Шпаргалка по Pandas / Pandas Cheatsheet

Краткий справочник по основным операциям

Содержание

1 Загрузка и Просмотр Данных

1.1. A Чтение и Запись

Основные операции ввода/вывода

Чтение данных из файлов различных форматов и запись DataFrame в файлы.

Чтение / Запись

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

1.2. B Первичный Осмотр DataFrame

Базовые атрибуты и методы

Команды для получения общего представления о структуре, типах данных и статистиках DataFrame (‘df‘).

Осмотр DataFrame (‘df‘)

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

2 Выборка Данных

2.1. A Базовый Выбор Столбцов

Доступ к столбцам по имени

Выбор одного или нескольких столбцов DataFrame по их названиям.

Выбор столбцов

```
s = df['col_name']
df_subset = df[['col1', 'col2']]
```

2.2. B Доступ по Меткам и Позициям: `.loc` vs `.iloc`

Ключевое различие

Два основных метода для индексации и выборки данных в Pandas:

- `.loc`: Выборка по **меткам** индекса и **названиям** столбцов. Правая граница среза **включается**.
- `.iloc`: Выборка по **целочисленным позициям** (индексам строк и столбцов, начиная с 0). Правая граница среза **не включается** (как в Python).

Примеры использования `.loc` и `.iloc`

Примеры .loc и .iloc

```
df.loc['label']
df.loc['start':'end']
df.loc[:, 'col_name']
df.loc['label', 'col_name']
df.loc[['l1', 'l3'], ['c1', 'c2']]
df.iloc[0]
df.iloc[0:5]
df.iloc[:, 0]
df.iloc[0, 0]
df.iloc[[0, 2], [0, 1]]
```

3 Фильтрация (Boolean Indexing)

3.1. A Фильтрация по Условиям

Выборка строк на основе логических условий

Используйте логические операторы & (И), | (ИЛИ), ~ (НЕ) для комбинирования условий. Условия необходимо оборачивать в круглые скобки `()`. Также полезны методы `isin()` и `between()`.

Примеры Булевой Индексации

```
df[df['score'] > 50]
df[(df['score'] > 50) & (df['attempts'] < 3)]
df[(df['city'] == 'Moscow') | (df['city'] ==
'London')]
df[df['city'].isin(['London', 'Paris', 'Tokyo'])]
df[df['age'].between(18, 30)]
df.loc[df['score'] > 90, ['name', 'score']]
df[~(df['city'] == 'Moscow')]
```

4 Сортировка

4.1. A Сортировка DataFrame

Упорядочивание данных

Сортировка строк DataFrame по значениям в одном или нескольких столбцах (`sort_values`) или по меткам индекса (`sort_index`).

Примеры сортировки

```
df.sort_values(by='score')
df.sort_values(by='score', ascending=False)
df.sort_values(by=['city', 'score'],
ascending=[True, False])
df.sort_index()
df.sort_index(ascending=False)
```

5 Работа с Дубликатами

5.1. A Обнаружение и Удаление Дубликатов

Идентификация и удаление повторяющихся строк

Методы для проверки наличия дубликатов (`duplicated`) и их удаления (`drop_duplicates`).

Примеры работы с дубликатами

```
df.duplicated()
df.duplicated(subset=['user_id', 'timestamp'])
df.drop_duplicates()
df.drop_duplicates(keep='last')
df.drop_duplicates(subset=['user_id'],
keep='first')
```

6 Применение Функций и Агрегация

6.1. A Применение Функций к Элементам, Строкам, Столбцам

Поэлементные и построчные/постолбцовые операции

Применение встроенных или пользовательских функций:

- **Поэлементные операции:** Векторизованные операции NumPy или базовые арифметические операции.
- `map()`: Применение функции или словаря к каждому элементу **Series**.
- `apply()`: Применение функции вдоль оси DataFrame (`axis=0` к столбцам, `axis=1` к строкам).
- `applymap()`: Применение функции к каждому элементу **DataFrame**. (Менее рекомендуется, часто есть векторизованные альтернативы).

