Глава II: Основи на Pandas

В тази глава ще се потопим в света на библиотеката Pandas - ключов инструмент за анализ и обработка на данни в Python. Ще разгледаме нейната философия и основни предимства, които я правят толкова популярна сред специалистите по данни. Ще научим как да инсталираме и импортираме библиотеката, както и ще се запознаем с двете основни структури от данни, които Pandas предлага: Series (едномерен етикетиран масив) и DataFrame (двумерна таблична структура). Ще разгледаме различни начини за създаване на тези структури от данни от познати Python обекти като списъци и речници, както и от NumPy масиви. Накрая, ще се научим как да инспектираме основните атрибути на Series и DataFrame, които ни дават важна информация за техния състав и структура, и ще направим първите си стъпки в създаването и разглеждането на прости DataFrame-и. След тази глава ще имате солидна основа за по-нататъшно изследване на мощните възможности на Pandas за анализ и манипулация на данни.

1. Какво представлява библиотеката Pandas?

Pandas е мощна и гъвкава библиотека за анализ и обработка на данни в Python. Тя е фундаментален инструмент за всеки, който работи с данни в Python, особено в области като анализ на данни, наука за данни (data science), машинно обучение и финансов анализ.

a) Философия на Pandas:

Основната философия на Pandas е да предостави бързи, гъвкави и изразителни структури от данни, които да улеснят работата с "релационни" или "маркирани" данни. Тези типове данни са често срещани в реалния свят и включват таблични данни (като електронни таблици или SQL таблици), времеви серии, наблюдавани данни и други.

Pandas се стреми да направи процеса на анализ на данни по-интуитивен и ефективен, като позволява на потребителите да извършват операции като:

- **Зареждане и записване на данни** от различни формати (CSV, Excel, SQL бази данни, JSON, HTML и други).
- **Почистване и подготовка на данни** (справяне с липсващи стойности, филтриране, сортиране, преобразуване на типове данни).
- **Трансформиране на данни** (сливане, свързване, преоформяне, агрегиране).
- **Анализиране на данни** (изчисляване на описателни статистики, групиране, прилагане на функции).
- **Визуализация на данни** (в комбинация с други библиотеки като Matplotlib и Seaborn).

б) Основни предимства на Pandas:

- **Мощни и гъвкави структури от данни:** Pandas въвежда две основни структури от данни:
  - **Series:** Едномерен, етикетиран масивоподобен обект, способен да съхранява данни от всякакъв тип (цели числа, числа с плаваща запетая, низове, булеви стойности и др.). Етикетите на данните се наричат "индекси".
  - **DataFrame:** Двумерна, таблична структура от данни с етикетирани редове и колони. Може да се разглежда като речник от Series обекти, които споделят общ индекс.
- **Лесна работа с липсващи данни:** Pandas предоставя удобни начини за представяне и обработка на липсващи стойности (често означени като `NaN`).
- **Интуитивни операции за сливане и свързване на данни:** Библиотеката предлага гъвкави методи за комбиниране на данни от различни източници.
- **Мощни инструменти за групиране:** Възможност за групиране на данни по определени критерии и прилагане на агрегиращи функции (сума, средна стойност, брой и др.).
- **Гъвкаво преоформяне на данни:** Лесно преминаване между различни представяния на данните (например от "дълъг" към "широк" формат).
- **Бърза производителност:** Много от основните операции в Pandas са имплементирани на C или Cython, което осигурява висока производителност, особено при работа с големи набори от данни.
- **Интеграция с други Python библиотеки:** Pandas работи добре с други научни библиотеки на Python като NumPy (за числени изчисления), Matplotlib и Seaborn (за визуализация), и scikit-learn (за машинно обучение). Всъщност, Pandas е изградена върху NumPy, което обяснява нейната бързина и ефективност при работа с масиви от данни.

в) Приложения на Pandas:

Pandas е широко използвана в различни области, включително:

- **Анализ на данни:** Изследване и разбиране на данни от различни източници.
- **Финансов анализ:** Работа с финансови времеви серии, анализ на пазарни данни.
- **Статистика:** Изчисляване на статистически мерки, тестване на хипотези.
- **Машинно обучение:** Подготовка и предварителна обработка на данни за модели на машинно обучение.
- **Биоинформатика:** Анализ на биологични данни.
- **Геопространствен анализ:** Работа с географски данни.
- **Бизнес анализ:** Извличане на прозрения от бизнес данни.
- **Уеб анализ:** Обработка и анализ на данни от уебсайтове.

Накратко, Pandas е незаменим инструмент за всеки, който работи с данни в Python. Нейните мощни и гъвкави структури от данни и богатият набор от функции улесняват процеса на анализ и манипулация на данни, което води до по-ефективна и продуктивна работа.

2. Инсталиране на Pandas

Pandas е външна библиотека, която не е част от стандартната инсталация на Python. За да я използвате, трябва да я инсталирате първо. Най-често използваният и препоръчителен начин за инсталиране на Python пакети, включително Pandas, е чрез пакетния мениджър **pip** (Pip Installs Packages).

