INSTITUT TEKNOLOGI DEL

Sitoluama, Laguboti, Toba Samosir

Session Date	:	01 Oktober 2019
Semester	:	Gasal
Subject	:	12S4054 – Penambangan Data
Week/Session	:	4/2
Topic	:	Data Preprocessing
Activity	:	Praktikum
Duration	:	120 minutes
Rules	:	Personal
Deliverable	:	*Softcopy*
Deadline	:	03 Oktober 2019; Pukul 21:00 WIB
Place to deliver	:	https://ecourse.del.ac.id/
Objective	:	Memahami penerapan pendekatan-pendekatan umum dalam *data cleaning*, *data integration*, *data reduction*, dan *data transformation*.
Lecturer	:	SGS
Instructor	:	DES

Referensi

Berikut merupakan referensi yang digunakan dalam materi:

1. Conda, https://conda.io/docs/ (diakses 30 September 2019).
2. Github, https://github.com/github (diakses 30 September 2018).
3. J. Han, M. Kamber, dan J. Pei. (2012). *Data Mining: Concepts and Techniques (3rd ed.)*, Elsevier.
4. F. Anthony. (2015). *Mastering Pandas: Master The Features and Capabilities of Pandas, A Data Analysis Toolkit for Python*, Packt Publishing.
5. J. VanderPlas. (2016). *Python Data Science: Handbook Essential Tools for Working with Data*, O'Reilly Media.
6. D. Y. Chen. (2017). *Pandas for Everyone: Python Data Analysis*, Addison-Wesley.
7. W. McKinney. (2017). *Python for Data Analysis Data Wrangling with Pandas, NumPy, and IPython (2nd Ed.)*. O'Reilly Media.

Petunjuk

Dalam mengerjakan praktikum ini, Anda harus memperhatikan beberapa hal berikut:

1. Praktikum ini adalah pekerjaan perorangan. Anda diwajibkan untuk mengerjakan sendiri.
2. Pengumpulan pekerjaan tidak dianjurkan terlambat.

Note: PENYIMPANGAN TERHADAP PETUNJUK KE-1 MEMBUAT ANDA KEHILANGAN 100 POIN, PENYIMPANGAN TERHADAP PETUNJUK KE-2 MEMBUAT ANDA KEHILANGAN 10 POIN PER HARI KETERLAMBATAN.

Deliverables

Kumpulkan *source code* pekerjaan Anda.

Deliverables (hierarchy):

```
Praktikum3_NIM.zip
└──Data_Preprocessing_NIM.ipynb
```

Kebutuhan

Untuk dapat menyelesaikan modul praktikum ini, Anda membutuhkan hal-hal berikut ini:

- Jupyter Notebooks
- Pandas
- Numpy
- Scikit-Learn

Pendahuluan

Basis data di dunia nyata saat ini sangat rentan terhadap data yang *noisy*, hilang, dan tidak konsisten dan kemungkinan berasal dari berbagai sumber heterogen. Data berkualitas rendah akan menghasilkan hasil penambangan berkualitas rendah. Untuk itu, tahap prapemrosesan diperlukan untuk menyiapkan data untuk analisis dengan menggunakan metode-metode penambangan data.

Ada beberapa teknik prapemrosesan data, antara lain: *data cleaning*, *data integration*, *data reduction*, dan *data transformation*. *Data cleaning* diterapkan untuk menghilangkan *noise* dan memperbaiki ketidakkonsistenan dalam data. *Data integration* diterapkan untuk memadukan data dari berbagai sumber ke penyimpanan data yang koheren seperti gudang data (*data warehouse*). *Data reduction* dapat mengurangi ukuran data dengan, misalnya, menggabungkan, menghilangkan, atau pengelompokan fitur atau variabel yang redundan. *Data transformation* (mis., normalisasi) dapat diterapkan, di mana data diskalakan agar berada dalam kisaran yang lebih kecil seperti $0{,}0$ hingga $1{,}0$. Ini dapat meningkatkan akurasi dan efisiensi algoritma penambangan data yang melibatkan pengukuran jarak. Tahap-tahap ini tidak saling eksklusif; dalam artian tahap-tahap ini sangat mungkin untuk diterapkan bersamaan. Misalnya, *data cleaning* dapat melibatkan *data transformation* untuk memperbaiki data yang salah, seperti dengan mentransformasi semua entri untuk variabel tanggal ke format umum.

Dalam praktikum ini, kita akan mengeksplorasi teknik-teknik sederhana dan umum dalam melakukan *data cleaning*, *data integration*, *data reduction*, dan *data transformation*. Perlu diingat bahwa Anda masih perlu melakukan eksplorasi lebih lanjut untuk mengetahui teknik-teknik lain dalam melakukan prapemrosesan data dan aplikasinya dalam kasus dunia nyata.

Uraian Latihan

Latihan 1 | Data Cleaning

Pada bagian ini, kita akan mempelajari beberapa pendekatan umum untuk mengatasi data yang hilang.

Latihan 1.1 | Handling Missing Data

Ada banyak cara untuk menangani data (nilai atribut atau variabel suatu objek data) yang hilang. Misalnya, kita dapat mengganti nilai yang hilang dengan nilai lain, mengisi nilai yang hilang menggunakan nilai objek data lain, atau menghapus objek data dari kumpulan data yang kita miliki.

