Python, pandas & friends

Dr. Uwe Ziegenhagen

September 28, 2017

Python and SciPy

Data Handling with pandas

Various pandas Examples

Pivotizing

Examples

About Me

- Dipl.-Kfm, M. Sc. & Ph.D. in Statistics
- Seven years experience in Private Banking and Private Equity
- since October '15: Analyst Credit & Treasury Operations at IKB Industriebank in Düsseldorf
- LaTeX enthusiast for about 20 years
- Treasurer for "Dingfabrik Köln e.V.", Cologne's fablab & makerspace

Pandas

- my introduction to scientific Python: data consistency and completeness checks with `pandas`
- "pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language."[1]
- Initiated 2008 by Wes McKinney while at AQR Capital Management for high performance quantitative analysis
- Important parts implemented in C/Cython, quite fast
- Current version is 0.20.3

[1] Source: `pandas.pydata.org`

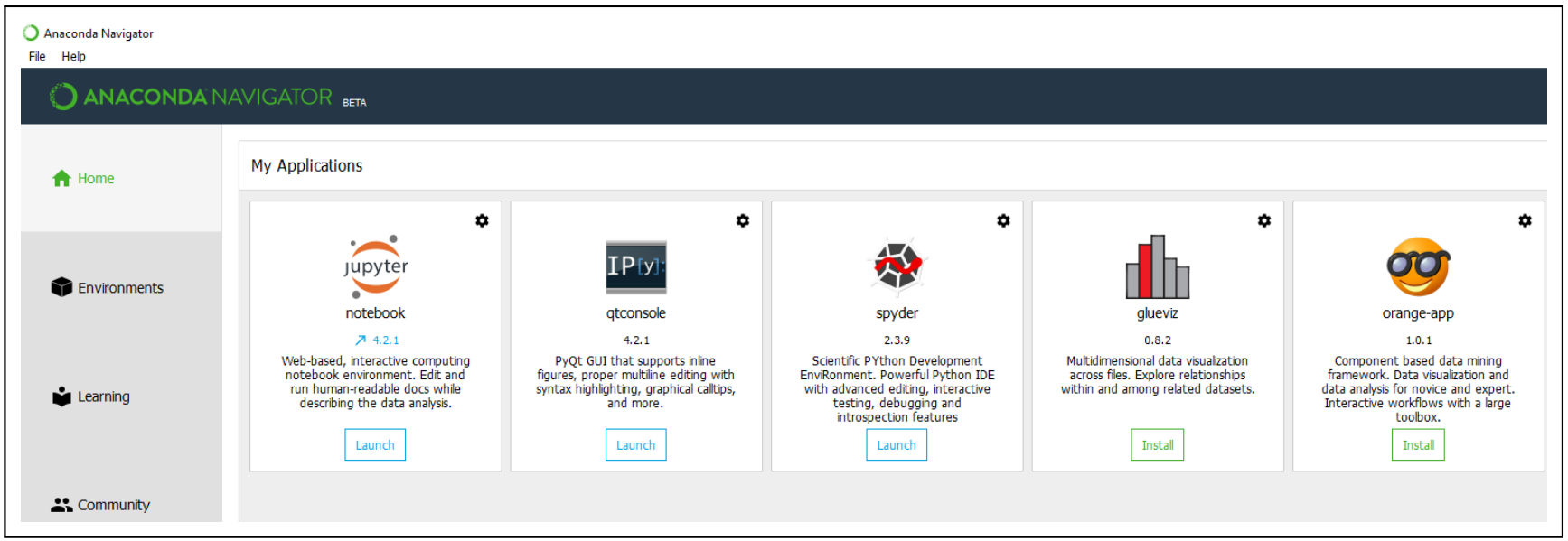
The SciPy Framework

Besides `pandas` there are

NumPy	matrices, vectors, algorithms
IPython	Matlab/Mathematica-like environment
Matplotlib	scientific plotting, basis for `seaborn` library
SymPy	symbolic mathematics
...	etc, etc

Scientific Python Distributions

- Linux/MacOS X bring Python, but not SciPy
- Install manually or use dedicated distribution
- personal recommendation: Anaconda
  - WinPython (`https://winpython.github.io`)
  - Anaconda (`https://www.continuum.io/downloads`)