а) Предпоставки:

- Уверете се, че имате инсталиран Python на вашата система. Можете да проверите версията на Python, като отворите командния ред (или терминал на macOS/Linux) и изпълните командата:

```
python --version
```

или

```
python3 --version
```

- Уверете се, че `pip` е инсталиран. Повечето съвременни инсталации на Python включват `pip` по подразбиране. Можете да проверите дали `pip` е инсталиран и неговата версия с командата:

```
pip --version
```

или

```
pip3 --version
```

- Ако `pip` не е инсталиран, ще трябва да го инсталирате отделно. Инструкции за инсталиране на `pip` могат да бъдат намерени в официалната документация на pip (https://pip.pypa.io/en/stable/installation/).

б) Стъпки за инсталиране на Pandas:

1) **Отворете командния ред (или терминал):**
   - **Windows:** Натиснете `Win + R`, въведете `cmd` и натиснете Enter.
   - **macOS:** Отворете "Terminal" от Applications -> Utilities -> Terminal.
   - **Linux:** Отворете терминал (обикновено с комбинация от клавиши като `Ctrl + Alt + T`).
2) **Използвайте `pip` за инсталиране на Pandas:** В командния ред (или терминал) изпълнете следната команда:

```
pip install pandas
```

или, ако имате няколко версии на Python, може да се наложи да използвате `pip3`:

```
pip3 install pandas
```

• `pip` ще се свърже с Python Package Index (PyPI), ще изтегли необходимите файлове за библиотеката Pandas и ще ги инсталира във вашата Python среда.

1) **Изчакване на процеса на инсталация:** Ще видите поредица от съобщения в конзолата, докато `pip` изтегля и инсталира Pandas и всички нейни зависимости (като NumPy, тъй като Pandas е изградена върху NumPy). Процесът може да отнеме няколко минути в зависимост от вашата интернет връзка.
2) **Проверка на инсталацията (по избор):** След като инсталацията завърши успешно, не е необходимо да правите нищо допълнително, за да използвате Pandas във вашите Python скриптове. Въпреки това, можете да проверите дали Pandas е инсталиран, като стартирате

Python интерпретатора и се опитате да импортирате библиотеката. След като Python интерпретаторът се стартира, изпълнете:

```
import pandas as pd

print(pd.__version__)
```

Ако не получите съобщение за грешка и видите отпечатана версията на Pandas, значи инсталацията е била успешна. Затворете интерпретатора с `exit()` или `Ctrl + D`.

в) Важни бележки:

- **Виртуални среди:** Препоръчително е да инсталирате Pandas (и други пакети) във виртуална среда, особено когато работите по различни проекти. Виртуалните среди помагат да се изолират зависимостите на всеки проект и да се избегнат конфликти между различни версии на едни и същи библиотеки. Ако използвате виртуална среда, уверете се, че сте я активирали, преди да изпълните командата `pip install pandas`.
- **Проблеми при инсталация:** Ако срещнете проблеми по време на инсталацията, уверете се, че имате актуална версия на `pip`. Можете да я обновите с командата:

```
pip install -- upgrade pip
```

или

```
pip3 install -- upgrade pip
```

- Също така, проверете дали имате необходимите права за запис в директорията, където се инсталират Python пакетите.

След като изпълните тези стъпки, библиотеката Pandas ще бъде успешно инсталирана и готова за използване във вашите Python проекти.

3. Импортиране на библиотеката Pandas

В Python, за да използваме функционалността на външни библиотеки като Pandas, трябва да ги импортираме в нашия скрипт или интерактивна сесия. Конвенцията при импортирането на библиотеката Pandas е да се използва псевдонимът `pd`. Това е широко прието и следвано от Python общността, особено сред хората, работещи с анализ на данни.

а) Синтаксис за импортиране:

За да импортирате библиотеката Pandas с псевдоним `pd`, използвате следната `import ... as ...` конструкция:

```
import pandas as pd
```

- **Какво се случва при импортирането:**

Когато Python интерпретаторът срещне тази команда, той зарежда модула `pandas` и го прави достъпен във вашия код чрез краткия и удобен псевдоним `pd`. След като библиотеката е импортирана по този начин, можете да достъпвате всички функции, класове и обекти, дефинирани в Pandas, като използвате `pd.` пред тях.

- **Пример:**

Да предположим, че искате да създадете обект от типа `Series` (една от основните структури от данни в Pandas). След като сте импортирали библиотеката като `pd`, можете да направите това по следния начин:

```
import pandas as pd


# Създаване на Series обект
my_series = pd.Series([10, 20, 30, 40, 50])


# Отпечатване на Series обекта
print(my_series)
```

В този пример, `pd.Series()` е функция от библиотеката Pandas, която използваме, за да създадем Series обект. Благодарение на конвенцията за импортиране като `pd`, кодът става по-кратък и по-лесен за четене, особено когато се използват много функции от библиотеката.

б) Защо се използва псевдонимът `pd`?

- **Краткост:** `pd` е много по-кратък от `pandas`, което намалява количеството на писане и прави кода по-компактен.
- **Четимост:** Използването на стандартен псевдоним прави кода по-лесен за разбиране от други програмисти, които също са запознати с конвенциите на Pandas.
- **Общностен стандарт:** Следването на общностните стандарти улеснява сътрудничеството и споделянето на код.

