

| 2025-10-29

| Библиотеки научных вычислений и анализа данных в экосистеме Python

| NumPy



Официальный сайт: <https://numpy.org/>

Документация: [NumPy documentation — NumPy v2.3 Manual](#)

Страница библиотеки в PyPI: [numpy · PyPI](#)

`pip install numpy`

Исходный код: [GitHub - numpy/numpy: The fundamental package for scientific computing with Python.](#)

- является де-факто фундаментом для всех библиотек научных вычислений в экосистеме Питона
- **предоставляет поддержку многомерных массивов (и, следовательно, матриц)**
- предназначена для работы с данными высоких размерностей и с большими объемами данных
- **частично** написана на С и С++, что обеспечивает высокую скорость и эффективный менеджмент памяти
- предоставляет свою систему типов данных (`dtypes`), совместима с типами С для гранулярной настройки потребления памяти и точности вычислений
- предоставляет связи с С, С++ и Fortran (т.е. в том числе полезным легаси)
- содержит набор готовых функций для математических вычислений, включая операции над матрицами/тензорами, линейную алгебру, преобразование Фурье и т.д.
- добавляет в Питон честные массивы (класс `ndarray`) — прирост производительности относительно встроенных списков до 50 раз

⚠ Warning

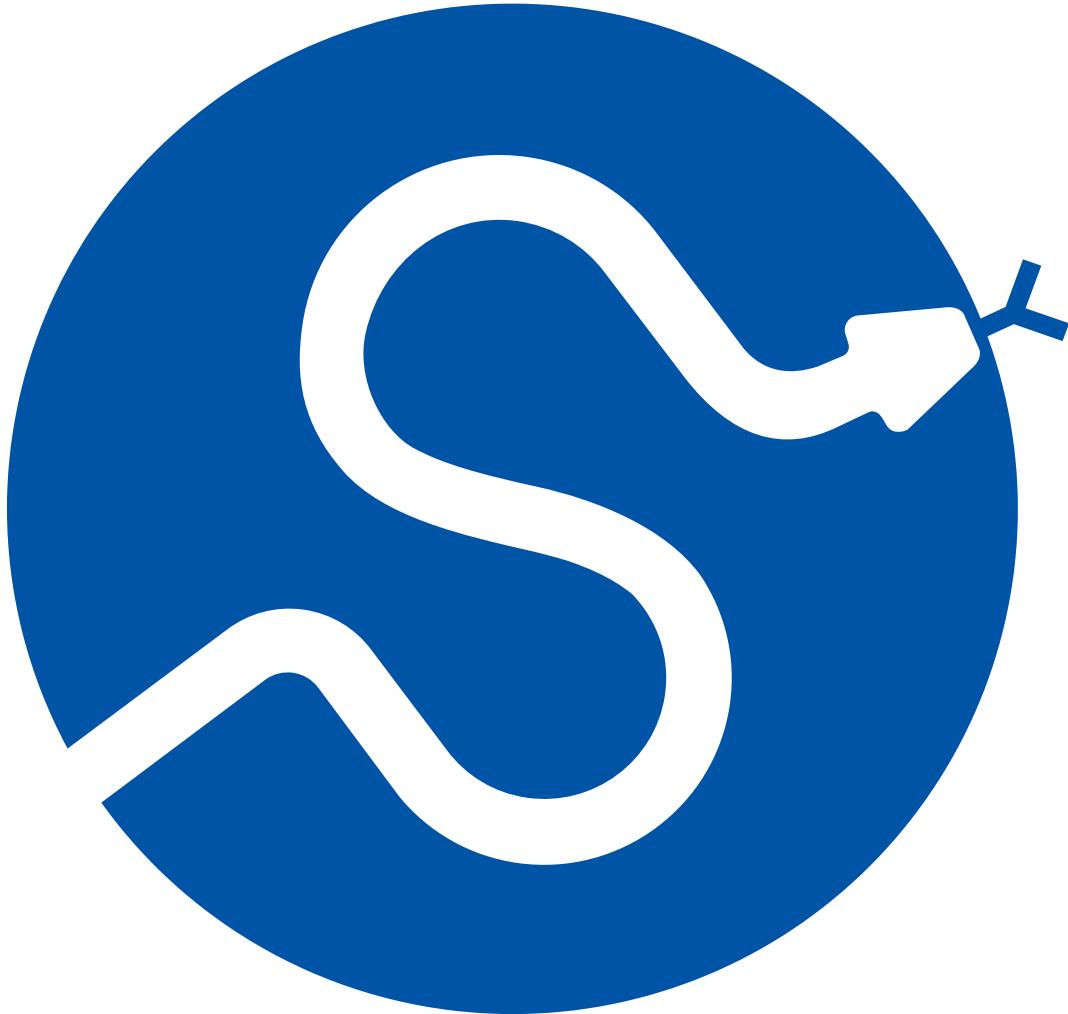
1. Массивы требуют непрерывных блоков памяти, что иногда бывает проблемой, особенно в случае со стандартными типами данных Питона
2. Библиотека вводит понятие `NaN` (Not a Number, что **не** эквивалентно Питоновскому `None`), из-за чего в ряде случаев могут возникать проблемы с совместимостью, т.к. в Питоне такой концепт в чистом виде отсутствует

💡 Tip

16 июня 2024 г. вышла версия 2.0, которая нарушила обратную совместимость со старым кодом; есть руководство [NumPy 2.0 migration guide — NumPy v2.3 Manual](#) по переходу на новую версию

Если совсем никак не чинится, то можно зафиксировать версию на, например, `numpy==1.26.4`

SciPy



Официальный сайт: <https://scipy.org/>

Документация: [SciPy documentation — SciPy v1.16.3 Manual](#)

Страница библиотеки в PyPI: [scipy · PyPI](#)

```
pip install scipy
```

Исходный код: [GitHub - scipy/scipy: SciPy library main repository](#)

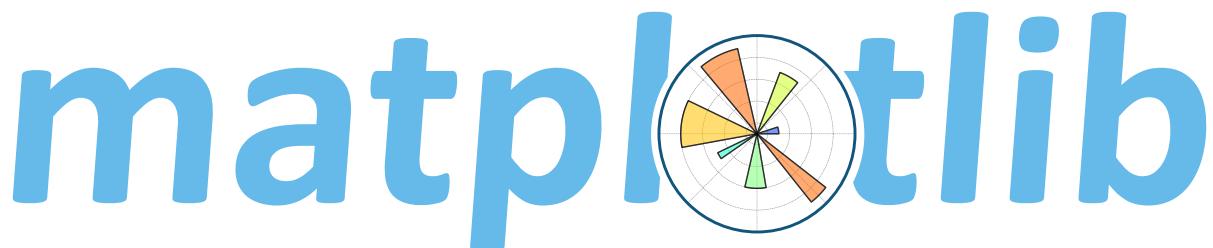
- построенная на базе NumPy библиотека научных и технических вычислений
- содержит готовые методы и алгоритмы статистического анализа, алгебры, оптимизации, цифровой обработки сигналов, интерполяции, многомерных преобразований, вычислений в n -мерных пространствах, обработки изображений (в виде многомерных массивов), кластеризации, работы с разреженными матрицами и т.д. (но, как и в случае с NumPy, это требует определенного математического понимания)
- предоставляет удобный инструментарий для организации параллельных вычислений
- есть возможность виртуализации «из коробки»

Subpackages

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the following table:

Subpackage	Description
<code>cluster</code>	Clustering algorithms
<code>constants</code>	Physical and mathematical constants
<code>fftpack</code>	Fast Fourier Transform routines
<code>integrate</code>	Integration and ordinary differential equation solvers
<code>interpolate</code>	Interpolation and smoothing splines
<code>io</code>	Input and Output
<code>linalg</code>	Linear algebra
<code>ndimage</code>	N-dimensional image processing
<code>odr</code>	Orthogonal distance regression
<code>optimize</code>	Optimization and root-finding routines
<code>signal</code>	Signal processing
<code>sparse</code>	Sparse matrices and associated routines
<code>spatial</code>	Spatial data structures and algorithms
<code>special</code>	Special functions
<code>stats</code>	Statistical distributions and functions

| matplotlib



Официальный сайт: <https://matplotlib.org/>

Документация: [Matplotlib documentation — Matplotlib 3.10.7 documentation](https://matplotlib.org/3.1.0/api/index.html)

[API: API Reference — Matplotlib 3.10.7 documentation](#)

Страница библиотеки в PyPI: [matplotlib · PyPI](#)

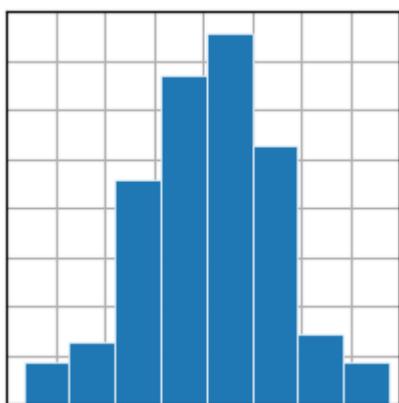
```
pip install matplotlib
```

Исходный код: [GitHub - matplotlib/matplotlib: matplotlib: plotting with Python](#)

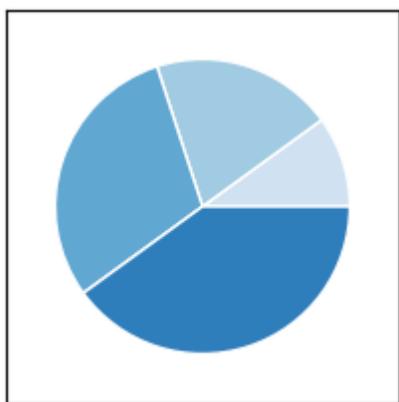
Библиотека научной визуализации: построения графиков, диаграмм, тепловых карт и т.д. в 2 и 3 измерениях

