# Pandas

October 11, 2024

An In-Depth Tutorial of the Pandas Framework

Pandas is a powerful and flexible open-source data analysis and manipulation library for Python. It provides high-performance data structures and tools for working with structured (tabular, multidimensional, potentially heterogeneous) and time-series data. If you're working with data in Python, pandas is your go-to library.

This tutorial aims to provide a comprehensive overview of pandas, covering its key features, data structures, and functionalities, complete with code examples to help you get started.

## 1 1. Getting Started with Pandas

Before diving into pandas, ensure you have it installed and know how to import it in your Python environment.

### 1.1 Installation

If you don't have pandas installed, you can install it using pip:

pip install pandas

Or, if you're using Anaconda:

conda install pandas

### 1.2 Importing Pandas

It's customary to import pandas under the alias pd:

[ ]:
```
import pandas as pd
```

## 2 2. Pandas Data Structures

Pandas introduces two new data structures to Python: Series and DataFrame.

### 2.1 Series

A Series is a one-dimensional labeled array capable of holding any data type.

[2]:
```
import pandas as pd
import numpy as np # another python library
```

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

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

### 2.2 DataFrame

A DataFrame is a two-dimensional labeled data structure with columns of potentially different types.

[3]:
```
import pandas as pd
import numpy as np

dates = pd.date_range("20210101", periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
print(df)
```

```
                   A         B         C         D
2021-01-01 -0.685007 -0.801989 -1.037318 -0.202605
2021-01-02 -0.646656 -0.089786  0.406266  0.603260
2021-01-03 -0.705890 -0.701637 -1.304500 -0.859583
2021-01-04  0.920052  0.905505  0.967911 -0.176353
2021-01-05  1.090478 -1.553111  0.549579 -1.141632
2021-01-06 -0.178044 -1.044312 -1.123455  2.668283
```

## 3 Creating Data Structures

### 3.1 From Lists and Dictionaries

#### 3.1.1 Series from List:

[8]:
```
s = pd.Series([1, 3, 5, 7, 9])
print(s)
```

```
0    1
1    3
2    5
3    7
4    9
dtype: int64
```

#### 3.1.2 DataFrame from Dictionary:

[6]:
```
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 27, 22, 32],
    'City': ['New York', 'Paris', 'London', 'Berlin']
}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age      City
0    Alice   24  New York
1      Bob   27     Paris
2  Charlie   22    London
3    David   32    Berlin
```

#### 3.1.3 From NumPy Arrays

[7]:
```
arr = np.array([[1, 2], [3, 4], [5, 6]])
df = pd.DataFrame(arr, columns=['Column1', 'Column2'])
print(df)
```

```
   Column1  Column2
0        1        2
1        3        4
2        5        6
```

### 3.2 Reading Data from Files

Pandas can read data from various file formats, including CSV, Excel, JSON, SQL, etc.

#### 3.2.1 Reading CSV:

[ ]:
```
df = pd.read_csv('data.csv') # Use actual path to file trying to be read
```

#### 3.2.2 Reading Excel:

[ ]:
```
df = pd.read_excel('data.xlsx', sheet_name='Sheet1')
```

## 4 4. Data Selection and Indexing

Understanding how to select data from pandas data structures is crucial.

### 4.1 Selecting Data in Series

[ ]:
```
s = pd.Series([1, 3, 5, 7, 9], index=['a', 'b', 'c', 'd', 'e'])

# Access by label
```

```
print(s['c'])  # Output: 5

# Access by integer location
print(s[2])    # Output: 5

# Slicing
print(s['b':'d'])  # Output: Series with indices 'b', 'c', 'd'
```

### 4.2 Selecting Data in DataFrames

#### 4.2.1 Selecting Columns:

[ ]:
```
print(df['Name'])  # Returns a Series
print(df[['Name', 'Age']])  # Returns a DataFrame
```

#### 4.2.2 Selecting Rows by Label with loc:

[ ]:
```
print(df.loc[0])  # First row
print(df.loc[0:2])  # Rows from index 0 to 2
```

#### 4.2.3 Selecting Rows by Position with iloc:

[ ]:
```
print(df.iloc[0])  # First row
print(df.iloc[0:2])  # Rows at positions 0 to 1
```