Latihan 1.1.1 | Recode/Replace

Fungsi `fillna` pada ***pandas*** dapat digunakan untuk mengkode ulang nilai yang hilang ke nilai lain. Sebagai contoh, misalkan kita ingin nilai-nilai yang hilang dikodekan ulang sebagai 0.

```
import pandas as pd
ebola = pd.read_csv('./data/country_timeseries.csv')
print(ebola.fillna(0).iloc[0:10, 0:5])
```

Latihan 1.1.2 | Fill Forward

Fungsi `fillna` pada ***pandas*** dengan *method* `ffill` dapat digunakan untuk mengisi maju. Saat kita mengisi maju suatu sel, nilai terakhir yang diketahui digunakan untuk mengganti nilai yang hilang berikutnya. Dengan cara ini, nilai yang hilang diganti dengan nilai yang diketahui / dicatat terakhir.

```
print(ebola.fillna(method='ffill').iloc[0:10, 0:5])
```

Jika kolom dimulai dengan nilai yang hilang, maka data itu akan tetap hilang karena tidak ada nilai sebelumnya untuk diisi.

Latihan 1.1.3 | Fill Backward

Fungsi `fillna` pada ***pandas*** dengan *method* `bfill` dapat digunakan untuk mengisi mundur. Saat kita mengisi mundur suatu sel, nilai “terbaru” digunakan untuk mengganti nilai yang hilang sebelumnya. Dengan cara ini, nilai yang hilang diganti dengan nilai “terbaru”.

```
print(ebola.fillna(method='bfill').iloc[:,0:5].tail())
```

Jika kolom berakhir dengan nilai yang hilang, maka kolom itu akan tetap hilang karena tidak ada nilai baru untuk diisi.

Latihan 1.1.4 | Interpolate

Interpolasi diterapkan dengan menggunakan nilai yang tersedia untuk mengisi sel tanpa nilai (nilai yang hilang). Meskipun ada banyak cara untuk mengisi nilai yang hilang, fungsi

`interpolate` di ***pandas*** mengisi nilai yang hilang secara linear. Secara khusus, fungsi ini memperlakukan nilai-nilai yang hilang seolah-olah mereka harus sama-sama diberi jarak.

```
print(ebola.interpolate().iloc[0:10,0:5])
```

Latihan 1.1.5 | Drop Missing Values

Cara terakhir untuk menangani data yang hilang adalah dengan melakukan pengamatan atau variabel dengan data yang hilang, kemudian kita dapat menggunakan metode `dropna` untuk menghilangkan data yang hilang dan menentukan parameter fungsi ini yang mengontrol bagaimana data akan dihilangkan.

```
print(ebola.shape)
```

Jika kita hanya menyimpan *case* yang lengkap dalam kumpulan data Ebola, kita hanya memiliki satu baris data.

```
ebola_dropna = ebola.dropna()
print(ebola_dropna.shape)
```

```
print(ebola_dropna)
```

Latihan 1.2 | Handling Noisy Data

Pada bagian ini, kita akan mempelajari tiga pendekatan penanganan data yang *noisy* yaitu *smoothing by bin mean*, *smoothing by bin boundary*, dan *smoothing by bin* median.

Diketahui suatu variabel *price* (dalam dolar): 4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34.

```
import numpy as np
import math

a = [4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34]  # array
sa = np.sort(array)  # sorted array
```

Inisiasilah beberapa keranjang atau *bin*.

```
# create bins
bin_mean = np.zeros((3,4))
bin_boundaries = np.zeros((3,4))
bin_median = np.zeros((3,4))
```

- ***Smoothing by bin mean***:
    - *Bin* 1: 9, 9, 9, 9
    - *Bin* 2: 23, 23, 23, 23

- *Bin* 3: 29, 29, 29, 29

```
# Bin mean
for i in range (0,12,4):
    k = int(i/4)
    mean = (sa[i] + sa[i+1] + sa[i+2] + sa[i+3])/4
    for j in range(4):
        bin_mean[k,j] = mean
print("Bin Mean: \n", bin_mean)
```

- ***Smoothing by bin boundaries***:
    - *Bin* 1: 4, 4, 4, 15
    - *Bin* 2: 21, 21, 25, 25
    - *Bin* 3: 26, 26, 26, 34

```
# Bin boundaries
for i in range (0,12,4):
    k = int(i/4)
    for j in range (4):
        if (sa[i+j]-sa[i]) < (sa[i+3]-sa[i+j]):
            bin_boundaries[k,j] = sa[i]
        else:
            bin_boundaries[k,j] = sa[i+3]
print("Bin Boundaries: \n", bin_boundaries)
```

- ***Smoothing by bin median***:
    - *Bin* 1: 9 9, 9, 9
    - *Bin* 2: 24, 24, 24, 24
    - *Bin* 3: 29, 29, 29, 29

```
# Bin median
for i in range (0,12,4):
    k = int(i/4)
    for j in range (4):
        bin_median[k,j] = sa[i+2]
print("Bin Median: \n",bin_median)
```

Apakah Anda dapat menulis code untuk mengimplementasi ketiga pendekatan tersebut dengan lebih baik?

Latihan 2 | Data Integration

Pada bagian ini, kita akan mempelajari beberapa pendekatan sederhana dalam mengintegrasikan data.

Latihan 2.1 | Concatenation

Salah satu cara (secara konseptual) yang lebih mudah untuk menggabungkan data adalah dengan *concatenation*. *Concatenation* dapat dianggap sebagai sebuah pendekatan untuk menambahkan baris atau kolom ke data Anda. Pendekatan ini dimungkinkan jika data Anda terbagi menjadi beberapa bagian atau jika Anda melakukan perhitungan yang ingin Anda tambahkan ke set data Anda yang sudah tersedia.

Concatenation dilakukan dengan menggunakan fungsi `concat` dari ***pandas***.

