Data Wrangling and Data Analysis

Data Extraction in Python

Hakim Qahtan

Department of Information and Computing Sciences

Utrecht University

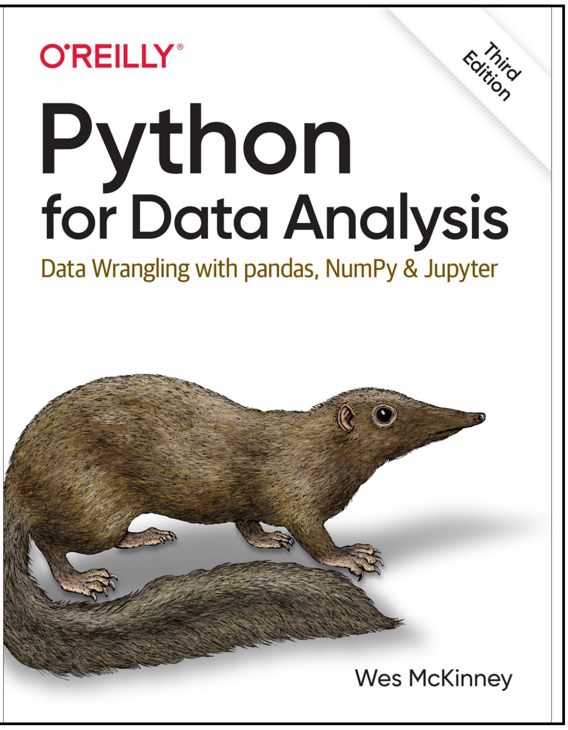
Utrecht University

1

Reading Material for Today

Python for Data Analysis, 3E

CH 5, and 6.1.



O'REILLY

Third Edition

Python for Data Analysis

Data Wrangling with pandas, NumPy & Jupyter

Wes McKinney

Utrecht University

2

- **So Far**
  - Data models and focused on the relational model
  - Creating and querying databases
  - Applying integrity constraints and ensuring effective database design
  - Data integration and similarity between atomic values and documents
- **Today**
  - Data manipulation in Python
  - Dataframes
  - Connecting to databases
  - Data profiling

3

Aren’t DBMSs Enough?

- DBMSs provides effective and efficient access for the data
- Performing data analysis is limited
- Programming languages (such as Python) provides libraries for wide range of data analysis techniques including different ML models

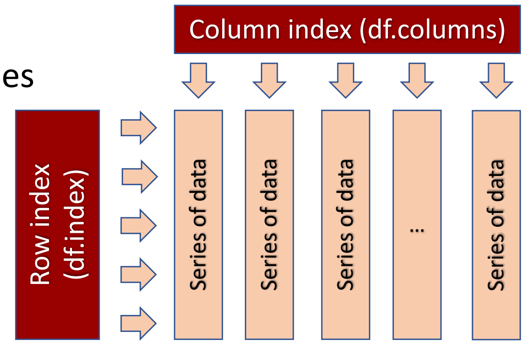
4

Data Extraction in Python

5

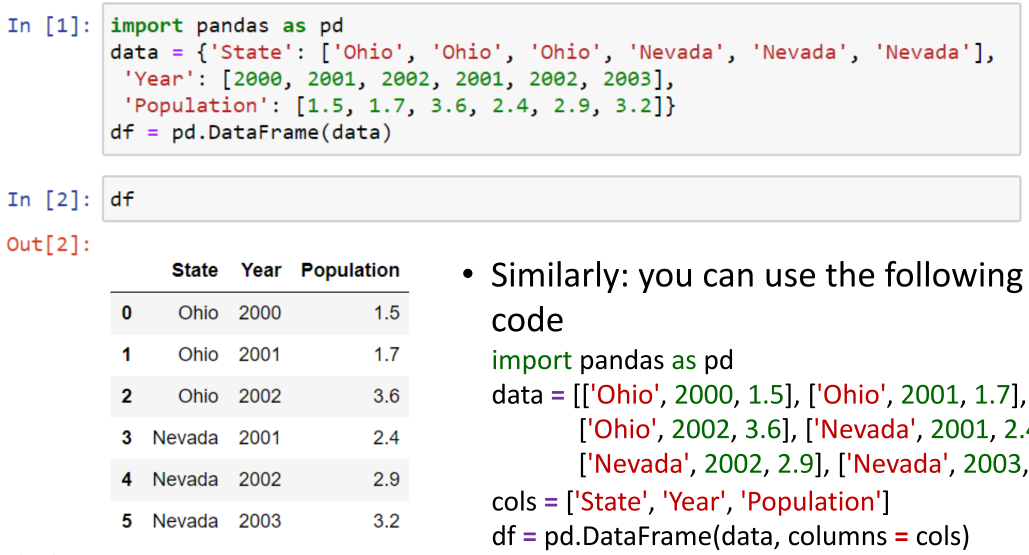
Pandas Dataframes

- The most popular way to handle data tables in Python is using Pandas dataframes
- DataFrame: a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.)
- Has columns and rows indexes
- Columns are made up of pandas series



6

Creating DataFrame



```
In [1]: import pandas as pd
        data = {'State': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
         'Year': [2000, 2001, 2002, 2001, 2002, 2003],
         'Population': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
        df = pd.DataFrame(data)

In [2]: df

Out[2]:
            State  Year  Population
        0    Ohio  2000         1.5
        1    Ohio  2001         1.7
        2    Ohio  2002         3.6
        3  Nevada  2001         2.4
        4  Nevada  2002         2.9
        5  Nevada  2003         3.2
```

- Similarly: you can use the following code

```
import pandas as pd
data = [['Ohio', 2000, 1.5], ['Ohio', 2001, 1.7],
        ['Ohio', 2002, 3.6], ['Nevada', 2001, 2.4],
        ['Nevada', 2002, 2.9], ['Nevada', 2003, 3.2]]
cols = ['State', 'Year', 'Population']
df = pd.DataFrame(data, columns = cols)
```

Utrecht University

Load DataFrame from CSV Files

- The simplest way is:

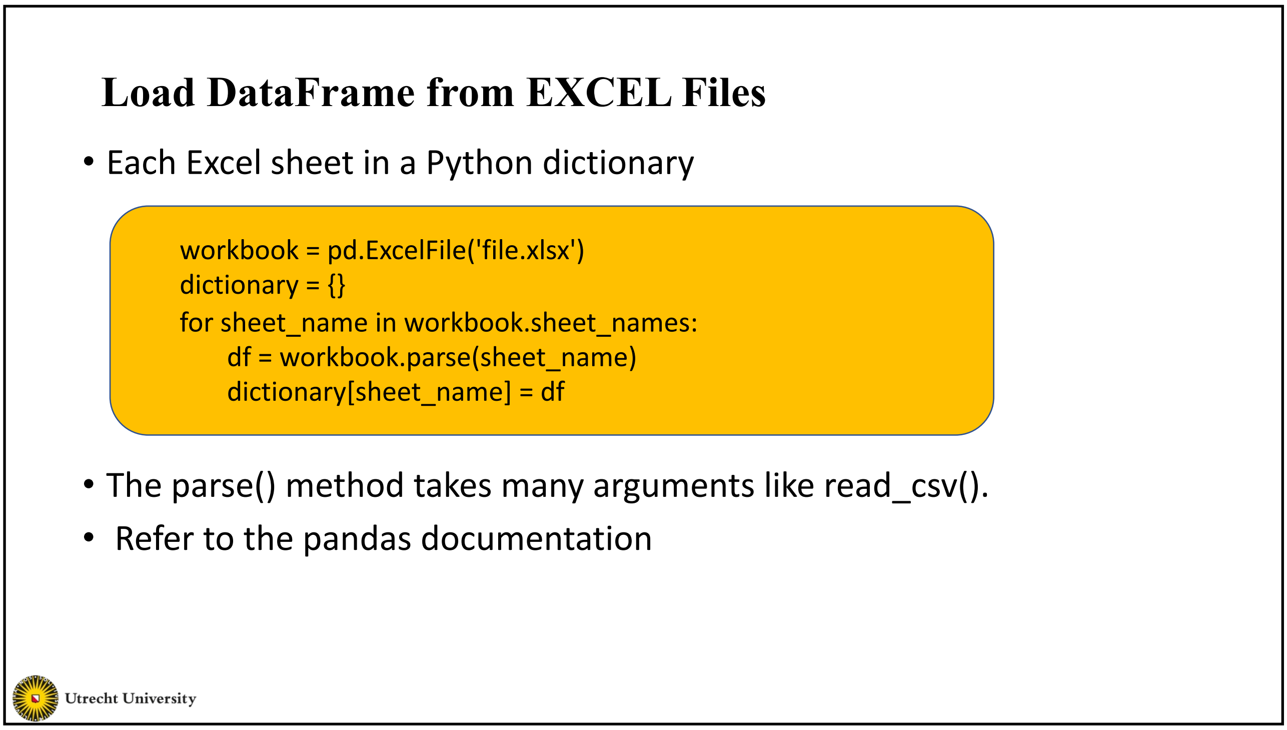
```
df = pd.read_csv('file.csv’)    # often works
```

- More options can be added when loading a csv file into a dataframe

```
df = pd.read_csv('movies.csv', header=0,
   index_col=0, quotechar='"',sep=",",
   na_values = ['na', '-', '.', ‘’])
```

- More options can be found in Pandas documentation
- Remeber to import the pandas library as pd

Utrecht University



Load DataFrame from EXCEL Files

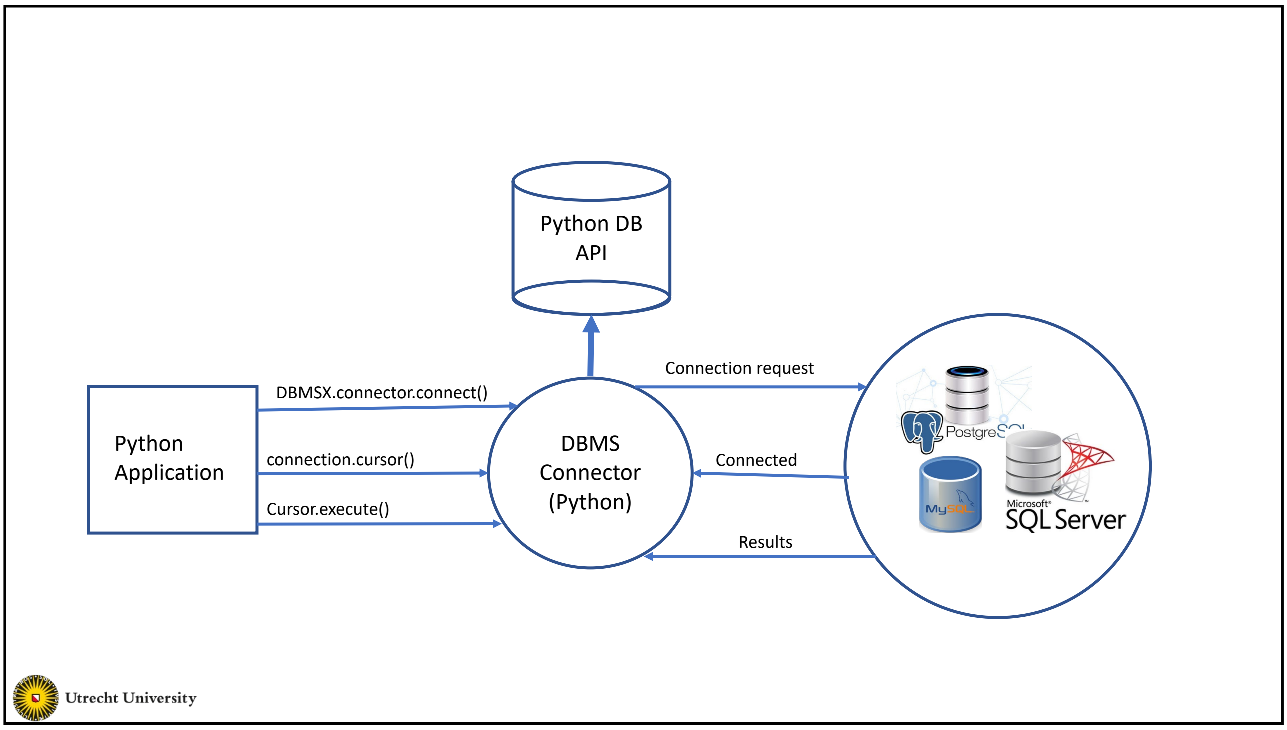
- Each Excel sheet in a Python dictionary

```
workbook = pd.ExcelFile('file.xlsx')
dictionary = {}
for sheet_name in workbook.sheet_names:
    df = workbook.parse(sheet_name)
    dictionary[sheet_name] = df
```

