2025

MODUL PRAKTIKUM 3

PANDAS FOR BIG DATA ANALYTICS

OPTIMIZED VERSION

Disusun Oleh:
Ryan F F Hakim
Silvi Nurinsan

Dipersembahkan Untuk:
Sains Data

Mata Kuliah
Analisis Big Data

APA ITU PANDAS?

Pandas adalah sebuah pustaka (library) Python yang bersifat open-source dan sangat populer digunakan untuk manipulasi dan analisis data. Pustaka ini menyediakan struktur data dan fungsi-fungsi yang efisien untuk melakukan berbagai operasi pada data, khususnya data tabular seperti spreadsheet atau tabel SQL. Nama "Pandas" sendiri merupakan singkatan dari "Panel Data" dan juga "Python Data Analysis". Pandas dikembangkan oleh Wes McKinney dan pertama kali dirilis pada tahun 2008, dan sejak saat itu telah berkembang menjadi salah satu alat paling penting dalam ekosistem ilmu data Python.

Pandas dibangun di atas pustaka NumPy, yang memudahkannya dalam manipulasi dan analisis data numerik. Pandas bertujuan menjadi blok bangunan tingkat tinggi yang fundamental untuk melakukan analisis data praktis di dunia nyata menggunakan Python.

PERAN PANDAS DALAM EKOSISTEM PYTHON UNTUK DATA SCIENCE

Pandas memainkan peran sentral dalam ekosistem Python untuk ilmu data karena kemampuannya untuk bekerja dengan baik bersama pustaka penting lainnya. Pustaka ini sering digunakan bersama dengan:

- **NumPy**: Untuk operasi numerik dan array multidimensi.
- **Matplotlib** dan **Seaborn**: Untuk visualisasi data dan pembuatan grafik.
- **SciPy**: Untuk analisis statistik.

- **Scikit-learn**: Untuk algoritma pembelajaran mesin.

Pandas berfungsi sebagai lapisan dasar untuk komputasi statistik di Python, melengkapi tumpukan ilmiah Python yang sudah ada dan meningkatkan jenis alat manipulasi data yang ditemukan dalam bahasa pemrograman statistik lainnya seperti R. Dalam alur kerja ilmu data, Pandas sangat penting untuk tugas-tugas seperti pembersihan data (data cleaning), pra-pemrosesan data (data preprocessing), transformasi data, dan eksplorasi data.

KEUNGGULAN PANDAS

Pandas menawarkan berbagai keunggulan yang menjadikannya pilihan utama bagi para ilmuwan data dan analis:

- **Penanganan Data yang Efisien**: Pandas menyediakan struktur data yang cepat dan fleksibel seperti DataFrame dan Series, yang memungkinkan penanganan data tabular dan deret waktu secara efisien.
- **Kemudahan Penggunaan**: Sintaks Pandas yang intuitif dan dokumentasi yang komprehensif membuatnya mudah dipelajari dan digunakan, bahkan untuk pemula. Pandas memungkinkan operasi data yang kompleks dilakukan hanya dengan beberapa baris kode.
- **Fungsi Manipulasi Data yang Kuat**: Pandas menyediakan berbagai fungsi untuk membersihkan, menggabungkan, membentuk ulang (reshaping), memfilter, mengelompokkan (grouping), dan mengagregasi data. Ini termasuk penanganan data yang hilang (missing data) dengan mudah, penyisipan dan penghapusan kolom, serta operasi "split-apply-combine" yang kuat.
- **Integrasi yang Mulus**: Pandas terintegrasi dengan baik dengan pustaka Python lainnya dalam ekosistem ilmu data, seperti NumPy, Matplotlib, dan Scikit-learn.
- **Dukungan Input/Output (I/O) yang Luas**: Pandas mendukung pembacaan dan penulisan data dari berbagai format file, termasuk CSV, Excel, database SQL, JSON, dan HDF5.
- **Fungsionalitas Deret Waktu**: Pandas memiliki fungsionalitas khusus untuk analisis deret waktu, seperti pembuatan rentang tanggal, konversi frekuensi, statistik jendela bergerak (moving window statistics), dan pergeseran tanggal.
- **Komunitas yang Besar dan Aktif**: Sebagai pustaka yang populer, Pandas memiliki komunitas pengguna dan pengembang yang besar dan aktif, yang berarti banyak sumber daya, tutorial, dan dukungan yang tersedia.