Latihan 2.1.1 | Adding Rows

Mari kita mulai dengan beberapa contoh kumpulan data sehingga Anda dapat melihat apa yang sebenarnya terjadi.

```
import pandas as pd

df1 = pd.read_csv('./data/concat_1.csv')
df2 = pd.read_csv('./data/concat_2.csv')
df3 = pd.read_csv('./data/concat_3.csv')
```

```
print(df1)
```

```
print(df2)
```

```
print(df3)
```

```
row_concat = pd.concat([df1, df2, df3])
print(row_concat)
```

Seperti yang Anda lihat, DataFrame ditumpuk tanpa memperhatikan konteks. Jika Anda melihat nama-nama baris (mis., indeks baris), mereka juga hanya versi *stacked* dari indeks baris asli. Jika kita menerapkan berbagai metode *subsetting* dari Tabel 1, tabel akan dimasukkan kembali seperti yang diharapkan.

```
# subset the 4th row of the concatenated dataframe
print(row_concat.iloc[3, ])
```

Tabel 1. Table of dataframe subsetting methods

Syntax	Selection Result
`df[column name]`	Single column
`df[[column1, column2, ...]]`	Multiple columns
`df.loc[row label]`	Row by row index label (row name)
`df.loc[[label1 , label2 , ...]]`	Multiple rows by index label
`df.iloc[row number]`	Row by row number

Syntax	Selection Result
`df.iloc[[row1, row2, ...]]`	Multiple rows by row number
`df.ix[label or number]`	Row by index label or number
`df.ix[[lab num1, lab num2, ...]]`	Multiple rows by index label or number
`df[bool]`	Row based on bool
`df[[bool1, bool2, ...]]`	Multiple rows based on bool
`df[start :stop: step ]`	Rows based on slicing notation

Jika kita membuat sebuah Series baru untuk ditambahkan ke DataFrame, Series tersebut tidak akan ditambahkan dengan benar.

```
# create a new row of data
new_row_series = pd.Series(['n1', 'n2', 'n3', 'n4'])
print(new_row_series)
```

Jika Series tersebut ditambahkan ke suatu DataFrame, maka Series ini tidak akan ditambahkan dengan benar.

```
# attempt to add the new row to a dataframe
print(pd.concat([df1, new_row_series]))
```

Hal pertama yang dapat kita perhatikan adalah nilai *NaN values*. Ini hanyalah cara Python mewakili 'nilai yang hilang'. Kita berharap untuk menambahkan nilai baru sebagai baris, tetapi hal ini tidak terjadi. Faktanya, kode kita tidak hanya menambahkan nilai sebagai baris, tetapi juga menciptakan sebuah kolom baru benar-benar tidak selaras dengan yang telah ada sebelumnya.

Jika kita berhenti sejenak untuk memikirkan apa yang sebenarnya terjadi, kita dapat melihat hasilnya benar-benar masuk akal. Pertama, jika kita melihat indeks baru yang ditambahkan, kita perhatikan bahwa mereka sangat mirip dengan bagaimana kita menggabungkan DataFrame sebelumnya. Indeks dari objek `new_row_series` analog dengan nomor baris DataFrame. Selanjutnya, karena Series tidak memiliki kolom yang cocok, `new_row_series` ditambahkan ke kolom baru.

Untuk memperbaiki masalah ini, kita dapat mengubah Series menjadi DataFrame. DataFrame ini berisi satu baris data, dan nama kolom adalah yang akan diikat oleh data.

```
# note the double brackets
new_row_df = pd.DataFrame([['n1', 'n2', 'n3', 'n4']], columns=['A', 'B', 'C', 'D'])
print(new_row_df)
```

```
print(pd.concat([df1, new_row_df]))
```

Bandingkan dengan penggunaan sebuah DataFrame.

```
print(df1.append(df2))
```

Bandingkan juga dengan penggunaan *single-row* DataFrame.

```
print(df1.append(new_row_df))
```

Terakhir, bandingkan dengan penggunaan Python Dictionary.

```
data_dict = {'A': 'n1',
             'B': 'n2',
             'C': 'n3',
             'D': 'n4'}

print(df1.append(data_dict, ignore_index=True))
```

Pada contoh terakhir, ketika kita menambahkan `dict` ke kerangka data, kita harus menggunakan parameter `ignore_index`. Jika kita melihat lebih dekat, Anda dapat melihat indeks baris juga bertambah 1, dan tidak mengulangi nilai indeks sebelumnya.

Jika kita hanya ingin menggabungkan atau menambahkan data secara bersamaan, kita dapat menggunakan parameter `ign_index` untuk mereset indeks baris setelah *concatenation*.

```
row_concat_i = pd.concat([df1, df2, df3], ignore_index=True)
print(row_concat_i)
```

Latihan 2.1.2 | Adding Columns

Penggabungan kolom sangat mirip dengan penggabungan baris. Perbedaan utama adalah parameter `axis` dalam fungsi `concat`.

```
col_concat = pd.concat([df1, df2, df3], axis=1)
print(col_concat)
```

```
    A   B   C   D   A   B   C   D    A    B    C    D
0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8
1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9
2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11
```

Jika kita mencoba untuk mengelompokkan berdasarkan nama kolom, kita akan mendapatkan hasil yang sama dengan ketika kita menggabungkan baris dan subset dengan indeks baris.

```
print(col_concat['A'])
```

```
    A   A    A
0  a0  a4   a8
1  a1  a5   a9
2  a2  a6  a10
3  a3  a7  a11
```

Menambahkan satu kolom ke DataFrame dapat dilakukan secara langsung tanpa menggunakan fungsi tertentu dari ***pandas***. Cukup berikan nama kolom baru vektor yang ingin Anda tetapkan ke kolom baru.

```
col_concat['new_col_list'] = ['n1', 'n2', 'n3', 'n4']
print(col_concat)
```

```
    A   B   C   D   A   B   C   D    A    B    C    D new_col_list
0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8           n1
1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9           n2
2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10           n3
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11           n4
```

```
col_concat['new_col_series'] = pd.Series(['n1', 'n2', 'n3', 'n4'])
print(col_concat)
```

```
    A   B   C   D   A   B   C   D    A    B    C    D new_col_list new_col_series
0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8           n1             n1
1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9           n2             n2
2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10           n3             n3
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11           n4             n4
```

Fungsi `concat` akan selalu berfungsi selama Anda memberikannya DataFrame.