- The parse() method takes many arguments like read_csv().
- Refer to the pandas documentation

Utrecht University

9



Python DB API

Python Application

DBMSX.connector.connect()

connection.cursor()

Cursor.execute()

DBMS Connector (Python)

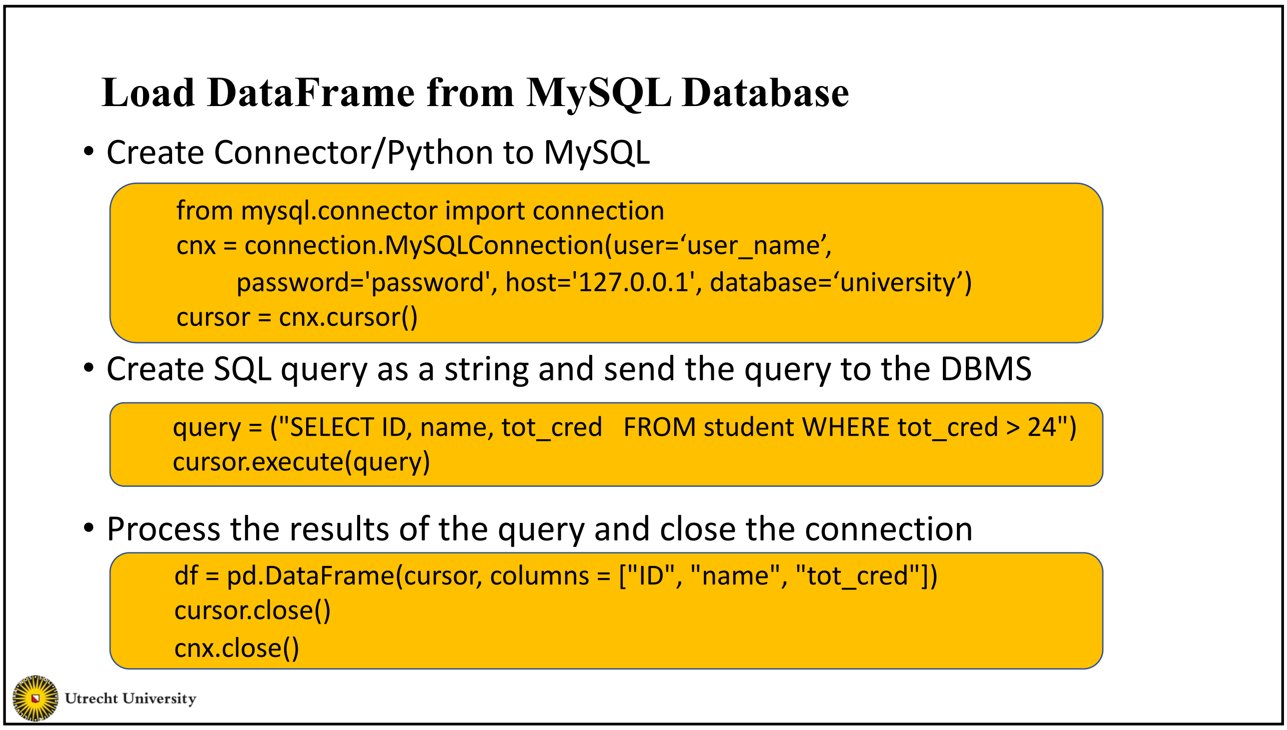
Connection request

Connected

Results

Utrecht University

10



Load DataFrame from MySQL Database

- Create Connector/Python to MySQL

```
from mysql.connector import connection
cnx = connection.MySQLConnection(user=‘user_name’,
        password='password', host='127.0.0.1', database=‘university’)
cursor = cnx.cursor()
```

- Create SQL query as a string and send the query to the DBMS

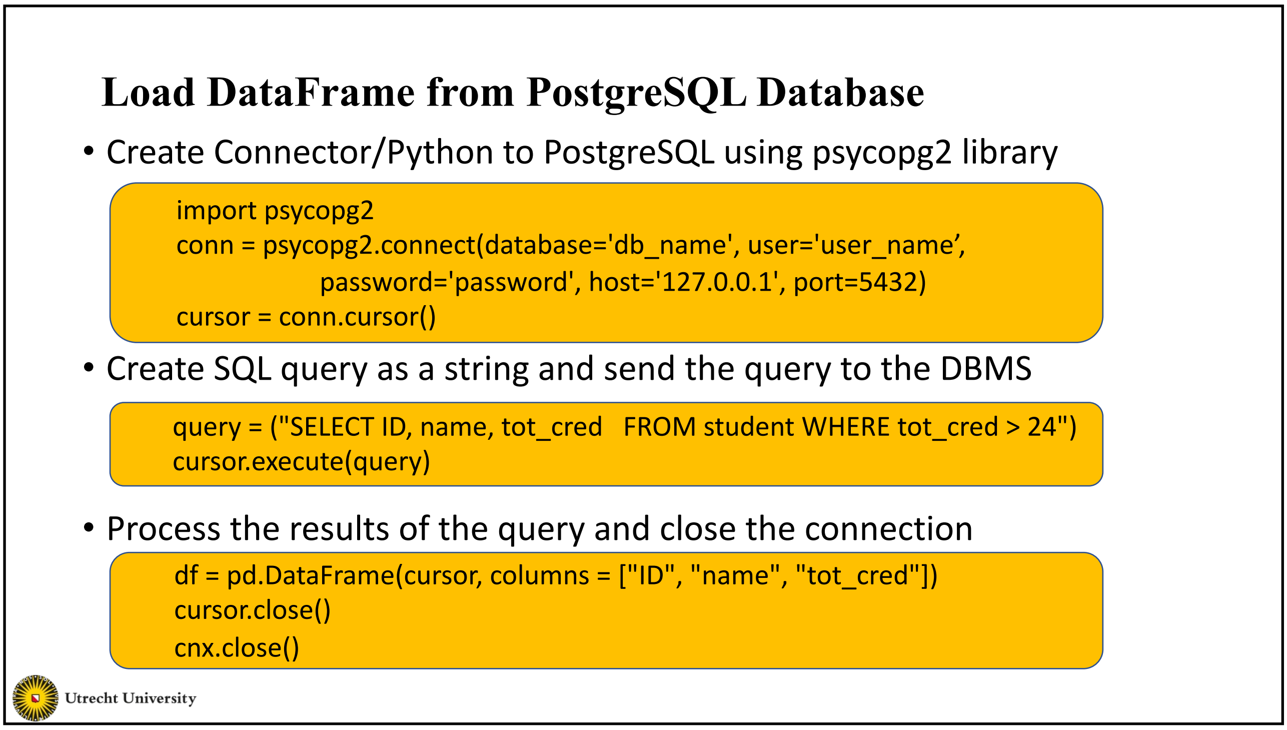
```
query = ("SELECT ID, name, tot_cred   FROM student WHERE tot_cred > 24")
cursor.execute(query)
```

