Глава VIII. Комбиниране и Сливане на DataFrame-и

В тази глава "Комбиниране и Сливане на DataFrame-и" ще разгледаме основните техники за обединяване на данни от няколко `DataFrame`-а в един. Това е ключова операция при работа с данни, които често са разпръснати в различни източници или са разделени по логически или времеви признаци.

Ето кратко резюме на темите, които ще бъдат разгледани:

1. **Конкатенация (`pd.concat()`):** Ще научим как да комбинираме `DataFrame`-и и `Series` обекти по вертикала (по редове) или хоризонтала (по колони). Ще разгледаме как да управляваме индексите по време на конкатенация (дали да ги игнорираме или да добавим йерархични индекси) и как да контролираме кои колони/редове да бъдат включени при непълно съвпадение (вътрешно или външно съединяване).
    - Комбиниране по редове (`axis=0`)
    - Комбиниране по колони (`axis=1`)
    - Работа с индекси (`ignore_index`, `keys`)
    - Типове съединяване (`join='inner'`, `join='outer'`)
2. **Сливане в стил бази данни (`pd.merge()`):** Ще се фокусираме върху сливането на `DataFrame`-и, подобно на операциите `JOIN` в SQL. Ще разгледаме различни начини за указване на ключови колони за сливане (използвайки `on`, `left_on`, `right_on`) и как да сливаме по индекс. Ще обсъдим различните типове сливания (`inner`, `outer`, `left`, `right`, `cross`) и как да управляваме дублиращи се имена на колони (чрез суфикси) и да добавяме индикаторна колона за произхода на редовете. Ще засегнем и валидирането на връзките между ключовите колони.
    - Ключове за сливане (`on`, `left_on`, `right_on`)
    - Сливане по индекс (`left_index=True`, `right_index=True`)
    - Типове сливания (`how='inner'`, `'outer'`, `'left'`, `'right'`, `'cross'`)
    - Индикатори и управление на суфикси (`indicator=True`, `suffixes`)
    - Валидиране на връзките (`validate`)
3. **Съединяване по индекс (`.join()`):** Ще разгледаме `.join()` като по-удобна и опростена функция за сливане на `DataFrame`-и, когато основният ключ за сливане е индексът. По същество, това е специализирана версия на `pd.merge()` с акцент върху индексите.
4. **Сравняване на `DataFrame`-и (`.compare()`):** Накрая, ще разгледаме метода `.compare()`, който позволява да се идентифицират разликите между два `DataFrame`-а със същата структура.

Чрез тези теми ще придобиете задълбочени познания за различните начини за комбиниране и сливане на данни в Pandas, което е съществено умение за подготовка и анализ на сложни набори от данни.

I. Конкатенация (pd.concat())

Конкатенацията в Pandas, осъществявана основно чрез функцията `pd.concat()`, представлява процес на **свързване или слепване на два или повече Pandas обекта (като `Series` или `DataFrame`) по определена ос**.

Представете си, че имате няколко отделни таблици с данни, които съдържат сходна информация, но за различни периоди от време, различни региони или различни аспекти на една и съща същност. Конкатенацията ви позволява да **комбинирате тези отделни набори от данни в единен, по-голям набор**, който може да бъде по-лесен за анализ и обработка.

С какво спомага конкатенацията:

- **Обединяване на разпръснати данни:** Когато данните са разделени в множество файлове или `DataFrame`-и, конкатенацията е основен инструмент за събирането им на едно място.
- **Добавяне на нови записи:** Ако имате нови данни, които искате да добавите към съществуващ `DataFrame` (например, нови наблюдения или записи), конкатенацията по редове е начинът да го направите.
- **Добавяне на нови характеристики:** Ако имате допълнителна информация (нови колони) за същите записи, но в отделен `DataFrame`, конкатенацията по колони може да ги обедини.
- **Създаване на по-големи набори от данни за анализ:** Комбинирането на данни може да разкрие по-широки тенденции и модели, които не биха били видими в по-малки, разпокъсани набори.
- **Улесняване на последваща обработка:** След като данните са обединени, е по-лесно да се прилагат общи операции за почистване, трансформация и анализ.

1. Комбиниране по редове (axis=0)

Комбинирането по редове (axis=0) с `pd.concat()` е процес на **вертикално слепване на Pandas обекти**. Представете си, че имате няколко `DataFrame`-а или `Series` обекта, които имат **едни и същи или сходни колони**, но съдържат данни за различни периоди, различни групи или различни части от цялостна информация. Когато комбинирате по редове, вие ги **нареждате един под друг**, увеличавайки броя на редовете в резултата.

а) Как работи:

- Функцията `pd.concat()` приема списък от Pandas обекти (може да бъдат `DataFrame`-и или `Series` обекти).
- Когато `axis=0` (което е и стойността по подразбиране), `concat()` се опитва да подравни колоните на обектите, които се комбинират.
- Ако колоните на различните обекти са едни и същи, те се подреждат една до друга в резултата.
- Ако някои обекти имат колони, които не съществуват в други, по подразбиране за тези липсващи колони ще бъдат попълнени `NaN` стойности в резултата.
- Индексът на резултата ще бъде комбинация от индексите на входните обекти. Ако има дублиращи се стойности на индекса между входните обекти, те ще бъдат запазени и в резултата.

б) Сценарии от реалния живот:

- **Комбиниране на данни от различни файлове:** Да кажем, че имате месечни отчети за продажби в отделни CSV файлове. Можете да прочетете всеки файл в `DataFrame` и след това да ги конкатенирате по редове, за да получите единен `DataFrame` с всички продажби за годината.
- **Добавяне на нови данни към съществуващ `DataFrame`:** Ако събирате данни на партиди или имате нови записи, които искате да добавите към вече съществуващ `DataFrame`, конкатенацията по редове е правилният подход.
- **Обединяване на данни от различни източници:** Може да имате данни за клиенти от една база данни и допълнителна информация за техните поръчки от друга. Ако тези данни могат да бъдат свързани по някакъв начин (например, ако имат общи колони и представляват различни записи), можете да ги конкатенирате (въпреки че в този случай сливането може да е по-подходящо, но конкатенацията пак би свършила работа, ако структурата е сходна).

в) Пример:

```
import pandas as pd

# Създаваме два DataFrame-а с еднакви колони
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print("DataFrame 1:\n", df1)

df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
print("\nDataFrame 2:\n", df2)

# Конкатенираме ги по редове (axis=0)
df_combined_rows = pd.concat([df1, df2])
print("\nDataFrame след конкатенация по редове:\n", df_combined_rows)

# Създаваме два DataFrame-а с различни колони
df3 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df4 = pd.DataFrame({'A': [5, 6], 'C': [7, 8]})
print("\nDataFrame 3:\n", df3)
print("\nDataFrame 4:\n", df4)

# Конкатенираме ги по редове (липсващите колони се попълват с NaN)
df_combined_diff_cols = pd.concat([df3, df4])
print("\nDataFrame след конкатенация по редове (различни колони):\n",
df_combined_diff_cols)
```

В първия пример, тъй като колоните са еднакви, данните просто се добавят една под друга. Във втория пример, където колоните са различни, резултатът съдържа всички колони от двата DataFrame-а, а за липсващите стойности се попълва NaN.

2. Комбиниране по колони (axis=1)

Комбинирането по колони (axis=1) с `pd.concat()` е процес на **хоризонтално слепване на Pandas обекти**. Представете си, че имате няколко `DataFrame`-а или `Series` обекта, които съдържат **данни за едни и същи редове, но с различни колони (различни характеристики или променливи)**. Когато комбинирате по колони, вие ги **нареждате един до друг**, увеличавайки броя на колоните в резултата.

а) Как работи:

- Подобно на комбинирането по редове, `pd.concat()` приема списък от Pandas обекти.
- Когато `axis=1`, `concat()` се опитва да подравни редовете на обектите, които се комбинират, **базирайки се на техния индекс**.
- Ако индексите на различните обекти съвпадат, съответните редове се слепват хоризонтално.
- Ако някои обекти имат индекси, които не съществуват в други, по подразбиране за тези липсващи редове ще бъдат попълнени `NaN` стойности за колоните от другия обект в резултата (това поведение може да бъде контролирано с параметъра `join`).
- Имената на колоните в резултата ще бъдат комбинация от имената на колоните на входните обекти. Ако има дублиращи се имена на колони, те ще бъдат запазени и ще трябва да бъдат обработени допълнително (например, чрез преименуване).

б) Сценарии от реалния живот:

- **Добавяне на допълнителни характеристики:** Да кажем, че имате `DataFrame` с основни данни за клиенти (име, възраст) и друг `DataFrame` с допълнителна информация за техните предпочитания (любим продукт, начин на плащане), като и двата `DataFrame`-а имат общ индекс (например, customer_id). Можете да ги конкатенирате по колони, за да получите един `DataFrame` с цялата информация за всеки клиент.
- **Комбиниране на резултати от различни изчисления:** Може да сте извършили различни анализи върху един и същ набор от данни и да искате да комбинирате резултатите (например, различни статистически метрики) в един `DataFrame`, където оригиналните данни служат като индекс.
- **Създаване на широк формат на данни:** В някои случаи може да имате данни в "дълъг" формат (където една променлива е представена в няколко реда) и да искате да ги преобразувате в "широк" формат (където всяка променлива става отделна колона). Конкатенацията по колони може да бъде част от този процес, особено след групиране и агрегиране.