| Наиболее популярные виды визуализации

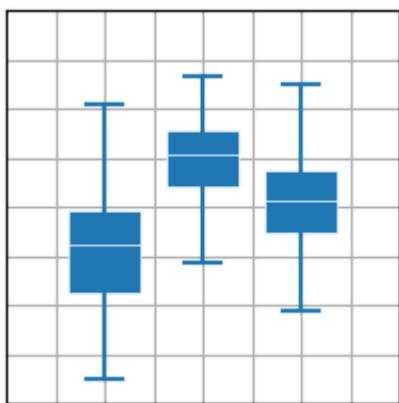
1. Гистограммы



2. Круговые диаграммы



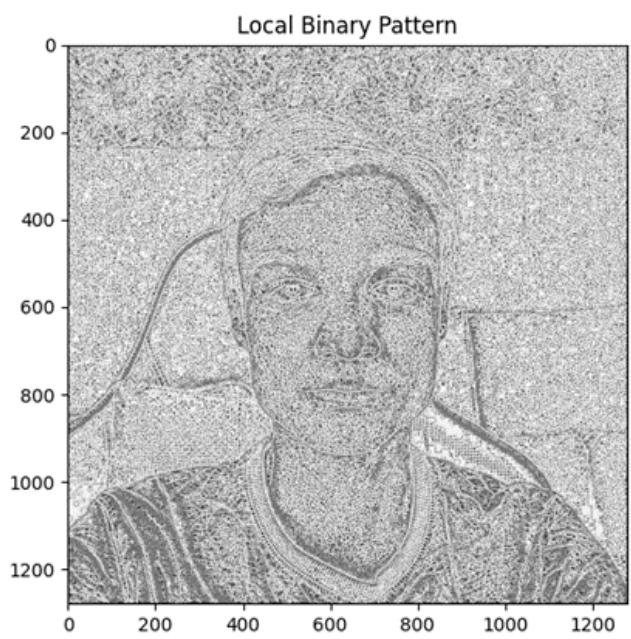
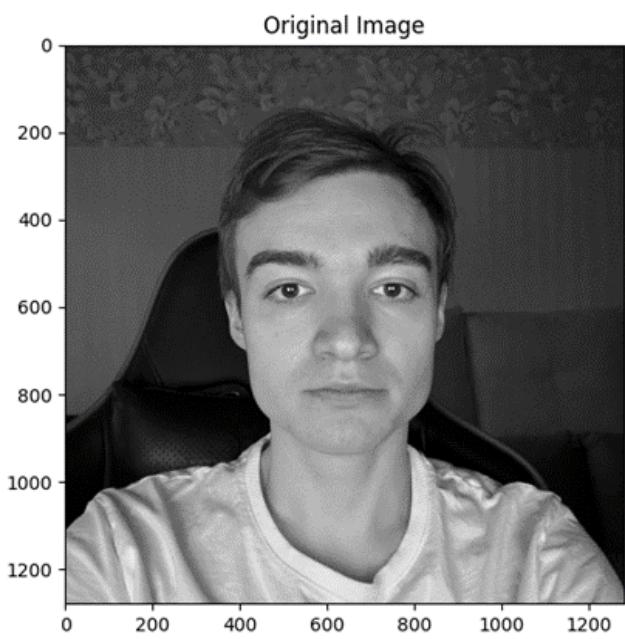
3. Ящики с усами



4. Потоковые карты (streamplots)

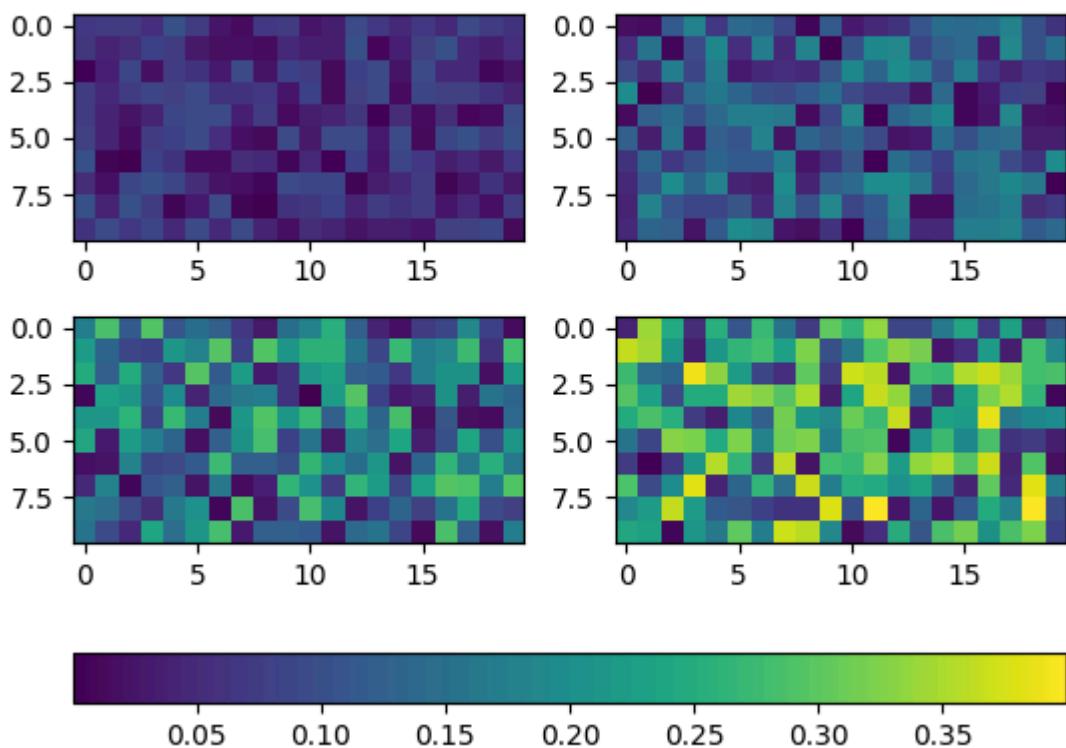


Позволяет визуализировать массивы данных при помощи заданного типа графика (что, например, открывает возможность вывода картинок на экран, если представить картинку как двумерный массив:



Есть возможность объединять множество графиков в одно изображение (т.е. строить витрины данных):

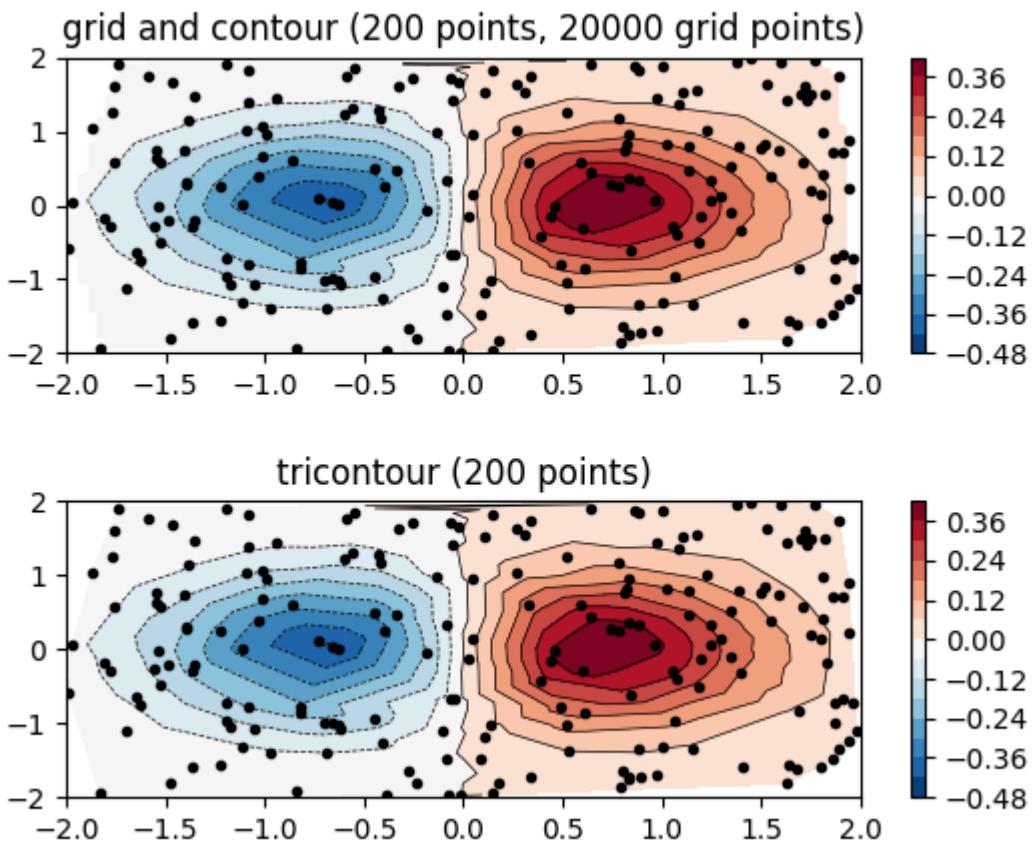
Multiple images



[Обширная галерея с примерами кода визуализации](#)

[Examples — Matplotlib 3.10.7 documentation](#)

[Contour plot of irregularly spaced data — Matplotlib 3.10.7 documentation](#)



```

import matplotlib.pyplot as plt
import numpy as np

import matplotlib.tri as tri

np.random.seed(19680801)
npts = 200
ngridx = 100
ngridy = 200
x = np.random.uniform(-2, 2, npts)
y = np.random.uniform(-2, 2, npts)
z = x * np.exp(-x**2 - y**2)

fig, (ax1, ax2) = plt.subplots(nrows=2)

# -----
# Interpolation on a grid
# -----
# A contour plot of irregularly spaced data coordinates
# via interpolation on a grid.

# Create grid values first.
xi = np.linspace(-2.1, 2.1, ngridx)
yi = np.linspace(-2.1, 2.1, ngridy)

```

```

# Linearly interpolate the data (x, y) on a grid defined by (xi, yi).
triang = tri.Triangulation(x, y)
interpolator = tri.LinearTriInterpolator(triang, z)
Xi, Yi = np.meshgrid(xi, yi)
zi = interpolator(Xi, Yi)

# Note that scipy.interpolate provides means to interpolate data on a grid
# as well. The following would be an alternative to the four lines above:
# from scipy.interpolate import griddata
# zi = griddata((x, y), z, (xi[None, :], yi[:, None]), method='linear')

ax1.contour(xi, yi, zi, levels=14, linewidths=0.5, colors='k')
cntr1 = ax1.contourf(xi, yi, zi, levels=14, cmap="RdBu_r")

fig.colorbar(cntr1, ax=ax1)
ax1.plot(x, y, 'ko', ms=3)
ax1.set(xlim=(-2, 2), ylim=(-2, 2))
ax1.set_title('grid and contour (%d points, %d grid points)' %
              (npts, ngridx * ngridy))

# -----
# Tricontour
# -----
# Directly supply the unordered, irregularly spaced coordinates
# to tricontour.

ax2.tricontour(x, y, z, levels=14, linewidths=0.5, colors='k')
cntr2 = ax2.tricontourf(x, y, z, levels=14, cmap="RdBu_r")

fig.colorbar(cntr2, ax=ax2)
ax2.plot(x, y, 'ko', ms=3)
ax2.set(xlim=(-2, 2), ylim=(-2, 2))
ax2.set_title('tricontour (%d points)' % npts)

plt.subplots_adjust(hspace=0.5)
plt.show()

```

- Легко встраивается в среды разработки (например, Jupyter Notebook)
- Легко экспортовать полученные изображения в нужном формате, разрешении и оформлении (что делает библиотеку незаменимой для оформления статей, отчетных работ, академических работ и т.п.)
- Многие методы схожи с Matlab

