

Шпаргалка по Pandas / Pandas Cheatsheet (XeLaTeX)

Создано с использованием LaTeX шаблона
March 30, 2025

Contents

1 Основные Структуры / Core Structures	1
2 Загрузка и Сохранение / IO	1
3 Первичный Осмотр / Basic Inspection	1
4 Очистка и Предобработка / Cleaning & Preprocessing	1
5 Выборка и Фильтрация / Selection & Filtering	2
6 Сортировка / Sorting	2
7 Трансформация и Применение Функций / Transformation & Applying Functions	2
8 Группировка и Агрегация / Grouping & Aggregation	3
9 Объединение / Merging & Joining	3
10 Визуализация / Visualization	4
11 Полезные ссылки и советы / Links & Tips	4

1 Основные Структуры / Core Structures

Series (1D) и DataFrame (2D)

Series: Одномерный индексированный массив. Аналог столбца таблицы или словаря Python. ● **DataFrame:** Двумерная табличная структура с индексированными строками и столбцами. Основной объект для анализа данных. Состоит из объектов Series.

Создание Series и DataFrame

```
1 import pandas as pd
2 import numpy as np
3
4 # Series (индекс по умолчанию)
5 s = pd.Series([10, 20, np.nan, 40])
6
7 # DataFrame из словаря Python
8 data = {'col_A': [1, 2, 3], 'col_B': ['X', 'Y', 'Z']}
9 dates_index = pd.date_range('20240101', periods=3)
10 df = pd.DataFrame(data, index=dates_index)
11 # print(s)
12 # print(df)
```

2 Загрузка и Сохранение / IO

Чтение и запись данных

Ключевые функции для взаимодействия с файлами.

```
pd.read_csv('f.csv', sep=',')
```

Чтение CSV файла. Важные параметры: 'sep', 'header', 'index_col', 'usecols', 'parse_dates', 'dtype'.

```
df.to_csv('out.csv', index=False)
```

Запись DataFrame в CSV. Важные параметры: 'sep', 'index', 'header', 'columns'.

```
pd.read_excel('f.xlsx', sheet_name=0)
```

Чтение из файла Excel. Важные параметры: 'sheet_name', 'header', 'index_col'.

```
df.to_excel('out.xlsx', sheet_name='Data')
```

Запись в Excel. Важные параметры: 'sheet_name', 'index', 'header'.

```
pd.read_sql(query, conn)
```

Чтение из SQL базы данных (требуется 'sqlalchemy' или аналог).

```
pd.read_json('f.json', orient='records')
```

Чтение из JSON файла/строки. 'orient' важен для структуры.

3 Первичный Осмотр / Basic Inspection

Первичный анализ DataFrame ('df')

Получение общего представления о данных.

Команды для осмотра 'df'

```
1 df.head(3)           # Первые N строк (по умолч. 5)
2 df.tail(3)           # Последние N строк (по умолч. 5)
3 df.shape              # Размеры (строки, столбцы) - кортеж
4 df.info()             # Сводка: индекс, столбцы, non-null count, типы, память
5 # df.info(memory_usage='deep') # Более точная оценка памяти
6 df.describe()         # Статистика для числовых столбцов (count, mean, std, min, max, ...)
7 # df.describe(include='object') # Статистика для object/string (count, unique, top, freq)
8 # df.describe(include='all')   # Статистика для всех типов
9 df.columns            # Список названий столбцов
10 df.index              # Индекс строк
11 df.dtypes             # Типы данных в каждом столбце
12 s.value_counts()      # Подсчет уникальных значений в Series (столбце)
13 # s.value_counts(dropna=False) # Включая NaN
14 df['col_A'].nunique()   # Количество уникальных значений в столбце
```

4 Очистка и Предобработка / Cleaning & Pre-processing

Подготовка данных к анализу

Обработка аномалий и приведение данных к нужному формату.

