на аналитические вопросы через визуализацию;
5. Интерактивный компонент: дашборд или HTML-отчёт с фильтрами;
6. Выводы: ключевые инсайты, ограничения анализа, рекомендации.

Критерии оценки проекта:

- Глубина разведочного анализа и обоснованность выбора методов визуализации;
- Качество и информативность визуализаций (соответствие принципам восприятия);
- Функциональность и удобство интерактивного компонента;
- Структурированность кода и отчёта, воспроизводимость результатов;
- Практическая ценность выводов и рекомендаций.

Программный код (шаблон проекта):

```
"""
                                    :

          : [                ],              [          ]
        : [                                 ]
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import plotly.express as px
from datetime import datetime

# ================= 1.
    =================
#              :
                    (
                     )
# df = pd.read_csv('air_quality.csv', parse_dates=['datetime'])
#

np.random.seed(42)
cities = ['             ', '      ', '                        ', '
                              ']
dates = pd.date_range('2023-01-01', '2023-12-31', freq='D')
data = []
for city in cities:
    base_pm = {'            ': 35, '      ': 25, '
                            ': 40, '                          ': 30}[
       city]
    seasonal = 10 * np.sin(2*np.pi*np.arange(len(dates))/365)  #

    noise = np.random.normal(0, 5, len(dates))
    pm25 = np.clip(base_pm + seasonal + noise, 5, 150)
    for i, date in enumerate(dates):
        data.append({
            'datetime': date,
            'city': city,
            'pm25': pm25[i],
            'temperature': 20 + 15*np.sin(2*np.pi*i/365) + np.random.
               normal(0, 3)
        })
df = pd.DataFrame(data)

# ================= 2.
    =================
print("                          :", df.shape)
print("\ n                :\n", df.isnull().sum())
print("\ n                                              :\n", df.
    describe())

#                                                 (                  )
# sns.heatmap(df.isnull(), cbar=False, yticklabels=False)
# plt.title('                                          ')
# plt.show()

#                              PM2.5
plt.figure(figsize=(10, 5))
sns.boxplot(data=df, x='city', y='pm25', palette='Set2')
plt.ylabel('PM2.5,       /    '); plt.xlabel('          ')
plt.title('
                 ')
```

```
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.savefig('eda_boxplot.png', dpi=300)
plt.show()

# ================ 3.
    ================
#
df['month'] = df['datetime'].dt.month
monthly = df.groupby(['city', 'month'])['pm25'].mean().reset_index()

fig_season = px.line(monthly, x='month', y='pm25', color='city',
                     title='                                    PM2.5',
                     labels={'month': '          ', 'pm25': 'PM2.5,
                                    /    '})
fig_season.update_xaxes(tickvals=list(range(1,13)),
                        ticktext=['      ','      ','      ','      '
                           ,'      ','      ',
                                  '      ','      ','      ','      '
                                     ,'      ','      ']))
fig_season.write_html('seasonal_trend.html')
fig_season.show()

#
fig_corr = px.scatter(df, x='temperature', y='pm25', color='city',
                      trendline='lowess',  #

                      title='                          PM2.5
                                                  ',
                      labels={'temperature': '                          ,
                           C '})
fig_corr.write_html('temp_correlation.html')
fig_corr.show()

# ================ 4.                                              (
                        ) ================
#
                           Streamlit (     .
               )
print("\ n                                    .
                      :")
print("    - eda_boxplot.png:                                      ")
print("    - seasonal_trend.html:                                       (
                           )")
print("    - temp_correlation.html:
                           ")

# ================ 5.               ================
summary = df.groupby('city')['pm25'].agg(['mean', 'std', 'max'])
print("\  n
                  :")
print(summary.round(2))
print("\  n                                        :")
```

```
print("1.                                                    PM2.5
                                                          ")
print("2.
                 (                                      )")
print("3.

                     ")
```

Индивидуальное задание: Выберите датасет по своей специализации (медицина, финансы, экология, социальные сети и т.д.). Выполните полный цикл: загрузка → EDA → углублённый анализ → интерактивный отчёт. Оформите результат в виде Jupyter Notebook с возможностью экспорта в HTML/PDF.

Вопросы для самопроверки:

1. Как проверить, что визуализация не вводит в заблуждение из-за выбора масштаба или агрегации?
2. Почему важно документировать все этапы предобработки данных в отчёте?
3. Как обеспечить воспроизводимость визуального анализа при обновлении исходных данных?

Список литературы

[1] Tufte E.R. The Visual Display of Quantitative Information. 2nd ed. Graphics Press, 2001. 197 p.

[2] Ware C. Information Visualization: Perception for Design. 4th ed. Morgan Kaufmann, 2021. 552 p.

[3] Knaflic C.N. Storytelling with Data: A Data Visualization Guide for Business Professionals. Wiley, 2015. 288 p.

[4] Hunter J.D. Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 2007. Vol. 9, No. 3, pp. 90-95.

[5] Waskom M.L. Seaborn: Statistical Data Visualization. Journal of Open Source Software, 2021. Vol. 6, No. 60, p. 3021.

[6] Plotly Technologies Inc. Collaborative data science. Montréal, QC, 2015. URL: https://plotly.com.

[7] Streamlit Inc. Streamlit Documentation. URL: https://docs.streamlit.io.

[8] Jordahl K. et al. GeoPandas: Python tools for geographic data. Zenodo, 2020. DOI: 10.5281/zenodo.3946805.

[9] McInnes L., Healy J., Melville J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:1802.03426, 2018.

[10] Python Software Foundation. Python 3.11 Documentation. URL: https://docs.python.org/3/.

[11] Harris C.R. et al. Array programming with NumPy. Nature, 2020. Vol. 585, pp. 357-362.

[12] Reback J. et al. pandas: Powerful Python data analysis toolkit. URL: https://pandas.pydata.org.