Из минусов: страдает интерактивность, нужно много кода, чтобы сделать комплексную витрину данных и заставить ее выглядеть визуально привлекательно

| seaborn



seaborn

Официальный сайт: <https://seaborn.pydata.org/>

Документация: [User guide and tutorial — seaborn 0.13.2 documentation](#)

API: [API reference — seaborn 0.13.2 documentation](#)

Страница библиотеки в PyPI: [seaborn · PyPI](#)

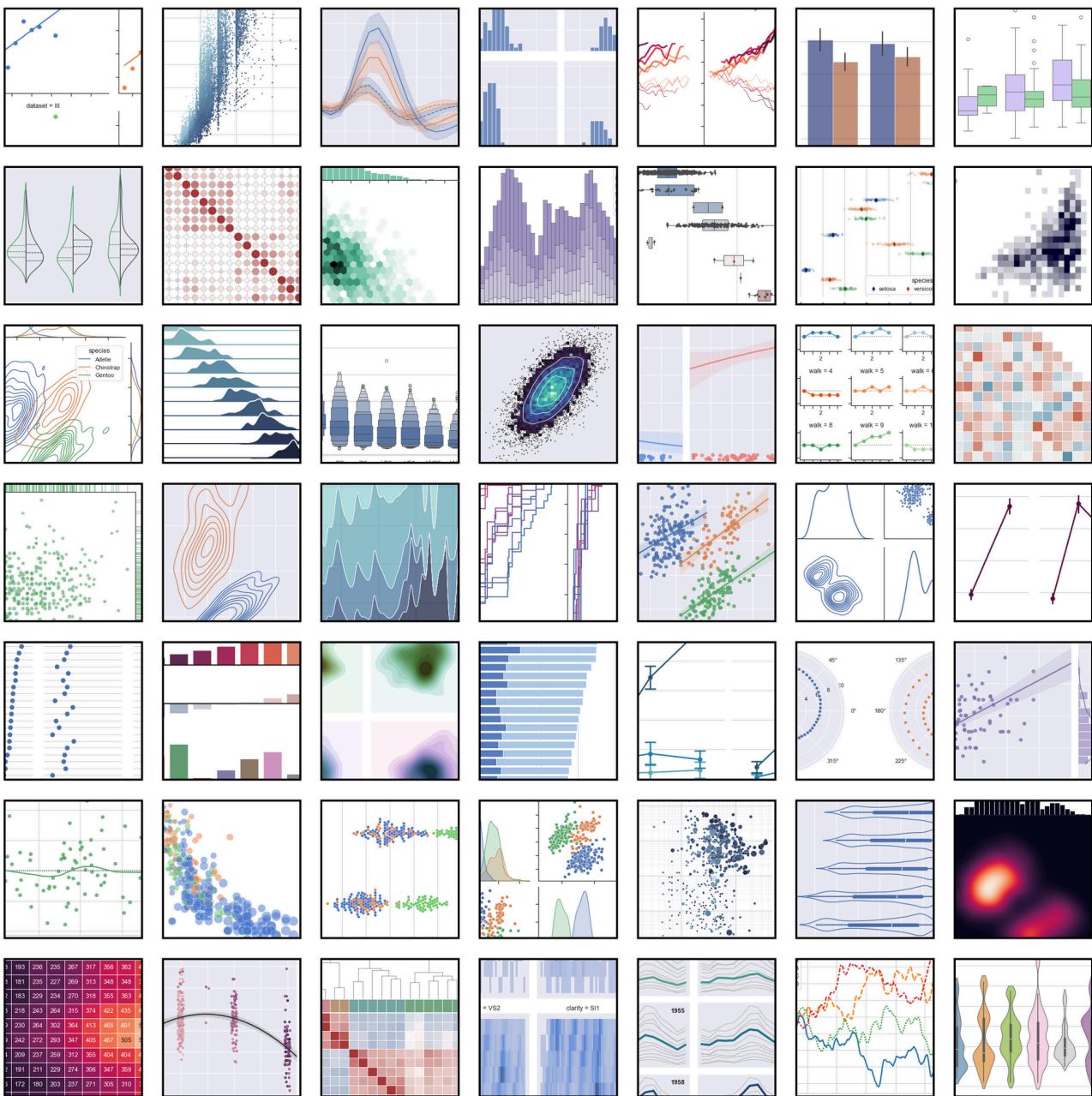
```
pip install seaborn
```

Исходный код: [GitHub - mwaskom/seaborn: Statistical data visualization in Python](#)

- высокоуровневый набор методов и функций для matplotlib, расширяющий возможности по визуализации
- добавляет большое число видов графиков и предоставляет удобный интерфейс методов их построения
- умеет работать сразу с структурами данных, что ускоряет визуализацию по сравнению с matplotlib (хорошо стыкуется с фреймами данных [pandas](#))
- поддерживает темы и (субъективно) предлагает более приятную цветовую палитру
- проще сделать графики интерактивными (но все еще сложно!)
- умеет визуализировать регрессионные модели и временные ряды

Галерея примеров

[Example gallery — seaborn 0.13.2 documentation](#)



Ящики с усами:

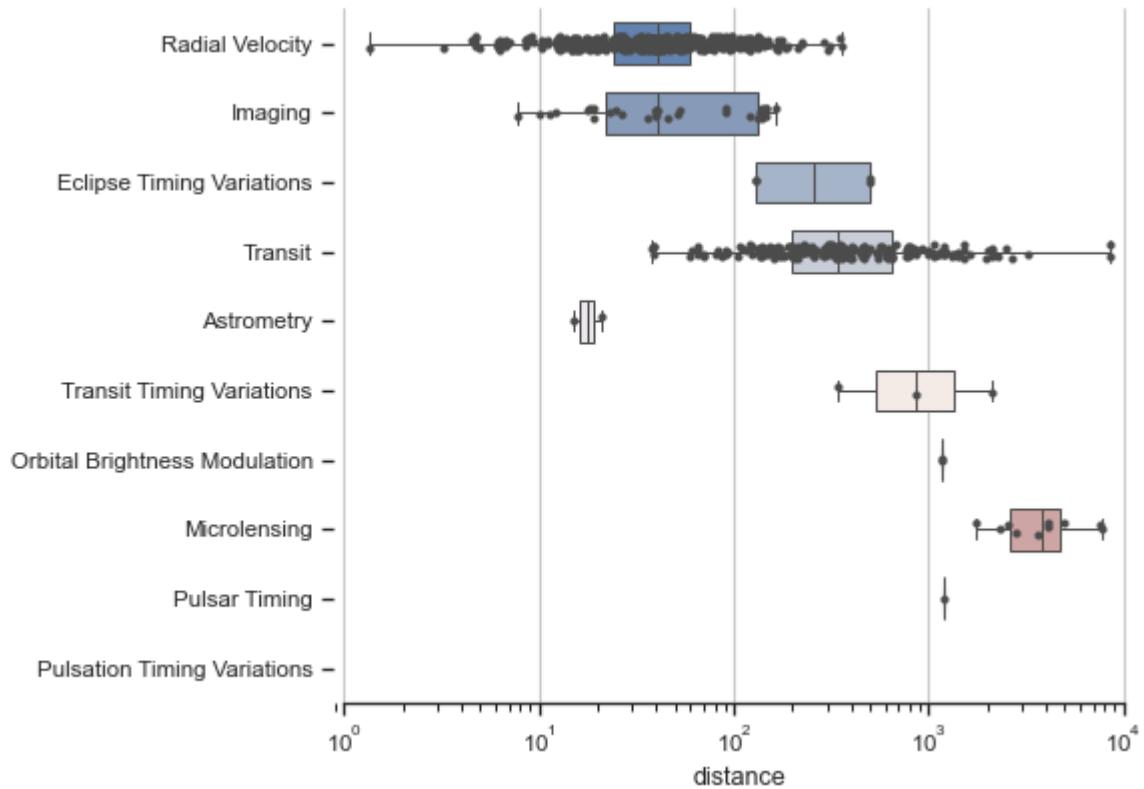
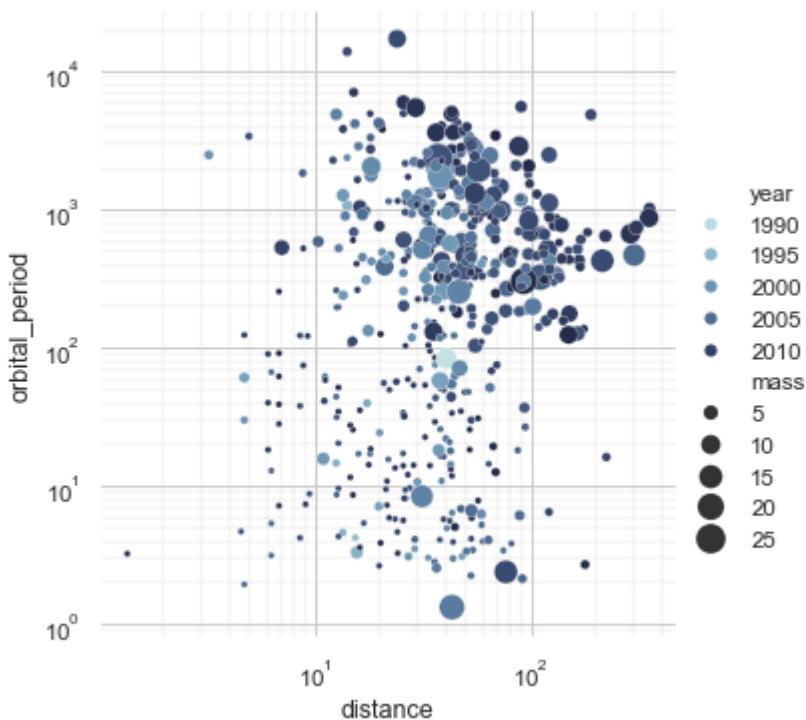
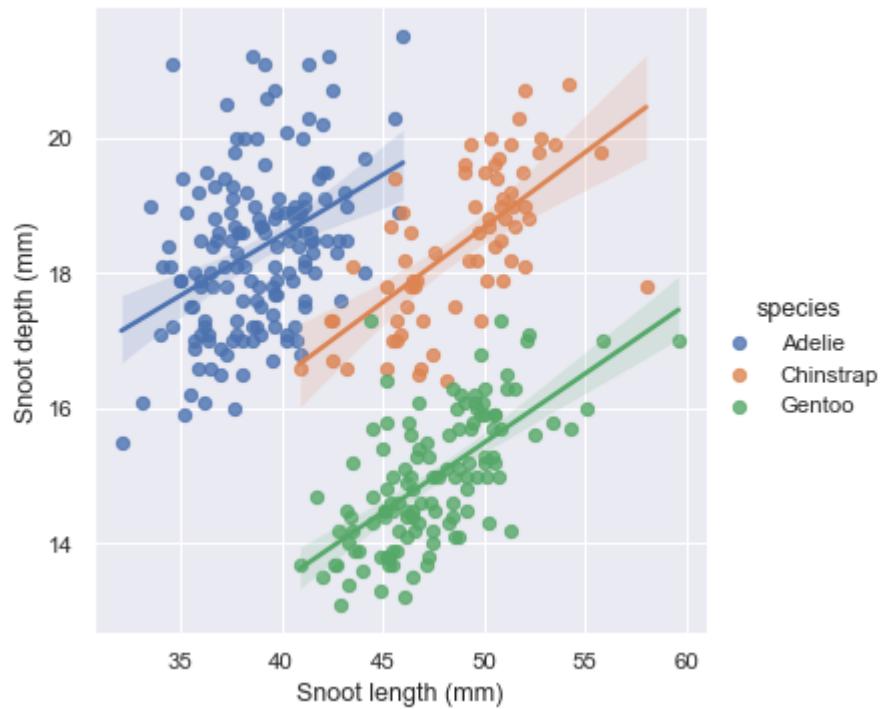