Примеры применения функций

```
df['new_col'] = df['col_A'] * 10
df['col_B_log'] = np.log(df['col_B'])
df['cat_code'] = df['category'].map({'A':1, 'B':2,
'C':3})
df[['col_A', 'col_B']].apply(np.sum, axis=0)
df['row_sum'] = df[['col_A',
'col_B']].apply(np.sum, axis=1)
df['col_C_processed'] = df['col_C'].apply(lambda
x: x**2 if x > 0 else 0)
```

6.2. B Встроенные Агрегирующие Функции

Расчет сводных статистик

Быстрый расчет основных статистик для Series или столбцов DataFrame. Игнорируют NaN по умолчанию.

Примеры агрегаций

```
df['score'].mean()
df['score'].median()
df['score'].sum()
df['score'].min()
df['score'].max()
df['score'].std()
df['score'].var()
df['score'].count()
df['score'].nunique()
df['category'].unique()
df['category'].value_counts()
df.mean()
```

6.3. C Работа со Строками (`.str`)

Векторизованные строковые операции

Аксессор `.str` предоставляет доступ к множеству методов для работы со строками в Series.

Примеры .str методов

```
df['name'].str.lower()
df['name'].str.upper()
df['name'].str.title()
df['address'].str.contains('Street')
df['email'].str.startswith('info@')
df['code'].str.endswith('Z')
df['product_id'].str.isdigit()
df['city'].str.replace(' ', '_')
df['full_name'].str.split(' ')
df['comment'].str.len()
```

6.4. D Работа с Датами/Временем (`.dt`)

Доступ к компонентам даты/времени

Аксессор `.dt` предоставляет доступ к компонентам даты и времени для Series с типом данных 'datetime64[ns]'. Предварительно столбец нужно преобразовать.

Примеры .dt атрибутов и методов

```
df['date_col'].dt.year
df['date_col'].dt.month
df['date_col'].dt.day
df['date_col'].dt.hour
df['date_col'].dt.minute
df['date_col'].dt.second
df['date_col'].dt.dayofweek
df['date_col'].dt.dayofyear
df['date_col'].dt.quarter
df['date_col'].dt.month_name()
df['date_col'].dt.day_name()
df['date_col'].dt.strftime('%Y-%m-%d')
```

7 Группировка и Сводные Таблицы

7.1. A GroupBy: Разделяй-Применяй-Объединяй

Агрегация данных по группам

Механизм `groupby()` позволяет:

1. **Разделить** данные на группы на основе значений в одном или нескольких столбцах.
2. **Применить** агрегирующую функцию (sum, mean, count, etc.) или трансформацию к каждой группе независимо.

3. **Объединить** результаты в новую структуру данных (Series или DataFrame).

Примеры GroupBy

```
grouped = df.groupby('category')
grouped_multi = df.groupby(['category', 'status'])
grouped['value'].sum()
grouped['value'].mean()
grouped['value'].agg(['mean', 'std', 'count'])
grouped_multi.agg(
    total_value=('value', 'sum'),
    avg_score=('score', 'mean'),
    max_score=('score', 'max'),
    unique_ids=('id', 'nunique')
)
grouped.size()
```

7.2. В Сводные Таблицы и Таблицы Сопряженности

Создание агрегированных таблиц

`pivot_table`: Создает сводную таблицу в стиле Excel. Позволяет агрегировать данные по двум или более категориальным переменным. ● `crosstab`: Вычисляет таблицу сопряженности (частот) для двух или более факторов.

Pivot Table и CrossTab

```
pd.pivot_table(df,
               values='score',
               index='category',
               columns='status',
               aggfunc=np.mean,
               fill_value=0)
pd.crosstab(df['category'], df['status'])
```

8 Работа с Пропусками (NaN)

8.1. А Обработка Пропущенных Значений (NaN)

Обнаружение, удаление и заполнение NaN

Стандартные подходы к работе с пропущенными данными ('NaN', 'None', 'NaT').