MENGAPA PANDAS PENTING UNTUK BIG DATA

Meskipun Pandas dirancang untuk analisis data dalam memori (in-memory), yang berarti kinerjanya optimal untuk dataset yang muat dalam RAM, perannya tetap penting dalam konteks data besar (big data). Berikut adalah beberapa alasannya:

- **Eksplorasi dan Prototyping Awal**: Sebelum menerapkan solusi data besar yang lebih kompleks, ilmuwan data sering menggunakan Pandas untuk menganalisis sampel data atau subset dari dataset besar. Ini memungkinkan eksplorasi data yang cepat dan pembuatan prototipe alur kerja analisis.
- **Teknik Optimasi Memori**: Pandas menyediakan beberapa teknik untuk mengoptimalkan penggunaan memori saat bekerja dengan dataset yang lebih besar dari biasanya. Ini termasuk memuat hanya kolom yang diperlukan, menggunakan tipe data yang lebih efisien (misalnya, **category** untuk data teks berulang, atau downcasting tipe numerik), dan memproses data dalam potongan (chunks).
- **Integrasi dengan Alat Big Data**: Pandas dapat diintegrasikan dengan alat pemrosesan data terdistribusi seperti Dask dan Apache Spark.
    - **Dask**: Dask menyediakan DataFrame paralel yang meniru API Pandas, memungkinkan pengguna untuk bekerja dengan dataset yang lebih besar dari memori menggunakan sintaks Pandas yang sudah dikenal. Dask mempartisi DataFrame Pandas menjadi beberapa bagian dan melakukan komputasi secara paralel.
    - **Apache Spark**: Dengan pembaruan Spark 3.2, Pandas API kini dapat dijalankan di atas Spark (melalui Koalas atau pandas API on Spark), memungkinkan pengguna memanfaatkan kekuatan pemrosesan terdistribusi Spark sambil menggunakan sintaks Pandas yang familiar.
- **Fleksibilitas untuk Pra-pemrosesan**: Bahkan dalam pipeline data besar, Pandas sering digunakan untuk langkah-langkah pra-pemrosesan dan pembersihan data pada node individual atau pada subset data sebelum digabungkan atau dianalisis lebih lanjut menggunakan kerangka kerja data besar.

Namun, penting untuk dicatat bahwa Pandas sendiri memiliki keterbatasan dalam menangani dataset yang sangat besar (misalnya, terabyte hingga petabyte) karena sifatnya yang berbasis memori. Dalam skenario seperti itu, alat seperti Spark atau Dask menjadi lebih cocok, meskipun Pandas masih dapat memainkan peran pendukung.

STRUKTUR DATA UTAMA PANDAS

Pandas memiliki dua struktur data utama yang menjadi dasar fungsionalitasnya:

1. **Series**:
    a. **Definisi**: Series adalah array berlabel satu dimensi yang mampu menampung semua jenis data (integer, string, float, objek Python, dll.). Setiap elemen dalam Series memiliki label yang disebut indeks.
    b. **Analogi**: Anda dapat menganggap Series sebagai satu kolom dalam spreadsheet atau tabel database.
    c. **Pembuatan**: Series dapat dibuat dari list Python, array NumPy, atau dictionary.

d. **Contoh**:

```
import pandas as pd

data_list = [10, 20, 30, 40, 50]

ser_from_list = pd.Series(data_list)

ser_from_list
```

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

```
data_dict = {'a': 1, 'b': 2, 'c': 3}

ser_from_dict = pd.Series(data_dict)

ser_from_dict
```

```
a    1
b    2
c    3
dtype: int64
```

2. **DataFrame**:

- **Definisi**: DataFrame adalah struktur data tabular dua dimensi dengan sumbu berlabel (baris dan kolom). DataFrame dapat menampung kolom-kolom dengan tipe data yang berbeda.
- **Analogi**: DataFrame mirip dengan spreadsheet Excel, tabel SQL, atau dictionary dari Series.
- **Komponen**: Terdiri dari data aktual, indeks baris, dan label kolom.
- **Pembuatan**: DataFrame dapat dibuat dari berbagai sumber, termasuk list, dictionary, Series, array NumPy, DataFrame lain, atau file eksternal (seperti CSV atau Excel).