Диаграмма рассеяния (scatterplot)

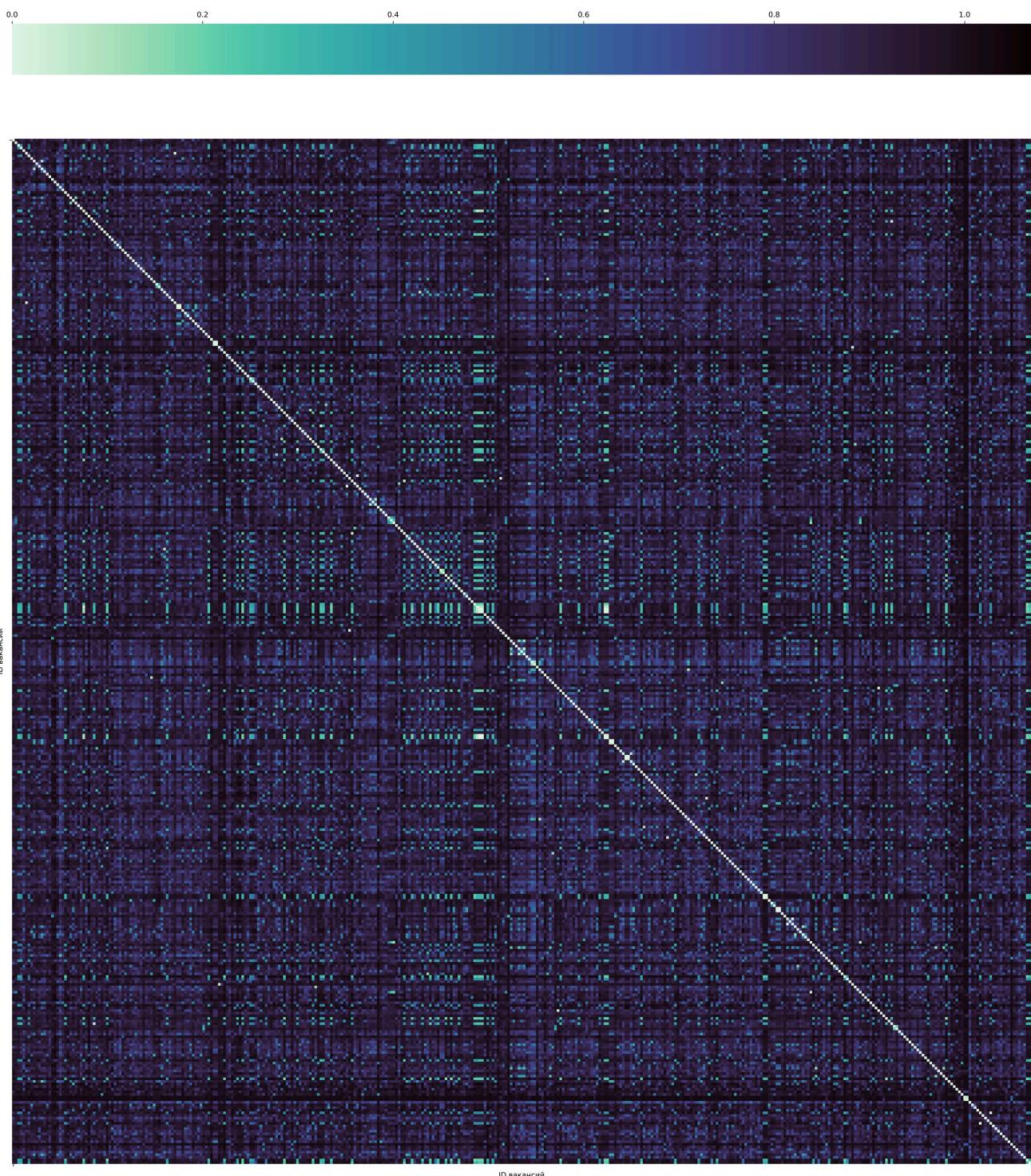


Регрессия



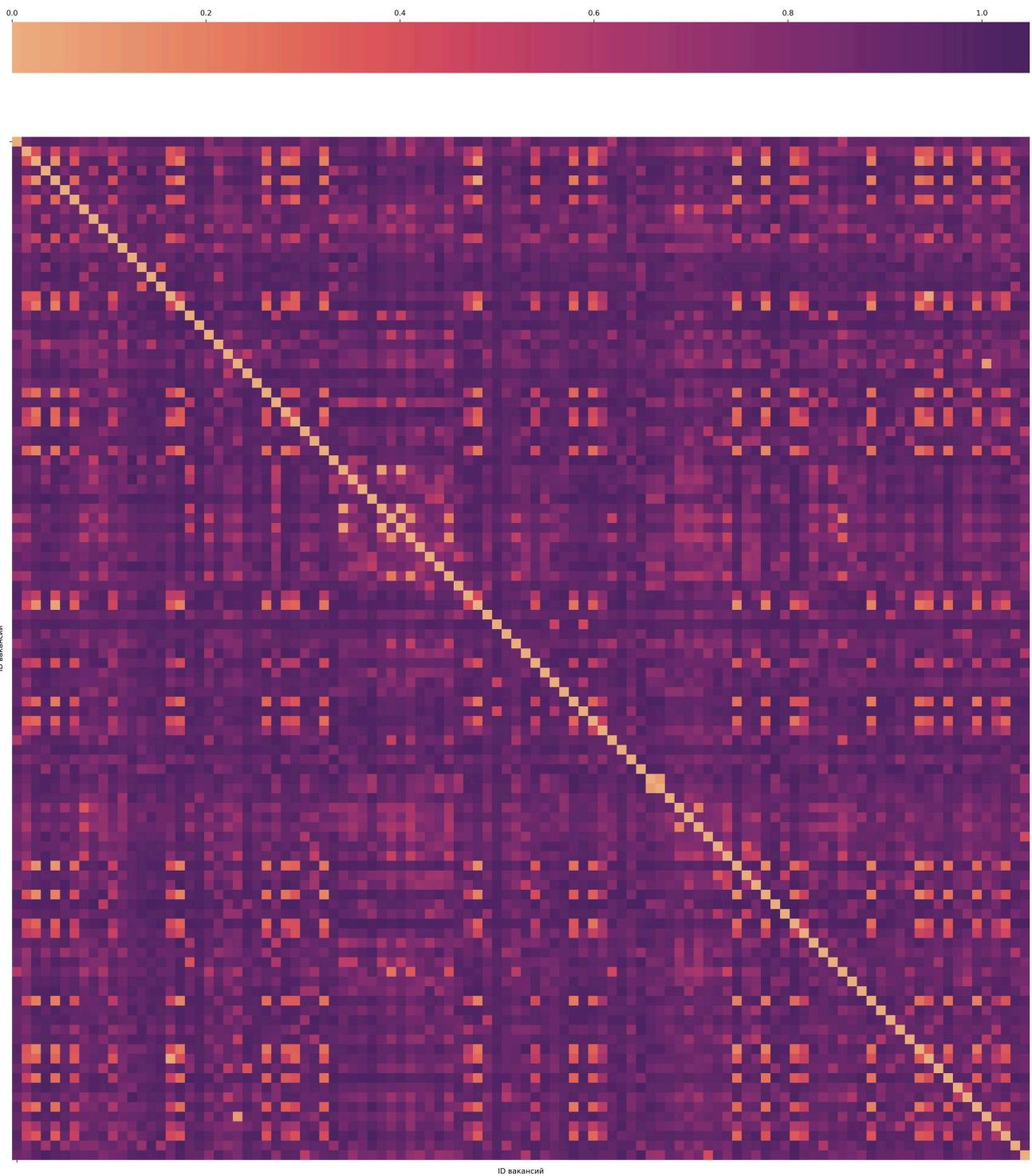
Тепловая карта



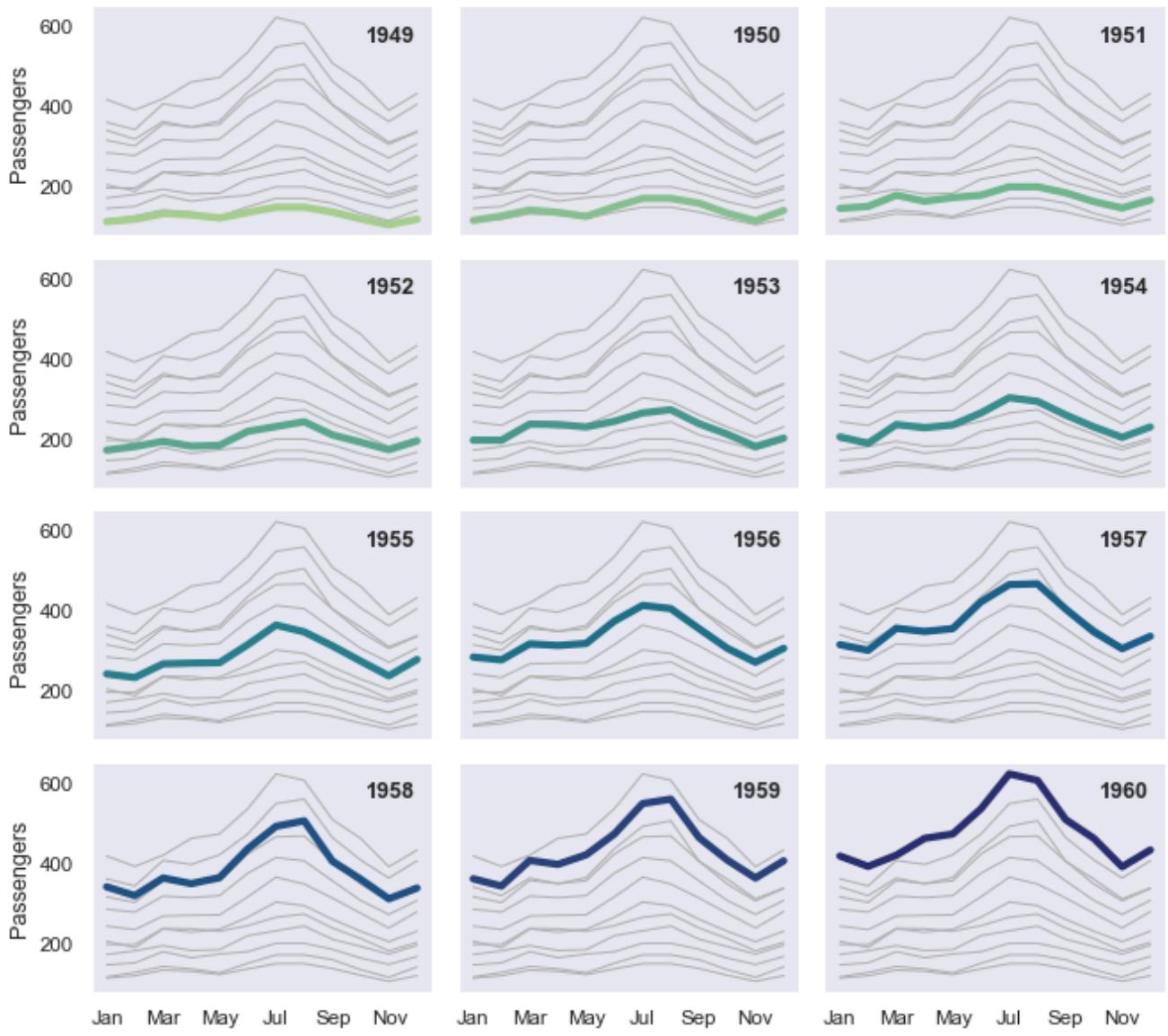


ID вакансий

ID вакансий



Временные ряды

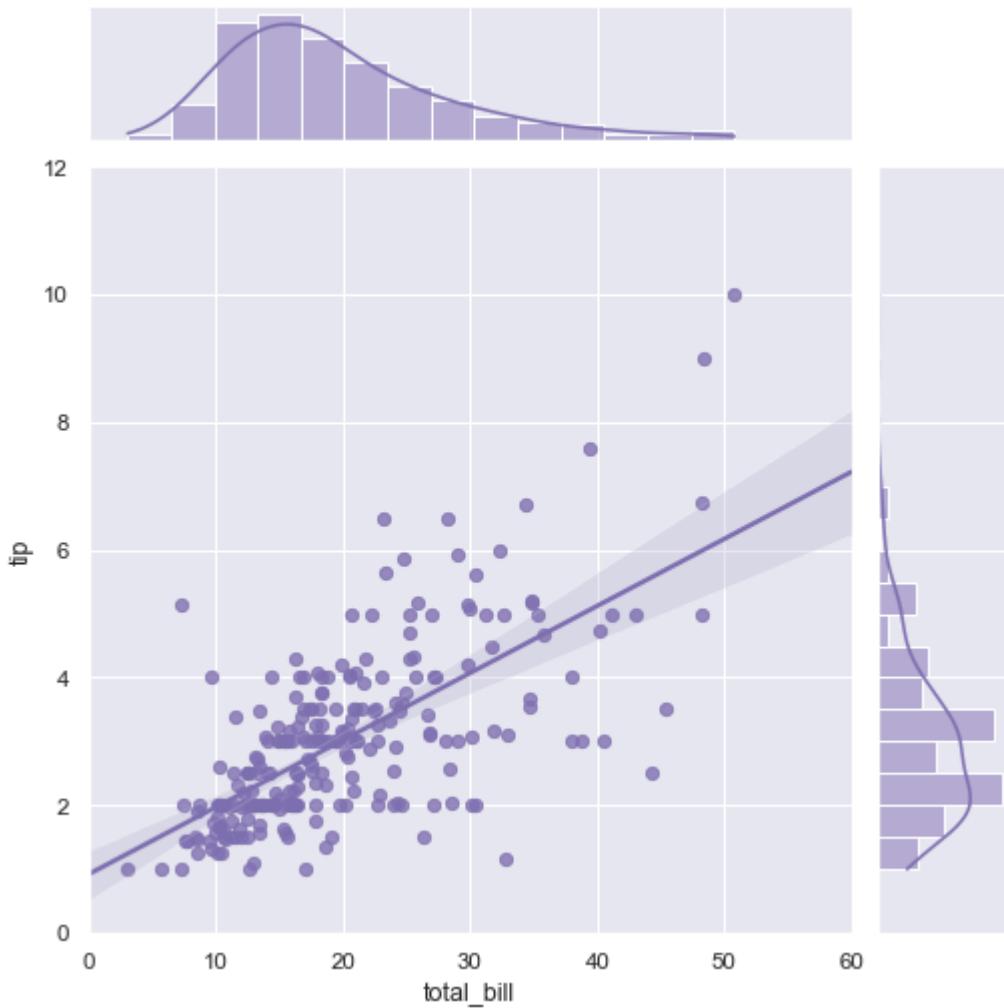


Комбинация линейной регрессии и распределений

```
import seaborn as sns
sns.set_theme(style="darkgrid")

tips = sns.load_dataset("tips")
g = sns.jointplot(x="total_bill", y="tip", data=tips,
                  kind="reg", truncate=False,
                  xlim=(0, 60), ylim=(0, 12),
                  color="m", height=7)
```

[seaborn.jointplot — seaborn 0.13.2 documentation](#) — список всех доступных аргументов для настройки



Радиальная проекция

```