- Process the results of the query and close the connection

```
df = pd.DataFrame(cursor, columns = ["ID", "name", "tot_cred"])
cursor.close()
cnx.close()
```

Utrecht University

11



Load DataFrame from PostgreSQL Database

- Create Connector/Python to PostgreSQL using psycopg2 library

```
import psycopg2
conn = psycopg2.connect(database='db_name', user='user_name’,
            password='password', host='127.0.0.1', port=5432)
cursor = conn.cursor()
```

- Create SQL query as a string and send the query to the DBMS

```
query = ("SELECT ID, name, tot_cred   FROM student WHERE tot_cred > 24")
cursor.execute(query)
```

- Process the results of the query and close the connection

```
df = pd.DataFrame(cursor, columns = ["ID", "name", "tot_cred"])
cursor.close()
cnx.close()
```

Utrecht University

12

Load DataFrame from SQLite Database

- Create Connector/Python to SQLite using sqlite3 library

```
import sqlite3
conn = sqlite3.connect(db_file)
cursor = conn.cursor()
```

- Create SQL query as a string and send the query to the DBMS
- Process the results of the query and close the connection
- Example

```
import sqlite3
conn = sqlite3.connect(``chinook.db”)
cursor = conn.cursor()
table = ``albums”
```

```
query = ``SELECT * FROM” + table
cur.execute(query)
rows = cur.fetchall()
  for row in rows:
    print(row)
```

Utrecht University

13

Working with Dataframes

- Consider the movies dataset extracted from imdb dataset
- Start by reading the csv file

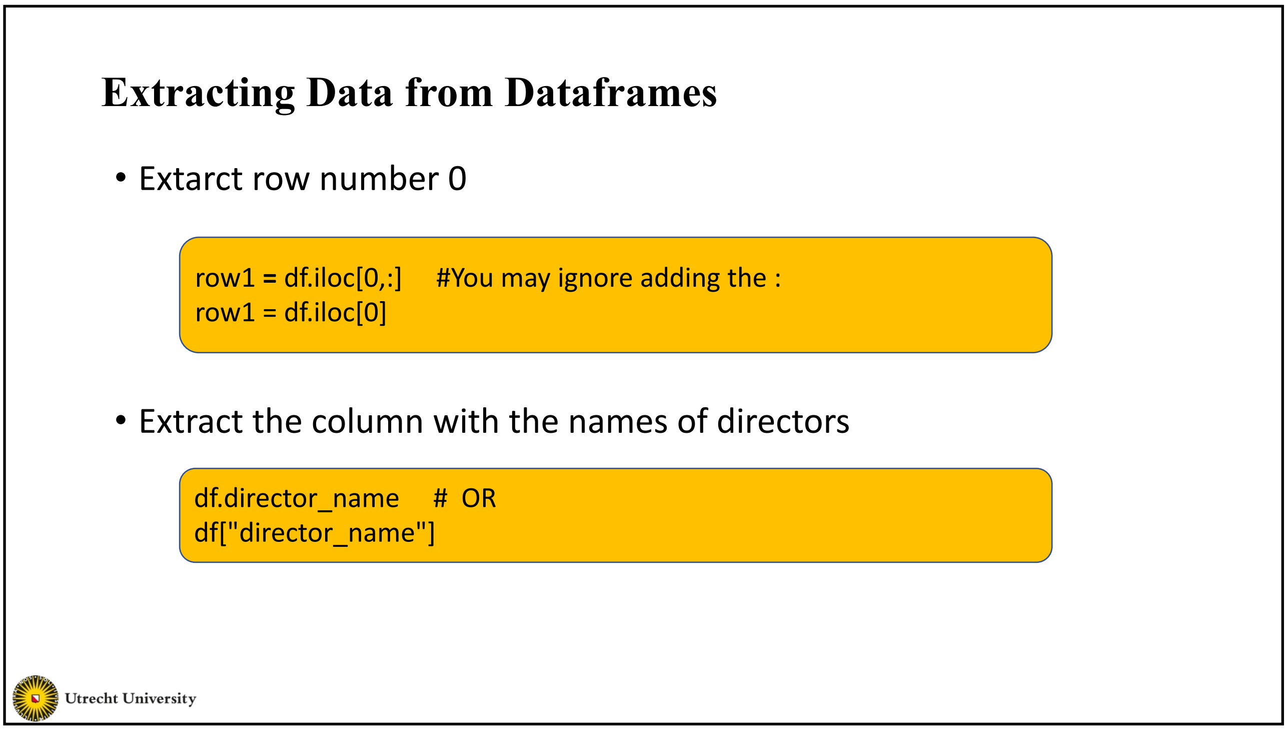
```
df = pd.read_csv(filepath_or_buffer = 'movies.csv', delimiter=',',
     doublequote=True, quotechar='"',na_values = ['na', '-', '.', ''],
     quoting=csv.QUOTE_ALL, encoding = "ISO-8859-1")
```

- Extract sub-table of the dataframe

```
df.info()               # index & data types
n = 4
dfh = df.head(n)        # get first n rows
dft = df.tail(n)        # get last n rows
dfs = df.describe()     # summary stats cols
top_left_corner_df = df.iloc[:5, :5]
```