Series and DataFrames

- central data structures in `pandas`

Creating Series and Dataframes

- just for the sake of completeness: Pandas objects **can** also be created manually

```
1  import pandas as pd
2
3  d = pd.DataFrame({'A': ['A0','A1','A2','A3'],
4    'Key': ['K0','K1','K2','K4']})
5  a = pd.Series([1,2,3,4,5,6,7,8,9,10])
6  b = pd.Series(['A','C','D','B','F','G','I','K','L','P'])
7  df = pd.concat([a,b], axis=1)
8  # alternativ
9  df = pd.DataFrame({'a': a,'b':b})
10 df = a.to_frame().join(b.to_frame())
11 df = pd.DataFrame(data=dict(a=a, b=b))
```

Reading Data

- manual creation of pandas objects not recommended except for testing purposes
- preferred way: read and write from data source

Command	Description
read_pickle	read Pickle objects
read_table	for general table-like formats
read_csv	Comma-Separated Values
read_fwf	for weird fixed-width formats
read_clipboard	read from clipboard
read_excel	read Excel files

other commands for HTML, JSON, HDF5, ...

Pandas Example I: Date conversion

- Assumption: Proprietary software uses **"14 Mar 1983"** as date format in CSV, German Excel understands it just "sometimes"
- Task:
  - Take the CSV data
  - Transform the "evil" dates and
  - Save the data in MS Excel format

```
1 import pandas as pd
2 data = pd.read_csv(somefile.csv)
3 data['datecol'] = pd.to_datetime(data['datecol'])
4 data.to_excel('somefile.xlsx')
```

Learning from the Example...

Reading Data

- `import pandas as pd`
  Load the pandas library
- `read_csv()`
  Load data in CSV format
- `pd.to_datetime(data['datecol'])`
  convert into Python `datetime` objects
- `to_excel()`
  save data in Excel format

Reading CSV

- CSV can be pretty "messed up":
  - column separator
  - decimal separator
  - text encoding
- see the specification:
  `http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html`

sep	specify the column separator
thousands	Thousands separator
encoding	hopefully UTF-8 (BOM!)
decimal	decimal separator
converters	converters={'A': str} for explicit conversion

Reading Excel

- Use `pd.read_excel()` to read XLSX files (which are just zipped XMLs)
- see the documentation: `http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_excel.html`
- to export to Excel use `pd.to_excel()` function
- remarks:
  - Excel export is much slower than CSV
  - Exporting "styled" Excel requires additional effort
  - Excel can be well controlled via COM (Common Object Model)

Querying DataFrames

Getting basic information

- For the next slides load `Northwind` data
- First task after loading data: do some sanity checks

```
1 customers = pd.read_excel('Northwind.xlsx',
2                           sheetname = 'Customers')
3
4 print(len(customers))   # Zeilenzahl
5 print(customers.head()) # die ersten 5 Zeilen
6 print(customers.tail()) # die letzten 5 Zeilen
7 print(list(customers)) # die Spaltennamen
```

Pandas Dataframe Operations

Selection and Filtering I

- pandas provides sophisticated methods to select, filter and transform rows and columns
- Select only certain columns

```
df = df[['colA', 'colB']]
```

- Select only first two rows (hint: index starts at 0)

```
df.iloc[:1]
```

- Select only rows where column value is greater

```
df[df['colA'] > 50]
```

Pandas Dataframe Operations

Selection and Filtering II

- Select only rows where column value is greater than 50 and smaller than 500

```
df[(df['colA'] > 50)| (df['colA'] < 500)]
```

- Select only rows where column value is not

```
df[~(df['colA'] == 'HelloWorld')]
```

- Select those rows, where column b is 'A' or 'B'

```
df = df[ (df['b'] == 'A')| (df['b'] == 'I')]
```

- more readable alternative via `isin()`

```
df = df[df['b'].isin(['A','I'])]
```

- or the opposite

```
df = df[~df['b'].isin(['A','I'])]
```

Pandas Dataframe Operations

Merging and Joining

- `merge()` provides SQL-like merging
- very handy to combine different datasets
- Supported are the followin join-types:
  - Left
  - Right
  - Inner
  - Full Outer
- `join()` is special alias for `merge()`, works on index, not columns (the default for merge)