import numpy as np
import pandas as pd
import seaborn as sns

sns.set_theme()

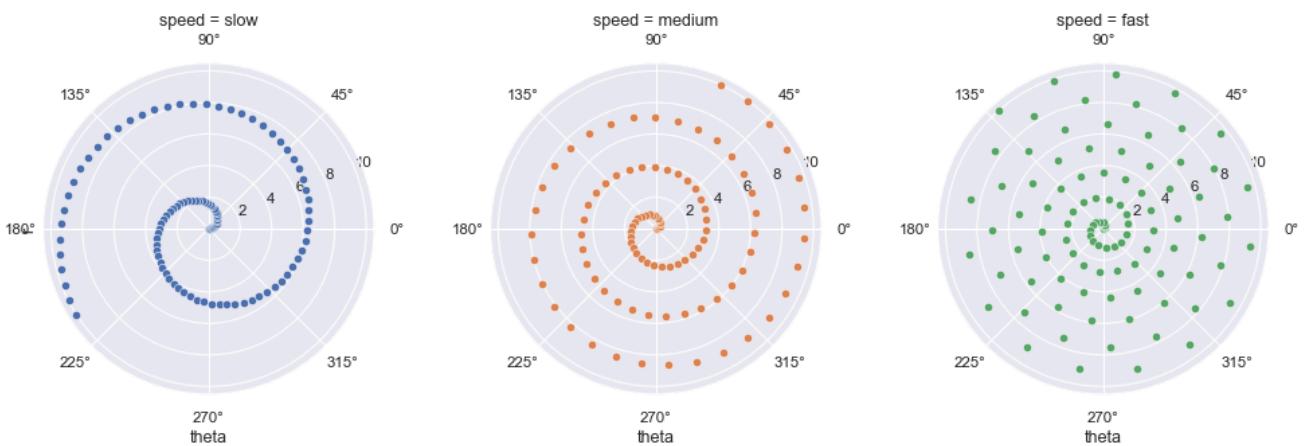
# Generate an example radial dataset
r = np.linspace(0, 10, num=100)
df = pd.DataFrame({'r': r, 'slow': r, 'medium': 2 * r, 'fast': 4 * r})

# Convert the dataframe to long-form or "tidy" format
df = pd.melt(df, id_vars=['r'], var_name='speed', value_name='theta')

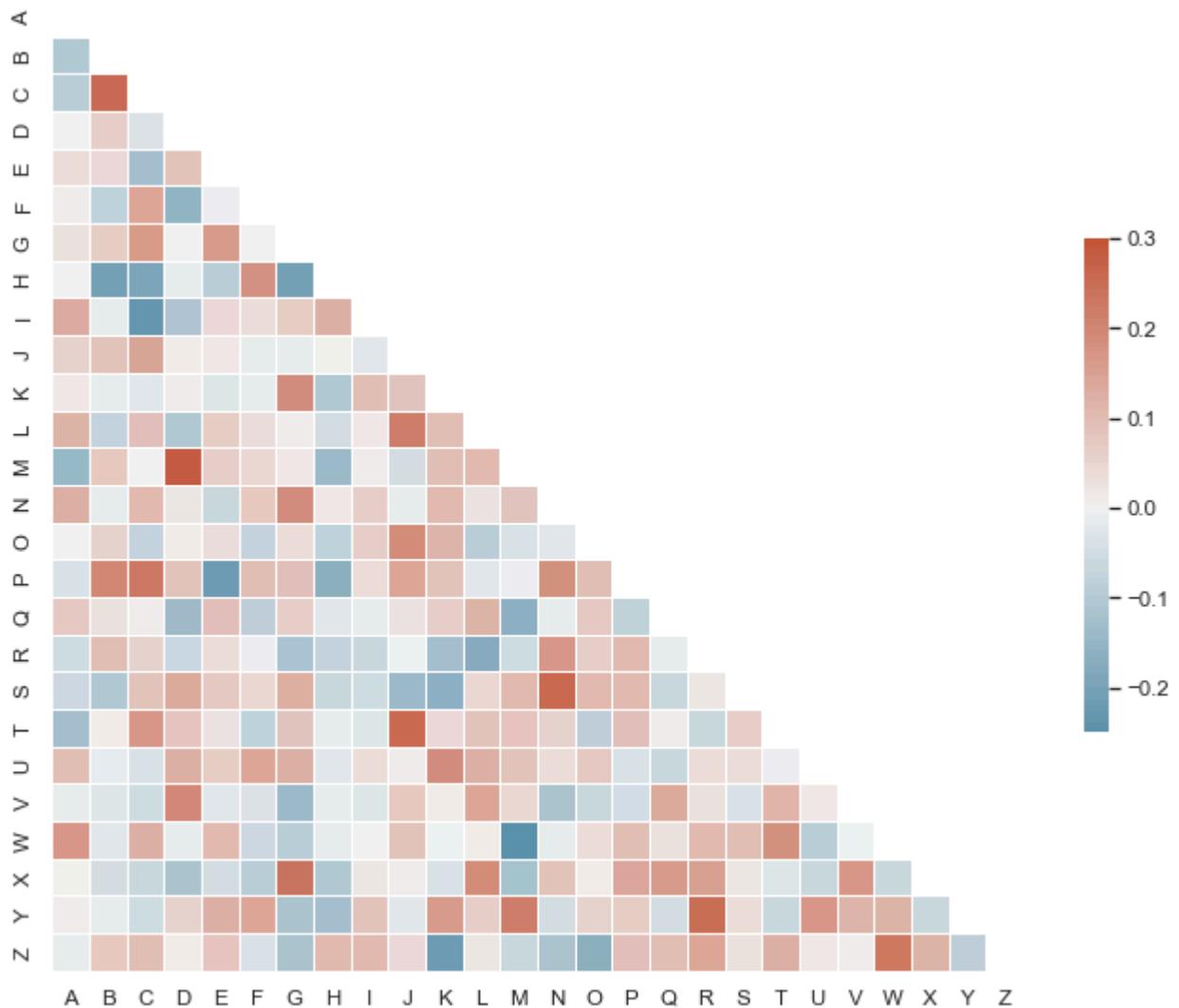
# Set up a grid of axes with a polar projection
g = sns.FacetGrid(df, col="speed", hue="speed",
                   subplot_kws=dict(projection='polar'), height=4.5,
                   sharex=False, sharey=False, despine=False)

# Draw a scatterplot onto each axes in the grid
g.map(sns.scatterplot, "theta", "r")