Akhirnya, kita dapat memilih untuk mengatur ulang indeks kolom sehingga kita tidak memiliki nama kolom yang digandakan.

```
print(pd.concat([df1, df2, df3], axis=1, ignore_index=True))
```

```
    0   1   2   3   4   5   6   7    8    9    10   11
0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8
1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9
2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11
```

Latihan 2.1.3 | Concatenation With Different Indices

Contoh-contoh yang ditunjukkan sejauh ini mengasumsikan kita sedang melakukan penggabungan baris atau kolom sederhana. Pada contoh tersebut kita juga berasumsi bahwa baris baru memiliki nama kolom yang sama atau kolom memiliki indeks baris yang sama. Di bagian ini kita akan melihat apa yang terjadi ketika indeks baris dan kolom tidak selaras.

Latihan 2.1.3.1 | Concatenate Rows with Different Columns

Modifikasi DataFrame dengan mengikuti instruksi sebagai berikut.

```
df1.columns = ['A', 'B', 'C', 'D']
df2.columns = ['E', 'F', 'G', 'H']
df3.columns = ['A', 'C', 'F', 'H']
```

```
print(df1)
```

```
    A   B   C   D
0  a0  b0  c0  d0
1  a1  b1  c1  d1
2  a2  b2  c2  d2
3  a3  b3  c3  d3
```

```
print(df2)
```

```
    E   F   G   H
0  a4  b4  c4  d4
1  a5  b5  c5  d5
2  a6  b6  c6  d6
3  a7  b7  c7  d7
```

```
print(df3)
```

```
     A    C    F    H
0   a8   b8   c8   d8
1   a9   b9   c9   d9
2  a10  b10  c10  d10
3  a11  b11  c11  d11
```

Jika kita mencoba untuk mengkonkatenasi bingkai data seperti yang kita lakukan di Latihan 2.1.1 | Adding Rows, DataFrame sekarang melakukan lebih dari sekadar menumpuk satu di atas yang lain. Kolom menyelaraskan diri, dan NaN mengisi area yang hilang.

```
row_concat = pd.concat([df1, df2, df3], sort=False)
print(row_concat)
```

```
     A    B    C    D    E    F    G    H
0   a0   b0   c0   d0  NaN  NaN  NaN  NaN
1   a1   b1   c1   d1  NaN  NaN  NaN  NaN
2   a2   b2   c2   d2  NaN  NaN  NaN  NaN
3   a3   b3   c3   d3  NaN  NaN  NaN  NaN
0  NaN  NaN  NaN  NaN   a4   b4   c4   d4
1  NaN  NaN  NaN  NaN   a5   b5   c5   d5
2  NaN  NaN  NaN  NaN   a6   b6   c6   d6
3  NaN  NaN  NaN  NaN   a7   b7   c7   d7
0   a8  NaN   b8  NaN  NaN   c8  NaN   d8
1   a9  NaN   b9  NaN  NaN   c9  NaN   d9
2  a10  NaN  b10  NaN  NaN  c10  NaN  d10
3  a11  NaN  b11  NaN  NaN  c11  NaN  d11
```

Salah satu cara untuk menghindari dimasukkannya nilai NaN adalah dengan menyimpan hanya kolom-kolom yang dipakai bersama-sama dengan daftar objek yang akan digabungkan. Sebuah parameter yang bernama `join`, dapat digunakan untuk menyelesaikan persoalan ini. Secara *default*, ia memiliki nilai '`outer`', yang berarti ia akan menyimpan semua kolom. Namun, kita dapat mengatur `join = 'inner'` untuk menjaga hanya kolom yang dibagi di antara set data. Jika kita mencoba untuk menyimpan hanya kolom dari ketiga DataFrame, kita akan mendapatkan DataFrame kosong karena tidak ada kolom yang sama.

```
print(pd.concat([df1, df2, df3], join='inner'))
```

```
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
```

Jika kita menggunakan DataFrame yang memiliki kolom yang sama, maka hanya kolom yang digunakan bersama-sama yang akan dikembalikan.

```
print(pd.concat([df1,df3], ignore_index=False, join='inner'))
```

```
     A    C
0   a0   c0
1   a1   c1
2   a2   c2
3   a3   c3
0   a8   b8
1   a9   b9
2  a10  b10
3  a11  b11
```

Latihan 2.1.3.2 | Concatenate Columns with Different Rows

Mari kita ambil DataFrame kita dan memodifikasinya lagi sehingga mereka memiliki indeks baris yang berbeda. Di sini, kita membangun modifikasi DataFrame yang sama dari Latihan 2.1.3.1 | Concatenate Rows with Different Columns.

```
df1.index = [0, 1, 2, 3]
df2.index = [4, 5, 6, 7]
df3.index = [0, 2, 5, 7]

print(df1)
```

```
    A   B   C   D
0  a0  b0  c0  d0
1  a1  b1  c1  d1
2  a2  b2  c2  d2
3  a3  b3  c3  d3
```

```
print(df2)
```

```
    E   F   G   H
4  a4  b4  c4  d4
5  a5  b5  c5  d5
6  a6  b6  c6  d6
7  a7  b7  c7  d7
```

```
print(df3)
```

```
     A    C    F    H
0   a8   b8   c8   d8
2   a9   b9   c9   d9
5  a10  b10  c10  d10
7  a11  b11  c11  d11
```