Pandas Dataframe Operations

Merging and Joining

- Default-command for `merge()`

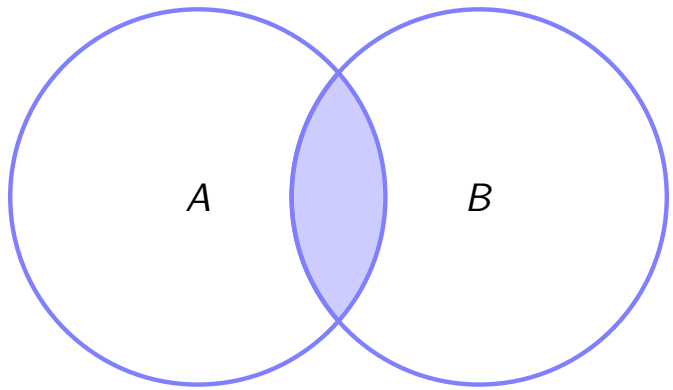
```
leftDataFrame.merge(rightDataFrame, how='inner',
on=None, left_on=None, right_on=None, left_index=False,
right_index=False, sort=False, suffixes=('_x', '_y'),
copy=True, indicator=False)
```

1. define the other DataFrame to merge
2. define how to merge
3. define the keys to use for the merge

Merging

Inner Join

- Select all data which is in A and B



left

	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

right

	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

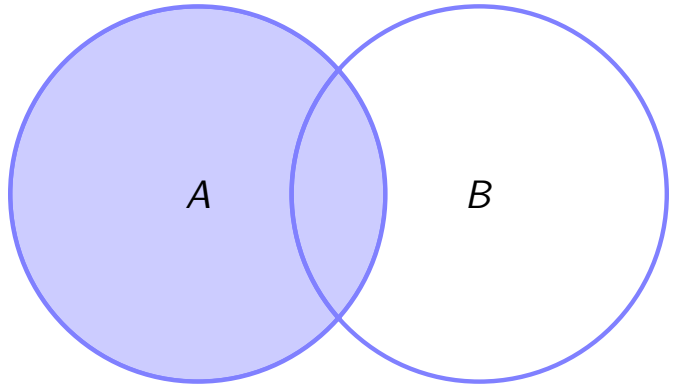
merged

	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2

Merging

Left Join

- Select all data which is in A and B



left

	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

right

	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

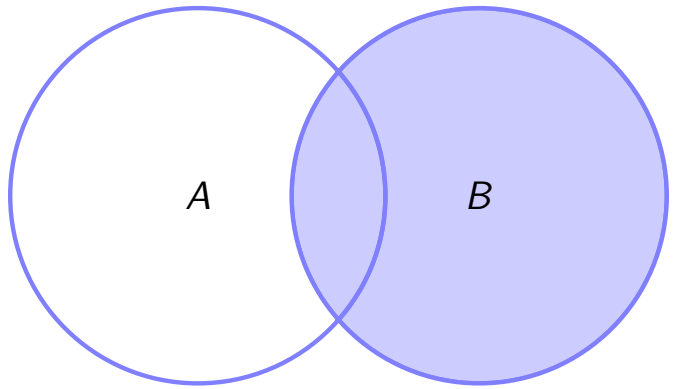
merged

	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	NaN	K4