в) Пример:

```
import pandas as pd

# Създаваме два DataFrame-а с еднакъв индекс
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]}, index=['row1', 'row2'])
print("DataFrame 1:\n", df1)
```

```
df2 = pd.DataFrame({'C': [5, 6], 'D': [7, 8]}, index=['row1', 'row2'])
print("\nDataFrame 2:\n", df2)

# Конкатенираме ги по колони (axis=1)
df_combined_cols = pd.concat([df1, df2], axis=1)
print("\nDataFrame след конкатенация по колони:\n", df_combined_cols)

# Създаваме два DataFrame-а с различен индекс
df3 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]}, index=['row1', 'row2'])
df4 = pd.DataFrame({'C': [5, 6], 'D': [7, 8]}, index=['row1', 'row3'])
print("\nDataFrame 3:\n", df3)
print("\nDataFrame 4:\n", df4)

# Конкатенираме ги по колони (липсващите редове се попълват с NaN)
df_combined_diff_index = pd.concat([df3, df4], axis=1)
print("\nDataFrame след конкатенация по колони (различен индекс):\n",
df_combined_diff_index)
```

В първия пример, тъй като индексите съвпадат, колоните на двата DataFrame-а просто се добавят една до друга. Във втория пример, където индексите не съвпадат, резултатът съдържа всички индекси от двата DataFrame-а, а за липсващите стойности се попълва NaN.

3. Работа с индекси (ignore_index, keys)

a) `ignore_index=True`:

Този параметър се използва, когато искате да **игнорирате оригиналните индекси** на `DataFrame`-ите или `Series`-ите, които се конкатенират, и да създадете **нов, последователен числов индекс** за резултата. Това е особено полезно, когато комбинирате данни, които може да имат дублиращи се индекси, и тези дубликати не са желани в крайния резултат.

- Сценарии от реалния живот:

Представете си, че имате месечни отчети за продажби в няколко `DataFrame`-а, всеки с числов индекс, започващ от 0. Когато ги конкатенирате по редове, бихте получили много дублиращи се стойности на индекса. Ако искате да получите единен `DataFrame` с уникален числов индекс за всички продажби, тогава `ignore_index=True` е подходящ.

- Пример:

```
import pandas as pd

# Създаваме два DataFrame-а с еднакви индекси
```

```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print("DataFrame 1:\n", df1)

df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
print("\nDataFrame 2:\n", df2)

# Конкатенираме ги по редове, игнорирайки оригиналните индекси
df_ignored_index = pd.concat([df1, df2], ignore_index=True)
print("\nDataFrame след конкатенация с ignore_index=True:\n",
df_ignored_index)
```

Забележете, че въпреки че df1 и df2 имаха индекси 0 и 1, резултатът df_ignored_index има нов, последователен индекс 0, 1, 2, 3.

б) keys:

Параметърът `keys` позволява да се създаде **йерархичен (MultiIndex) индекс** за резултата от конкатенацията. Когато подадете списък от стойности на `keys`, всяка стойност от списъка ще бъде свързана с оригиналния индекс на съответния обект в списъка, който се подава на `pd.concat()`. Това е полезно, когато искате да запазите информация за произхода на данните след конкатенация.

- Сценарии от реалния живот:

Да кажем, че комбинирате данни за продажби от различни региони, които се намират в отделни `DataFrame`-и. Можете да използвате имената на регионите като стойности в параметъра `keys`, за да създадете MultiIndex, където първото ниво ще бъде името на региона, а второто ниво ще бъде оригиналният индекс в рамките на този регион.

- Пример:

```
import pandas as pd

# Създаваме два DataFrame-а с еднакви колони и индекси
df_north = pd.DataFrame({'Продукт': ['А', 'Б'], 'Продажби': [100, 150]},
index=[0, 1])
df_south = pd.DataFrame({'Продукт': ['А', 'В'], 'Продажби': [120, 90]},
index=[0, 1])
print("DataFrame Север:\n", df_north)
print("\nDataFrame Юг:\n", df_south)

# Конкатенираме ги по редове, използвайки 'keys' за създаване на
MultiIndex
```

```
df_sales_by_region = pd.concat([df_north, df_south], keys=['Север',
'Юг'])
print("\nDataFrame след конкатенация с keys:\n", df_sales_by_region)

# Можете да достъпвате данните по ниво на индекса
print("\nПродажби за регион Север:\n", df_sales_by_region.loc['Север'])
print("\nПродажба на продукт 'Б' в регион Север:\n",
df_sales_by_region.loc[('Север', 1)])
```

В този пример, параметърът keys=['Север', 'Юг'] създаде MultiIndex, където първото ниво е 'Север' за данните от df_north и 'Юг' за данните от df_south. Оригиналните индекси (0 и 1) станаха второто ниво на MultiIndex.

Можете да комбинирате използването на keys с ignore_index=True. В този случай, ще получите MultiIndex, но оригиналните вътрешни индекси ще бъдат заменени с нов числов индекс в рамките на всяка група, дефинирана от keys.

Разбирането на `ignore_index` и `keys` е важно за ефективното комбиниране на данни и за запазване на важна информация за произхода на данните, както и за избягване на проблеми с дублиращи се индекси.

4. Типове съединяване (join='inner', join='outer')

Параметъра `join` в `pd.concat()`, е този който определя как да се обработят индексите (при `axis=1`) или колоните (при `axis=0`), когато те не съвпадат напълно между обектите, които се конкатенират. Възможните стойности за `join` са `'inner'` и `'outer'`.

а) `join='outer'` (стойност по подразбиране):

`'outer'` съединяването (външно съединяване) е поведението по подразбиране на `pd.concat()`. Когато използвате `'outer'`, резултатът ще съдържа **всички уникални индекси (при `axis=1`) или колони (при `axis=0`)** от всички конкатенирани обекти. За стойностите, където даден индекс (или колона) не съществува в един от входните обекти, ще бъдат попълнени **`NaN` (Not a Number)** стойности.

- Сценарии от реалния живот:

Представете си, че имате два `DataFrame`-а с информация за клиенти, но всеки съдържа малко по-различен набор от клиенти (различен индекс). Когато ги конкатенирате по редове (`axis=0`) с `join='outer'`, резултатът ще съдържа всички клиенти от двата `DataFrame`-а, а за колоните, които не съществуват в един от `DataFrame`-ите за определен клиент, ще има `NaN`.

Аналогично, ако конкатенирате по колони (`axis=1`) два `DataFrame`-а с несъвпадащи индекси, резултатът ще съдържа всички уникални индекси от двата, а за липсващите комбинации от индекс и колона ще има `NaN`.

- Пример (конкатенация по редове с `join='outer'`):

```
import pandas as pd

# Два DataFrame-а с различни колони
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print("DataFrame 1:\n", df1)

df2 = pd.DataFrame({'A': [5, 6], 'C': [7, 8]})
print("\nDataFrame 2:\n", df2)

# Конкатенираме по редове с join='outer' (по подразбиране)
df_outer_rows = pd.concat([df1, df2], axis=0, join='outer')
print("\nКонкатенация по редове (join='outer'):\n", df_outer_rows)
```

Забележете, че резултатът съдържа колони 'A', 'B' и 'C'. За редовете, които идват от df1, колоната 'C' е NaN, а за редовете от df2, колоната 'B' е NaN.

- Пример (конкатенация по колони с `join='outer'`):

```
import pandas as pd

# Два DataFrame-а с различни индекси
df1 = pd.DataFrame({'A': [1, 2]}, index=['row1', 'row2'])
print("DataFrame 1:\n", df1)

df2 = pd.DataFrame({'B': [3, 4]}, index=['row1', 'row3'])
print("\nDataFrame 2:\n", df2)

# Конкатенираме по колони с join='outer' (по подразбиране)
df_outer_cols = pd.concat([df1, df2], axis=1, join='outer')
print("\nКонкатенация по колони (join='outer'):\n", df_outer_cols)
```

Резултатът съдържа всички уникални индекси ('row1', 'row2', 'row3'). За липсващите комбинации от индекс и колона има NaN.

б) join='inner':

`'inner'` съединяването (вътрешно съединяване) връща резултат, който съдържа само **тези индекси (при `axis=1`) или колони (при `axis=0`)**, които са **общи за всички конкатенирани обекти**. Всички други индекси или колони, които съществуват само в някои от обектите, ще бъдат изключени от резултата.

- Сценарии от реалния живот:

Ако имате два `DataFrame`-а с информация за клиенти, но с потенциално различни набори от колони и/или редове, и искате да запазите само тези редове (при `axis=0`) или колони (при `axis=1`), които са налични и в двата `DataFrame`-а, тогава `'inner'` съединяването е подходящо.

- Пример (конкатенация по редове с `join='inner'`):

```
import pandas as pd

# Два DataFrame-а с частично съвпадащи колони
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print("DataFrame 1:\n", df1)

df2 = pd.DataFrame({'A': [5, 6], 'C': [7, 8]})
print("\nDataFrame 2:\n", df2)

# Конкатенираме по редове с join='inner'
df_inner_rows = pd.concat([df1, df2], axis=0, join='inner')
print("\nКонкатенация по редове (join='inner'):\n", df_inner_rows)
```

Резултатът съдържа само колоната 'A', която е обща и за двата `DataFrame`-а. Колоните 'B' и 'C' са изключени.

- Пример (конкатенация по колони с `join='inner'`):

```
import pandas as pd

# Два DataFrame-а с частично съвпадащи индекси
df1 = pd.DataFrame({'A': [1, 2]}, index=['row1', 'row2'])
print("DataFrame 1:\n", df1)

df2 = pd.DataFrame({'B': [3, 4]}, index=['row1', 'row3'])
print("\nDataFrame 2:\n", df2)
```

```
# Конкатенираме по колони с join='inner'
df_inner_cols = pd.concat([df1, df2], axis=1, join='inner')
print("\nКонкатенация по колони (join='inner'):\n", df_inner_cols)
```

Резултатът съдържа само реда с индекс 'row1', който е общ и за двата DataFrame-а. Редът 'row2' (от df1) и 'row3' (от df2) са изключени.

Разбирането на `'outer'` и `'inner'` съединяването е важно за контролиране на това как се обработват несъвпадащи етикети (индекси или колони) при конкатениране и за получаване на желания набор от данни в резултата.

Казус 1: Комбиниране на данни за трафика на уебсайт от различни устройства

Представете си, че събирате данни за трафика на вашия уебсайт и ги съхранявате отделно за настолни компютри и мобилни устройства в два различни CSV файла. И двата файла имат еднакви колони: 'Дата', 'Посетители', 'Преглеждания на страници'. Искате да анализирате общия трафик на уебсайта, като комбинирате тези данни в един `DataFrame`.