Ketika kita menggabungan sepanjang `axis=1`, kita mendapatkan hasil yang sama dari meggabungkan sepanjang `axis=0`. DataFrame baru akan menambahkan kolom dan

dicocokkan dengan indeks baris masing-masing. Indikator nilai yang hilang muncul di area di mana indeks tidak selaras.

```
col_concat = pd.concat([df1, df2, df3], axis=1)
print(col_concat)
```

	A	B	C	D	E	F	G	H	A	C	F	H
0	a0	b0	c0	d0	NaN	NaN	NaN	NaN	a8	b8	c8	d8
1	a1	b1	c1	d1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	a2	b2	c2	d2	NaN	NaN	NaN	NaN	a9	b9	c9	d9
3	a3	b3	c3	d3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	a4	b4	c4	d4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	a5	b5	c5	d5	a10	b10	c10	d10
6	NaN	NaN	NaN	NaN	a6	b6	c6	d6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	a7	b7	c7	d7	a11	b11	c11	d11

Seperti yang kita lakukan ketika menggabungkan baris, kita dapat memilih untuk menyimpan hasilnya ketika ada indeks yang cocok dengan menggunakan `join = 'inner'`.

```
print(pd.concat([df1, df3], axis=1, join='inner'))
```

	A	B	C	D	A	C	F	H
0	a0	b0	c0	d0	a8	b8	c8	d8
2	a2	b2	c2	d2	a9	b9	c9	d9

Latihan 2.2 | Merging Multiple Data Sets

Bagian sebelumnya menyinggung beberapa konsep basis data. Parameter `join = 'inner'` dan parameter *default* `join='outer'` diadopsi dari konsep menggabungkan tabel ketika bekerja dengan basis data relasional.

Alih-alih hanya memiliki indeks baris atau kolom yang ingin kita konkatenasi, akan ada saatnya Anda memiliki 2 atau lebih DataFrame yang ingin Anda gabungkan berdasarkan pada *common data values*. Ini dikenal di dunia basis data sebagai melakukan "join".

Pandas memiliki perintah `pd.join` yang menggunakan `pd.merge` di bawahnya. Fungsi `join` akan menggabungkan objek DataFrame dengan indeks, tetapi fungsi `merge` jauh lebih eksplisit dan fleksibel. Jika Anda hanya berencana untuk menggabungkan kerangka data dengan indeks baris, Anda dapat melihat dalam dokumentasi fungsi join.

Kita akan menggunakan data survei dalam serangkaian contoh ini.

```
person = pd.read_csv('./data/survey_person.csv')
site = pd.read_csv('./data/survey_site.csv')
survey = pd.read_csv('./data/survey_survey.csv')
visited = pd.read_csv('./data/survey_visited.csv')

print(person)
print(survey)
print(site)
print(visited)
```

```
      ident   personal    family
0      dyer    William      Dyer
1        pb      Frank   Pabodie
2      lake   Anderson      Lake
3       roe  Valentina   Roerich
4  danforth      Frank  Danforth

    taken person quant  reading
0     619   dyer   rad     9.82
1     619   dyer   sal     0.13
2     622   dyer   rad     7.80
3     622   dyer   sal     0.09
4     734     pb   rad     8.41
5     734   lake   sal     0.05
6     734     pb  temp   -21.50
7     735     pb   rad     7.22
8     735    NaN   sal     0.06
9     735    NaN  temp   -26.00
10    751     pb   rad     4.35
11    751     pb  temp   -18.50
12    751   lake   sal     0.10
13    752   lake   rad     2.19
14    752   lake   sal     0.09
15    752   lake  temp   -16.00
16    752    roe   sal    41.60
17    837   lake   rad     1.46
18    837   lake   sal     0.21
```

```
19    837    roe   sal    22.50
20    844    roe   rad    11.25

    name    lat    long
0   DR-1 -49.85 -128.57
1   DR-3 -47.15 -126.72
2  MSK-4 -48.87 -123.40

   ident   site       dated
0    619   DR-1  1927-02-08
1    622   DR-1  1927-02-10
2    734   DR-3  1939-01-07
3    735   DR-3  1930-01-12
4    751   DR-3  1930-02-26
5    752   DR-3         NaN
6    837  MSK-4  1932-01-14
7    844   DR-1  1932-03-22
```

Saat ini, data kita dibagi menjadi beberapa bagian, di mana setiap bagian adalah unit observasional. Jika kita ingin melihat tanggal di setiap situs dengan latlong situs. Kita harus menggabungkan beberapa DataFrame. Kita melakukan ini dengan fungsi `merge` di ***pandas***. Fungsi `merge` merupakan sebuah metode DataFrame.

Ketika kita memanggil metode ini, DataFrame yang dipanggil akan dirujuk ke yang ada di 'kiri'. Dalam fungsi `merge`, parameter pertama adalah 'right' DataFrame. Parameter berikutnya adalah bagaimana hasil akhir penggabungan terlihat. Lihat Tabel 2 untuk lebih jelasnya. Selanjutnya, kita mengatur parameter `on`. Ini menentukan kolom mana yang cocok. Jika kolom kiri dan kanan bukan nama yang sama, kita dapat menggunakan parameter `left_on` dan `right_on` sebagai gantinya.

Tabel 2. How the Pandas how Parameter Relates to SQL

Pandas	SQL	Description
left	left outer	Keep all the keys from the left
right	right outer	Keep all the keys from the right
outer	full outer	Keep all the keys from both left and right
inner	inner	Keep only the keys that exist in the left and right

Latihan 2.2.1 | One-to-One Merge

Dalam jenis penggabungan yang paling sederhana, kitai memiliki dua DataFrame di mana kita ingin menggabungkan satu kolom ke kolom lainnya, dan di mana kolom yang ingin kita gabungkan tidak mengandung nilai duplikat apa pun.

