Шпаргалка по Pandas / Pandas Cheatsheet (XeLaTeX)

Краткий справочник по основным операциям

April 2, 2025

Contents

Необходимые импорты

Стандартные импорты для работы с Pandas и NumPy.

Импорты

```python
import pandas as pd
import numpy as np
# df = pd.DataFrame(...) # Предполагаемый DataFrame
# s = pd.Series(...) # Предполагаемый Series
```

1 Загрузка и Просмотр

Чтение и Запись

Основные операции ввода/вывода данных.

Чтение / Запись

```python
# Чтение
df_csv = pd.read_csv('file.csv', sep=',')
df_excel = pd.read_excel('file.xlsx')
# Запись
df.to_csv('output.csv', index=False)
df.to_excel('output.xlsx', index=False)
```

Первичный Осмотр

Базовые команды для понимания структуры данных.

Осмотр DataFrame (`df`)

```python
df.head()
df.tail(3)
df.shape
df.info()
df.describe()
df.describe(include='object')
df.columns
df.dtypes
```

2 Выборка: .loc vs .iloc

Доступ по Меткам и Позициям

`.loc`: по меткам индекса/названиям столбцов (включая правую границу среза).
● `.iloc`: по целочисленным позициям (НЕ включая правую границу среза).

Примеры .loc и .iloc

```python
# Выбор столбца(ов)
s = df['col_name']
df_subset = df[['col1', 'col2']]

# --- .loc (по МЕТКАМ) ---
df.loc['label'] # Строка
df.loc['start':'end'] # Срез строк (включительно)
df.loc[:, 'col_name'] # Столбец
df.loc['label', 'col_name'] # Значение
df.loc[['l1', 'l3'], ['c1', 'c2']] # По спискам

# --- .iloc (по ПОЗИЦИЯМ) ---
df.iloc[0] # Первая строка
df.iloc[0:5] # Первые 5 строк (не вкл. 5)
df.iloc[:, 0] # Первый столбец
df.iloc[0, 0] # Значение [0, 0]
df.iloc[[0, 2], [0, 1]] # По спискам позиций
```

3 Фильтрация (Boolean Indexing)

Выборка по Условиям

Используйте & (И), | (ИЛИ), ~ (НЕ), `isin()`, `between()`. Оборачивайте условия в `()`.

Примеры фильтрации

```python
df[df['score'] > 50]
df[(df['score'] > 50) & (df['attempts'] < 3)]
df[df['city'].isin(['London', 'Paris'])]
df[df['age'].between(18, 30)]
# Явное использование .loc
df.loc[df['score'] > 90, ['name', 'score']]
```

4 Сортировка

Упорядочивание Данных

Сортировка по значениям (`sort_values`) или индексу (`sort_index`).

Примеры сортировки

```python
# По значениям
df.sort_values(by='score') # Возрастание
df.sort_values(by='score', ascending=False) # Убывание
df.sort_values(by=['city', 'score'], ascending=[True, False])
# По индексу
df.sort_index(ascending=False)
# На месте (inplace=True)
# df.sort_values(by='score', inplace=True)
```

5 Удаление Дубликатов

Работа с Дубликатами

Обнаружение (`duplicated`) и удаление (`drop_duplicates`).

Примеры работы с дубликатами

```python
df.duplicated() # Проверка (bool Series)
df.duplicated(subset=['col1', 'col2'])
df.drop_duplicates() # Удалить (оставить первое)
df.drop_duplicates(keep='last') # Оставить последнее
df.drop_duplicates(subset=['user_id'], keep='first')
# На месте (inplace=True)
# df.drop_duplicates(inplace=True)
```

6 Функции и Агрегация

Применение Функций

`apply` (строки/столбцы), `map` (Series), `applymap` (DataFrame). Встроенные агрегации.

Применение функций

```
df['new_col'] = df['col_A'] * 10
df[['col_A', 'col_B']].apply(np.sum, axis=0)
df['cat_code'] = df['cat_name'].map({'A':1,
  'B':2})
# numeric_df.applymap(lambda x: x + 1)
```

Встроенные Агрегации

Расчет базовых статистик для Series или столбцов DataFrame.

Агрегирующие функции

```
df['score'].mean()
df['score'].sum()
df['score'].min()
df['score'].max()
df['score'].count() # Не-NA значения
df['category'].nunique() # Уникальные значения
```

Работа со Строками и Датами

Использование аксессоров `.str` и `.dt`.