> **В повечето Python скриптове и Jupyter Notebook, които работят с Pandas, ще видите библиотеката да бъде импортирана по този начин в самото начало. Това е първата стъпка, преди да можете да използвате каквато и да е функционалност, предоставена от Pandas.**

4. Основни структури от данни в Pandas: Series и DataFrame

Pandas въвежда две основни структури от данни, които са изключително мощни и гъвкави за работа с различни видове данни:

а) Series:

- **Едномерна структура:** Series представлява едномерна (1D) етикетирана масивоподобна структура. Може да съдържа данни от всякакъв тип (цели числа, числа с плаваща запетая, низове, булеви стойности, Python обекти и др.).
- **Етикетиран индекс:** За разлика от обикновените Python списъци или NumPy масиви, елементите в Series имат асоциирани етикети, наречени **индекс**. По подразбиране, индексът е поредица от цели числа, започваща от 0, подобно на Python списъците. Въпреки това, можете изрично да зададете персонализиран индекс с различни типове данни (например низове, дати и др.).
- **Визуализация:** Можете да си представите Series като колона от данни в електронна таблица или SQL таблица, заедно с нейните етикети на редовете (индекса).

б) DataFrame:

- **Двумерна структура:** DataFrame е двумерна (2D) таблична структура от данни с етикетирани редове (индекс) и етикетирани колони.
- **Речник от Series:** Може да се разглежда като речник от Series обекти, където всеки Series представлява една колона, а всички Series в DataFrame споделят общ индекс (етикетите на редовете).
- **Таблични данни:** DataFrame е най-често използваната структура в Pandas и е много подобна на таблица в електронна таблица (например Excel) или SQL таблица. Всяка колона може да съдържа данни от различен тип.
- **Гъвкавост:** DataFrame предлага голяма гъвкавост за работа с данни, включително селектиране, филтриране, сливане, преоформяне и много други операции.

в) Връзката между Series и DataFrame:

DataFrame може да бъде разглеждан като колекция от Series обекти, които са подредени в колони и споделят общ индекс. Всяка колона в DataFrame е всъщност един Series.

Примери за аналогия:

Структура от данни	Аналогия	Измерения	Етикетиране
`Python List`	Едномерна подредена колекция от елементи	1D	Позиционен индекс (0, 1, 2...)
`NumPy Array`	Многомерен хомогенен масив от елементи	N-D	Позиционен индекс (0, 1, 2...)
`Pandas Series`	Едномерна етикетирана колона от данни	1D	Персонализиран индекс
`Pandas DataFrame`	Двумерна таблица от данни	2D	Индекси на редове и колони

5. Създаване на Series

Можем да създадем `Series` обект по няколко начина, в зависимост от типа на данните, които имаме и как искаме да бъде етикетиран индексът.

a) Създаване на Series от Python списък:

Когато създаваме `Series` от Python списък, Pandas автоматично създава целочислен индекс по подразбиране,

```
import pandas as pd


# Създаване на Series от списък
data_list = [10, 20, 30, 40, 50]
series_from_list = pd.Series(data_list)


print("Series от списък:")
print(series_from_list)
```

```
Изход:
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

В изхода виждаме данните от списъка и автоматично генерирания индекс (0 до 4). dtype: int64 показва типа на данните в Series (64-битови цели числа).

Можем също така да зададем персонализиран индекс, като подадем списък с етикети на аргумента `index`:

```
import pandas as pd


# Създаване на Series със зададен индекс
data_list = [10, 20, 30, 40, 50]
index_labels = ['a', 'b', 'c', 'd', 'e']
series_with_index = pd.Series(data_list, index=index_labels)


print("\nSeries със зададен индекс:")
```

```
print(series_with_index)
```

```
Изход:


a    10
b    20
c    30
d    40
e    50
dtype: int64
```

Сега елементите на Series са етикетирани с предоставените низови стойности.

б) Създаване на Series от NumPy array:

Създаването на `Series` от NumPy array е много подобно на създаването от списък. Отново, ако не е зададен индекс, ще бъде създаден целочислен индекс по подразбиране.

```
import pandas as pd
import numpy as np

# Създаване на NumPy array
numpy_array = np.array([5, 10, 15, 20])

# Създаване на Series от NumPy array
series_from_array = pd.Series(numpy_array)

print("\nSeries от NumPy array:")
print(series_from_array)
```

```
Изход:
0     5
1    10
2    15
3    20
```

```
dtype: int64
```

Можем и тук да зададем персонализиран индекс:

```
import pandas as pd
import numpy as np

# Създаване на NumPy array
numpy_array = np.array([5, 10, 15, 20])
index_labels = [100, 101, 102, 103]
series_with_array_index = pd.Series(numpy_array, index=index_labels)

print("\nSeries от NumPy array със зададен индекс:")
print(series_with_array_index)
```

```
Изход:
100     5
101    10
102    15
103    20
dtype: int64
```

в) Създаване на Series от Python речник:

Когато създаваме `Series` от Python речник, ключовете на речника автоматично стават индекс на Series, а стойностите на речника стават стойности на Series.