Операции с NaN

```
df.isnull()
df.isna()
df.notnull()
df.isnull().sum()
df.dropna()
df.dropna(axis=1)
df.dropna(subset=['col_A', 'col_B'])
df.dropna(thresh=2)
df.fillna(0)
df['col_A'].fillna('Unknown')
df['col_B'].fillna(df['col_B'].mean())
df.fillna(method='ffill')
df.fillna(method='bfill')
```

9 Объединение DataFrames

9.1. А Комбинирование DataFrames

Основные методы объединения

- `pd.concat()`: "Склеивание" объектов Pandas вдоль оси (строк `axis=0` или столбцов `axis=1`). Полезно для добавления строк/столбцов.
- `pd.merge()`: Объединение в стиле SQL баз данных. Соединяет строки из двух DataFrame на основе одного или нескольких общих столбцов (**ключей**) или индексов.
- `df.join()`: Удобный метод для объединения по **индексу** (или по ключу в одном DataFrame и индексу в другом). Часто используется как обертка над `merge`.

Примеры объединения

Предполагаем наличие 'df1' и 'df2'.

Merge, Concat, Join

```
combined_rows = pd.concat([df1, df2], axis=0,
ignore_index=True)
combined_cols = pd.concat([df1, df2], axis=1)
merged_inner = pd.merge(df1, df2, on='key',
how='inner')
merged_left = pd.merge(df1, df2, on='key',
how='left')
joined_index = df1.join(df2, how='inner',
lsuffix='_df1', rsuffix='_df2')
```

10 Изменение Формы (Reshaping)

10.1. А Изменение Структуры: `stack`, `unstack`, `melt`

Преобразование между "широким" и "длинным" форматами

- `stack()`: "Укладывает" столбцы DataFrame в индекс, создавая Series (или DataFrame с MultiIndex). Переход от "широкого" к "длинному" формату.
- `unstack()`: Обратная операция к `stack()`. "Поднимает" уровень индекса в столбцы. Переход от "длинного" к "широкому" формату.

- `melt()`: "Расплавляет" DataFrame, преобразуя столбцы в строки. Полезно для создания "длинного" формата данных из "широкого".

Примеры Reshaping

```
df_melted = pd.melt(df,
                    id_vars=['id', 'name'],
                    value_vars=['Score_Q1',
                    'Score_Q2'],
                    var_name='Quarter',
                    value_name='Score')
```

11 Временные Ряды (Time Series)

11.1. А Основные Операции с Временными Рядами

Работа с датами и временем как индексом

Pandas предоставляет мощные инструменты для работы с временными рядами, особенно когда индекс DataFrame имеет тип 'DatetimeIndex'.

Операции с Time Series

```
date_rng = pd.date_range(start='2024-01-01',
end='2024-01-10', freq='D')
time_rng = pd.date_range('2024-01-01', periods=5,
freq='H')
df['previous_value'] = df['value'].shift(1)
df['change'] = df['value'] - df['value'].shift(1)
```

Доступ к компонентам даты/времени через `.dt` (см. VI.D) работает и для 'DatetimeIndex'.

12 Преобразование Типов Данных

12.1. А Изменение Типов Данных Столбцов

Явное приведение типов

Преобразование столбцов к нужным типам данных для корректной обработки или экономии памяти.

Приведение типов

```
print(df.dtypes)
df['numeric_col'] =
pd.to_numeric(df['numeric_col'], errors='coerce')
df['string_col'] = df['string_col'].astype(str)
df['category_col'] =
df['category_col'].astype('category')
print(df.dtypes)
```

13 Базовая Визуализация

13.1. A Быстрая Визуализация с `.plot()`

Встроенные методы для графиков

Pandas интегрируется с Matplotlib, позволяя быстро строить базовые графики прямо из DataFrame или Series с помощью метода `.plot()`. Необходим импорт 'matplotlib.pyplot'.

Примеры '.plot()'

```
df['value_ts'].plot(kind='line', title='Trend Over
Time', figsize=(10, 4))
df['numeric_col'].plot(kind='hist', bins=30,
title='Distribution')
df.plot(kind='scatter', x='col_A', y='col_B',
title='Scatter Plot A vs B')
df['numeric_col'].plot(kind='kde', title='Density
Plot')
```