Pandas Dataframes

een kennismaking

Wat is een pandas dataframe?

Een vorm van dataopslag / presentatie / bewerkingen, vergelijkbaar met een spreadsheet.

Enkele eigenschappen:

- 2 dimensionaal (rijen en kolommen)
- header (“eerste” rij)
- index (“eerste” kolom), default start = 0

Een voorbeeld

	Stad	Provincie	Aantal inwoners
0	Nijmegen	Gelderland	182000
1	Arnhem	Gelderland	176000
2	Groningen	Groningen	200000

- Rij/kolom intersectie: data (value). Kan van type int, string, boolean etc.. zijn
- Voorwaarde: geen verschillende data-typen binnen 1 kolom

Maken van een dataframe

Veel mogelijkheden:

- met python lists and dicts
- met pandas Series
- met ndarray (van “numpy”)
- met een csv bestand
- met een SQL query

Maken dataframe: met python lists

Vanuit een (python) “list van lists” met Dataframe functie:

```
data =
[['Nijmegen','Gelderland',182000],['Arnhem','Gelderland',176000],['Groningen','Groningen',200000]]


df = pd.DataFrame(data, columns=['Stad', 'Provincie','Aantal inwoners'])


print(df)

        Stad   Provincie  Aantal inwoners

0   Nijmegen  Gelderland           182000

1     Arnhem  Gelderland           176000

2  Groningen   Groningen           200000


df.index = df.index + 1
```

Maken dataframe: met python dict

Vanuit een (python) dictionary met Dataframe functie:

```
 data = {
          'Stad': ['Nijmegen','Arnhem','Groningen'],
          'Provincie': ['Gelderland','Gelderland','Groningen'],
          'Aantal inwoners': [182000,176000,200000]
        }
df = pd.DataFrame(data, columns=data.keys())
print(df)
```

```
        Stad   Provincie  Aantal inwoners
0   Nijmegen  Gelderland           182000
1     Arnhem  Gelderland           176000
2  Groningen   Groningen           200000
```

Maken dataframe: gebruik van een csv bestand (steden.csv)

```
$ cat steden.csv:
    Stad,Provincie,Aantal inwoners
    Nijmegen,Gelderland,182000
    Arnhem,Gelderland,176000
    Groningen,Groningen,200000
```

```
$ cat maak_df.py
    import pandas as pd
    df =
    pd.read_csv('steden.csv')
    print(df)
```

```
$ python3 maak_df.py
        Stad   Provincie  Aantal inwoners
0   Nijmegen  Gelderland           182000
1     Arnhem  Gelderland           176000
2  Groningen   Groningen           200000
```

Maken dataframe: met een SQL query

```
query = f'SELECT * FROM plants'

mycursor.execute(query)

result = mycursor.fetchall()

df = pd.DataFrame(result)

# Maak column namen

df.columns = [i[0] for i in mycursor.description]

# start index met 1

df.index = df.index + 1
```

Verkrijgen waarden

```
Toon column:
 df['Stad']
0     Nijmegen
1       Arnhem
2    Groningen


Toon row:
df[1:2]
     Stad   Provincie  Aantal inwoners
1  Arnhem  Gelderland           176000
```

Verkrijg *enkele* waarden - df.at / df.iat

```
df.at[0,'Stad']
    Nijmegen

df.at[2,’Aantal inwoners’]
    200000

df.iat[2,2]
    200000
df.at gaat uit van labels
df.iat gaat uit van integers
```

Verkrijg row, column waarden - df.loc

```
df.loc[:,'Stad']

            Stad

    0    Nijmegen

    1      Arnhem

    2   Groningen

df.loc[1:2,"Stad":"Aantal inwoners"]

              Stad   Provincie   Aantal inwoners

    1       Arnhem  Gelderland            176000

    2    Groningen   Groningen            200000
```

df.loc is *inclusive* en gaat uit van de labels: je kunt dus ook strings gebruiken.