```
import pandas as pd

# Създаване на речник
data_dict = {'apple': 1, 'banana': 2, 'cherry': 3, 'date': 4}

# Създаване на Series от речник
series_from_dict = pd.Series(data_dict)
```

```
print("\nSeries от речник:")
print(series_from_dict)
```

```
Изход:
apple     1
banana    2
cherry    3
date      4
dtype: int64
```

Забележете, че индексът е съставен от ключовете на речника ('apple', 'banana', 'cherry', 'date'), а съответните стойности са стойностите на речника (1, 2, 3, 4).

Ако искаме да контролираме реда на индекса при създаване на Series от речник, можем да подадем списък с желаните ключове на аргумента `index`. Ако някой ключ от списъка не съществува в речника, съответната стойност в Series ще бъде `NaN` (Not a Number), което представлява липсваща стойност.

```
import pandas as pd

# Създаване на речник
data_dict = {'apple': 1, 'banana': 2, 'cherry': 3}
index_order = ['banana', 'apple', 'date']

# Създаване на Series от речник със зададен ред на индекса
series_with_dict_index_order = pd.Series(data_dict, index=index_order)

print("\nSeries от речник със зададен ред на индекса:")
print(series_with_dict_index_order)
```

```
Изход:

banana    2.0
apple     1.0
```

```
date      NaN
dtype: float64
```

В този случай, редът на елементите в Series съответства на реда в `index_order`. Ключът 'date' не съществува в `data_dict`, затова съответната стойност е `NaN`, а типът на данните е преобразуван във `float64`, за да може да представи `NaN`.

Тези примери показват основните начини за създаване на `Series` обекти в Pandas от различни Python структури. Разбирането на тези методи е важна стъпка към ефективната работа с Pandas.

6. Създаване на DataFrame

`DataFrame` е двумерна, таблична структура от данни, която може да бъде създадена от различни източници. Ето някои от най-често срещаните начини:

a) Създаване на DataFrame от речник от Series или списъци:

Ако имаме речник, където ключовете представляват имената на колоните, а стойностите са `Series` обекти или списъци (които ще бъдат преобразувани в `Series`), можем да създадем `DataFrame`. Всички `Series` или списъци трябва да имат еднаква дължина (или Pandas ще запълни липсващите стойности с `NaN`).

```
import pandas as pd

# Създаване на речник от списъци
data_dict_lists = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 28],
    'City': ['New York', 'London', 'Paris']
}
df_from_dict_lists = pd.DataFrame(data_dict_lists)
print("DataFrame от речник от списъци:\n", df_from_dict_lists)

# Създаване на речник от Series
data_dict_series = {
    'Product': pd.Series(['A', 'B', 'C']),
    'Price': pd.Series([10.5, 20.3, 5.0]),
```

```
    'Quantity': pd.Series([5, 10, 2])
}
df_from_dict_series = pd.DataFrame(data_dict_series)
print("\nDataFrame от речник от Series:\n", df_from_dict_series)
Изход:
DataFrame от речник от списъци:
      Name  Age      City
0    Alice   25  New York
1      Bob   30    London
2  Charlie   28     Paris

DataFrame от речник от Series:
   Product  Price  Quantity
0        A   10.5         5
1        B   20.3        10
2        C    5.0         2
```

В тези примери, ключовете на речника стават имената на колоните в DataFrame, а стойностите (списъците или Series) стават съдържанието на тези колони. Pandas автоматично присвоява целочислен индекс по подразбиране.

Можем да зададем персонализиран индекс, като подадем списък на аргумента `index`:

```
import pandas as pd

data_dict_lists_indexed = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 28],
    'City': ['New York', 'London', 'Paris']
}
index_labels = ['person1', 'person2', 'person3']
df_from_dict_lists_indexed = pd.DataFrame(data_dict_lists_indexed, index=index_labels)
print("\nDataFrame от речник от списъци със зададен индекс:\n", df_from_dict_lists_indexed)
```

```
Изход:


DataFrame от речник от списъци със зададен индекс:
          Name  Age      City
person1  Alice   25  New York
person2    Bob   30    London
person3  Charlie   28     Paris
```

б) Създаване на DataFrame от списък от речници:

Ако имаме списък, където всеки елемент е речник, представящ един ред от данните, можем да създадем `DataFrame`. Ключовете на речниците ще станат имената на колоните. Ако някои речници нямат определени ключове, Pandas ще запълни съответните стойности с `NaN`.

```
import pandas as pd


# Създаване на списък от речници
data_list_of_dicts = [
    {'Name': 'Alice', 'Age': 25, 'City': 'New York'},
    {'Name': 'Bob', 'Age': 30, 'City': 'London'},
    {'Name': 'Charlie', 'City': 'Paris', 'Occupation': 'Engineer'}
]
df_from_list_of_dicts = pd.DataFrame(data_list_of_dicts)
print("\nDataFrame от списък от речници:\n", df_from_list_of_dicts)
```

```
Изход:
DataFrame от списък от речници:
      Name  Age      City Occupation
0    Alice   25  New York        NaN
1      Bob   30    London        NaN
```

```
2  Charlie  NaN     Paris     Engineer
```

Забележете, че за третия речник липсва ключ 'Age', а за първите два липсва 'Occupation', което води до `NaN` стойности в съответните клетки.

Можем да контролираме реда на колоните, като подадем списък с желаните имена на колони на аргумента `columns`:

```
import pandas as pd


data_list_of_dicts_cols = [
    {'Name': 'Alice', 'Age': 25, 'City': 'New York'},
    {'Name': 'Bob', 'Age': 30, 'City': 'London'},
    {'Name': 'Charlie', 'City': 'Paris', 'Occupation': 'Engineer'}
]
df_from_list_of_dicts_cols = pd.DataFrame(data_list_of_dicts_cols, columns=['Name', 'Age', 'City'])
print("\nDataFrame от списък от речници със зададени колони:\n", df_from_list_of_dicts_cols)
```

```
Изход:


DataFrame от списък от речници със зададени колони:
      Name  Age      City
0    Alice   25  New York
1      Bob   30    London
2  Charlie  NaN     Paris
```

В този случай, колоната 'Occupation' не е включена, а за липсващата стойност 'Age' за Charlie се появява `NaN`.

в) Създаване на DataFrame от NumPy array:

Можем да създадем `DataFrame` и от двумерен NumPy array. В този случай трябва да предоставим имената на колоните и евентуално индекса.

```
import pandas as pd
import numpy as np

# Създаване на NumPy array
numpy_array_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Създаване на DataFrame от NumPy array с имена на колони
column_names = ['col1', 'col2', 'col3']
df_from_array = pd.DataFrame(numpy_array_2d, columns=column_names)
print("\nDataFrame от NumPy array с имена на колони:\n", df_from_array)

# Създаване на DataFrame от NumPy array с имена на колони и индекс
index_labels_array = ['row1', 'row2', 'row3']
df_from_array_indexed = pd.DataFrame(numpy_array_2d, columns=column_names, index=index_labels_array)
print("\nDataFrame от NumPy array с имена на колони и индекс:\n", df_from_array_indexed)
```

```
Изход:


DataFrame от NumPy array с имена на колони:
    col1  col2  col3
0     1     2     3
1     4     5     6
2     7     8     9


DataFrame от NumPy array с имена на колони и индекс:
      col1  col2  col3
row1     1     2     3
row2     4     5     6
row3     7     8     9
```

Ако не предоставим имена на колони или индекс, Pandas ще използва целочислен индекс по подразбиране.

г) Създаване на DataFrame от други формати:

Pandas може да чете данни от много други формати, като CSV файлове, Excel файлове, SQL бази данни, JSON файлове, HTML таблици и други. Функции като `pd.read_csv()`, `pd.read_excel()`, `pd.read_sql()`, `pd.read_json()`, `pd.read_html()` се използват за тази цел и връщат `DataFrame` обекти. Тези методи ще бъдат разгледани по-късно.

Тези примери показват основните начини за създаване на `DataFrame` обекти в Pandas от различни типове данни. Изборът на метод зависи от формата, в който са налични вашите данни.

7. Разглеждане на основни атрибути на Series и DataFrame

Разбира се, след като вече знаем как да създаваме `Series` и `DataFrame` обекти, е важно да се запознаем с техните основни атрибути, които ни дават важна информация за структурата и данните, които съдържат. Тези атрибути са достъпни чрез точкова нотация (`.attribute_name`).

Ще разгледаме следните основни атрибути, които са общи или специфични за `Series` и `DataFrame`:

- `.index`
- `.values`
- `.dtype` (за Series) / `.dtypes` (за DataFrame)
- `.shape`
- `.size`
- `.name` (само за Series)

а) Атрибут `.index`:

- **Series:** Атрибутът `.index` връща индексния обект на `Series`. Индексът е последователност от етикети, които идентифицират всеки елемент в `Series`.
- **DataFrame:** Атрибутът `.index` връща индексния обект, който съдържа етикетите на редовете в `DataFrame`.

```
import pandas as pd
import numpy as np


# Създаване на Series
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print("Series:\n", s)
print("\nSeries index:", s.index)
```

```
# Създаване на DataFrame
data = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data, index=['row1', 'row2'])
print("\nDataFrame:\n", df)
print("\nDataFrame index:", df.index)
```

```
Изход:
Series:
a    10
b    20
c    30
dtype: int64

Series index: Index(['a', 'b', 'c'], dtype='object')

DataFrame:
       col1  col2
row1     1     3
row2     2     4

DataFrame index: Index(['row1', 'row2'], dtype='object')
```

Както виждате, .index връща обект от тип Index, който съдържа етикетите. Типът на данните в индекса може да бъде различен (например 'object' за низове, 'int64' за цели числа, 'datetime64[ns]' за дати).

б) Атрибут `.values`:

- **Series:** Атрибутът `.values` връща NumPy array, съдържащ данните в `Series`.
- **DataFrame:** Атрибутът `.values` връща NumPy array, съдържащ всички данни в `DataFrame`.