#### 4.2.4 Boolean Indexing:

[ ]:
```
print(df[df['Age'] > 25])
```

## 5 5. Data Manipulation

### 5.1 Handling Missing Data

#### 5.1.1 Detecting Missing Data:

[ ]:
```
df.isnull()  # Returns a DataFrame of booleans
```

#### 5.1.2 Dropping Missing Data:

[ ]:
```
df.dropna()  # Drops any rows with missing data
```

#### 5.1.3 Filling Missing Data:

[ ]:
```
df.fillna(value=0)  # Replaces missing data with 0
```

### 5.2 Data Transformation

#### 5.2.1 Applying Functions:

[12]:
```
# Apply a NumPy function
df = pd.DataFrame(np.random.randn(6, 4), columns=list('ABCD'))
print(df.apply(np.abs))
```

```
          A         B         C         D
0  1.037090  0.006674  0.132463  1.321959
1  1.031727  1.103459  0.347871  0.667284
2  0.226428  0.023311  0.501185  2.010616
3  0.696185  0.694298  0.212074  0.057956
4  1.673134  0.150017  1.551994  0.238564
5  0.492555  2.491170  0.619272  0.991288
```

#### 5.2.2 Applying Custom Functions:

[15]:
```
# Apply a lambda function to each column
df['E'] = df['A'].apply(lambda x: x ** 2)
```

### 5.3 Sorting and Ranking

#### 5.3.1 Sorting by Label:

[14]:
```
df.sort_index(axis=0, ascending=False)  # Sort rows descending
```

[14]:
```
           A         B         C         D         E
5 -0.492555 -2.491170  0.619272 -0.991288  0.242611
4 -1.673134 -0.150017 -1.551994  0.238564  2.799378
3  0.696185  0.694298  0.212074  0.057956  0.484673
2  0.226428 -0.023311  0.501185  2.010616  0.051270
1 -1.031727  1.103459  0.347871 -0.667284  1.064461
0 -1.037090  0.006674 -0.132463  1.321959  1.075555
```

#### 5.3.2 Sorting by Values:

[ ]:
```
df.sort_values(by='Age')
```

## 6 6. Merging and Joining DataFrames

Pandas provides several functions for combining DataFrames.

### 6.1 Concatenation

[18]:
```
df1 = pd.DataFrame({'A': ['A0', 'A1'], 'B': ['B0', 'B1']})
df2 = pd.DataFrame({'A': ['A2', 'A3'], 'B': ['B2', 'B3']})

result = pd.concat([df1, df2])
```

```
print(result)
```

```
    A   B
0  A0  B0
1  A1  B1
0  A2  B2
1  A3  B3
```

### 6.2 Merging on Keys

[19]:
```
left = pd.DataFrame({'key': ['K0', 'K1'], 'A': ['A0', 'A1']})
right = pd.DataFrame({'key': ['K0', 'K2'], 'B': ['B0', 'B2']})

merged = pd.merge(left, right, on='key', how='inner')
print(merged)
```

```
  key   A   B
0  K0  A0  B0
```

## 7 7. GroupBy Operations

GroupBy allows you to split data into groups, apply a function to each group independently, and then combine the results.

### 7.1 Aggregation

[20]:
```
df = pd.DataFrame({
    'Category': ['A', 'A', 'B', 'B'],
    'Data': [1, 2, 3, 4]
})

grouped = df.groupby('Category')
print(grouped.sum())
```

```
          Data
Category
A            3
B            7
```

### 7.2 Filtering, Transformation, and Applying

#### 7.2.1 Filtering Groups:

[21]:
```
grouped.filter(lambda x: x['Data'].sum() > 4)
```

[21]:
```
  Category  Data
2        B     3
3        B     4
```

### 7.2.2 Transforming Data:

[26]:
```
df['Transformed'] = grouped.transform(lambda x: x - x.mean())
print(df['Transformed'])
```

```
0   -0.5
1    0.5
2   -0.5
3    0.5
Name: Transformed, dtype: float64
```

### 7.2.3 Applying Functions:

[23]:
```
grouped.apply(lambda x: x.max() - x.min())
```

[23]:

| Category | Data |
|---|---|
| A | 1 |
| B | 1 |

# 8 Pivot Tables and Cross-tabulations