Файл `desktop_traffic.csv`:

```
Дата,Посетители,Преглеждания на страници
2024-01-01,1500,5000
2024-01-02,1650,5500
2024-01-03,1400,4800
```

Файл `mobile_traffic.csv`:

```
Дата,Посетители,Преглеждания на страници
2024-01-02,1800,6200
2024-01-03,1950,6800
2024-01-04,1700,5900
```

Задача:

Прочетете тези два файла в Pandas `DataFrame`-и и ги комбинирайте по редове в един `DataFrame`, наречен `all_traffic`, като добавите допълнителна колона 'Устройство', указваща произхода на данните ('Настолен' или 'Мобилен').

Решение на Казус 1:

```
import pandas as pd

# Прочитаме данните от CSV файловете
df_desktop = pd.read_csv('desktop_traffic.csv')
df_mobile = pd.read_csv('mobile_traffic.csv')

# Добавяме колона 'Устройство' към всеки DataFrame
df_desktop['Устройство'] = 'Настолен'
df_mobile['Устройство'] = 'Мобилен'

# Комбинираме DataFrame-ите по редове, игнорирайки оригиналните индекси
all_traffic = pd.concat([df_desktop, df_mobile], ignore_index=True)

# Извеждаме резултата
print("Комбинирани данни за трафика:\n", all_traffic)
```

Разбор на решението:

1. Използваме `pd.read_csv()` за да прочетем данните от двата CSV файла в съответните `DataFrame`-и (`df_desktop` и `df_mobile`).
2. Създаваме нова колона 'Устройство' във всеки `DataFrame` и присвояваме съответната стойност ('Настолен' или 'Мобилен'), за да можем да проследим произхода на данните след комбинирането.
3. Използваме `pd.concat()` с аргумент `axis=0` (по подразбиране, но е добре да се указва изрично за яснота) за да комбинираме `df_desktop` и `df_mobile` по редове.
4. Задаваме `ignore_index=True`, защото не искаме да запазваме оригиналните индекси (които може да се дублират) и искаме да създадем нов, последователен индекс за комбинирания `DataFrame`.

Казус 2: Събиране на резултати от анкети през различни периоди

Представете си, че провеждате онлайн анкета и събирате резултати през два различни периода. Резултатите от всеки период се съхраняват в отделен `DataFrame`. Първият `DataFrame` (`survey_period_1`) съдържа колоните 'Потребителски ID', 'Въпрос 1', 'Въпрос 2'. Вторият `DataFrame` (`survey_period_2`) съдържа колоните 'Потребителски ID', 'Въпрос 2', 'Въпрос 3'. За някои потребители може да има отговори и в двата периода. Искате да комбинирате тези резултати по колони, като запазите всички потребители и попълните `NaN` за въпросите, на които потребителят не е отговорил през съответния период.

Данни за `survey_period_1`:

```
data_period_1 = {'Потребителски ID': [1, 2, 3],
                 'Въпрос 1': ['Да', 'Не', 'Да'],
                 'Въпрос 2': ['Харесвам', 'Не харесвам', 'Харесвам']}
survey_period_1 = pd.DataFrame(data_period_1)
print("Резултати от период 1:\n", survey_period_1)
```

Данни за `survey_period_2`:

```
data_period_2 = {'Потребителски ID': [2, 3, 4],
                 'Въпрос 2': ['Много харесвам', 'Харесвам',
'Неутрален'],
                 'Въпрос 3': ['A', 'B', 'C']}
survey_period_2 = pd.DataFrame(data_period_2)
print("\nРезултати от период 2:\n", survey_period_2)
```

Задача:

Комбинирайте `survey_period_1` и `survey_period_2` по колони, използвайки 'Потребителски ID' като основа за подравняване на редовете, и добавете ниво на йерархичен индекс ('Период') към имената на колоните, за да разграничите въпросите от различните периоди.

Решение на Казус 2:

```
import pandas as pd

# Задаваме 'Потребителски ID' като индекс за двата DataFrame-а, за да ги
подравним при конкатенация по колони
survey_period_1_indexed = survey_period_1.set_index('Потребителски ID')
survey_period_2_indexed = survey_period_2.set_index('Потребителски ID')

# Комбинираме DataFrame-ите по колони (axis=1), използвайки 'keys' за
създаване на MultiIndex на колоните
all_survey_data = pd.concat([survey_period_1_indexed,
survey_period_2_indexed], axis=1, keys=['Период 1', 'Период 2'])

# Извеждаме резултата
```

```
print("\nКомбинирани резултати от анкетата:\n", all_survey_data)
```

Разбор на решението:

1. Използваме `.set_index('Потребителски ID')` за да зададем колоната 'Потребителски ID' като индекс и за двата `DataFrame`-а. Това е важно, защото при конкатенация по колони (`axis=1`), Pandas подравнява редовете въз основа на индекса.
2. Използваме `pd.concat()` с аргумент `axis=1` за да комбинираме `survey_period_1_indexed` и `survey_period_2_indexed` по колони.
3. Задаваме `keys=['Период 1', 'Период 2']`. Това създава MultiIndex за колоните, където първото ниво указва периода на анкетата, а второто ниво са оригиналните имена на колоните.
4. По подразбиране, `join='outer'` се използва, което означава, че ще бъдат запазени всички потребителски ID-та от двата `DataFrame`-а. За потребители, които са участвали само в единия период, ще има `NaN` стойности за въпросите от другия период.

Тези два казуса илюстрират как `pd.concat()` може да се използва за комбиниране на данни по редове и колони, както и как параметрите `ignore_index` и `keys` могат да бъдат полезни за управление на индексите и добавяне на контекст към комбинираните данни.

С това завършваме разглеждането на конкатенацията. В следващата глава ще преминем към сливане в стил бази данни (pd.merge()), което е по-мощен и гъвкав начин за комбиниране на DataFrame-и въз основа на връзки между колони.

II. Сливане в стил бази данни (pd.merge())

Сливането в стил бази данни (`pd.merge()`) е изключително мощен и гъвкав инструмент в Pandas за **комбиниране на два `DataFrame`-а въз основа на връзки между колони или индекси**. То наподобява операциите `JOIN` в SQL и позволява да се свързват редове от два `DataFrame`-а, когато стойностите в определени колони (или индекси) съвпадат.

За разлика от `pd.concat()`, който просто слепва `DataFrame`-ите един до друг, `pd.merge()` **съпоставя редовете въз основа на зададени ключове** и комбинира колоните от двата `DataFrame`-а в един резултатен `DataFrame`.

Основни концепции при `pd.merge()`:

- **Ключове за сливане:** Това са колоните или индексите, които се използват за определяне кои редове от двата `DataFrame`-а да бъдат комбинирани.
- **Типове сливане (`how`):** Този параметър определя кои редове да бъдат включени в резултатния `DataFrame` в зависимост от това дали ключовете съществуват и в двата входни `DataFrame`-а.

- **Обработка на колизии на имена на колони:** Когато двата `DataFrame`-а имат колони с еднакви имена (които не са ключовете за сливане), Pandas автоматично добавя суфикси към имената на колоните в резултата, за да ги разграничи. Можете да контролирате тези суфикси.
- **Индикация за произход на редовете:** Можете да добавите специална колона, която указва от кой от входните `DataFrame`-и произлиза всеки ред в резултата.
- **Валидиране на връзките:** Pandas позволява да се проверят очакваните връзки между ключовите колони (например, дали е връзка "един към един", "един към много" и т.н.).

В следващите части ще разгледаме подробно всеки от тези аспекти, като започнем с указването на ключовете за сливане. Ще видим как да използваме параметрите `on`, `left_on`, и `right_on` за да определим кои колони да се използват като ключове в левия и десния `DataFrame`. След това ще разгледаме сливането по индекс и различните типове сливания (`inner`, `outer`, `left`, `right`, `cross`). Накрая ще обсъдим как да управляваме суфиксите, да добавяме индикаторна колона и да валидираме връзките между данните.

1. Ключове за сливане (on, left_on, right_on)

Нека разгледаме подробно как да указваме ключовете за сливане при използване на `pd.merge()` с помощта на параметрите `on`, `left_on` и `right_on`. Тези параметри определят кои колони от двата `DataFrame`-а ще бъдат използвани за съпоставяне на редовете при сливането.

а) Параметър `on`:

Параметърът `on` се използва, когато **имената на колоните, които искате да използвате като ключове за сливане, са еднакви и в двата `DataFrame`-а**. Подавате му един низ (името на колоната) или списък от низове (имената на няколко колони). Сливането ще се извърши по всички посочени колони.

➢ Сценарии от реалния живот:

Представете си, че имате два `DataFrame`-а: единият съдържа информация за поръчки ('order_id', 'customer_id', 'order_date') и другият съдържа информация за клиенти ('customer_id', 'customer_name', 'email'). И двата `DataFrame`-а имат колона 'customer_id'. За да комбинирате информацията за поръчките с информацията за клиентите, можете да използвате 'customer_id' като ключ за сливане.

➢ Пример:

```
import pandas as pd

# DataFrame с информация за поръчки
orders = pd.DataFrame({
    'order_id': [1, 2, 3, 4, 5],
    'customer_id': [101, 102, 101, 103, 104],
    'order_date': ['2024-01-01', '2024-01-05', '2024-01-10', '2024-01-
15', '2024-01-20'],
```

```
    'product': ['A', 'B', 'A', 'C', 'B']
})
print("DataFrame 'orders':\n", orders)

# DataFrame с информация за клиенти
customers = pd.DataFrame({
    'customer_id': [101, 102, 103, 105],
    'customer_name': ['Alice', 'Bob', 'Charlie', 'David'],
    'email': ['alice@example.com', 'bob@example.com',
'charlie@example.com', 'david@example.com']
})
print("\nDataFrame 'customers':\n", customers)

# Сливане на двата DataFrame-а по колона 'customer_id'
merged_orders_customers = pd.merge(orders, customers, on='customer_id')
print("\nDataFrame след сливане по 'customer_id':\n",
merged_orders_customers)
```

В този пример, pd.merge(orders, customers, on='customer_id') слива orders (левия DataFrame) и customers (десния DataFrame) въз основа на съвпадащите стойности в колоната 'customer_id', която съществува и в двата.