Verkrijg row, column waarden - df.iloc

```
df.iloc[1:2,0:2]
```

```
             Stad   Provincie
1         Arnhem  Gelderland
```

df.iloc is *exclusive* en gaat uit van integer lokaties

Verkrijg waarden - df.query

```
df.query("`Aantal inwoners` > 150000 and `Aantal inwoners` < 200000")
```

	Stad	Provincie	Aantal inwoners
0	Nijmegen	Gelderland	182000
1	Arnhem	Gelderland	176000

Verkrijg waarden - head, tail

```
df.head(2)
```

	Stad	Provincie	Aantal inwoners
0	Nijmegen	Gelderland	182000
1	Arnhem	Gelderland	176000

```
df.tail(2)
```

	Stad	Provincie	Aantal inwoners
1	Arnhem	Gelderland	176000
2	Groningen	Groningen	200000

Verkrijg waarden - mean()

```
        A  B
     0  4  3
     1  8  7
>>> df.mean()      # default axis is row (0)
     A    6.0
     B    5.0
>>> df.mean(axis=1)
     0    3.5
     1    7.5
>>> df['Gemiddelde'] = df.mean(axis=1)
>>> df
        A  B  Gemiddelde
     0  4  3         3.5
     1  8  7         7.5
```

Verkrijg waarden - sum()

```
    A  B
0   4  3
1   8  7
>>> df.sum()       # default axis is row (0)
A    12
B    10
>>> df.sum(axis=1)
0    7
1   15
```

Maak wijziging permanent: inplace=True

```
>>> df
   A  B
0  4  5
1  8  1
>>> df.drop(['B'],axis=1)
   A
0  4
1  8
>>> df
   A  B
0  4  5
1  8  1
```

```
>>> df
   A  B
0  4  5
1  8  1
>>> df.drop(['B'],axis=1,inplace=True)
>>> df
   A
0  4
1  8
```

Samenvoegen van dataframes: pd.concat()

```
>>> df1
   A  B
0  6  4
1  8  2
>>> df2
   A  B  C
0  0  4  2
>>> pd.concat([df1,df2],axis=0, ignore_index=True) # voeg samen per row
   A  B    C
0  6  4  NaN
1  8  2  NaN
2  0  4  2.0
```

Verkrijg alle data: to_string()

print(df) geeft een *maximaal* aantal columns en rows in de output.

Als er meer rows en/of columns zijn wordt dit getoond met …

Dit kun je overigens beinvloeden met pd.set_options().

Om *alle* data te zien gebruik je df.to_string()

Drop columns

```
df.drop(columns="Provincie")

          Stad  Aantal inwoners
   0   Nijmegen           182000
   1     Arnhem           176000
   2  Groningen           200000

df.drop(columns=df.loc[:,"Provincie":"Aantal inwoners"]):

          Stad
   0   Nijmegen
   1     Arnhem
   2  Groningen
```

Drop rows

```
df.drop(df.index[2:])
```

```
       Stad   Provincie  Aantal inwoners
0  Nijmegen  Gelderland           182000
1    Arnhem  Gelderland           176000
```

Datetime index

```
date_series = pd.date_range(datetime(2023, 10, 10, hour=0, minute=0), periods=268, freq='5min')

num_series = np.random.randint(100, size=268)

date_df = pd.DataFrame({'datetime':pd.to_datetime(date_series),'values':num_series})

date_df2 = date_df.set_index('datetime')
```

NaN

Als een invoer veld “leeg” is krijgt deze in het dataframe automatisch de “waarde” NaN.

En hierop kun je checken, bv.:

```
if pd.isna(df.loc[row,column]):
```

```
(of pd.notna voor het omgekeerde effect)
```

Een praktijk voorbeeld

Twee csv files combineren:

- csv-file met inverter “measurement” definities (SC naam, Prov naam, string-nummer, modbus registernummer, factor, unit, active, …)
- csv-file (gezipped) met actuele (elke 5 minuten) modbus registernummers en de resp. waarden

Gemeenschappelijk: modbus registernummer

Idee: zoek voor modbus registernummers de bijbehorende actuele waarde en schrijf die samen met de juiste vars naar een .prom (Prometheus) file.

Bier?

(daarna oefeningen!)

Oefeningen

Download oefening.py

```
$ wget www.kwalinux.nl/lugn_134/oefening.py
```

Run in een tweede terminal:

```
python3 oefening.py setup
```

Volg nu de instructies op het scherm om je virtual environment op te zetten. Is je virtual environment opgezet dan ben je klaar om de oefeningen te doen.

Gebruik 1 terminal met de virtual environment en de python3 interpreter om te oefenen en in de andere terminal run je:

```
$ python3 oefening.py oefen | less -r
```

Succes!