Работа с пропусками (NaN)

```
1 df.isnull()          # Boolean DataFrame: True где NaN
2 df.isnull().sum()    # Количество NaN в каждом
                       # столбце
3 df.notnull()         # Boolean DataFrame: True где НЕ
                       # NaN
4 df.dropna()          # Удалить строки с любым NaN
5 # df.dropna(axis=1)  # Удалить столбцы с любым NaN
6 # df.dropna(subset=['col_A']) # Удалить строки с
                       # NaN только в 'col_A'
7 df.fillna(0)         # Заменить все NaN на 0
8 # df['col_A'].fillna(df['col_A'].mean()) # Замена
                       # средним по столбцу
```

Работа с дубликатами

```
1 df.duplicated() # Boolean Series: True для
                 # дублирующихся строк (кроме первого вхождения)
2 # df.duplicated(subset=['col_A', 'col_B']) #
                 # Проверка дубликатов по подмножеству столбцов
3 df.drop_duplicates() # Удалить дублирующиеся
                       # строки
4 # df.drop_duplicates(keep='last') # Оставить
                       # последнее вхождение
```

Изменение типов и переименование

```
1 # Изменение типа столбца
2 # df['col_A'].astype(float)
3 # df['date_col'] = pd.to_datetime(df['date_col'],
3 # format='%Y-%m-%d')
4 # df['category_col'].astype('category') # Экономия
5 # памяти для категориальных
6
6 # Переименование столбцов/индекса
7 # df.rename(columns={'old_name': 'new_name'},
7 # index={'old_idx': 'new_idx'})
```

5 Выборка и Фильтрация / Selection & Filtering

Доступ к данным

Индексация:

[] (столбцы), .loc[] (метки), .iloc[] (позиции).

Выбор столбцов и подмножеств

```
1 df['col_A']          # Выбор одного столбца (Series)
2 df.col_A             # Альтернатива (если имя валидно)
3 df[['col_A', 'col_B']] # Выбор нескольких столбцов
                       # (DataFrame)
```

Выбор строк/значений по МЕТКАМ (.loc)

```
1 # df.loc[label]      # Строка по метке индекса
2 # df.loc[start:end]  # Срез строк по меткам
                       # (включая end)
3 # df.loc[:, 'col_A'] # Все строки, столбец
                       # 'col_A'
4 # df.loc[label, 'col_A'] # Конкретное значение
```

Выбор строк/значений по ПОЗИЦИЯМ (.iloc)

```
1 # df.iloc[0]         # Первая строка (позиция 0)
2 # df.iloc[0:2]       # Первые две строки (срез до
                       # 2, НЕ включая 2)
3 # df.iloc[:, 0]      # Все строки, первый столбец
                       # (позиция 0)
4 # df.iloc[0, 0]      # Значение в [0, 0]
5 # df.iloc[[0, 2], [0, 1]] # Выбор по спискам
                       # позиций строк/столбцов
```

Фильтрация по условию (Boolean Indexing)

Мощный способ выбора строк на основе логических условий.

Примеры Boolean Indexing

```
1 df[df['col_A'] > 10] # Строки, где значение в
                       # col_A > 10
2 # Условия: & (И), | (ИЛИ), ~ (НЕ). Скобки
                       # обязательны!
3 df[(df['col_A'] > 10) & (df['col_B'] == 'x')]
4 df[df['col_B'].isin(['x', 'y'])] # Строки, где
                       # col_B равно 'x' или 'y'
5 # df[df['col_A'].between(10, 20)] # Значения между
                       # 10 и 20 включительно
6
7 # Использование .loc с булевым массивом
                       # (предпочтительнее для явности)
8 # df.loc[df['col_A'] > 10, ['col_B', 'col_C']] #
                       # Фильтр + выбор столбцов
```

6 Сортировка / Sorting

Сортировка данных

Упорядочивание строк DataFrame. По значениям

(sort_values) и по индексу

(sort_index).

Примеры сортировки

```
1 # Сортировка по значениям одного или нескольких
   # столбцов
2 df.sort_values(by='col_A') # По возрастанию
3 df.sort_values(by='col_A', ascending=False) # По
   # убыванию
4 # df.sort_values(by=['col_A', 'col_B']) # По
   # нескольким столбцам
5
6 # Сортировка по индексу
7 df.sort_index(ascending=False)
8
9 # Важно: сортировка возвращает НОВЫЙ DataFrame.
10 # Для изменения на месте: inplace=True
```

7 Трансформация и Применение Функций / Transformation & Applying Functions

Изменение и создание данных

Применение функций, работа со специальными типами, создание новых признаков.