б) Параметри `left_on` и `right_on`:

Параметрите `left_on` и `right_on` се използват, когато **имената на колоните, които искате да използвате като ключове за сливане, са различни в двата `DataFrame`-а**. `left_on` приема името на колоната от левия `DataFrame`, а `right_on` приема името на съответната колона от десния `DataFrame`.

- Сценарии от реалния живот:

Да кажем, че имате `DataFrame` с информация за служители ('emp_id', 'emp_name') и друг `DataFrame` със заплати ('worker_id', 'salary'). Въпреки че данните се отнасят до едни и същи хора, идентификаторите на служителите имат различни имена в двата `DataFrame`-а ('emp_id' и 'worker_id'). За да ги слеете, трябва да укажете коя колона от кой `DataFrame` да се използва като ключ.

- Пример:

```
import pandas as pd

# DataFrame с информация за служители
employees = pd.DataFrame({
    'emp_id': [1, 2, 3, 4],
```

```
    'emp_name': ['Eve', 'Fred', 'Grace', 'Heidi']
})
print("DataFrame 'employees':\n", employees)

# DataFrame със заплати
salaries = pd.DataFrame({
    'worker_id': [2, 4, 1, 3],
    'salary': [60000, 75000, 55000, 62000]
})
print("\nDataFrame 'salaries':\n", salaries)

# Сливане на двата DataFrame-а, използвайки различни имена на ключовите колони
merged_employees_salaries = pd.merge(employees, salaries,
left_on='emp_id', right_on='worker_id')
print("\nDataFrame след сливане по различни ключови колони:\n",
merged_employees_salaries)
```

В този пример, pd.merge(employees, salaries, left_on='emp_id', right_on='worker_id') слива employees (левия) с salaries (десния) като съпоставя стойностите от колоната 'emp_id' в employees със стойностите от колоната 'worker_id' в salaries.След сливането, ще забележите, че и двете ключови колони ('emp_id' и 'worker_id') се появяват в резултата. Ако една от тях вече не е необходима, можете да я премахнете с .drop() след сливането.

Можете също да слеете по множество колони, като подадете списък от имена на колони на on, left_on или right_on (като се уверите, че left_on и right_on имат списъци с еднаква дължина).

2. Сливане по индекс (left_index=True, right_index=True)

Нека разгледаме как да извършваме сливане на `DataFrame`-и, използвайки техните **индекси** като ключове за сливане. Това се постига с помощта на булевите параметри `left_index=True` и `right_index=True` в `pd.merge()`.

Тези параметри са полезни в случаите, когато:

- Няма подходящи общи колони за сливане, но има логическа връзка между редовете в двата `DataFrame`-а, установена чрез техния индекс.
- Индексът съдържа уникални идентификатори, които могат да бъдат използвани за свързване на данните.

- Вече сте задали определени колони като индекс и искате да слеете данни въз основа на тези индекси.

а) Как работят left_index=True и right_index=True:

- Когато зададете `left_index=True`, `pd.merge()` ще използва индекса на левия `DataFrame` като ключ за сливане.
- Когато зададете `right_index=True`, `pd.merge()` ще използва индекса на десния `DataFrame` като ключ за сливане.
- Можете да използвате един от тези параметри или и двата едновременно.

б) Сценарии от реалния живот 1:

Комбиниране на допълнителна информация по ID, зададен като индекс.

Представете си, че имате `DataFrame` с основни данни за продукти, където 'Product_ID' е зададен като индекс. Имате и друг `DataFrame` с допълнителна информация (например, рейтинг) за същите продукти, където също 'Product_ID' е зададен като индекс. Можете да ги слеете, използвайки индексите.

в) Пример 1:

```
import pandas as pd

# DataFrame с основни данни за продукти (Product_ID като индекс)
product_info = pd.DataFrame({
    'Name': ['Laptop', 'Smartphone', 'Tablet'],
    'Category': ['Electronics', 'Electronics', 'Electronics']
}, index=[101, 102, 103])
print("DataFrame 'product_info':\n", product_info)

# DataFrame с допълнителна информация (Product_ID като индекс)
product_ratings = pd.DataFrame({
    'Rating': [4.5, 4.8, 4.2],
    'Reviews': [150, 220, 180]
}, index=[102, 103, 104])
print("\nDataFrame 'product_ratings':\n", product_ratings)

# Сливане по индекс (ляв и десен)
merged_products = pd.merge(product_info, product_ratings,
left_index=True, right_index=True)
```

```
print("\nDataFrame след сливане по индекс (inner join по
подразбиране):\n", merged_products)

# Можете да укажете и друг тип сливане (например, outer)
merged_products_outer = pd.merge(product_info, product_ratings,
left_index=True, right_index=True, how='outer')
print("\nDataFrame след сливане по индекс (outer join):\n",
merged_products_outer)
```

В този пример, сливането се извършва въз основа на съвпадащите стойности на индекса в product_info и product_ratings. При вътрешно сливане (по подразбиране), се запазват само редовете, за които има съвпадение на индекса и в двата DataFrame-а. При външно сливане, се запазват всички редове, а липсващите стойности се попълват с NaN.

г) Сценарии от реалния живот 2:

Сливане на данни, където ключът в единия DataFrame е колона, а в другия - индекс.

Понякога може да се наложи да слеете `DataFrame`, където ключът за сливане в единия е обикновена колона, а в другия - индекс. В този случай, комбинирате използването на `left_on` (или `right_on`) с `right_index=True` (или `left_index=True`).

д) Пример 2:

```
import pandas as pd

# DataFrame с информация за поръчки (customer_id като колона)
orders = pd.DataFrame({
    'order_id': [1, 2, 3],
    'customer_id': [101, 102, 101],
    'item': ['A', 'B', 'C']
})
print("DataFrame 'orders':\n", orders)

# DataFrame с информация за клиенти (customer_id като индекс)
customers = pd.DataFrame({
    'name': ['Alice', 'Bob'],
    'city': ['New York', 'London']
}, index=[101, 102])
```

```
print("\nDataFrame 'customers':\n", customers)

# Сливане, използвайки колона от 'orders' и индекс от 'customers'
merged_data = pd.merge(orders, customers, left_on='customer_id',
right_index=True)
print("\nDataFrame след сливане (колона от ляво, индекс от дясно):\n",
merged_data)
```

В този случай, сливаме orders (ляв) с customers (десен), като съпоставяме стойностите от колоната 'customer_id' в orders със стойностите на индекса в customers.

3. Типове сливания (how='inner', 'outer', 'left', 'right', 'cross')

Нека разгледаме различните типове сливания, които могат да бъдат извършени с `pd.merge()` чрез параметъра `how`. Този параметър определя кои редове ще бъдат включени в резултатния `DataFrame` в зависимост от наличието на съвпадащи ключове в левия и десния `DataFrame`.

В контекста на сливане, нека условно наречем първия `DataFrame`, подаден на `pd.merge()`, "ляв", а втория - "десен".

Ето основните типове сливания:

a) `how='inner'` (вътрешно сливане):

- Резултатът съдържа **само редовете, за които има съвпадащи ключове и в левия, и в десния `DataFrame`.**
- Ако даден ключ съществува няколко пъти в един или и в двата `DataFrame`-а, ще се получи **декартово произведение** на съвпадащите редове.
- Това е **стойността по подразбиране** за параметъра `how`.

➢ Сценарии от реалния живот:

Ако имате списък с поръчки и списък с клиенти и искате да получите само тези поръчки, които са направени от съществуващи клиенти (чиито ID-та присъстват и в двата списъка), използвате `inner` join.

➢ Пример:

```
import pandas as pd

orders = pd.DataFrame({'order_id': [1, 2, 3, 4], 'customer_id': [101,
102, 101, 103]})
customers = pd.DataFrame({'customer_id': [101, 102, 104], 'name':
['Alice', 'Bob', 'Charlie']})
```

```
merged_inner = pd.merge(orders, customers, on='customer_id',
how='inner')
print("Вътрешно сливане (inner):\n", merged_inner)
```

Резултатът съдържа само поръчките с customer_id 101 и 102, защото това са единствените ID-та, които присъстват и в orders, и в customers.

б) `how='outer'` (външно сливане):

- Резултатът съдържа **всички редове от левия и десния `DataFrame`**.
- За редовете, за които няма съвпадение в другия `DataFrame` по ключа, съответните колони ще бъдат попълнени с **`NaN`**.

➢ Сценарии от реалния живот:

Ако искате да видите всички поръчки и всички клиенти, независимо дали има съответствие между тях (например, за да идентифицирате поръчки без клиент или клиенти без поръчки), използвате `outer` join.

➢ Пример:

```
merged_outer = pd.merge(orders, customers, on='customer_id',
how='outer')
print("\nВъншно сливане (outer):\n", merged_outer)
```

Резултатът съдържа всички поръчки и всички клиенти. За поръчка 4 (customer_id 103), информацията за клиента е NaN. За клиент Charlie (customer_id 104), информацията за поръчката е NaN.

в) `how='left'` (ляво сливане):

- Резултатът съдържа **всички редове от левия `DataFrame`** и съответстващите редове от десния `DataFrame` (ако има такива).
- Ако няма съвпадение в десния `DataFrame` по ключа, съответните колони от десния `DataFrame` ще бъдат попълнени с **`NaN`**.

➢ Сценарии от реалния живот:

Ако искате да запазите всички поръчки и да добавите информация за клиента, ако е налична, използвате `left` join (като `orders` е левият `DataFrame`).

➢ Пример:

```
merged_left = pd.merge(orders, customers, on='customer_id', how='left')
print("\nЛяво сливане (left):\n", merged_left)
```

Резултатът съдържа всички поръчки. За поръчка 4 (customer_id 103), информацията за клиента е `NaN`, защото няма клиент с ID 103 в `customers`.