- **Contoh**:

```
import pandas as pd

data = {'Nama': ['Alice', 'Bob', 'Charlie'],
        'Usia': [25, 30, 22],
        'Kota': ['Jakarta', 'Bandung', 'Surabaya']}

df = pd.DataFrame(data)

df
```

	Nama	Usia	Kota
0	Alice	25	Jakarta
1	Bob	30	Bandung
2	Charlie	22	Surabaya

MEMULAI DENGAN PANDAS

1. Import Library Pandas

```
import pandas as pd
import numpy as np    # Sering digunakan bersama pandas
```

2. Membuat Series
    a. Dari list python

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])

s
```

b. Dengan indeks kustom:

```
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])

s
```

3. Membuat DataFrame
   a. Dari dictionary python (keys menjadi nama kolom)

```
data = {'Nama': ['Alice', 'Bob', 'Charlie'],
        'Usia': [20, 21, 22]}

df = pd.DataFrame(data)

df
```

	Nama	Usia
0	Alice	20
1	Bob	21
2	Charlie	22

   b. Dari list of lists dengan nama kolom

```
data_list = [['Alice', 20], ['Bob', 21], ['Charlie', 22]]

df = pd.DataFrame(data_list, columns=['Nama', 'Usia'])

df
```

	Nama	Usia
0	Alice	20
1	Bob	21
2	Charlie	22

c. Membaca data dari file eksternal

```
df = pd.read_csv('path_file')    # Misal ingin membaca csv

df
```

INSPEKSI DATA

1. Melihat data awal dan akhir

```
df.head(n)    # Menampilkan 'n' baris pertama (default 5)
```

```
df.tail(n)    # Menampilkan 'n' baris terakhir (default 5)
```

2. Dimensi DataFrame

```
df.shape   # Mengembalikan tuple (jumlah baris, jumlah kolom)
```

3. Informasi Umum DataFrame

```
df.info()   # Memberikan ringkasan DataFrame, termasuk tipe data per kolom dan penggunaan memori
```

4. Statistik Deskriptif

```
df.describe()   # Menghasilkan statistik deskriptif (count, mean, std, min, max, kuartil) untuk kolom numerik
```

```
df.describe(include='all')   # Untuk menyertakan kolom non-numerik
```

5. Tipe Data Kolom

```
df.dtypes   # Menampilkan tipe data dari setiap kolom
```

6. Nilai Unik dan Frekuensi

```
df['nama_kolom'].unique()   # Menampilkan nilai unik dalam satu kolom
```

```
df['nama_kolom'].nunique()   # Menghitung jumlah nilai unik dalam satu kolom
```

```
df['nama_kolom'].value_counts()   # Menghitung frekuensi kemunculan setiap nilai unik
```

SELEKSI DAN PENGINDEKSAN DATA

1. Seleksi Kolom

```
df['nama_kolom']  # Menghasilkan satu kolom (Series)
```

```
df.nama_kolom  # Jika nama kolom valid sebagai identifier python
```

```
df[['kolom1', 'kolom2']]  # Menghasilkan DataFrame
```

2. Seleksi Baris

```
df.loc[label_baris]  # Menggunakan .loc[] (berbasis label/indeks)
```

```
df.loc[label_baris_awal:label_baris_akhir]  # Slicing inklusif
```

```
df.iloc[posisi_baris]  # Menggunakan .iloc[] (berbasis posisi integer)
```

```
df.iloc[posisi_baris_awal:posisi_baris_akhir]  # Slicing eksklusif
```

3. Seleksi Baris dan Kolom Bersamaan