```



Матрица корреляции

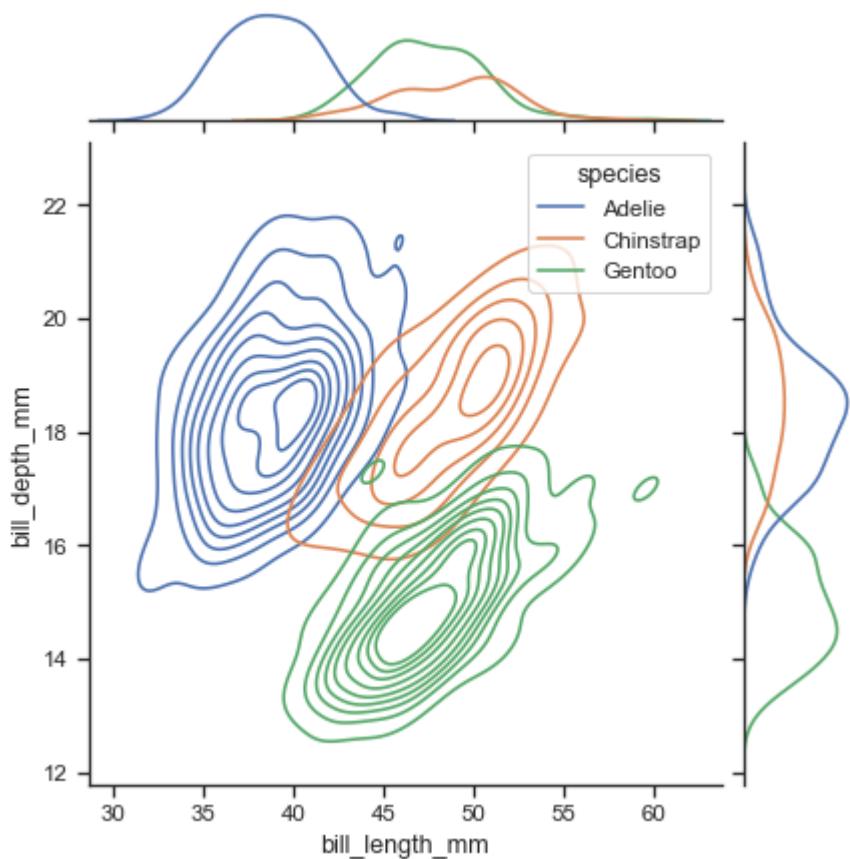
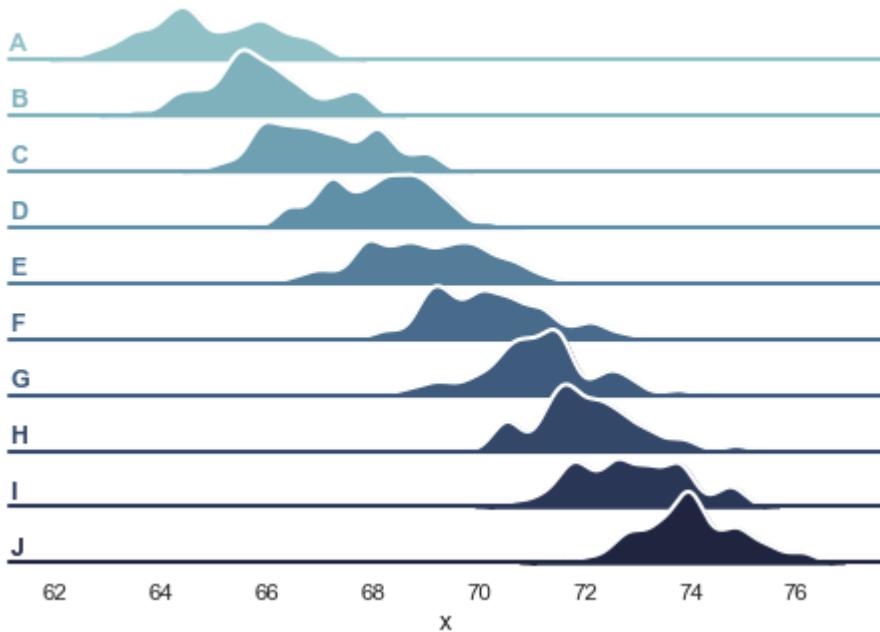


Confusion matrix

Confusion Matrix

Output Class	Target Class					
	BRCA	KIRC	LUAD	LUSC	UCEC	Total
BRCA	342 41.0%	2 0.2%	3 0.4%	4 0.5%	1 0.1%	97.2% 2.8%
KIRC	3 0.4%	211 25.3%	0 0.0%	0 0.0%	0 0.0%	98.6% 1.4%
LUAD	4 0.5%	1 0.1%	54 6.5%	13 1.6%	3 0.4%	72.0% 28.0%
LUSC	2 0.2%	1 0.1%	8 1.0%	79 9.5%	0 0.0%	87.8% 12.2%
UCEC	0 0.0%	0 0.0%	0 0.0%	0 0.0%	104 12.5%	100% 0.0%
	97.4% 2.6%	98.1% 1.9%	83.1% 16.9%	82.3% 17.7%	96.3% 3.7%	94.6% 5.4%

Ridge plot («хребтовый» график)



Комбинация: иерархическая кластеризация (визуализация дендрограммой) + тепловая карта

```
import pandas as pd
import seaborn as sns
sns.set_theme()

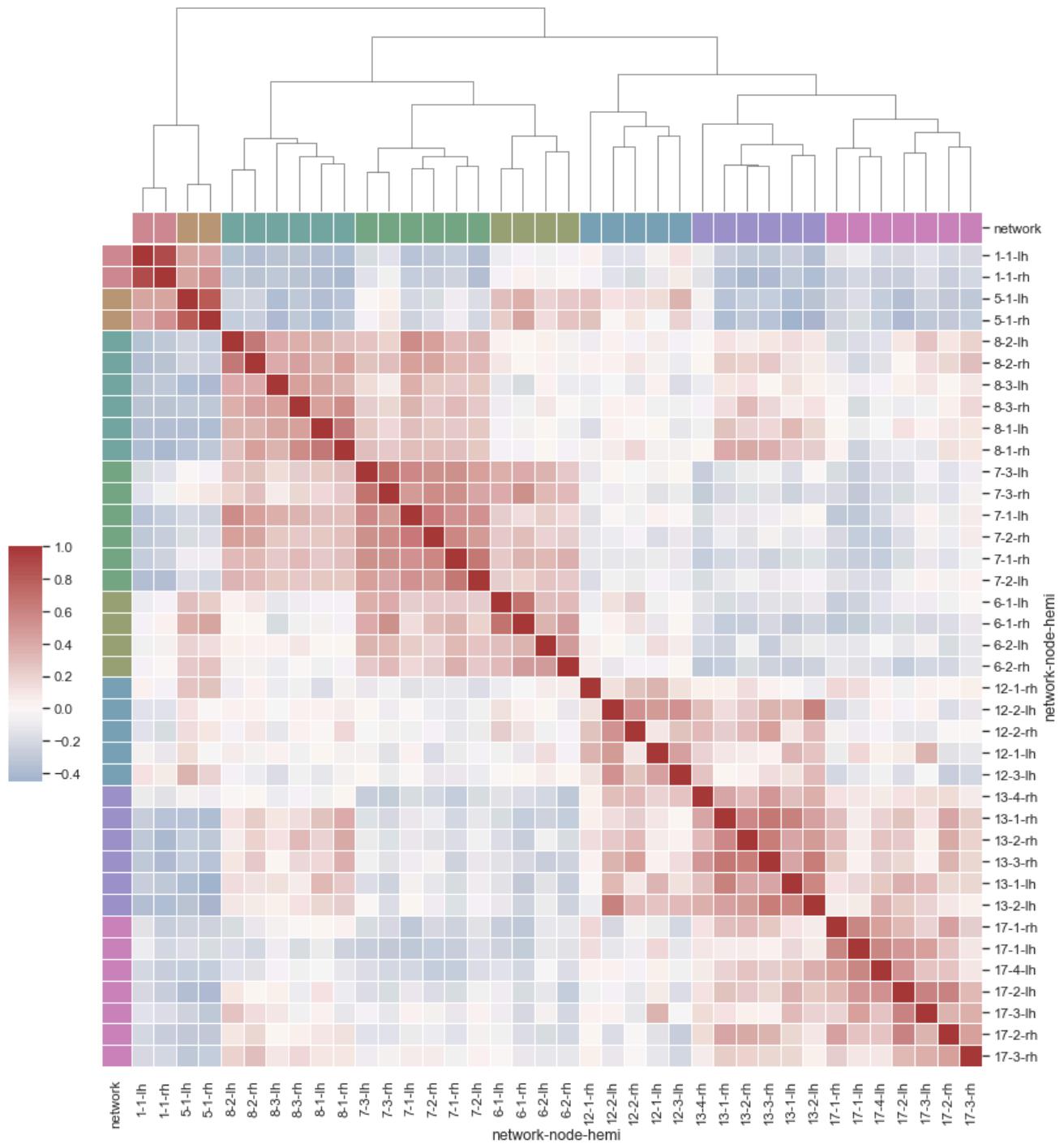
# Load the brain networks example dataset
df = sns.load_dataset("brain_networks", header=[0, 1, 2], index_col=0)
```

```
# Select a subset of the networks
used_networks = [1, 5, 6, 7, 8, 12, 13, 17]
used_columns = (df.columns.get_level_values("network")
                  .astype(int)
                  .isin(used_networks))
df = df.loc[:, used_columns]

# Create a categorical palette to identify the networks
network_pal = sns.husl_palette(8, s=.45)
network_lut = dict(zip(map(str, used_networks), network_pal))

# Convert the palette to vectors that will be drawn on the side of the
matrix
networks = df.columns.get_level_values("network")
network_colors = pd.Series(networks, index=df.columns).map(network_lut)

# Draw the full plot
g = sns.clustermap(df.corr(), center=0, cmap="vlag",
                    row_colors=network_colors, col_colors=network_colors,
                    dendrogram_ratio=(.1, .2),
                    cbar_pos=(.02, .32, .03, .2),
                    linewidths=.75, figsize=(12, 13))
```



| pandas



Официальный сайт: <https://pandas.pydata.org/>

Документация: [pandas documentation — pandas 2.3.3 documentation](#)

Страница библиотеки в PyPI: [pandas · PyPI](#)

```
pip install pandas
```

Исходный код: [GitHub - pandas-dev/pandas: Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more](#)

Маккинни, У. Python и анализ данных. Первичная обработка данных с применением pandas, NumPy и Jupiter : справочник / У. Маккинни ; перевод с английского А. А. Слинкина. — 3-е изд. — Москва : ДМК Пресс, 2023. — 536 с. — ISBN 978-5-93700-174-0. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/348086> (дата обращения: 29.10.2025). — Режим доступа: для авториз. пользователей.