г) `how='right'` (дясно сливане):

- Резултатът съдържа **всички редове от десния `DataFrame`** и съответстващите редове от левия `DataFrame` (ако има такива).
- Ако няма съвпадение в левия `DataFrame` по ключа, съответните колони от левия `DataFrame` ще бъдат попълнени с **`NaN`**.

➢ Сценарии от реалния живот:

Ако искате да видите всички клиенти и да добавите информация за поръчките им, ако има такива, използвате `right` join (като `customers` е десният `DataFrame`).

➢ Пример:

```
merged_right = pd.merge(orders, customers, on='customer_id',
how='right')
print("\nДясно сливане (right):\n", merged_right)
```

Резултатът съдържа всички клиенти. За клиент David (customer_id 105), информацията за поръчката е NaN, защото няма поръчка с customer_id 105 в orders.

д) `how='cross'` (кръстосано сливане):

- Резултатът представлява **декартово произведение на редовете от левия и десния `DataFrame`**.
- Всеки ред от левия `DataFrame` се комбинира с всеки ред от десния `DataFrame`.
- Този тип сливане **не изисква указване на ключови колони** (`on`, `left_on`, `right_on`) и ще генерира броя на редовете, равен на броя на редовете в левия `DataFrame`, умножен по броя на редовете в десния `DataFrame`.
- Кръстосаното сливане беше въведено в Pandas 1.2.0.

➢ Сценарии от реалния живот:

Може да се използва за генериране на всички възможни комбинации, например, всички възможни двойки продукти от два различни списъка.

➢ Пример:

```
products_a = pd.DataFrame({'name_a': ['A', 'B']})
products_b = pd.DataFrame({'name_b': ['X', 'Y', 'Z']})

merged_cross = pd.merge(products_a, products_b, how='cross')
print("\nКръстосано сливане (cross):\n", merged_cross)
```

Резултатът съдържа всички възможни комбинации от продукти от products_a и products_b.

4. Индикатори и управление на суфикси (indicator=True, suffixes)

Нека разгледаме параметрите `indicator=True` и `suffixes` в `pd.merge()`, които помагат съответно да разберем произхода на редовете след сливане и да управляваме имената на колони с еднакви имена (но не са ключовете за сливане).

a) Параметър indicator=True:

Когато зададете `indicator=True`, `pd.merge()` добавя специална колона към резултатния `DataFrame`, наречена `_merge` (по подразбиране). Тази колона съдържа категориална информация за произхода на всеки ред въз основа на ключовете за сливане:

- `'left_only'`: Редът присъства само в левия `DataFrame`.
- `'right_only'`: Редът присъства само в десния `DataFrame`.
- `'both'`: Редът има съвпадащи ключове и присъства и в двата `DataFrame`-а.

Параметърът `indicator` може също да приеме низов аргумент, който ще бъде използван като име на тази индикаторна колона.

➢ Сценарии от реалния живот:

След сливане на данни за поръчки и клиенти, може да искате бързо да идентифицирате кои поръчки нямат съответстващ клиент, кои клиенти нямат поръчки и кои поръчки имат съответстващи клиенти. Индикаторната колона улеснява този анализ.

➢ Пример:

```
import pandas as pd

orders = pd.DataFrame({'order_id': [1, 2, 3, 4], 'customer_id': [101,
102, 101, 103]})
customers = pd.DataFrame({'customer_id': [101, 102, 104], 'name':
['Alice', 'Bob', 'Charlie']})

# Сливане с добавяне на индикаторна колона
merged_indicator = pd.merge(orders, customers, on='customer_id',
how='outer', indicator=True)
print("Сливане с индикаторна колона:\n", merged_indicator)

# Можете да използвате и различно име за индикаторната колона
merged_indicator_named = pd.merge(orders, customers, on='customer_id',
how='outer', indicator='произход')
```

```
print("\nСливане с преименувана индикаторна колона:\n",
merged_indicator_named)
```

В резултата ще видите колона _merge (или произход), която показва откъде произлиза всеки ред.

б) Параметър `suffixes`:

Когато сливате два `DataFrame`-а, които имат колони с **едни и същи имена, но тези колони не са ключовете за сливане**, Pandas автоматично добавя суфикси към имената на тези колони в резултата, за да ги разграничи. По подразбиране, суфиксите са `'_x'` за колоните от левия `DataFrame` и `'_y'` за колоните от десния `DataFrame`.

Параметърът `suffixes` приема **кортеж от два низа**, указващи суфиксите, които да се използват съответно за левия и десния `DataFrame`.

- Сценарии от реалния живот:

Представете си, че имате информация за продукти от два различни източника, и двата `DataFrame`-а съдържат колона 'цена'. Когато ги слеете, Pandas ще добави суфикси, за да разграничи цената от единия източник от цената от другия. Можете да използвате `suffixes` да зададете по-описателни суфикси, например `'_източник1'` и `'_източник2'`.

- Пример:

```
import pandas as pd

products_source1 = pd.DataFrame({
    'product_id': [1, 2],
    'name': ['Laptop', 'Smartphone'],
    'price': [1200, 800]
})
print("DataFrame 'products_source1':\n", products_source1)

products_source2 = pd.DataFrame({
    'product_id': [1, 3],
    'name': ['Laptop', 'Tablet'],
    'price': [1250, 350]
})
print("\nDataFrame 'products_source2':\n", products_source2)

# Сливане по 'product_id', колоната 'name' и 'price' ще получат суфикси
```

```
merged_products_default_suffixes = pd.merge(products_source1,
products_source2, on='product_id', how='outer')
print("\nСливане с подразбиращи се суфикси:\n",
merged_products_default_suffixes)

# Сливане с указване на суфикси
merged_products_custom_suffixes = pd.merge(products_source1,
products_source2, on='product_id', how='outer', suffixes=('_източник1',
'_източник2'))
print("\nСливане с потребителски суфикси:\n",
merged_products_custom_suffixes)
```

В резултата с подразбиращи се суфикси, колоните 'name' и 'price' от левия DataFrame са преименувани на 'name_x' и 'price_x', а тези от десния - на 'name_y' и 'price_y'. Във втория случай, използвахме suffixes за да зададем по-ясни суфикси.

Използването на `indicator=True` и `suffixes` може значително да подобри разбирането и последващата обработка на резултатите от сливането, особено при работа със сложни набори от данни от различни източници.

5. Валидиране на връзките (validate)

Нека разгледаме параметъра `validate` в `pd.merge()`, който позволява да се провери дали връзките между ключовите колони в левия и десния `DataFrame` отговарят на очакван тип. Ако валидацията не успее, ще бъде върната грешка `MergeError`.

Параметърът `validate` приема един от следните низови стойности:

- `'one_to_one'` или `'1:1'`: Проверява дали има най-много един ред с всяка стойност на ключа във всеки от `DataFrame`-ите.
- `'one_to_many'` или `'1:m'`: Проверява дали има най-много един ред с всяка стойност на ключа в левия `DataFrame`.
- `'many_to_one'` или `'m:1'`: Проверява дали има най-много един ред с всяка стойност на ключа в десния `DataFrame`.
- `'many_to_many'` или `'m:m'`: Не извършва валидация (това е полезно, когато знаете, че може да има множество съвпадения).

Използването на `validate` е полезно за откриване на неочаквани дубликати в ключовите колони, които могат да доведат до неправилни резултати от сливането.

а) Сценарии от реалния живот 1:

Проверка за уникалност на потребителски ID-та при сливане на данни за поръчки и клиенти.

Ако очаквате всеки клиент да има уникален `customer_id` във вашия `customers DataFrame`, можете да използвате `validate='one_to_many'` при сливане с `orders DataFrame` (където един клиент може да има много поръчки), за да се уверите, че няма дублиращи се `customer_id` в `customers`.

- Пример 1 (валидна връзка):

```
import pandas as pd

orders = pd.DataFrame({'order_id': [1, 2, 3], 'customer_id': [101, 102, 101]})
customers = pd.DataFrame({'customer_id': [101, 102, 103], 'name': ['Alice', 'Bob', 'Charlie']})

# Очакваме един клиент да може да има много поръчки (1:m)
merged_valid = pd.merge(orders, customers, on='customer_id', validate='m:1')
print("Сливане с валидация (валидна връзка):\n", merged_valid)
```

В този случай, валидацията 'm:1' (много поръчки към един клиент) е успешна, защото всеки customer_id в customers е уникален.

- Пример 2 (невалидна връзка):

```
import pandas as pd

orders = pd.DataFrame({'order_id': [1, 2, 3], 'customer_id': [101, 102, 101]})
customers_duplicate = pd.DataFrame({'customer_id': [101, 102, 101], 'name': ['Alice', 'Bob', 'Alice']})

# Очакваме един клиент да може да има много поръчки (m:1), но има дублиращи се customer_id в customers
try:
    merged_invalid = pd.merge(orders, customers_duplicate, on='customer_id', validate='m:1')
    print("Сливане с валидация (невалидна връзка):\n", merged_invalid)
except pd.errors.MergeError as e:
    print(f"Грешка при сливане: {e}")
```

В този пример, customers_duplicate съдържа дублиращи се customer_id. Затова, валидацията 'm:1' ще предизвика MergeError, тъй като очакваме най-много един ред за всяка стойност на ключа в десния DataFrame.

б) Сценарии от реалния живот 2:

Проверка за уникалност на ID-та на поръчки при сливане на детайли за поръчки.

Ако имате два `DataFrame`-а с детайли за поръчки, които трябва да имат уникални `order_id`, можете да използвате `validate='one_to_one'` при сливането им, за да се уверите, че няма дублиращи се ID-та в нито един от тях.