Создание новых столбцов

```
1 # На основе существующих
2 # df['new_col'] = df['col_A'] * 2
3 # df['col_C'] = df['col_A'] + df['col_B'] #
   # Поэлементные операции
4
5 # С использованием np.where (аналог IF в
   # SQL/Excel)
6 # df['flag'] = np.where(df['col_A'] > 10, 'High',
   # 'Low')
```

Применение функций (apply, map, applymap)

```
1 # apply: применяет функцию к строкам (axis=1) или
   # столбцам (axis=0)
2 # df.apply(np.sum, axis=0) # Сумма по каждому
   # столбцу (Series)
3 # df.apply(lambda row: row['col_A'] *
   # row.name.day, axis=1) # Пример сложной функции по
   # строкам
4
5 # map: работает поэлементно на Series (для замены
   # или преобразования)
6 # s.map({'X': 1, 'Y': 2}) # Замена значений по
   # словарю
7 # s.map('Value: {}'.format) # Применение строковой
   # функции
8
9 # applymap: работает поэлементно на DataFrame
   # (применяет функцию к каждому элементу)
10 # df_numeric.applymap(lambda x: x**2) # Квадрат
   # каждого элемента
11 # ВАЖНО: Старайтесь использовать векторизованные
   # операции Pandas/NumPy вместо apply/applymap, если
   # это возможно (они быстрее!).
```

8 Группировка и Агрегация / Grouping & Aggregation

Split-Apply-Combine (Разделяй-Применяй-Объединяй)

Основа 'groupby': 1. **Split**: Данные делятся на группы по ключу. 2. **Apply**: Функция применяется к каждой группе. 3. **Combine**: Результаты объединяются.

GroupBy: Группировка и агрегация

Расчет сводных статистик по группам.

Работа со строками (.str)

```
1 # Доступ к строковым методам через аксессор .str
  для Series
2 # s_text = pd.Series(['apple', 'Banana ', 'kiwi
  '])
3 # s_text.str.lower()      # -> ['apple', ' banana
  ', 'kiwi ']
4 # s_text.str.strip()      # -> ['apple', 'Banana',
  'kiwi']
5 # s_text.str.contains('a')# -> [True, True, False]
6 # s_text.str.split('a')   # -> [['', 'pple'], ['
  B', 'n', 'n', ''], ['kiwi ']]
7 # s_text.str.replace('a', 'X') # -> ['Xpple', '
  BXnana ', 'kiwi ']
8 # s_text.str.len()        # Длина каждой строки
```

Работа с датами (.dt)

```
1 # Доступ к компонентам даты/времени через аксессор
  .dt для Series (типа datetime)
2 # s_dates =
  pd.to_datetime(pd.Series(['2024-01-05',
  '2024-02-10']))
3 # s_dates.dt.year         # -> [2024, 2024]
4 # s_dates.dt.month_name() # -> ['January',
  'February']
5 # s_dates.dt.dayofweek    # -> [4, 5]
  (Понедельник=0, Воскресенье=6)
6 # s_dates.dt.date         # Только дата (без
  времени)
7 # (s_dates - pd.Timedelta(days=1)) # Вычитание
  временного интервала
```

Примеры GroupBy

```
1 # Группировка по одному или нескольким столбцам
2 grouped = df.groupby('key_col')
3 # grouped = df.groupby(['key1', 'key2']) #
  Группировка по нескольким ключам
4
5 # Применение агрегирующих функций
6 # grouped.mean() # Среднее для всех числовых
  столбцов в каждой группе
7 # grouped['data_col'].sum() # Сумма для
  конкретного столбца в каждой группе
8 # grouped.size() # Размер каждой группы (включая
  NaN в ключах)
9 # grouped.count() # Количество НЕ-NaN значений в
  каждой группе/столбце
10
11 # Несколько агрегаций сразу с .agg()
12 # grouped['data_col'].agg(['sum', 'mean', 'std'])
13 # grouped.agg({
14 #     'data_col1': ['mean', 'min', 'max'], #
  Несколько функций к одному столбцу
15 #     'data_col2': 'nunique',              # Одна
  функция к другому
16 #     'data_col3': lambda x: x.max() - x.min() #
  Пользовательская lambda-функция
17 # })
18
19 # Применение функции к группам без агрегации
  (.transform)
20 # Часто используется для заполнения пропусков
  средним по группе или для нормализации
21 # df['group_mean'] =
  df.groupby('key_col')['data_col'].transform('mean')
22 # df['normalized'] = df['data_col'] /
  df.groupby('key_col')['data_col'].transform('sum')
23
24 # Фильтрация групп (.filter)
25 # Оставить только те группы, где среднее по
  data_col > 10
26 # filtered_groups =
  df.groupby('key_col').filter(lambda g:
  g['data_col'].mean() > 10)
```

Сводные таблицы / Pivot Tables

Аналог сводных таблиц в Excel для агрегации и изменения формы данных.