```
import pandas as pd
import numpy as np

# Използване на създадените по-горе Series и DataFrame
print("\nSeries values:\n", s.values)
```

```
print("\nType of Series values:", type(s.values))


print("\nDataFrame values:\n", df.values)
print("\nType of DataFrame values:", type(df.values))
```

```
Изход:


Series values:
 [10 20 30]


Type of Series values: <class 'numpy.ndarray'>


DataFrame values:
 [[1 3]
 [2 4]]


Type of DataFrame values: <class 'numpy.ndarray'>
```

Атрибутът .values ни дава достъп до същинските данни, съхранени в Series или DataFrame, като NumPy array, което позволява използването на NumPy функционалности върху тези данни.

в) Атрибут `.dtype` (за Series) / `.dtypes` (за DataFrame):

- **Series:** Атрибутът `.dtype` връща типа на данните (data type) на елементите в `Series`. Всички елементи в един `Series` обикновено са от един и същ тип.
- **DataFrame:** Атрибутът `.dtypes` връща `Series`, съдържащ типа на данните за всяка колона в `DataFrame`. Тъй като всяка колона може да съдържа данни от различен тип, `.dtypes` е `Series`, където индексът са имената на колоните, а стойностите са съответните типове данни.

```
import pandas as pd


# Създаване на Series с различен тип данни
s_mixed = pd.Series([1, 'two', 3.0])
print("\nSeries с различни типове данни:\n", s_mixed)
```

```
print("Series dtype:", s_mixed.dtype)


# Използване на създадения по-горе DataFrame
print("\nDataFrame dtypes:\n", df.dtypes)
```

```
Изход:
Series с различни типове данни:
0       1
1     two
2     3.0
dtype: object

Series dtype: object


DataFrame dtypes:
col1    int64
col2    int64
dtype: object
```

Когато `Series` съдържа елементи от различни типове, Pandas обикновено избира най-общия тип, който може да ги побере (в случая 'object' за смесени числови и низови данни). За `DataFrame`, `.dtypes` показва типа на данните за всяка колона ('int64' в нашия пример).

г) Атрибут `.shape`:

- **Series:** Атрибутът `.shape` връща кортеж, съдържащ броя на елементите в `Series`. Тъй като `Series` е едномерна структура, кортежът ще съдържа само едно число.
- **DataFrame:** Атрибутът `.shape` връща кортеж, съдържащ броя на редовете и броя на колоните в `DataFrame` (в реда (rows, columns)).

```
import pandas as pd


# Използване на създадените по-горе Series и DataFrame
print("\nSeries shape:", s.shape)
print("DataFrame shape:", df.shape)
```

```
Изход:
Series shape: (3,)
DataFrame shape: (2, 2)
```

`s.shape` показва, че `Series` `s` има 3 елемента. `df.shape` показва, че `DataFrame` `df` има 2 реда и 2 колони.

д) Атрибут `.size`:

- **Series:** Атрибутът `.size` връща общия брой на елементите в `Series`.
- **DataFrame:** Атрибутът `.size` връща общия брой на елементите в `DataFrame` (броя на редовете, умножен по броя на колоните).

```
import pandas as pd


# Използване на създадените по-горе Series и DataFrame
print("\nSeries size:", s.size)
print("DataFrame size:", df.size)
```

```
Изход:
Series size: 3
DataFrame size: 4
```

`s.size` е 3, тъй като `Series` `s` има 3 елемента. `df.size` е 4, тъй като `DataFrame` `df` има 2 реда * 2 колони = 4 елемента.

е) Атрибут `.name` (само за Series):

- **Series:** Атрибутът `.name` съдържа името на `Series`. Това е полезно, когато `Series` представлява колона в `DataFrame`. Името може да бъде зададено при създаването на `Series` или по-късно.
- **DataFrame:** `DataFrame` няма атрибут `.name` по същия начин, тъй като представлява двумерна структура с множество колони. Имената на колоните се съхраняват в атрибута `.columns` (който ще разгледаме по-късно).

```
import pandas as pd


# Създаване на Series с име
```

```
named_series = pd.Series([100, 200, 300], name='Values')

print("\nNamed Series:\n", named_series)

print("Named Series name:", named_series.name)
```

```
Изход:

Named Series:

 0    100

1    200

2    300

Name: Values, dtype: int64


Named Series name: Values
```

В този пример, `Series` `named_series` има зададено име 'Values', което се появява в изхода при печатане на `Series` и може да бъде достъпено чрез `.name`.

Разбирането и използването на тези основни атрибути е важна стъпка при работата с `Series` и `DataFrame` обекти в Pandas, тъй като те ни помагат да получим бърз преглед на данните и тяхната структура.

8. Първи стъпки: създаване и разглеждане на прости DataFrame-и

Разбира се, нека направим първите си стъпки в създаването и разглеждането на прости `DataFrame`-и. Ще използваме знанията си от предходните секции, за да създадем няколко основни `DataFrame`-а и да разгледаме техните атрибути и съдържание.

В тази секция ще създадем няколко прости `DataFrame`-а, използвайки различни методи, които вече разгледахме, и ще използваме атрибутите, с които се запознахме, за да ги инспектираме.

Пример 1: Създаване на DataFrame от речник от списъци