- Пример 3 (проверка за one-to-one връзка):

```
import pandas as pd

order_details_1 = pd.DataFrame({'order_id': [1, 2, 3], 'item': ['A',
'B', 'C']})
order_details_2 = pd.DataFrame({'order_id': [3, 4, 5], 'quantity': [2,
1, 3]})

# Очакваме всяка поръчка да има уникален ID и в двата DataFrame-а (1:1)
merged_one_to_one_valid = pd.merge(order_details_1, order_details_2,
on='order_id', validate='1:1', how='outer')
print("Сливане с валидация (one-to-one, валидна):\n",
merged_one_to_one_valid)

order_details_duplicate_1 = pd.DataFrame({'order_id': [1, 2, 2], 'item':
['A', 'B', 'C']})
order_details_duplicate_2 = pd.DataFrame({'order_id': [3, 4, 5],
'quantity': [2, 1, 3]})

try:
    merged_one_to_one_invalid_left = pd.merge(order_details_duplicate_1,
order_details_duplicate_2, on='order_id', validate='1:1', how='outer')
    print("Сливане с валидация (one-to-one, невалидна отляво):\n",
merged_one_to_one_invalid_left)
except pd.errors.MergeError as e:
    print(f"Грешка при сливане (one-to-one, ляво дублиране): {e}")
```

Използването на validate е добра практика, за да се гарантира целостта на данните след сливане и да се хванат потенциални проблеми с дубликати в ключовите колони рано в процеса на анализ.

Казус 1:

Анализ на продажби на дребно с информация за продукти и клиенти

Представете си, че работите за компания за електронна търговия и имате два основни набора от данни:

1. **Информация за поръчките (`orders.csv`):** Съдържа детайли за всяка направена поръчка, включително ID на поръчката, ID на клиента, ID на продукта и количество.
2. **Информация за клиентите (`customers.csv`):** Съдържа детайли за регистрираните клиенти, включително ID на клиента и информация за контакт (имейл, град).

Искате да анализирате кои продукти се купуват най-често от клиенти от определени градове. За целта трябва да комбинирате тези два набора от данни.

Файл `orders.csv`:

```
OrderID,CustomerID,ProductID,Quantity
1,101,A100,2
2,102,B200,1
3,101,C300,3
4,103,A100,1
5,102,B200,2
6,104,D400,1
```

Файл `customers.csv`:

```
CustomerID,Email,City
101,alice@example.com,New York
102,bob@example.com,London
103,charlie@example.com,New York
104,david@example.com,Paris
```

Задача:

1. Прочетете файловете `orders.csv` и `customers.csv` в Pandas `DataFrame`-и.

2. Слейте тези два `DataFrame`-а, за да свържете всяка поръчка с информацията за клиента, който я е направил. Използвайте `CustomerID` като ключ за сливане.
3. Анализирайте резултата, за да определите кои продукти са закупени най-много пъти от клиенти в Ню Йорк.

Решение:

```
import pandas as pd

# 1. Прочитане на данните
df_orders = pd.read_csv('orders.csv')
df_customers = pd.read_csv('customers.csv')

print("DataFrame 'orders':\n", df_orders)
print("\nDataFrame 'customers':\n", df_customers)

# 2. Сливане на DataFrame-ите
# Използваме 'CustomerID' като ключ за сливане, тъй като това е общата колона
merged_df = pd.merge(df_orders, df_customers, on='CustomerID')

print("\nDataFrame след сливане:\n", merged_df)

# 3. Анализ на резултата
# Филтрираме само поръчките от клиенти в Ню Йорк
new_york_orders = merged_df[merged_df['City'] == 'New York']

# Групираме по 'ProductID' и сумираме 'Quantity', за да видим кои продукти са купени най-много пъти
product_sales_ny =
new_york_orders.groupby('ProductID')['Quantity'].sum()

# Сортираме резултата в низходящ ред, за да видим най-продаваните продукти
top_products_ny = product_sales_ny.sort_values(ascending=False)
```

```
print("\nПродажби на продукти в Ню Йорк (сортирани):\n",
top_products_ny)
```

Разбор на решението:

1. Използваме `pd.read_csv()` за да прочетем данните от `orders.csv` и `customers.csv` в съответните `DataFrame`-и (`df_orders` и `df_customers`).
2. Използваме `pd.merge()` за да комбинираме `df_orders` (левия `DataFrame`) и `df_customers` (десния `DataFrame`). Параметърът `on='CustomerID'` указва, че сливането трябва да се извърши въз основа на съвпадащите стойности в колоната 'CustomerID', която присъства и в двата `DataFrame`-а. По подразбиране, `how='inner'`, така че ще бъдат включени само поръчки от клиенти, които присъстват и в двата набора от данни.
3. След сливането, `merged_df` съдържа информация както за поръчките, така и за клиентите, свързани чрез 'CustomerID'.
4. За да анализираме продажбите в Ню Йорк, първо филтрираме `merged_df`, за да запазим само редовете, където стойността в колоната 'City' е 'New York'.
5. След това, използваме `.groupby('ProductID')` за да групираме филтрираните данни по 'ProductID' и `.sum()['Quantity']` за да изчислим общото количество за всеки продукт, закупен от клиенти в Ню Йорк.
6. Накрая, използваме `.sort_values(ascending=False)` за да сортираме резултата по общо количество в низходящ ред, което ни показва кои продукти са закупени най-много пъти от клиенти в Ню Йорк.

Този казус демонстрира как `pd.merge()` позволява да се комбинират данни от различни таблици въз основа на общ ключ, което е често срещана операция при подготовката на данни за анализ.

Казус 2:

Комбиниране на информация за представянето на служители от две различни системи

Представете си, че вашата компания използва две различни системи за управление на данни за служители:

1. **Система за оценка на представянето (`performance.csv`):** Съдържа информация за резултатите от годишните оценки на служителите, включително уникален идентификатор на служителя (`EmpID`), име на служителя (`EmployeeName`) и оценка (`PerformanceScore`).
2. **Система за информация за служителите (`employee_info.csv`):** Съдържа обща информация за служителите, включително различен уникален идентификатор на служителя (`StaffID`), име на служителя (`Name`), отдел (`Department`) и местоположение (`Location`).

Искате да комбинирате тези данни, за да получите общ преглед на представянето на служителите заедно с тяхната основна информация. Забележете, че идентификаторите на служителите имат различни имена в двете системи (`EmpID` и `StaffID`), а имената на служителите също са с леко различно име (`EmployeeName` и `Name`).

Файл `performance.csv`:

```
EmpID,EmployeeName,PerformanceScore
101,Alice Smith,92
102,Bob Johnson,78
103,Charlie Brown,85
104,David Lee,95
```

Файл `employee_info.csv`:

```
StaffID,Name,Department,Location
101,Alice Smith,Sales,New York
102,Robert Johnson,Marketing,London
103,Charlie Brown,Engineering,Seattle
105,Eve Williams,HR,Chicago
```

Задача:

1. Прочетете файловете `performance.csv` и `employee_info.csv` в Pandas `DataFrame`-и.
2. Слейте тези два `DataFrame`-а, свързвайки информацията за представянето с общата информация за служителите. Използвайте `EmpID` от `performance` и `StaffID` от `employee_info` като ключове за сливане.
3. Тъй като има колони с подобни имена ('EmployeeName' и 'Name'), използвайте параметъра `suffixes` при сливането, за да ги разграничите като '_Performance' и '_Info' съответно.
4. Покажете резултата от сливането.

Решение:

```
import pandas as pd

# 1. Прочитане на данните
df_performance = pd.read_csv('performance.csv')
df_employee_info = pd.read_csv('employee_info.csv')

print("DataFrame 'performance':\n", df_performance)
print("\nDataFrame 'employee_info':\n", df_employee_info)
```

```
# 2. Сливане на DataFrame-ите
# Използваме left_on и right_on за да укажем различните имена на ключовите колони
merged_df = pd.merge(df_performance, df_employee_info, left_on='EmpID', right_on='StaffID',
                     suffixes=('_Performance', '_Info'), how='left')

print("\nDataFrame след сливане:\n", merged_df)

# По желание: Можем да премахнем дублиращата се колона 'StaffID' след сливането
merged_df = merged_df.drop(columns=['StaffID'])
print("\nDataFrame след премахване на 'StaffID':\n", merged_df)
```

Разбор на решението:

1. Използваме `pd.read_csv()` за да прочетем данните от `performance.csv` и `employee_info.csv` в съответните `DataFrame`-и (`df_performance` и `df_employee_info`).
2. Използваме `pd.merge()` за да комбинираме двата `DataFrame`-а.
   - `left_on='EmpID'` указва колоната от левия `DataFrame` (`df_performance`), която ще се използва като ключ.
   - `right_on='StaffID'` указва колоната от десния `DataFrame` (`df_employee_info`), която ще се използва като ключ.
   - `suffixes=('_Performance', '_Info')` указва суфиксите, които ще бъдат добавени към колоните с еднакви имена (в случая 'EmployeeName' от левия и 'Name' от десния) след сливането.
   - `how='left'` указва, че искаме да запазим всички служители от `df_performance` и да добавим информация от `df_employee_info`, ако има съвпадение по ID. Ако няма съвпадение, колоните от десния `DataFrame` ще бъдат попълнени с `NaN`.
3. След сливането, получаваме `DataFrame`, който съдържа информация за представянето и обща информация за служителите. Забележете, че колоните с имената на служителите са разграничени с добавените суфикси.
4. По желание, можем да премахнем колоната 'StaffID', тъй като вече имаме 'EmpID', която представлява същия идентификатор.

Този казус илюстрира как `pd.merge()` може да се използва за сливане на `DataFrame`-и, когато ключовите колони имат различни имена, и как параметърът `suffixes` помага за управление на колизиите на имена на колони, което е често срещано при интегриране на данни от различни системи.

III. Сравняване на DataFrame-и (.compare())

Метода `.compare()` в Pandas е метод, който се използва за **сравняване на два `DataFrame`-а** и идентифициране на разликите между тях. Този метод връща нов `DataFrame`, който показва стойностите, които са различни в сравняваните `DataFrame`-и, както и от кой от тях произлиза всяка от различните стойности.

1. Основни характеристики на `.compare()`:

- Извиква се върху един `DataFrame` и приема другия `DataFrame` като аргумент.
- Сравнява елемент по елемент `DataFrame`-ите.
- За да бъдат сравнени, двата `DataFrame`-а трябва да имат **еднаква форма (същия брой редове и колони)** и **едни и същи имена на колоните и индекси**.
- Резултатът е `DataFrame` с **MultiIndex** за колоните. Външното ниво на MultiIndex съдържа имената на колоните, където има разлики. Вътрешното ниво съдържа две стойности: `'self'` (стойността от `DataFrame`-а, върху който е извикан `.compare()`) и `'other'` (стойността от `DataFrame`-а, подаден като аргумент).
- Стойностите, които са еднакви и в двата `DataFrame`-а, ще бъдат представени като `NaN` в резултата.