Utrecht University

14



Extracting Data from Dataframes

- Extarct row number 0

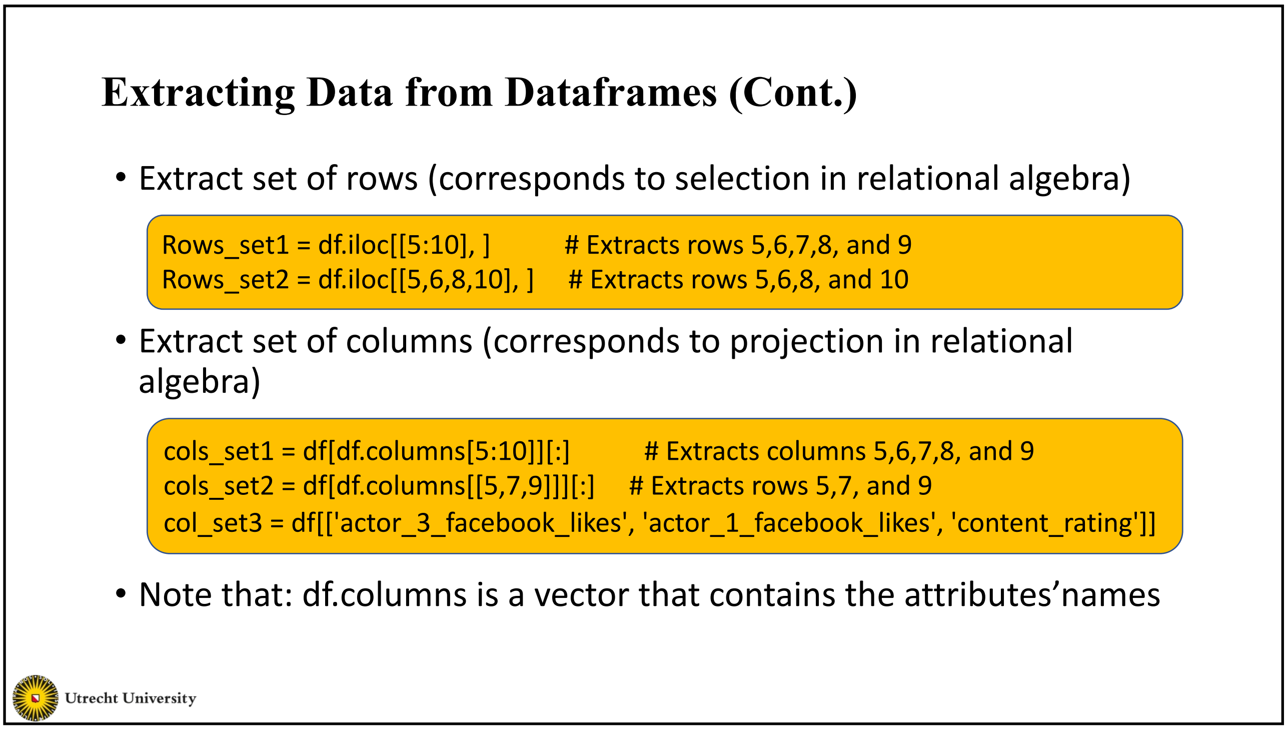
```
row1 = df.iloc[0,:]     #You may ignore adding the :
row1 = df.iloc[0]
```

- Extract the column with the names of directors

```
df.director_name     #  OR
df["director_name"]
```

Utrecht University

15



Extracting Data from Dataframes (Cont.)

- Extract set of rows (corresponds to selection in relational algebra)

```
Rows_set1 = df.iloc[[5:10], ]          # Extracts rows 5,6,7,8, and 9
Rows_set2 = df.iloc[[5,6,8,10], ]    # Extracts rows 5,6,8, and 10
```

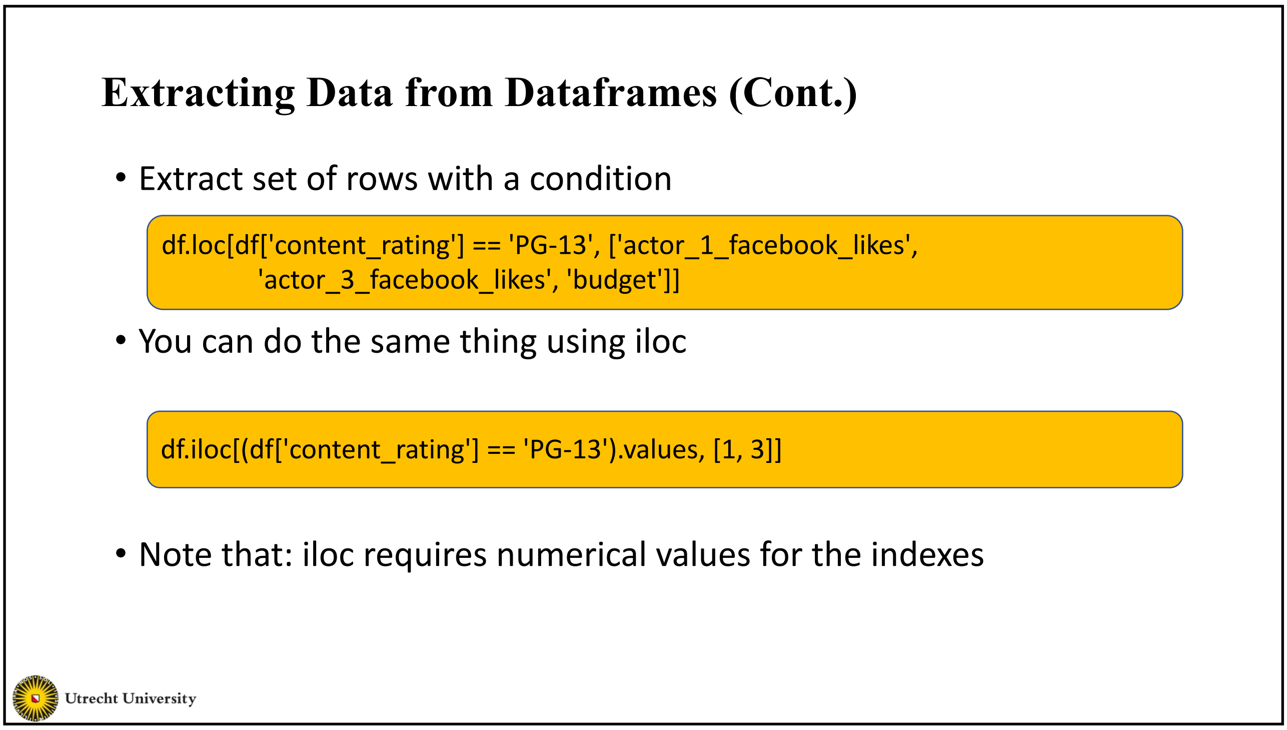
- Extract set of columns (corresponds to projection in relational algebra)

```
cols_set1 = df[df.columns[5:10]][:]         # Extracts columns 5,6,7,8, and 9
cols_set2 = df[df.columns[[5,7,9]]][:]     # Extracts rows 5,7, and 9
col_set3 = df[['actor_3_facebook_likes', 'actor_1_facebook_likes', 'content_rating']]
```