Untuk contoh ini, kita akan memodifikasi DataFrame `visited` sehingga tidak ada nilai situs yang diduplikasi.

```
visited_subset = visited.ix[[0, 2, 6], ]
```

Kita dapat menampilkan *one-to-one merge* sebagai berikut.

```
# the default value for 'how' is 'inner'
# so it doesn't need to be specified
o2o_merge = site.merge(visited_subset, left_on='name', right_on='site')
print(o2o_merge)
```

```
    name    lat    long  ident   site       dated
0   DR-1 -49.85 -128.57    619   DR-1  1927-02-08
1   DR-3 -47.15 -126.72    734   DR-3  1939-01-07
2  MSK-4 -48.87 -123.40    837  MSK-4  1932-01-14
```

Seperti yang Anda lihat, kita baru saja membuat DataFrame baru dari dua kerangka data terpisah dimana baris dicocokkan berdasarkan kumpulan kolom tertentu. Dalam SQL, kolom yang digunakan untuk mencocokkan disebut "*key*".

Latihan 2.2.2 | Many-to-One Merge

Jika kita memilih untuk melakukan penggabungan yang sama, tetapi kali ini tanpa menggunakan *subsetted visited* DataFrame, kita akan melakukan *many-to-one merge*. Dalam penggabungan semacam ini, salah satu DataFrame memiliki nilai *key* yang berulang. DataFrame yang berisi *single observation* kemudian akan diduplikasi dalam penggabungan.

```
m2o_merge = site.merge(visited, left_on='name', right_on='site')
print(m2o_merge)
```

```
    name    lat    long  ident   site       dated
0   DR-1 -49.85 -128.57    619   DR-1  1927-02-08
1   DR-1 -49.85 -128.57    622   DR-1  1927-02-10
2   DR-1 -49.85 -128.57    844   DR-1  1932-03-22
3   DR-3 -47.15 -126.72    734   DR-3  1939-01-07
4   DR-3 -47.15 -126.72    735   DR-3  1930-01-12
5   DR-3 -47.15 -126.72    751   DR-3  1930-02-26
6   DR-3 -47.15 -126.72    752   DR-3         NaN
7  MSK-4 -48.87 -123.40    837  MSK-4  1932-01-14
```

Seperti yang Anda lihat, informasi situs (`name`, `lat`, dan `long`) digandakan dan dicocokkan dengan data `visited`.

Latihan 2.2.3 | Many-to-Many Merge

Terakhir, akan ada saat-saat ketika kita ingin melakukan pencocokan berdasarkan beberapa kolom. Sebagai contoh, misalkan kita memiliki dua DataFrame yang berasal dari `person` digabungkan dengan `survey`, dan DataFrame lain yang berasal dari `visited` digabungkan dengan `survey`.

```
ps = person.merge(survey, left_on='ident', right_on='person')
vs = visited.merge(survey, left_on='ident', right_on='taken')
print(ps)
```

```
    ident   personal    family  taken person quant  reading
0    dyer    William       Dyer    619   dyer   rad     9.82
1    dyer    William       Dyer    619   dyer   sal     0.13
2    dyer    William       Dyer    622   dyer   rad     7.80
3    dyer    William       Dyer    622   dyer   sal     0.09
4      pb      Frank    Pabodie    734     pb   rad     8.41
5      pb      Frank    Pabodie    734     pb  temp   -21.50
6      pb      Frank    Pabodie    735     pb   rad     7.22
7      pb      Frank    Pabodie    751     pb   rad     4.35
8      pb      Frank    Pabodie    751     pb  temp   -18.50
9    lake   Anderson       Lake    734   lake   sal     0.05
10   lake   Anderson       Lake    751   lake   sal     0.10
11   lake   Anderson       Lake    752   lake   rad     2.19
12   lake   Anderson       Lake    752   lake   sal     0.09
13   lake   Anderson       Lake    752   lake  temp   -16.00
14   lake   Anderson       Lake    837   lake   rad     1.46
15   lake   Anderson       Lake    837   lake   sal     0.21
16    roe  Valentina    Roerich    752    roe   sal    41.60
17    roe  Valentina    Roerich    837    roe   sal    22.50
18    roe  Valentina    Roerich    844    roe   rad    11.25
```

```
print(vs)
```

```
    ident   site       dated  taken person quant  reading
0     619   DR-1  1927-02-08    619   dyer   rad     9.82
1     619   DR-1  1927-02-08    619   dyer   sal     0.13
2     622   DR-1  1927-02-10    622   dyer   rad     7.80
3     622   DR-1  1927-02-10    622   dyer   sal     0.09
4     734   DR-3  1939-01-07    734     pb   rad     8.41
5     734   DR-3  1939-01-07    734   lake   sal     0.05
6     734   DR-3  1939-01-07    734     pb  temp   -21.50
7     735   DR-3  1930-01-12    735     pb   rad     7.22
8     735   DR-3  1930-01-12    735    NaN   sal     0.06
9     735   DR-3  1930-01-12    735    NaN  temp   -26.00
10    751   DR-3  1930-02-26    751     pb   rad     4.35
11    751   DR-3  1930-02-26    751     pb  temp   -18.50
12    751   DR-3  1930-02-26    751   lake   sal     0.10
13    752   DR-3         NaN    752   lake   rad     2.19
14    752   DR-3         NaN    752   lake   sal     0.09
15    752   DR-3         NaN    752   lake  temp   -16.00
16    752   DR-3         NaN    752    roe   sal    41.60
17    837  MSK-4  1932-01-14    837   lake   rad     1.46
18    837  MSK-4  1932-01-14    837   lake   sal     0.21
19    837  MSK-4  1932-01-14    837    roe   sal    22.50
20    844   DR-1  1932-03-22    844    roe   rad    11.25
```