Merging

Right Join

- Select all data which is in B



left

	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

right

	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

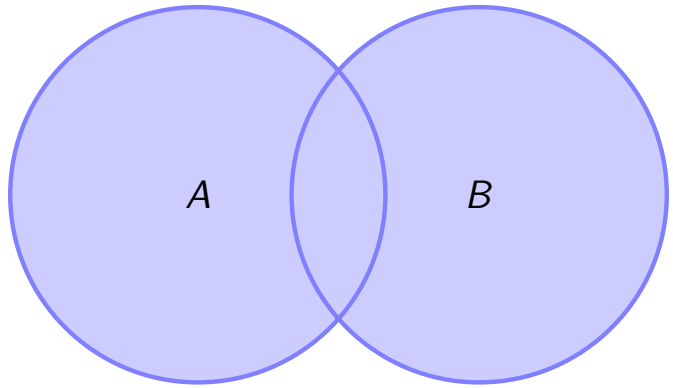
merged

	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	NaN	B3	K5

Merging

Full Outer Join

- Select all data which is in A or B



left

	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

right

	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

merged

	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	NaN	K4
4	NaN	B3	K5

Example: Merging Rows into Columns

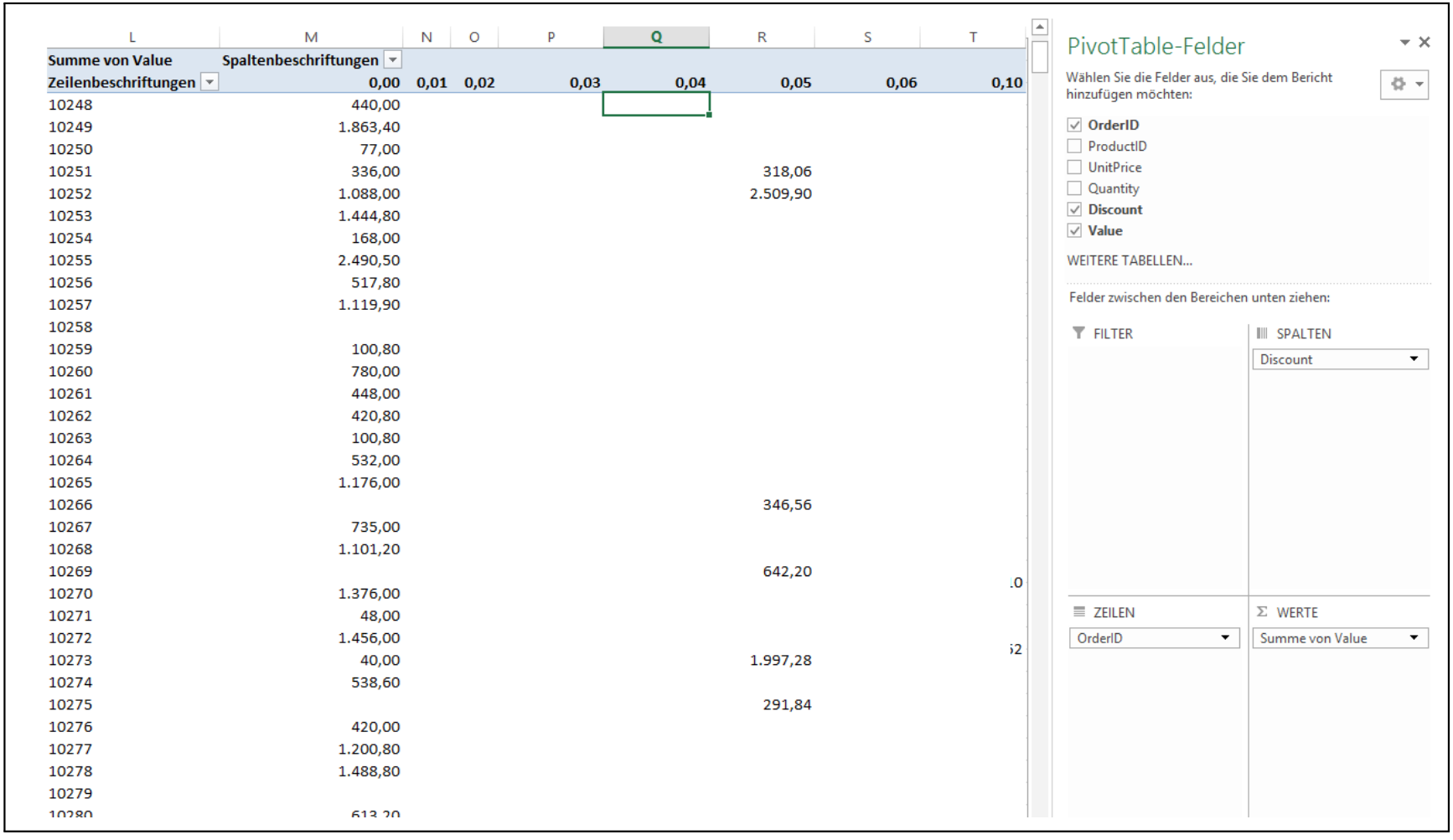
Spalte	Wert
ColA	Andi
ColB	Berni
ColC	Cesar
ColA	Dorian
ColB	Ernst
ColC	Frank

```
1 import pandas as pd
2 daten = pd.read_excel('combine.xlsx')
3 result = pd.DataFrame(columns=['ColA', 'ColB', 'ColC'])
4 for i, row in daten.iterrows():
5     result.loc[i // 3, row['Spalte']] = row['Wert']
6
7 print(result)
```