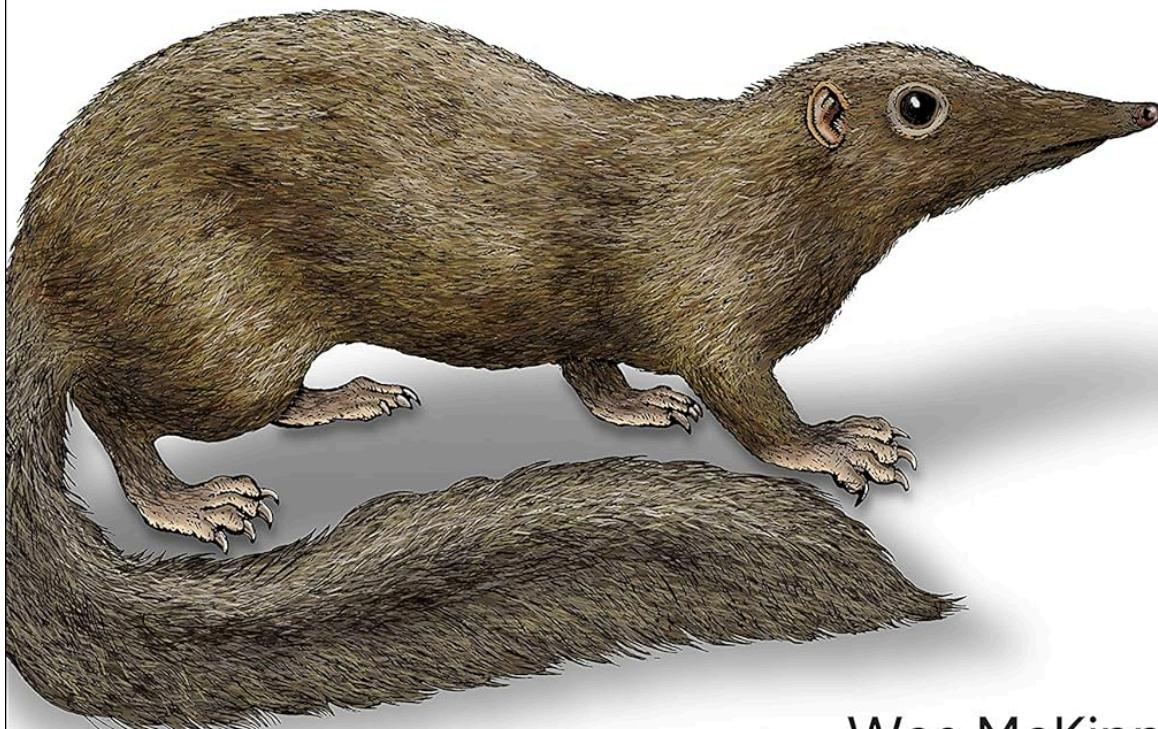
[Python for Data Analysis: Data Wrangling... by McKinney, Wes](#)

O'REILLY®

Third
Edition

Python for Data Analysis

Data Wrangling with pandas, NumPy & Jupyter



Wes McKinney

Вводный праймер для начинающих: [10 minutes to pandas — pandas 2.3.3 documentation](#)

- основное назначение библиотеки — работа с большими массивами данных
- готовые интеграции с наиболее распространенными форматами данных (как для хранения, так и для манипуляции, включая CSV, TSV, JSON, SQL, BigQuery, HDF5, Excel <через [openpyxl](#) [openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm](#)

The pandas I/O API is a set of top level `[reader]` functions accessed like `pandas.read_csv()` that generally return a pandas object. The corresponding `[writer]` functions are object methods that are accessed like `Dataframe.to_csv()`. Below is a table containing available `[readers]` and `[writers]`.

Format	Type	Data Description	Reader	Writer
text	CSV		<code>read_csv</code>	<code>to_csv</code>
text		Fixed-Width Text File	<code>read_fwf</code>	NA
text	JSON		<code>read_json</code>	<code>to_json</code>
text	HTML		<code>read_html</code>	<code>to_html</code>
text	LaTeX		<code>Styler.to_latex</code>	NA
text	XML		<code>read_xml</code>	<code>to_xml</code>
text		Local clipboard	<code>read_clipboard</code>	<code>to_clipboard</code>
binary	MS Excel		<code>read_excel</code>	<code>to_excel</code>
binary	OpenDocument		<code>read_excel</code>	NA
binary	HDF5 Format		<code>read_hdf</code>	<code>to_hdf</code>
binary	Feather Format		<code>read_feather</code>	<code>to_feather</code>
binary	Parquet Format		<code>read_parquet</code>	<code>to_parquet</code>
binary	ORC Format		<code>read_orc</code>	<code>to_orc</code>
binary	Stata		<code>read_stata</code>	<code>to_stata</code>
binary	SAS		<code>read_sas</code>	NA
binary	SPSS		<code>read_spss</code>	NA

- функциональность включает просмотр данных, выборки, анализ аномалий, отсутствующих данных, слияние, группировку, реструктуризацию (reshaping, изменение числа измерений), временные ряды, категориальные данные и манипуляцию ими, визуализацию в виде графиков и диаграмм
- основными концепциями и объектами в pandas являются **датафреймы (фреймы данных / DataFrames) и серии/ряды (Series)**

Документация:

- [Series — pandas 2.3.3 documentation](#)
- [DataFrame — pandas 2.3.3 documentation](#)

Датафрейм — двумерная структура данных, серия — одномерная структура данных.

Каждый датафрейм можно разбить на серии. Несколько серий можно объединить в датафрейм.

И датафрейм, и серия имеют индекс, который не является частью данных. Все объекты в pandas используют С-подобные типы (заимствованные из numpy `dtypes` и полагающиеся на их реализацию в той библиотеке).

Одним из больших преимуществ pandas является гранулярный контроль над типами данных на уровне серий (например, вы можете задать для колонки тип данных `int8`). Настоящее преимущество такого подхода раскрывается при использовании категориальных данных.

Категория — специальный ярлык, который соотносится с отдельной областью (массивом) в памяти, заменяя всех вхождения категории в датафрейме или серии на индекс этой категории в хранилище категорий. Это позволяет заменять повторяющиеся, например, строковые данные на `int8`. Подробнее о категориальных данных: [Categorical data — pandas 2.3.3 documentation](#)

Встроенный мониторинг потребления памяти:

- [pandas.Series.memory_usage — pandas 2.3.3 documentation](#)
- [pandas.DataFrame.memory_usage — pandas 2.3.3 documentation](#)
- обеспечивает простое представление сложных структур данных: есть возможность агрегации данных из разных источников, с разными типами, с разной полнотой и удобный инструмент для изучения (`explore`), анализа, манипуляции и визуализации
- все необходимые методы уже включены в библиотеку, т.е. она предлагает полный цикл анализа данных
- легко масштабируется под задачи, готова к работе с большими данными

Из минусов:

- `easy to learn, hard to master`, пользоваться можно начать относительно легко, но продвинутые функции и методы использования библиотеки требуют отдельного изучения
- документация не всегда успевает за библиотекой, требует понимания, что именно и для каких целей вы ищете
- несовместима с тензорами и иными структурами высоких размерностей: несмотря на то, что построена на базе `pint`, многомерные массивы гораздо лучше обрабатываются в самой `pint`, т.к. датафреймы и серии — дву- и одномерные структуры данных

Версия / Период	Ключевые нововведения, связанные с PyArrow	Статус
До Pandas 2.0	NumPy — единственный бэкенд.	Устарело
Pandas 2.0	Введен <code>dtype_backend="pyarrow"</code> для операций ввода-вывода. Новые типы <code>dtype="int64[pyarrow]"</code> .	Актуально
Будущее (Pandas 3.0)	Строки на бэкенде PyArrow станут поведением по умолчанию	В планах

```
# Для серии
ser = pd.Series([1, 2, None], dtype="int64[pyarrow]")

# Для всего DataFrame при создании
df = pd.DataFrame([[1, 2], [3, 4]], dtype="uint64[pyarrow]")
```

Операции со строками показывают наибольший прирост производительности — до 3-16 раз для методов вроде `.str.contains()` или `.str.startswith()`, поскольку PyArrow использует оптимизированные строковые типы вместо медленных объектов Python.