- Note that: df.columns is a vector that contains the attributes’names

Utrecht University

16



Extracting Data from Dataframes (Cont.)

- Extract set of rows with a condition

```
df.loc[df['content_rating'] == 'PG-13', ['actor_1_facebook_likes',
            'actor_3_facebook_likes', 'budget']]
```

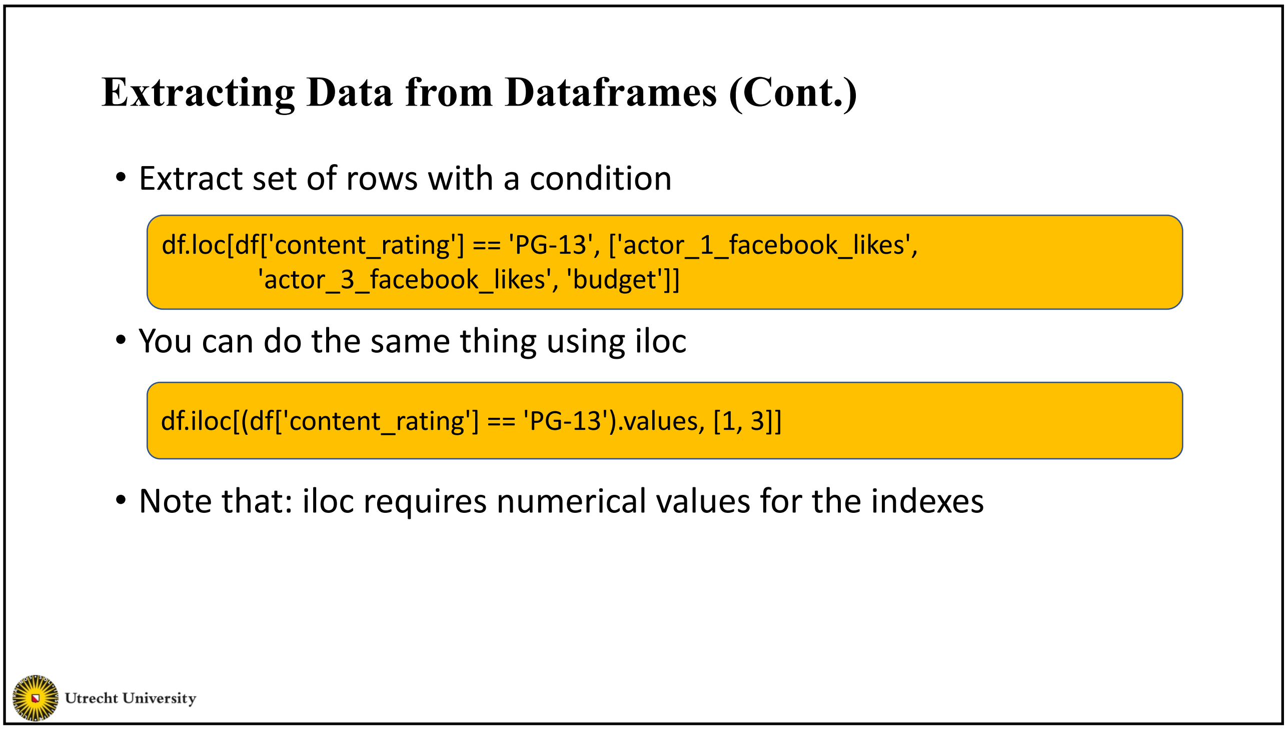
- You can do the same thing using iloc

```
df.iloc[(df['content_rating'] == 'PG-13').values, [1, 3]]
```

- Note that: iloc requires numerical values for the indexes

Utrecht University

17



Extracting Data from Dataframes (Cont.)

- Extract set of rows with a condition

```
df.loc[df['content_rating'] == 'PG-13', ['actor_1_facebook_likes',
            'actor_3_facebook_likes', 'budget']]
```