Пример pivot_table

```
1 # pd.pivot_table(df,
2 #                 values='Value_Column', # Столбец
  для агрегации
3 #                 index='Row_Index_Column', #
  Столбец(ы) для индекса строк
4 #                 columns='Column_Index_Column', #
  Столбец(ы) для названий столбцов
5 #                 aggfunc=np.sum, # Функция
  агрегации (sum, mean, count, ...)
6 #                 fill_value=0) # Значение для
  заполнения NaN после агрегации
7 #
8 # pd.crosstab(df['col_A'], df['col_B']) # Таблица
  сопряженности (частот)
```

Часто результатом groupby или pivot_table является DataFrame с MultiIndex!

9 Объединение / Merging & Joining

Объединение DataFrames

Комбинирование данных из разных источников. **merge()**

(SQL JOIN), **join()**

(по индексам), **concat()**

(склеивание).

Примеры объединения

```
1 # df1, df2 - примеры DataFrame
2 # merge: аналог SQL JOIN (по столбцам)
3 # pd.merge(df1, df2, on='key_col', how='inner') #
  Inner join по ключу
4 # pd.merge(df1, df2, left_on='key1',
  right_on='key2', how='left') # Left join по разным
  ключам
5 # how: 'inner' (по умолч.), 'outer', 'left',
  'right'
6
7 # concat: склеивание таблиц по оси (строк или
  столбцов)
8 # pd.concat([df1, df2], axis=0) # Склеить строки
  (ось 0)
9 # pd.concat([df1, df2], axis=1) # Склеить столбцы
  (ось 1) - важно совпадение индексов
10
11 # join: объединение по индексам (или индекс с
  ключом)
12 # df1.join(df2.set_index('key_col'), on='key_col',
  how='inner')
```

10 Визуализация / Visualization

Быстрая визуализация

Простые графики для первичного анализа с помощью 'plot()'. Использует Matplotlib под капотом.

Пример простого графика

```
1 # Требуется matplotlib: pip install matplotlib
2 import matplotlib.pyplot as plt
3
4 # df['numeric_col'].hist(bins=30) # Гистограмма
5 # df.plot(kind='scatter', x='col_A', y='col_B') #
  Диаграмма рассеяния
6 #
  df.groupby('category_col')['value_col'].mean().plot(kind='bar')
  # Столбчатая диаграмма средних по группам
7 # plt.show() # Отобразить график (часто не нужно в
  Jupyter)
8 # plt.savefig('img/my_plot.png') # Сохранить
  график
9 # plt.close()
```

11 Полезные ссылки и советы / Links & Tips

Полезные ресурсы

Pandas Documentation
Официальная документация^a

Stack Overflow [pandas]
Вопросы и ответы сообщества

Pandas GitHub
Исходный код библиотеки

^aPandas Development Team 2024.

Цитата / Quote

“ There should be one – and preferably only one –obvious way to do it.”
– The Zen of Python (import this) ”

Советы по производительности и Best Practices

- **Векторизация > Итерация:** Используйте встроенные функции Pandas/NumPy вместо циклов 'for' или 'iterrows()'. Векторизованные операции работают на порядки быстрее.
- **Тип 'category':** Для столбцов с небольшим количеством уникальных строковых значений используйте 'df[col].astype('category')'. Это значительно экономит память и ускоряет операции (особенно 'groupby').
- **'loc' vs '[]' для присваивания:** При изменении данных используйте 'loc' для избежания 'SettingWithCopyWarning'. Например: 'df.loc[mask, 'col'] = value'.
- **Работа с MultiIndex:** 'groupby' и 'pivot_table' часто возвращают MultiIndex. Изучите методы работы с ним ('reset_index()', 'unstack()', 'stack()'; выборка через кортежи).
- **Оценка памяти:** Используйте 'df.info(memory_usage='deep')' для более точной оценки занимаемой памяти DataFrame, особенно при наличии строковых данных.

References

[1] Pandas Development Team. *pandas documentation*. <https://pandas.pydata.org/docs/>. Accessed: 2025-03-30. 2024.