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
                  index=['cobra', 'viper', 'sidewinder'],
                  columns=['max_speed', 'shield'])

# out:
      max_speed  shield
cobra          1        2
viper          4        5
sidewinder     7        8
```

```
df.loc[df['shield'] > 6]

#out:
      max_speed  shield
sidewinder     7        8
```

```
mydict = [{‘a’: 1, ‘b’: 2, ‘c’: 3, ‘d’: 4},
          {‘a’: 100, ‘b’: 200, ‘c’: 300, ‘d’: 400},
          {‘a’: 1000, ‘b’: 2000, ‘c’: 3000, ‘d’: 4000}]

df = pd.DataFrame(mydict)
```

```
df
      a      b      c      d
0    1      2      3      4
1   100    200    300    400
2  1000   2000   3000   4000
```

```
df.iloc[:3]
      a      b      c      d
0    1      2      3      4
1   100    200    300    400
2  1000   2000   3000   4000
```

```
df.iloc[1:3, 0:3]
      a      b      c
1   100    200    300
2  1000   2000   3000
```

```
df.iloc[:, lambda df: [0, 2]]
      a      c
```

```
0      1      3
1    100    300
2   1000   3000
```

```
df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])
```

```
df
```

```
   A   B
0  4  9
1  4  9
2  4  9
```

```
df.apply(np.sqrt)
```

```
      A      B
0  2.0  3.0
1  2.0  3.0
2  2.0  3.0
```

```
df.apply(np.sum, axis=0)
```

```
A    12
B    27
dtype: int64
```

```
df.apply(np.sum, axis=1)
```

```
0    13
1    13
2    13
dtype: int64
```

```
df.apply(lambda x: pd.Series([1, 2], index=['foo', 'bar']), axis=1)
```

```
    foo  bar
0    1    2
1    1    2
2    1    2
```

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon', 'Parrot', 'Parrot'], 'Max Speed': [380., 370., 24., 26.]})
```

```
df
```

```
      Animal  Max Speed
0    Falcon     380.0
1    Falcon     370.0
2    Parrot      24.0
3    Parrot      26.0
```

```
df.groupby(['Animal']).mean()
```

```
          Max Speed
Animal
```

```

Falcon      375.0
Parrot       25.0

df = pd.DataFrame({'Animal': ['Falcon', 'Falcon', 'Parrot', 'Parrot'],
                   'Max Speed': [380., 370., 24., 26.]})

df.groupby("Animal", group_keys=True).apply(lambda x: x)
    Animal  Max Speed
Animal
Falcon  0   Falcon      380.0
        1   Falcon      370.0
Parrot  2   Parrot      24.0
        3   Parrot      26.0

```

| Polars



Официальный сайт: <https://www.pola.rs/>

Документация: [Introduction - Polars](#)

Страница библиотеки в PyPI: [polars · PyPI](#)

```
pip install polars
```

Исходный код: [GitHub - pola-rs/polars: Dataframes powered by a multithreaded, vectorized query engine, written in Rust](#)

- написана на Rust
- существенно быстрее pandas во многих реальных сценариях
- использует немного другие типы данных (расширяет набор, доступный pandas, в частности, имеет отдельный тип под строки)

```
import polars as pl

q = (
    pl.scan_csv("iris.csv")
    .filter(pl.col("sepal_length") > 5)
    .group_by("species")
    .agg(pl.all().sum())
)

df = q.collect()
```

- добавляет особый интерфейс к объектам DataFrame — LazyFrame (обеспечивает lazy-вычисления и запросы к датафреймам, см. стратегию [Ленивые вычисления — Википедия](#))
- использует Arrow ([Apache Arrow | Apache Arrow](#)), причем используется реализация, написанная на Rust ([GitHub - jorgecarleitao/arrow2: Transmute-free Rust library to work with the Arrow format](#))

💡 Tip

Также рекомендуется изучить [pyarrow · PyPI](#)

[Python — Apache Arrow v22.0.0](#) — связку Apache Arrow с Питоном, ее можно

использовать как бэкенд для pandas в том числе

В сложивших условиях на текущий момент максимальное число плюсов у комбинации pandas+pyarrow

| Лабораторная работа №4: Научные вычисления и анализ данных с библиотеками Python

Общие требования:

1. Использовать векторизованные операции вместо циклов, выбирать корректные типы данных для минимизации использования памяти
2. Все графики должны содержать подписи, легенды и заголовки
3. Если работаем с датасетами, то нужно корректно обрабатывать пропущенные значения и аномалии в данных

| Задание 1: Сравнительный анализ производительности NumPy и чистого Python

Требования к реализации:

1. Создайте наборы данных разного размера (10^3 , 10^4 , 10^5 , 10^6 элементов) для тестирования:

```
def generate_test_datasets() -> dict:
    return {
        'small': np.random.random(1000),
        'medium': np.random.random(10000),
        'large': np.random.random(100000),
        'xlarge': np.random.random(1000000)
    }
```

2. Реализуйте и сравните выполнение математических операций:

1. Поэлементное возведение в квадрат
 2. Вычисление синуса
 3. Суммирование всех элементов
 4. Поиск максимального элемента
3. Для каждой операции создайте две версии — на чистом Python и с использованием NumPy:

```
def py_square(data):
    return [x ** 2 for x in data]
```

```
def np_square(data):
    return np.square(data)
```

3. Измерьте время выполнения и потребление памяти для каждого подхода:

```
import time
import memory_profiler

def benchmark_operations():
    datasets = generate_test_datasets()
    results = {}

    for name, data in datasets.items():
        start_time = time.time()
        result = operation(data)
        end_time = time.time()

        mem_usage = memory_profiler.memory_usage((operation, (data,)))

        results[name] = {
            'time': end_time - start_time,
            'memory': max(mem_usage) - min(mem_usage)
        }

    return results
```

- Постройте сравнительные графики производительности используя matplotlib:
 - График времени выполнения в зависимости от размера данных
 - График потребления памяти в зависимости от размера данных
 - Тепловую карту (heatmap) относительного ускорения NumPy над чистым Python

Задание 2: Анализ и визуализация данных с pandas и seaborn

Требования к реализации:

- Загрузите датасет Titanic (`seaborn.load_dataset('titanic')`)
- Используя pandas, выполните:
 - Анализ структуры данных: типы столбцов, пропущенные значения
 - Статистическое описание числовых признаков
 - Группировку данных по полу и классу каюты с расчетом выживаемости
 - Создание новых признаков (feature engineering):

```
def create_features(df):  
    # добавим возрастные группы  
    df['age_group'] = pd.cut(df['age'],  
                             bins=[0, 18, 30, 50, 100],  
                             labels=['child', 'young', 'adult', 'senior'])  
  
    df['family_size'] = df['sibsp'] + df['parch'] # добавим размер семьи  
    return df
```

- Сравните производительность операций с разными типами данных:
 - Стандартные типы pandas (`object`, `int64`, `float64`)
 - Категориальные типы (`category`)
 - Типы PyArrow (`string[pyarrow]`, `int64[pyarrow]`)
- Используя seaborn, создайте комплексную витрину данных:
 - Матрицу корреляций, визуализированную тепловой картой (heatmap)
 - Распределение возрастов пассажиров с разбивкой по полу и выживаемости
 - Количество выживших в разрезе класса каюты и порта посадки
 - Ящик с усами (boxplot) стоимости билета по классам
- Реализуйте интерактивную визуализацию с возможностью фильтрации:

```
def create_interactive_dashboard(df):  
    fig = plt.figure(figsize=(15, 10))  
  
    age_filter = (18, 60)  
    class_filter = [1, 2, 3]
```

```

filtered_df = df[
    (df['age'].between(*age_filter)) &
    (df['class'].isin(class_filter))
]

gs = GridSpec(2, 2, figure=fig)
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])
ax3 = fig.add_subplot(gs[1, :])

# Графики с использованием filtered_df
return fig

```

Задание 3: Переход от NumPy/pandas на Polars и оптимизация больших данных

Требования к реализации:

- Создайте большой датасет (1 миллион записей или больше) для тестирования:

```

def generate_large_dataset(n_rows=1000000):
    return pd.DataFrame({
        'id': range(n_rows),
        'timestamp': pd.date_range('2020-01-01', periods=n_rows,
        freq='1min'),
        'category': np.random.choice(['A', 'B', 'C', 'D'], n_rows),
        'value1': np.random.normal(0, 1, n_rows),
        'value2': np.random.exponential(1, n_rows),
        'value3': np.random.randint(0, 100, n_rows)
    })

```

- Реализуйте каждую из перечисленных ниже операций в трех вариантах:

- NumPy/pandas
- pandas + PyArrow
- Polars

Операции:

- Фильтрация и агрегация
- Группировка (`.groupby()`) с вычислением статистик (`.agg(['sum', 'mean', 'count'])`)
- JOIN нескольких таблиц
- Скользящее среднее (rolling averages)
- Временные ряды (изменение частоты временного ряда — resampling)

☰ Пример на Polars

```
def polars_analysis(df):
    pl_df = pl.from_pandas(df)

    result = (pl_df
              .filter(pl.col('value1') > 0)
              .group_by('category')
              .agg([
                  pl.mean('value2').alias('avg_value'),
                  pl.std('value2').alias('std_value'),
                  pl.count().alias('count')
              ])
              .sort('avg_value', descending=True)
    )
    return result
```

6. Сравните производительность и использование памяти:

1. Время выполнения каждой операции
2. Пиковое использование памяти
3. Удобство API и читаемость кода

7. Реализуйте бенчмарк с генерацией отчета:

```
def generate_comparison_report(results):
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    # Графики сравнения производительности
    performance_data = prepare_performance_data(results)

    # Heatmap относительной производительности
    sns.heatmap(performance_data, annot=True, fmt='.2f',
                cmap='RdYlGn', center=1, ax=axes[0, 0])

    # График использования памяти
    plot_memory_usage(results, ax=axes[0, 1])

    return fig
```

| Задание 4: Научные вычисления с SciPy

1. Решите задачу оптимизации для различных функций:

1. Минимизация многомерной функции (Розенброка)
2. Поиск корней системы уравнений
3. Линейное программирование

☰ Оптимизация функции Розенброка

```
def rosenbrock(x):
    """Функция Розенброка"""
    return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

def benchmark_optimization():
    methods = ['BFGS', 'CG', 'Nelder-Mead', 'Powell']
    results = {}

    for method in methods:
        start_time = time.time()
        result = scipy.optimize.minimize(rosenbrock,
                                         x0=np.random.random(10)*2,
                                         method=method)
        end_time = time.time()

        results[method] = {
            'time': end_time - start_time,
            'iterations': result.nit,
            'success': result.success,
            'minimum': result.fun
        }

    return results
```

2. Реализуйте цифровую обработку сигнала:
 1. Загрузите аудиофайл или сгенерируйте тестовый сигнал
 2. Примените Фурье-анализ для выделения частот
 3. Используйте фильтры для очистки сигнала от шума
 4. Визуализируйте исходный и обработанный сигналы

Задание 5: Анализ набора данных качества вина

==UCI Wine Quality Dataset (красные вина): <https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv>

```
    'Высокое'])
return wine_data
```

Задачи:

1. Исследование характеристик:
 1. Распределение показателей качества вин
 2. Анализ выбросов в химических показателях
 3. Изучение корреляций между свойствами вина
2. Сравнительный анализ:
 1. Сравнение химического состава вин разного качества
 2. Влияние кислотности на общую оценку
 3. Анализ связи алкоголя и качества
3. Гипотезы и проверки:
 1. Влияние уровня сахара на воспринимаемое качество
 2. Связь между pH и кислотностью
 3. Статистическая проверка различий между группами качества (например, статистическая значимость различия содержания алкоголя в винах разного качества)