Kita dapat melakukan *many-to-many merge* dengan meneruskan beberapa kolom untuk disesuaikan dengan list Python (struktur data list).

```
ps_vs = ps.merge(vs,

                 left_on=['ident', 'taken', 'quant', 'reading'],
                 right_on=['person', 'ident', 'quant', 'reading'])
```

Perhatikan data baris pertama.

```
print(ps_vs.loc[0, ])
```

```
ident_x          dyer
personal      William
family           Dyer
taken_x           619
person_x         dyer
quant             rad
reading          9.82
ident_y           619
site             DR-1
dated      1927-02-08
taken_y           619
person_y         dyer
Name: 0, dtype: object
```

Pandas akan secara otomatis menambahkan akhiran ke nama sebuah kolom jika ada tabrakan dalam nama. Dalam output, `_x` merujuk ke nilai-nilai dari kerangka data kiri, dan akhiran `_y` berasal dari nilai-nilai di DataFrame yang tepat.

Latihan 3 | Data Reduction

Pada bagian ini, kita akan mempelajari satu pendekatan umum untuk melakukan pengurangan dimensionalitas, yaitu *principal component analysis* (PCA).

PCA adalah pelopor metode *subspace* linear pada reduksi dimensi. PCA berusaha menemukan transformasi ortogonal yang memproyeksikan data ke suatu *subspace* yang meminimalkan korelasi hasil proyeksi. *Subspace* ini disebut sebagai *principal subspace*. PCA merupakan suatu metode *unsupervised* dimana label pada data pelatihan tidak digunakan untuk melakukan pembelajaran. Dalam hal klasifikasi, karenanya, proyeksi PCA boleh jadi tidak optimal. Proyeksi PCA selalu berusaha mempertahankan semua jenis variasi secara maksimal tanpa menghiraukan factor-faktor yang memunculkan variasi tersebut.

```
import pandas as pd
from sklearn.decomposition import PCA

data = pd.read_csv('./data/d_data.csv')

pca = PCA(n_components=2).fit_transform(data)
print(pca)
```

```
[[  7.84170828 -12.59183041]
 [  9.2638174   -1.06657762]
 [ 20.96602518  -3.92729715]
 [ 12.43201934 -25.18583558]
 [-16.01355999  16.63692306]
 [  5.13191019   2.38929446]
```

```
 [ -9.33623593  -5.27897273]
 [ -6.58085739 -16.319062  ]
 [-12.23414033   0.30674688]
 [  7.51111002   6.77052294]
 [-36.217585    -2.34725768]
 [-14.32048033  -9.2296796 ]
 [  5.13301929  18.98084587]
 [-17.49829341   5.33277962]
 [ 13.78517421  20.25751047]
 [  1.51513576  25.29789293]
 [ 11.83756983   1.57183023]
 [ 21.19439155  -3.83927071]
 [ -4.41072868 -17.75856299]]
```

Latihan 4 | Data Transformation

Pada bagian ini, kita akan mempelajari beberapa pendekatan sederhana dalam mentransformasi data.

Latihan 4.1 | Aggregate

Agregasi adalah proses mengambil beberapa nilai dan mengembalikan satu nilai. Menghitung rata-rata aritmatika adalah contohnya, di mana rata-rata dari beberapa nilai adalah nilai tunggal.

Latihan 4.1.1 | Basic One-Variable Grouped Aggregation

Agregasi kadang-kadang bisa disebut sebagai *summarization*. Kedua istilah tersebut berarti bahwa beberapa bentuk *data reduction* terlibat. Misalnya, ketika Anda menghitung statistik ringkasan, seperti rata-rata, Anda mengambil beberapa nilai dan menggantinya dengan nilai tunggal. Jumlah data sekarang lebih kecil.

Catatan: ***TSV** is a **file** extension for a tab-delimited **file** used with spreadsheet software. **TSV** stands for Tab Separated Values. **TSV files** are used for raw data and can be imported into and exported from spreadsheet software.*

```
# load the gapminder data
import pandas as pd

df = pd.read_csv('./data/gapminder.tsv', sep='\t')

# calculate the average life expectancy for each year
avg_life_exp_by_year = df.groupby('year').lifeExp.mean()
print(avg_life_exp_by_year)
```

```
year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
```

```
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

Pernyataan sebelumnya menggunakan notasi titik untuk subset kolom `lifeExp`; persis sama dengan menggunakan notasi braket `[]`.

```
avg_life_exp_by_year = df.groupby('year')['lifeExp'].mean()
```

Fungsi `groupby` dapat dianggap sebagai suatu cara untuk membuat subset dari setiap nilai unik kolom (atau pasangan unik dari kolom). Misalnya, kita bisa mendapatkan daftar nilai unik di kolom.

```
# get a list of unique years in the data
years = df.year.unique()
print(years)
```

```
[1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007]
```

Kita dapat melihat masing-masing data dan subsetnya, sebagai berikut.

```
# subset the data for the year 1952
y1952 = df.loc[df.year == 1952, :]
print(y1952.head())
```

```
       country continent  year  lifeExp       pop     gdpPercap
0  Afghanistan      Asia  1952   28.801   8425333   779.445314
12     Albania    Europe  1952   55.230   1282697  1601.056136
24     Algeria    Africa  1952   43.077   9279525  2449.008185
36      Angola    Africa  1952   30.015   4232095  3520.610273
48   Argentina  Americas  1952   62.485  17876956  5911.315053
```

Akhirnya, kita dapat menampilkan fungsi pada subset data. Di sini kita mengambil rata-rata dari `lifeExp`.