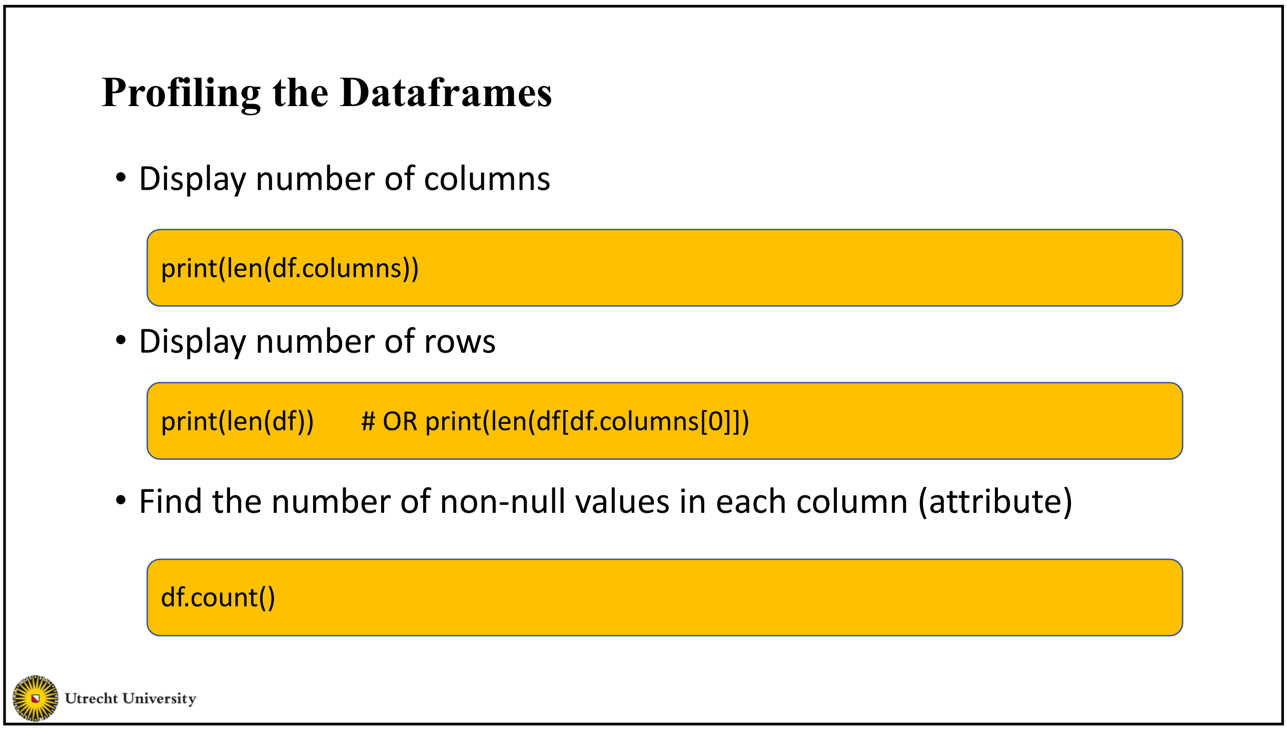
- You can do the same thing using iloc

```
df.iloc[(df['content_rating'] == 'PG-13').values, [1, 3]]
```

- Note that: iloc requires numerical values for the indexes

Utrecht University

18



Profiling the Dataframes

- Display number of columns

```
print(len(df.columns))
```

- Display number of rows

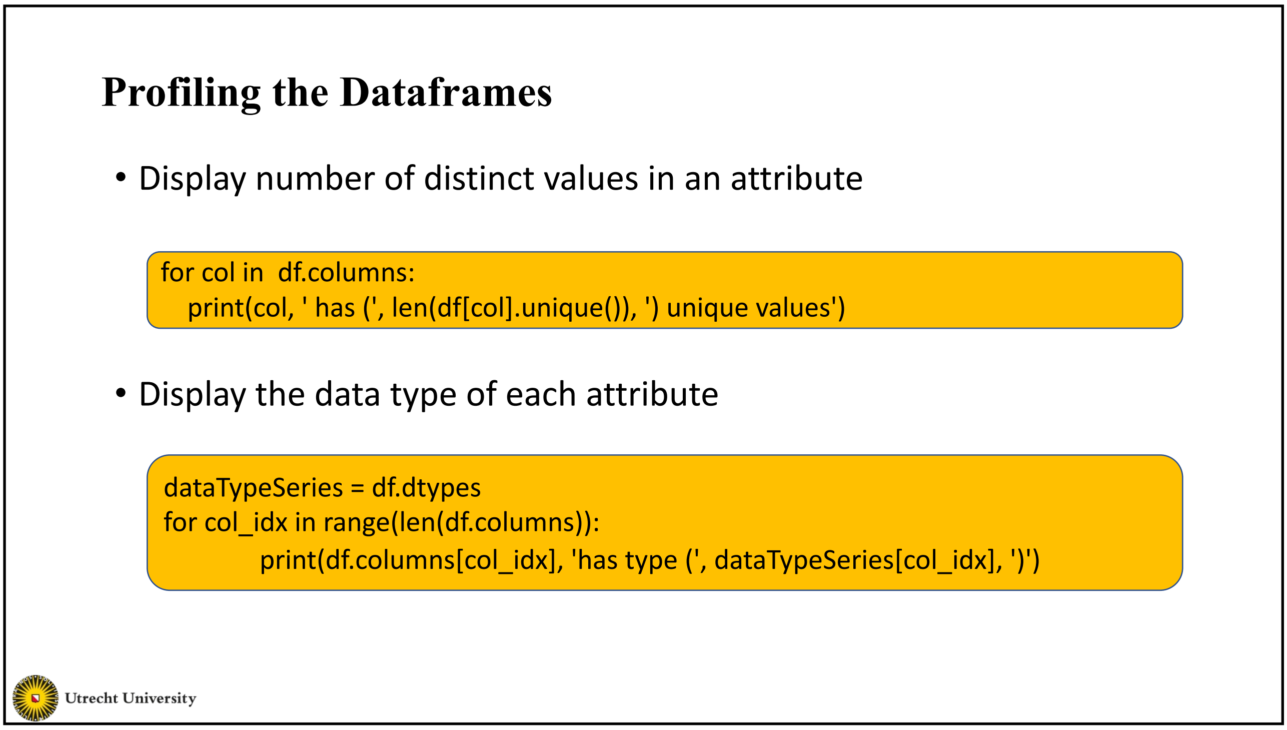
```
print(len(df))      # OR print(len(df[df.columns[0]])
```

- Find the number of non-null values in each column (attribute)

```
df.count()
```