а) Сценарии от реалния живот:

Представете си, че имате два `DataFrame`-а, съдържащи данни за продажби за един и същ период, но получени от два различни източника или след извършване на някаква обработка. Искате да проверите дали има несъответствия между тези два набора от данни, за да идентифицирате потенциални грешки или различия в процесите.

б) Пример:

```
import pandas as pd

# Два DataFrame-а с еднаква структура, но някои разлики в данните
df1 = pd.DataFrame({
    'ID': [1, 2, 3],
    'Продукт': ['A', 'B', 'C'],
    'Цена': [10.0, 20.0, 15.0],
    'Количество': [5, 2, 3]
})
print("DataFrame 1:\n", df1)

df2 = pd.DataFrame({
    'ID': [1, 2, 3],
```

```
    'Продукт': ['A', 'B', 'D'],
    'Цена': [10.0, 22.0, 15.0],
    'Количество': [5, 2, 4]
})
print("\nDataFrame 2:\n", df2)

# Сравняване на двата DataFrame-а
comparison_df = df1.compare(df2)
print("\nРезултат от сравнението:\n", comparison_df)
```

В резултата `comparison_df`, ще видите MultiIndex за колоните ('Продукт', 'Цена', 'Количество'), тъй като това са колоните, в които има разлики. За всяка от тези колони, ще има две под-колони: 'self' (стойността от `df1`) и 'other' (стойността от `df2`). Редовете, в които няма разлики за дадена колона, ще имат `NaN` стойности.

Можете да филтрирате резултата, за да видите само редовете с разлики:

```
# Филтриране на редовете, където има поне една разлика
differences = comparison_df[comparison_df.notna().any(axis=1)]
print("\nСамо редовете с разлики:\n", differences)
```

2. Допълнителни параметри на `.compare()`:

- **`align_axis`:** Определя по коя ос да се подравняват `DataFrame`-ите преди сравнението. По подразбиране е `1` (по колони). Може да бъде `0` за подравняване по индекс.
- **`keep_shape`:** Булев флаг. Ако е `True`, резултатният `DataFrame` ще има същата форма като оригиналните, като разликите ще бъдат показани, а еднаквите стойности ще бъдат `NaN`. Ако е `False` (по подразбиране), резултатът ще съдържа само колоните с разлики.
- **`result_names`:** Списък от два низа, указващи имената за нивата 'self' и 'other' на MultiIndex колоните. По подразбиране са `('self', 'other')`.

➢ Пример с keep_shape=True и result_names:

```
comparison_df_full = df1.compare(df2, keep_shape=True,
result_names=('Първи', 'Втори'))
print("\nРезултат от сравнението с keep_shape=True и потребителски
имена:\n", comparison_df_full)
```

В този случай, резултатът ще има всички оригинални колони, а разликите ще бъдат показани под нива 'Първи' и 'Втори', докато еднаквите стойности ще бъдат NaN.

Методът `.compare()` е полезен инструмент за бързо идентифициране на различия между два `DataFrame`-а с еднаква структура, което е важно за валидация на данни, проследяване на промени или интегриране на данни от различни източници.

Казус 1:

Откриване на несъответствия в инвентара между две седмични справки

Представете си, че управлявате склад и генерирате седмични справки за наличните количества от всеки продукт. Имате две такива справки: една от края на миналата седмица (`inventory_week1.csv`) и една от края на текущата седмица (`inventory_week2.csv`). И двата файла съдържат информация за ID на продукта, име на продукта и налично количество. Искате да използвате Pandas, за да сравните тези две справки и да идентифицирате всички продукти, за които има разлика в наличното количество между двете седмици.

Файл `inventory_week1.csv`:

```
ProductID,ProductName,Quantity
101,Laptop,50
102,Smartphone,100
103,Tablet,75
104,Charger,150
```

Файл `inventory_week2.csv`:

```
ProductID,ProductName,Quantity
101,Laptop,48
102,Smartphone,100
103,Tablet,80
105,Headphones,120
```

Задача:

1. Прочетете файловете `inventory_week1.csv` и `inventory_week2.csv` в Pandas `DataFrame`-и.
2. За да можете да ги сравните коректно, задайте колоната `ProductID` като индекс и за двата `DataFrame`-а.
3. Сравнете двата `DataFrame`-а, използвайки метода `.compare()`.
4. Изведете само продуктите, за които има разлика в количеството между двете седмици.
5. Изведете и продуктите, които присъстват само в една от справките (те също ще бъдат отчетени като разлики поради липсващи стойности в другата справка след подравняването по индекс).

Решение:

```
import pandas as pd

# 1. Прочитане на данните
df_week1 = pd.read_csv('inventory_week1.csv')
df_week2 = pd.read_csv('inventory_week2.csv')

print("Инвентар - край на седмица 1:\n", df_week1)
print("\nИнвентар - край на седмица 2:\n", df_week2)

# 2. Задаване на 'ProductID' като индекс
df_week1 = df_week1.set_index('ProductID')
df_week2 = df_week2.set_index('ProductID')

print("\nИнвентар - край на седмица 1 (с индекс):\n", df_week1)
print("\nИнвентар - край на седмица 2 (с индекс):\n", df_week2)

# 3. Сравняване на двата DataFrame-а
comparison_df = df_week1.compare(df_week2, result_names=('Седмица 1',
'Седмица 2'))

print("\nРезултат от сравнението:\n", comparison_df)

# 4. Извеждане на продукти с разлика в количеството
quantity_diff =
comparison_df[comparison_df['Quantity'].notna().any(axis=1)]
print("\nПродукти с разлика в количеството:\n", quantity_diff)

# 5. Извеждане на всички продукти с разлики (включително тези, които
присъстват само в една справка)
all_diff = comparison_df[comparison_df.notna().any(axis=1)]
print("\nВсички продукти с разлики (вкл. нови и липсващи):\n", all_diff)
```

Разбор на решението:

1. Прочитаме данните от двата CSV файла в съответните `DataFrame`-и (`df_week1` и `df_week2`).
2. Използваме `.set_index('ProductID')` за да зададем 'ProductID' като индекс и за двата `DataFrame`-а. Това е ключова стъпка, защото `.compare()` ще сравнява редовете въз основа на съвпадащи индекси. Ако продуктите присъстват само в една от справките, след задаването на индекса те ще бъдат подравнени, а липсващите стойности ще бъдат `NaN`, което ще бъде отчетено като разлика при сравнението.
3. Използваме метода `.compare(df_week2, result_names=('Седмица 1', 'Седмица 2'))` върху `df_week1`, като подаваме `df_week2` като аргумент. Параметърът `result_names` указва имената на нивата на MultiIndex колоните в резултата, което прави интерпретацията по-лесна.
4. За да изведем само продуктите с разлика в количеството, филтрираме `comparison_df`, като проверяваме дали има не-`NaN` стойности в колоната 'Quantity'. `.notna().any(axis=1)` връща `True` за редовете, където поне една от стойностите ('Седмица 1' или 'Седмица 2') в под-колоните на 'Quantity' е различна от `NaN`.
5. За да изведем всички продукти с разлики (включително тези, които присъстват само в една от справките), прилагаме същата логика за филтриране, но върху целия `comparison_df`, като проверяваме за не-`NaN` стойности във всички колони. Продукт 'Headphones' ще се появи като разлика, защото го няма в `df_week1` (ще има `NaN` за неговите стойности в 'Седмица 1'). Аналогично, ако продукт е присъствал в `df_week1`, но го няма в `df_week2`, ще има `NaN` за неговите стойности в 'Седмица 2'.

Този казус показва как `.compare()` може да бъде ефективен инструмент за бързо идентифициране на промени и несъответствия в набори от данни, които имат обща структура и могат да бъдат подравнени.

Казус 2:

Проследяване на промени в цените на акции

Представете си, че сте финансов анализатор и искате да проследите как са се променили цените на определени акции между края на предходния месец (`stock_prices_month_end.csv`) и текущия момент (`current_stock_prices.csv`). И двата файла съдържат информация за борсовия символ на акцията (Ticker), името на компанията (Company) и цената на затваряне (ClosingPrice). Искате да използвате Pandas, за да сравните тези два набора от данни и да идентифицирате акциите, за които е имало промяна в цената.

Файл `stock_prices_month_end.csv`:

```
Ticker,Company,ClosingPrice
AAPL,Apple Inc.,170.50
GOOGL,Alphabet Inc.,2700.20
MSFT,Microsoft Corp.,285.75
AMZN,Amazon.com Inc.,3200.10
TSLA,Tesla Inc.,850.30
```

Файл `current_stock_prices.csv`:

```
Ticker,Company,ClosingPrice
AAPL,Apple Inc.,172.10
GOOGL,Alphabet Inc.,2700.20
MSFT,Microsoft Corp.,283.50
AMZN,Amazon.com Inc.,3250.00
NVDA,NVIDIA Corp.,750.80
```

Задача:

1. Прочетете файловете `stock_prices_month_end.csv` и `current_stock_prices.csv` в Pandas `DataFrame`-и.
2. За да сравните цените на едни и същи акции, задайте колоната `Ticker` като индекс и за двата `DataFrame`-а.
3. Сравнете двата `DataFrame`-а, използвайки метода `.compare()`.
4. Изведете само акциите, за които има промяна в цената на затваряне.
5. В резултата покажете борсовия символ, старата цена (от края на месеца) и новата цена (текуща).
6. Включете и акциите, които са били добавени или премахнати от списъка (те също ще се появят като разлики поради липсващи стойности след подравняването по борсов символ).