Pivotizing data

	A	B	C	D	E	F
1	OrderID	ProductID	UnitPrice	Quantity	Discount	Value
2	10248	11	14,00	12	0	168,00
3	10248	42	9,80	10	0	98,00
4	10248	72	34,80	5	0	174,00
5	10249	14	18,60	9	0	167,40
6	10249	51	42,40	40	0	1.696,00
7	10250	41	7,70	10	0	77,00
8	10250	51	42,40	35	0,15	1.261,40
9	10250	65	16,80	15	0,15	214,20
10	10251	22	16,80	6	0,05	95,76
11	10251	57	15,60	15	0,05	222,30
12	10251	65	16,80	20	0	336,00
13	10252	20	64,80	40	0,05	2.462,40
14	10252	33	2,00	25	0,05	47,50
15	10252	60	27,20	40	0	1.088,00
16	10253	31	10,00	20	0	200,00
17	10253	39	14,40	42	0	604,80
18	10253	49	16,00	40	0	640,00

Pivotizing data



Summe von Value	Spaltenbeschriftungen							
Zeilenbeschriftungen	0,00	0,01	0,02	0,03	0,04	0,05	0,06	0,10
10248	440,00							
10249	1.863,40							
10250	77,00							
10251	336,00					318,06		
10252	1.088,00					2.509,90		
10253	1.444,80							
10254	168,00							
10255	2.490,50							
10256	517,80							
10257	1.119,90							
10258								
10259	100,80							
10260	780,00							
10261	448,00							
10262	420,80							
10263	100,80							
10264	532,00							
10265	1.176,00							
10266						346,56		
10267	735,00							
10268	1.101,20							
10269						642,20		
10270	1.376,00							
10271	48,00							
10272	1.456,00							
10273	40,00					1.997,28		
10274	538,60							
10275						291,84		
10276	420,00							
10277	1.200,80							
10278	1.488,80							
10279								
10280	613,20							

PivotTable-Felder

Wählen Sie die Felder aus, die Sie dem Bericht hinzufügen möchten:

- [x] OrderID
- [ ] ProductID
- [ ] UnitPrice
- [ ] Quantity
- [x] Discount
- [x] Value

WEITERE TABELLEN...

Felder zwischen den Bereichen unten ziehen:

FILTER | SPALTEN: Discount

ZEILEN: OrderID | WERTE: Summe von Value

Pivotizing data

```
1
2 ordersdetails['Value'] = ordersdetails['UnitPrice'] *
      ordersdetails['Quantity'] * ordersdetails['
      DiscountFactor']
3
4 print(pd.pivot_table(ordersdetails, index='OrderID',values
      ='Value',columns='Discount',aggfunc=np.sum, margins=
      True))
```

index corresponds to Excel's **rows** ("Zeilen")

values corresponds to Excel's **values** ("Werte")

columns corresponds to Excel's **columns** ("Spalten")

aggfunc corresponds to the applied Excel function

margins corresponds to Excel's **totals** ("Summen")

Example: Creating Tax Donation Receipts

- Donations to Dingfabrik are tax-deductible
- Manual creation error-prone and labor-intensive
- Last year: complicated mix (Python, MySQL, LaTeX)
- This year: pandas, much easier
- Interested? `http://uweziegenhagen.de/?p=3359`

Example: Checking the Payment Status

- Treasurer task: check payments from Dingfabrik members
- Annoying job, lots of Excel "Mouse Schubsing"
- Idea: Analyze payment data with pandas, merge with master data
- Interested? `http://uweziegenhagen.de/?p=3350`