```
y1952_mean = y1952.lifeExp.mean()
print(y1952_mean)
```

```
49.05761971830987
```

Fungsi `groupby` pada dasarnya mengulangi proses ini untuk setiap kolom tahun, dan dengan mudah mengembalikan semua hasil dalam satu DataFrame.

Tentu saja, `mean` bukan satu-satunya jenis fungsi agregasi yang dapat Anda gunakan. Ada banyak metode bawaan di ***pandas*** yang dapat Anda gunakan dengan `groupby`.

Latihan 4.1.2 | Built-in Aggregation Methods

Tabel 3 menyajikan daftar metode dan fungsi bawaan pada ***pandas*** yang tidak eksklusif, yang dapat Anda gunakan untuk mengagregasi data Anda.

Tabel 3. Methods and Functions that can be used with `groupby`

Pandas Method	`numpy/scipy` function	Description
`count`	`np.count_nonzero`	Frequency count not including `NaN` values
`size`		Frequency count with `NaN` values
`mean`	`np.mean`	Mean of the values
`std`	`np.std`	Sample standard deviation
`min`	`np.min`	Minimum values
`quantile(q=0.25)`	`np.percentile(q=0.25)`	25th percentile of the values
`quantile(q=0.50)`	`np.percentile(q=0.50)`	50th percentile of the values
`quantile(q=0.75)`	`np.percentile(q=0.75)`	75th percentile of the values

max	np.max	Maximum value
sum	np.sum	Sum of the values
var	np.var	Unbiased variance
sem	scipy.stats.sem	Unbiased standard error of the mean
describe	scipy.stats.describe	Count, mean, standard deviation, minimum, 25%, 50%, 75%, and maximum
first		Returns the first row
last		Returns the last row
nth		Returns the nth row (Python starts counting from 0)

Misalnya, kita dapat menghitung beberapa statistik ringkasan secara bersamaan dengan `describe`.

```
# group by continent and describe each group
continent_describe = df.groupby('continent').lifeExp.describe()
print(continent_describe)
```

continent	count	mean	std	min	25%	50%	75%	max
Africa	624.0	48.865330	9.150210	23.599	42.37250	47.7920	54.41150	76.442
Americas	300.0	64.658737	9.345088	37.579	58.41000	67.0480	71.69950	80.653
Asia	396.0	60.064903	11.864532	28.801	51.42625	61.7915	69.50525	82.603
Europe	360.0	71.903686	5.433178	43.585	69.57000	72.2410	75.45050	81.757
Oceania	24.0	74.326208	3.795611	69.120	71.20500	73.6650	77.55250	81.235

Latihan 4.1.3 | Aggregation Functions

Anda juga dapat menggunakan fungsi agregasi yang tidak tercantum dalam kolom "**Pandas Method**" pada Tabel 3. Sebagai pengganti langsung memanggil metode agregasi, Anda dapat memanggil `agg` atau `aggregate`. Saat menggunakan `agg` atau `aggregate`, Anda akan menggunakan fungsi-fungsi yang tercantum dalam kolom "fungsi numpy / scipy" pada Tabel 3.

Latihan 4.2 | Normalization

Saat kita menormalisasi (mentransformasi) data, kita meneruskan nilai dari DataFrame kita ke suatu fungsi. Fungsi kemudian "mentransformasi" data. Tidak seperti `aggregate`, yang dapat

mengambil banyak nilai dan mengembalikan nilai tunggal (teragregasi), `transform` mengambil beberapa nilai dan mengembalikan *one-to-one transformation* dari nilai tersebut. Artinya, fungsi ini tidak mengurangi jumlah data. Perhatikan kembali *course slide*: **Data Preprocessing** untuk keterangan lebih lanjut.

```
# load the gapminder data
import pandas as pd

df = pd.read_csv('./data/gapminder.tsv', sep='\t')

def my_zscore(x):
    '''Calculates the z-score of provided data
    'x' is a vector or series of values.
    '''
    return((x - x.mean()) / x.std())
```

Sekarang, kita dapat menggunakan fungsi ini untuk mentransformasikan data yang kita miliki.

```
transform_z = df.groupby('year').lifeExp.transform(my_zscore)
```

Perhatikan bentuk DataFrame yang kita miliki , dan bentuk `transform_z`. Keduanya memiliki jumlah baris dan data yang sama.

```
# note the number of rows in our data
print(df.shape)
```

```
(1704, 6)
```

```
# note the number of values in our transformation
print(transform_z.shape)
```

```
(1704,)
```

Scipy *library* memiliki fungsi `zscore` sendiri. Mari kita gunakan fungsi `zscore` dalam `groupby transform`, daripada dalam `groupby`.

```
# import the zscore function from scipy.stats
from scipy.stats import zscore

# calculate a grouped zscore
sp_z_grouped = df.groupby('year').lifeExp.transform(zscore)

# calculate a nongrouped zscore
sp_z_nogroup = zscore(df.lifeExp)
```

Perhatikan bahwa tidak semua nilai `zscore` sama.

```
# grouped z-score
print(transform_z.head())
```

```
0   -1.656854
1   -1.731249
2   -1.786543
3   -1.848157
4   -1.894173
Name: lifeExp, dtype: float64
```

```
# grouped z-score using scipy
print(sp_z_grouped.head())
```

```
0   -1.662719
1   -1.737377
2   -1.792867
3   -1.854699
4   -1.900878
Name: lifeExp, dtype: float64
```

```
# nongrouped z-score
print(sp_z_nogroup[:5])
```

```
[-2.37533395 -2.25677417 -2.1278375  -1.97117751 -1.81103275]
```

Hasil yang diperoleh mirip. Namun, ketika kita menghitung *z-score* di luar `groupby`, kita mendapatkan *z-score* yang dihitung pada seluruh data set, tidak dibagi berdasarkan kelompok (*group*).