Решение:

```
import pandas as pd

# 1. Прочитане на данните
df_month_end = pd.read_csv('stock_prices_month_end.csv')
df_current = pd.read_csv('current_stock_prices.csv')

print("Цени на акции - край на месеца:\n", df_month_end)
print("\nТекущи цени на акции:\n", df_current)

# 2. Задаване на 'Ticker' като индекс
df_month_end = df_month_end.set_index('Ticker')
df_current = df_current.set_index('Ticker')
```

```
print("\nЦени на акции - край на месеца (с индекс):\n", df_month_end)
print("\nТекущи цени на акции (с индекс):\n", df_current)

# 3. Сравняване на двата DataFrame-а
comparison_df = df_month_end.compare(df_current, result_names=('Край на
месеца', 'Текуща'))

print("\nРезултат от сравнението:\n", comparison_df)

# 4. Филтриране на акциите с промяна в цената
price_change =
comparison_df[comparison_df['ClosingPrice'].notna().any(axis=1)]

print("\nАкции с промяна в цената:\n", price_change)

# 5. Извеждане на борсов символ и цените
price_changes_summary = price_change[['ClosingPrice']]
print("\nПромени в цените на акциите:\n", price_changes_summary)

# 6. Включване на всички разлики (вкл. нови и липсващи акции)
all_changes = comparison_df[comparison_df.notna().any(axis=1)]
print("\nВсички промени (вкл. цени и нови/липсващи акции):\n",
all_changes)
```

Разбор на решението:

1. Прочитаме данните от двата CSV файла в `DataFrame`-и (`df_month_end` и `df_current`).
2. Задаваме колоната 'Ticker' като индекс и за двата `DataFrame`-а, за да можем да ги сравним по борсов символ.
3. Използваме `.compare()` с подходящи `result_names` за по-ясна интерпретация на резултата.
4. Филтрираме `comparison_df`, за да запазим само редовете, където има не-`NaN` стойности в колоната 'ClosingPrice', което означава, че е имало промяна в цената.
5. Избираме само под-колоната 'ClosingPrice' от филтрирания резултат, за да покажем старата и новата цена.
6. За да видим всички промени, включително акции, които са добавени (NVDA) или премахнати (TSLA), филтрираме `comparison_df` за всички редове, където има поне една не-`NaN` стойност в която и да е колона. Акция 'NVDA' ще се появи с `NaN` за 'Край на месеца', а 'TSLA' с `NaN` за 'Текуща'.

Този казус илюстрира как `.compare()` може да се използва за проследяване на промени в данни във времето или между различни набори от данни, където е важно да се идентифицират както количествени промени, така и появата или изчезването на записи.

ВЪПРОСИ И ЗАДАЧИ

Универсални Входни Данни (не са задължителни):

```
import pandas as pd

# DataFrame #1
data1 = {'ID': [1, 2, 3, 4],
         'Име': ['Алиса', 'Боб', 'Чарли', 'Дейвид'],
         'Група': ['A', 'B', 'A', 'C'],
         'Стойност1': [10, 20, 15, 25]}
df_universal1 = pd.DataFrame(data1)
df_universal1 = df_universal1.set_index('ID') # За задачи, изискващи
индекс

# DataFrame #2
data2 = {'ID': [3, 4, 5, 6],
         'Име': ['Чарли', 'Дейвид', 'Ева', 'Фред'],
         'Статус': ['Активен', 'Неактивен', 'Активен', 'Активен'],
         'Стойност2': [100, 200, 150, 250]}
df_universal2 = pd.DataFrame(data2)
df_universal2 = df_universal2.set_index('ID') # За задачи, изискващи
индекс

# DataFrame #3 (с дублиращи се колони/индекси за някои задачи)
data3 = {'ID': [1, 2, 1, 3],
         'Име': ['Алиса', 'Боб', 'Алиса', 'Чарли'],
         'Данни': [True, False, True, True],
         'Стойност1': [5, 10, 7, 12]}
df_universal3 = pd.DataFrame(data3)
# Някои задачи може да изискват df_universal3.set_index('ID')
```

```
# Series #1, #2, #3 за конкатенация
series_uni1 = pd.Series(['Един', 'Два'])
series_uni2 = pd.Series(['Три', 'Четири'])
series_uni3 = pd.Series(['Пет', 'Шест'], index=[2, 3]) # Различен индекс
за тестване

# DataFrame с дублиращи се имена на колони за сливане със суфикси
df_suffix1 = pd.DataFrame({'ID': [1, 2], 'Име': ['А', 'Б'], 'Стойност':
[10, 20]})
df_suffix2 = pd.DataFrame({'ID': [1, 3], 'Име': ['А', 'В'], 'Стойност':
[100, 300]})

# DataFrame-и за сравнение
df_compare1 = pd.DataFrame({'Код': [1, 2, 3], 'Цена': [10.0, 20.0,
15.0], 'Наличност': [5, 2, 3]})
df_compare2 = pd.DataFrame({'Код': [1, 2, 4], 'Цена': [10.0, 22.0,
15.0], 'Наличност': [5, 3, 3]})
df_compare1_indexed = df_compare1.set_index('Код')
df_compare2_indexed = df_compare2.set_index('Код')
```

Как да използвате тези данни за всяка задача:

- **Конкатенация:** Използвайте `df_universal1`, `df_universal2`, `series_uni1`, `series_uni2`, `series_uni3`. Може да се наложи да създадете допълнителни `DataFrame`-и с различна структура от тези универсални, ако задачата го изисква (например, за комбиниране по колони с несъвпадащи редове).
- **Сливане (`pd.merge()`):** Използвайте `df_universal1`, `df_universal2`, `df_universal3`, `df_suffix1`, `df_suffix2`. Задайте кои колони да бъдат ключове (`on`, `left_on`, `right_on`) или използвайте индекси (`left_index=True`, `right_index=True`).
- **Съединяване (`.join()`):** Използвайте `df_universal1` (след задаване на индекс), `df_universal2` (след задаване на индекс), `df_suffix1` (след задаване на индекс), `df_suffix2` (след задаване на индекс). Използвайте параметъра `on` за съединяване по колона, ако е необходимо.
- **Сравняване (`.compare()`):** Използвайте `df_compare1`, `df_compare2`, `df_compare1_indexed`, `df_compare2_indexed`. Уверете се, че `DataFrame`-ите имат еднаква форма и индекси/колони, когато е необходимо.

I. Конкатенация (`pd.concat()`):

1. Дадени са два `DataFrame`-а с еднакви колони ('Име', 'Възраст'). Напишете код, който ги комбинира по редове в нов `DataFrame`.
2. Имате два `DataFrame`-а с еднакви редове, но различни колони. Напишете код, който ги комбинира по колони.
3. Дадени са три `Series` обекта. Напишете код, който ги конкатенира по редове, като игнорира оригиналните им индекси и създава нов, последователен индекс.
4. Имате два `DataFrame`-а с еднакви колони, но с потенциално дублиращи се индекси. Конкатенирайте ги по редове, като създадете йерархичен индекс, където първото ниво указва произхода на данните (например, 'DataFrame 1', 'DataFrame 2').
5. Дадени са два `DataFrame`-а с частично съвпадащи колони. Конкатенирайте ги по редове, използвайки вътрешно съединяване (`inner join`), така че в резултата да присъстват само общите колони.
6. Дадени са два `DataFrame`-а с частично съвпадащи индекси. Конкатенирайте ги по колони, използвайки външно съединяване (`outer join`), така че в резултата да присъстват всички редове от двата `DataFrame`-а (липсващите стойности да бъдат `NaN`).

II. Сливане в стил бази данни (`pd.merge()`):

1. Дадени са два `DataFrame`-а: `df_служители` с колони ('ID', 'Име', 'Отдел') и `df_заплати` с колони ('ID', 'Заплата'). Слейте ги, използвайки колоната 'ID' като ключ.
2. Имате два `DataFrame`-а: `df_поръчки` с колони ('Номер на поръчка', 'ID на клиент') и `df_клиенти` с колони ('CustomerID', 'Име на клиент'). Слейте ги, използвайки съответните колони за ID на клиент като ключове.
3. Дадени са два `DataFrame`-а, където ключът за сливане в единия е обикновена колона, а в другия е индекс. Слейте ги.
4. Извършете ляво (`left`) сливане на два `DataFrame`-а, като запазите всички редове от левия `DataFrame` и добавите съответстващите от десния (ако има такива).
5. Извършете дясно (`right`) сливане на два `DataFrame`-а, като запазите всички редове от десния `DataFrame` и добавите съответстващите от левия (ако има такива).
6. Извършете външно (`outer`) сливане на два `DataFrame`-а и покажете как да идентифицирате редовете, които съществуват само в единия от оригиналните `DataFrame`-и, използвайки параметъра `indicator=True`.
7. Слейте два `DataFrame`-а, които имат колони с еднакви имена (но не са ключовете за сливане), и използвайте параметъра `suffixes` за да ги разграничите.
8. Слейте два `DataFrame`-а и използвайте параметъра `validate` за да проверите дали връзката между ключовите колони е 'one-to-many'.

III. Съединяване по индекс (`.join()`):

1. Дадени са два `DataFrame`-а, където колоната 'ID' е зададена като индекс и за двата. Съединете ги, използвайки метода `.join()`.
2. Съединете два `DataFrame`-а по индекс, като използвате вътрешно съединяване (`inner join`).
3. Съединете два `DataFrame`-а по индекс, като използвате ляво съединяване (`left join`) и задайте различни суфикси за колоните с еднакви имена.
4. Имате `DataFrame` с колона 'ProductID' и друг `DataFrame`, където 'ProductID' е индекс. Използвайте `.join()` за да ги съедините.

IV. Сравняване на `DataFrame`-и (`.compare()`):

1. Дадени са два `DataFrame`-а с еднаква структура, но с някои различаващи се стойности. Използвайте `.compare()` за да идентифицирате разликите.

2. Сравнете два `DataFrame`-а и изведете само редовете, където има поне една разлика.
3. Сравнете два `DataFrame`-а, като запазите оригиналната форма на резултата (еднаквите стойности да бъдат `NaN`) и зададете потребителски имена за нивата на MultiIndex колоните.
4. Обяснете в какви сценарии е полезно да се използва методът `.compare()`.