```
import pandas as pd


# Създаване на данни като речник от списъци

data = {

    'Име': ['Алиса', 'Борис', 'Ваня'],
```

```
    'Възраст': [25, 30, 27],
    'Град': ['София', 'Пловдив', 'Варна']
}

# Създаване на DataFrame
df1 = pd.DataFrame(data)

# Разглеждане на DataFrame
print("DataFrame 1:\n", df1)

# Разглеждане на основни атрибути
print("\nИндекс на DataFrame 1:", df1.index)
print("Стойности на DataFrame 1 (като NumPy array):\n", df1.values)
print("Типове данни на колоните в DataFrame 1:\n", df1.dtypes)
print("Форма на DataFrame 1 (ред, колони):", df1.shape)
print("Размер на DataFrame 1 (брой елементи):", df1.size)
print("Имена на колоните в DataFrame 1:", df1.columns)
```

```
Изход:


DataFrame 1:
      Име  Възраст    Град
0  Алиса       25   София
1  Борис       30  Пловдив
2   Ваня       27    Варна

Индекс на DataFrame 1: RangeIndex(start=0, stop=3, step=1)
Стойности на DataFrame 1 (като NumPy array):
 [['Алиса' 25 'София']
 ['Борис' 30 'Пловдив']
 ['Ваня' 27 'Варна']]
```

```
Типове данни на колоните в DataFrame 1:
 Име         object
Възраст      int64
Град        object
dtype: object
Форма на DataFrame 1 (ред, колони): (3, 3)
Размер на DataFrame 1 (брой елементи): 9
Имена на колоните в DataFrame 1: Index(['Име', 'Възраст', 'Град'],
dtype='object')
```

В този пример създадохме `DataFrame` с три колони ('Име', 'Възраст', 'Град') и три реда. Виждаме автоматично създадения целочислен индекс, стойностите като двумерен NumPy array, типовете данни на всяка колона (object за низове и int64 за цели числа), формата (3 реда, 3 колони) и общия брой на елементи (3 * 3 = 9). Атрибутът `.columns` връща обект `Index`, съдържащ имената на колоните.

Пример 2: Създаване на DataFrame от списък от речници със зададен индекс

```
import pandas as pd


# Създаване на данни като списък от речници
data_list = [
    {'Име': 'Елена', 'Оценка': 5.5},
    {'Име': 'Георги', 'Оценка': 6.0},
    {'Име': 'Иван', 'Оценка': 4.5}
]


# Създаване на DataFrame със зададен индекс
df2 = pd.DataFrame(data_list, index=['студент1', 'студент2', 'студент3'])


# Разглеждане на DataFrame
print("\nDataFrame 2:\n", df2)


# Разглеждане на основни атрибути
print("\nИндекс на DataFrame 2:", df2.index)
```

```
print("Типове данни на колоните в DataFrame 2:\n", df2.dtypes)
print("Форма на DataFrame 2:", df2.shape)
```

```
Изход:


DataFrame 2:
            Име  Оценка
студент1  Елена     5.5
студент2 Георги     6.0
студент3   Иван     4.5


Индекс на DataFrame 2: Index(['студент1', 'студент2', 'студент3'],
dtype='object')
Типове данни на колоните в DataFrame 2:
 Име         object
Оценка      float64
dtype: object
Форма на DataFrame 2: (3, 2)
```

Тук създадохме `DataFrame` от списък от речници и изрично зададохме индекс с низови етикети. Типът на данните за колоната 'Оценка' е `float64`, тъй като стойностите са числа с плаваща запетая. Формата е (3 реда, 2 колони).

Пример 3: Създаване на DataFrame от NumPy array с имена на колони

```
import pandas as pd
import numpy as np


# Създаване на NumPy array
data_array = np.array([[10, 20], [30, 40], [50, 60]])


# Задаване на имена на колони
columns = ['Първа', 'Втора']
```

```
# Създаване на DataFrame
df3 = pd.DataFrame(data_array, columns=columns)

# Разглеждане на DataFrame
print("\nDataFrame 3:\n", df3)

# Разглеждане на основни атрибути
print("\nИмена на колоните в DataFrame 3:", df3.columns)
print("Тип на данните в DataFrame 3:\n", df3.dtypes)
```

```
Изход:
DataFrame 3:
   Първа  Втора
0     10     20
1     30     40
2     50     60

Имена на колоните в DataFrame 3: Index(['Първа', 'Втора'], dtype='object')
Тип на данните в DataFrame 3:
Първа    int64
Втора    int64
dtype: object
```

В този случай създадохме `DataFrame` от двумерен NumPy array и предоставихме списък с имена на колони. Pandas автоматично създаде целочислен индекс. Типът на данните в колоните е `int64`.

Тези прости примери илюстрират как да създавате `DataFrame`-и от различни видове данни и как да използвате основните атрибути, за да получите информация за тяхната структура и съдържание. В следващите глави ще разгледаме по-подробно как да работим с данните в `DataFrame`-ите, включително селектиране, филтриране, манипулиране и анализ.

Казус 1:

Създайте Pandas Series обект, съдържащ следните данни: `[15, 22, 38, 45]`, с индекс `['A', 'B', 'C', 'D']`. След това изведете индекса и стойностите на Series обекта.

Решение на Казус 1:

```
import pandas as pd

data = [15, 22, 38, 45]
index_labels = ['A', 'B', 'C', 'D']
series1 = pd.Series(data, index=index_labels)

print("Създаден Series:\n", series1)
print("\nИндекс на Series:", series1.index)
print("Стойности на Series (като NumPy array):", series1.values)
```

Казус 2:

Създайте Pandas DataFrame обект от следния речник:

```
data = {'Име': ['Петър', 'Мария', 'Иван'],
        'Години': [28, 24, 32],
        'Град': ['София', 'Варна', 'Пловдив']}
```

Изведете формата и типа на данните за всяка колона на създадения DataFrame.

Решение на Казус 2:

```
import pandas as pd

data = {'Име': ['Петър', 'Мария', 'Иван'],
        'Години': [28, 24, 32],
        'Град': ['София', 'Варна', 'Пловдив']}
df2 = pd.DataFrame(data)

print("Създаден DataFrame:\n", df2)
print("\nФорма на DataFrame:", df2.shape)
print("Типове данни на колоните:\n", df2.dtypes)
```

Казус 3:

Имате следния NumPy array:

```
import numpy as np
data_array = np.array([[1, 2.5, 'A'], [4, 5.0, 'B'], [7, 8.5, 'C']])
```

Създайте Pandas DataFrame от този NumPy array с имена на колоните съответно: 'Колона 1', 'Колона 2', 'Колона 3'. След това изведете размера на DataFrame-а.

Решение на Казус 3:

```
import pandas as pd
import numpy as np

data_array = np.array([[1, 2.5, 'A'], [4, 5.0, 'B'], [7, 8.5, 'C']])
columns = ['Колона 1', 'Колона 2', 'Колона 3']
df3 = pd.DataFrame(data_array, columns=columns)

print("Създаден DataFrame:\n", df3)
print("\nРазмер на DataFrame (брой елементи):", df3.size)
```

Казус 4:

Създайте Pandas Series обект от следния речник:

```
data_dict = {'Ябълка': 10, 'Банан': 15, 'Портокал': 12}
```

Изведете името на индекса и името на Series обекта (ако зададете такова).

Решение на Казус 4:

```
import pandas as pd

data_dict = {'Ябълка': 10, 'Банан': 15, 'Портокал': 12}
```

```
series4 = pd.Series(data_dict, name='Цени на плодове')


print("Създаден Series:\n", series4)
print("\nИндекс на Series:", series4.index)
print("Име на Series:", series4.name)
```

Казус 5:

Обяснете с прости думи разликата между Pandas Series и Pandas DataFrame.

Решение на Казус 5:

Pandas **Series** е като едномерен списък или колона от данни, но с етикети (индекс) за всеки елемент. Представете си го като единична колона в Excel.

Pandas **DataFrame** е като двумерна таблица или електронна таблица с редове и колони, където всяка колона може да има различен тип данни. Може да се разглежда като колекция от Series обекти, които споделят общ индекс (етикетите на редовете).

Въпроси

1. Каква е основната цел на библиотеката Pandas в Python? Опишете накратко нейната философия.
2. Избройте поне три основни предимства на използването на Pandas за анализ на данни.
3. Какви са двете основни структури от данни, които Pandas предоставя? Опишете ги накратко и посочете основната разлика между тях.
4. Каква е конвенцията при импортиране на библиотеката Pandas в Python код? Защо се използва тази конвенция?
5. Обяснете как можете да създадете Pandas Series обект от Python списък, NumPy array и Python речник. Как се определя индексът в тези случаи по подразбиране и как можете да го персонализирате?
6. Опишете поне три начина за създаване на Pandas DataFrame обект. Дайте кратък пример за всеки метод.
7. Каква информация можете да получите от следните атрибути на Series и DataFrame: `.index`, `.values`, `.dtype` (за Series) / `.dtypes` (за DataFrame), `.shape`, `.size`, `.name` (за Series)?

Задачи

1. **Създаване на Series:** Създайте Pandas Series, съдържащ месеците от годината (като низове) като стойности, а като индекс използвайте съкратените им имена (Jan, Feb, Mar, ... Dec).
2. **Създаване на DataFrame от списък от речници:** Създайте DataFrame от следния списък от речници:

```
data = [
    {'Продукт': 'Лаптоп', 'Цена': 1200.50, 'Наличност': 10},
    {'Продукт': 'Мишка', 'Цена': 25.99, 'Наличност': 50},
    {'Продукт': 'Клавиатура', 'Цена': 75.00, 'Наличност': 25}
]
```

   Изведете имената на колоните и формата на DataFrame-а.

3. **Създаване на DataFrame от NumPy array:** Създайте DataFrame от случаен NumPy array с размери 5x3 (използвайте `np.random.rand(5, 3)`). Задайте имената на колоните като 'Колона A', 'Колона B', 'Колона C'. Изведете първите два реда от DataFrame-а (подсказка: използвайте slicing).
4. **Разглеждане на атрибути:** Създайте прост DataFrame с няколко колони от различен тип данни (например int, float, string). Изведете типа на данните за всяка колона, както и общия брой на елементи в DataFrame-а.
5. **Персонализиран индекс:** Създайте DataFrame с данни за трима студенти (име, факултетен номер, среден успех). Използвайте факултетните номера като индекс на DataFrame-а. Изведете индекса на DataFrame-а.