Аксессоры .str и .dt

```
# Строки (.str)
df['name'].str.lower()
df['address'].str.contains('Street')
df['city'].str.replace(' ', '_')
# Даты (.dt, столбец должен быть datetime)
# df['date_col'] = pd.to_datetime(df['date_col'])
df['date_col'].dt.year
df['date_col'].dt.month_name()
df['date_col'].dt.dayofweek # Пн=0, Вс=6
```

7 Группировка и Сводные Таблицы

GroupBy (Разделяй-Применяй-Объединяй)

Группировка данных с последующей агрегацией.

Примеры GroupBy

```
grouped = df.groupby('category')
grouped_multi = df.groupby(['cat', 'status'])
grouped['value'].sum()
grouped['value'].agg(['mean', 'std', 'count'])
grouped_multi.agg({
    'value': 'sum',
    'score': ['mean', 'max'],
    'id': 'nunique'})
grouped.size() # Размер групп
# df['g_mean'] =
  df.groupby('cat')['val'].transform('mean')
```

Сводные Таблицы

Использование `pivot_table` и `crosstab` для агрегации и анализа частот.

Pivot Table и CrossTab

```
# Pivot Table
pd.pivot_table(df, values='score',
  index='category',
               columns='status', aggfunc=np.mean,
               fill_value=0)
# Cross Tabulation (Таблица сопряженности)
pd.crosstab(df['category'], df['status'])
```

8 Работа с Пропусками (NaN)

Обработка NaN

Обнаружение, удаление и заполнение пропущенных данных.

Операции с NaN

```
df.isnull().sum() # Количество NaN
df.dropna() # Удалить строки с NaN
df.dropna(axis=1) # Удалить столбцы с NaN
df.dropna(subset=['col_A']) # По столбцу
df.fillna(0) # Заполнить нулем
df['col_A'].fillna(df['col_A'].mean()) # Средним
df.fillna(method='ffill') # Пред. значением
df.fillna(method='bfill') # След. значением
```

9 Объединение DataFrames

Merge, Concat, Join

Комбинирование таблиц. `merge` (SQL-like), `concat` (склеивание), `join` (по индексам).

Примеры объединения

```
# Merge (по ключу 'key')
pd.merge(df1, df2, on='key', how='inner')
# Concat (склеить строки)
pd.concat([df1, df2], axis=0)
# Concat (склеить столбцы)
pd.concat([df1, df2], axis=1)
# Join (по индексам)
df1.join(df2, how='inner')
```

10 Изменение Формы (Reshaping)

Stack, Unstack, Melt

Преобразование структуры DataFrame.

Reshaping

```
# Stack (столбцы -> индекс)
stacked = df.stack()
# Unstack (индекс -> столбцы)
unstacked = stacked.unstack()
# Melt ("расплавление" столбцов)
pd.melt(df, id_vars=['id'], value_vars=['A', 'B'],
        var_name='variable', value_name='value')
```

11 Временные Ряды (Time Series)

Работа с Датами/Временем

Преобразование, доступ к компонентам, создание диапазонов, смещение, ресемплинг.

Операции Time Series

```
# Преобразование и установка индекса
df['time_col'] = pd.to_datetime(df['time_col'])
df.set_index('time_col', inplace=True)
# Доступ к компонентам (если индекс)
# year = df.index.year
# Создание диапазона
rng = pd.date_range('2024-01-01', periods=5,
freq='D')
# Смещение
df.shift(1)
# Ресемплинг (агрегация)
df['value'].resample('M').sum() # По месяцам
df['value'].resample('W').mean() # По неделям
```

12 Преобразование Типов

Изменение Типов Данных

Явное приведение столбцов к нужным типам (`astype`, `to_numeric`).

Приведение типов

```
# В числовой (с заменой ошибок на NaN)
df['num'] = pd.to_numeric(df['num'],
errors='coerce')
# В строку
df['col_str'] = df['col_str'].astype(str)
# В категорию (экономия памяти)
df['cat'] = df['cat'].astype('category')
# В datetime (см. раздел Time Series)
# df['date'] = pd.to_datetime(df['date'])
df.dtypes # Проверка типов
```

13 Базовая Визуализация

Быстрые Графики `.plot()`

Создание простых графиков для EDA (требует 'matplotlib').

Примеры `.plot()`

```
# import matplotlib.pyplot as plt
# Линейный график
df['value'].plot(kind='line', title='Trend')
# Гистограмма
df['num_col'].plot(kind='hist', bins=20)
# Столбчатая диаграмма
df.groupby('cat')['val'].mean().plot(kind='bar')
# Диаграмма рассеяния
df.plot(kind='scatter', x='col_A', y='col_B')
# plt.show() # Показать график
```