```
df.loc[label_baris, ['kolom1', 'kolom2']]  # Seleksi baris dan kolom bersamaan
```

```
df.iloc[posisi_baris, [posisi_kolom1, posisi_kolom2]]  # Seleksi baris dan kolom
```

4. Seleksi Kondisional (Boolean Indexing)

```
df[df['nama_kolom'] > nilai]  # Boolean indexing
```

```
df[(df['kolom1'] > nilai1) & (df['kolom2'] == 'kategori')]  # gunakan & untuk AND, | untuk OR
```

OPERASI DASAR PADA DATA

1. Menambahkan Kolom Baru

```
df['kolom_baru'] = df['kolom_lama'] * 2 # operasi ditentukan pengguna
```

```
df['kolom_baru_skalar'] = 'nilai_default'
```

2. Menghapus Kolom atau Baris

```
df.drop('nama_kolom', axis=1, inplace=False)   # (axis=1 untuk kolom)
```

```
df.drop(label_baris, axis=0, inplace=False)   # (axis=0 untuk baris)
```

3. Mengganti Nama Kolom

```
df.rename(columns={'nama_lama': 'nama_baru'}, inplace=False)
```

4. Mengganti Nilai yang Hilang (Missing value atau NaN)

```
df.isnull().sum()  # (menghitung jumlah NaN per kolom)
```

```
df.isna().sum()  # (menghitung jumlah NaN per kolom)
```

```
df.dropna(axis=0)  # (hapus baris dengan NaN)
```

```
df.dropna(axis=1)  # (hapus kolom dengan NaN)
```

```
df.fillna(value)  # (mengisi semua NaN dengan value)
```

```
df['nama_kolom'].fillna(df['nama_kolom'].mean(), inplace=True)

# (mengisi NaN di kolom tertentu dengan mean kolom tersebut)
```

5. Mengurutkan Data

```
df.sort_values(by='nama_kolom', ascending=True)
```

```
df.sort_values(by=['kolom1', 'kolom2'], ascending=[True, False])
```

```
df.sort_index(ascending=False)
```

GROUPING DAN AGREGASI

1. Konsep groupby()
    a. Memecah data menjadi grup berdasarkan kriteria tertentu
    b. Menerapkan fungsi agregasi pada setiap grup
    c. Menggabungkan hasil menjadi struktur data baru
2. Operasi

```
grouped = df.groupby('nama_kolom_kategori')
```

3. Fungsi Agregasi Dasar

```
grouped.sum()  # Summasion
```

```
grouped.mean()  # Mean
```

```
grouped.count()  # Menghitung
```

```
grouped.min()  # Nilai Minimum
```

```
grouped.max()  # Nilai Maximum
```

```
grouped.std()  # Nilai Standard Deviation
```

4. Agregasi dengan Beberapa Fungsi

```
df.groupby('nama_kolom_kategori').agg(['sum', 'mean', 'count'])
```

5. Agregasi pada Beberapa Kolom

```
df.groupby('nama_kolom_kategori').agg({'kolom_numerik1': 'sum', 'kolom_numerik2': 'mean'})
```

6. Grouping degan Beberaoa Kolom

```
df.groupby(['kolomm_grup1', 'kolom_grup2']).mean()
```

INPUT/OUTPUT DATA

1. Membaca Data dari File (Review)

```
# Membaca CSV

pd.read_csv('file_path')
```

```
# Membaca Excel

pd.read_excel('file_path')
```

2. Menyimpan DataFrame ke File

```
# Menyimpan file ke CSV

df.to_csv('output_file.csv')
```

```
# Menyimpan file ke Excel

df.to_excel('output_file.xlsx')
```

STUDI KASUS LATIHAN PRAKTIKUM

Untuk latihan praktikum tersedia pada link berikut:

https://colab.research.google.com/drive/1_I8pL0juzwFZuueSp7Amy-7yZQAbWKmE?usp=sharing