Utrecht University

19



Profiling the Dataframes

- Display number of distinct values in an attribute

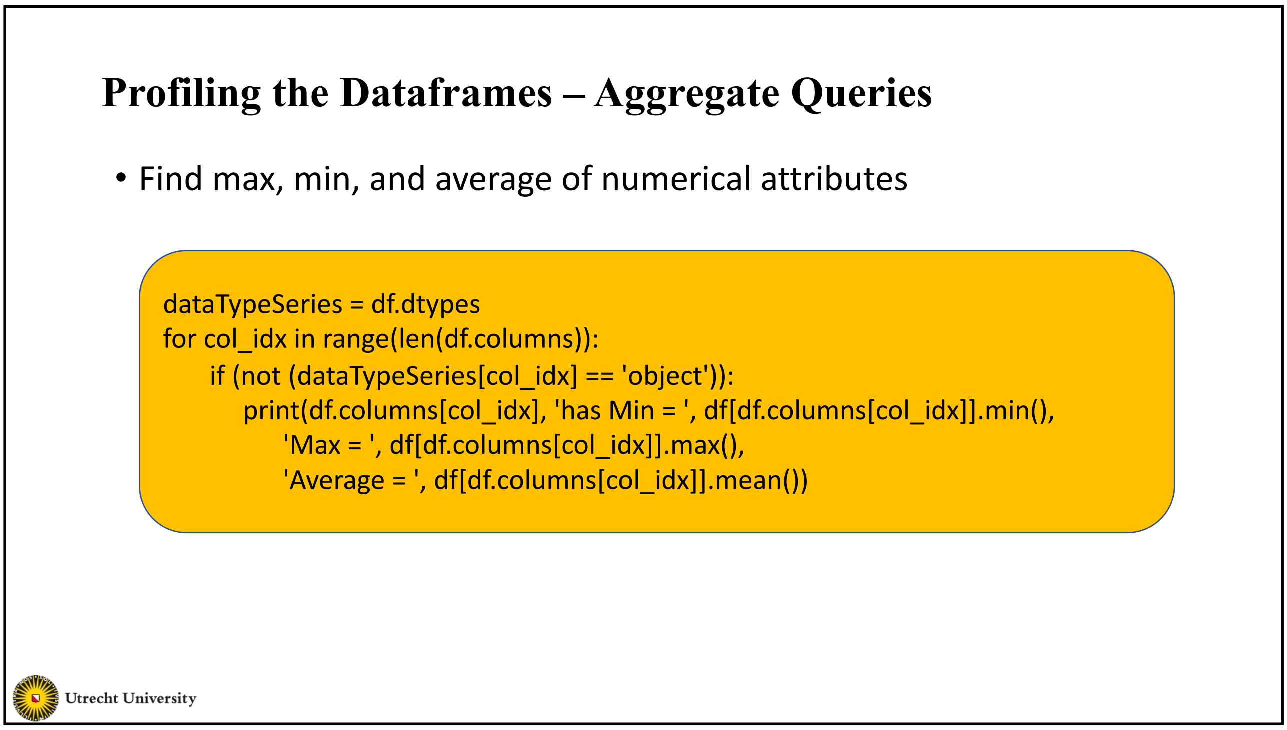
```
for col in  df.columns:
    print(col, ' has (', len(df[col].unique()), ') unique values')
```

- Display the data type of each attribute

```
dataTypeSeries = df.dtypes
for col_idx in range(len(df.columns)):
            print(df.columns[col_idx], 'has type (', dataTypeSeries[col_idx], ')')
```

Utrecht University

20



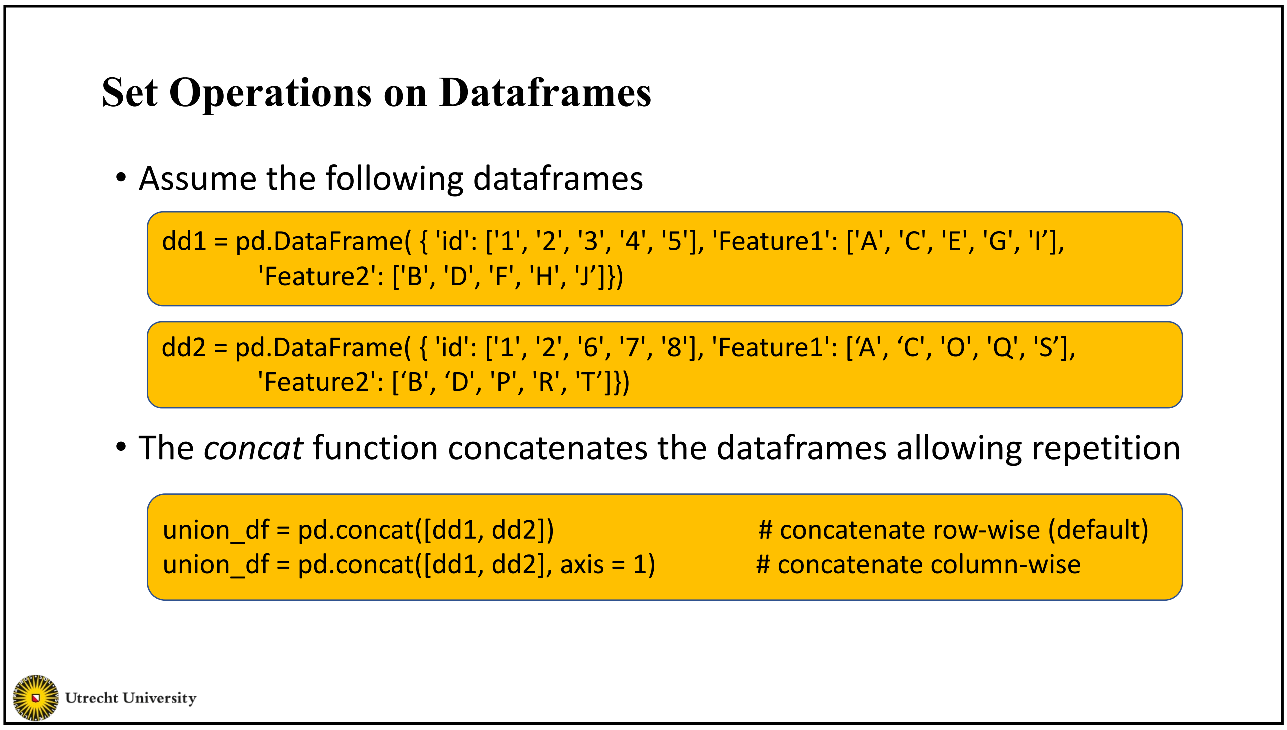
Profiling the Dataframes – Aggregate Queries

- Find max, min, and average of numerical attributes

```
dataTypeSeries = df.dtypes
for col_idx in range(len(df.columns)):
    if (not (dataTypeSeries[col_idx] == 'object')):
        print(df.columns[col_idx], 'has Min = ', df[df.columns[col_idx]].min(),
            'Max = ', df[df.columns[col_idx]].max(),
            'Average = ', df[df.columns[col_idx]].mean())
```

Utrecht University

21



Set Operations on Dataframes

- Assume the following dataframes

```
dd1 = pd.DataFrame( { 'id': ['1', '2', '3', '4', '5'], 'Feature1': ['A', 'C', 'E', 'G', 'I’],
        'Feature2': ['B', 'D', 'F', 'H', 'J’]})
```

```
dd2 = pd.DataFrame( { 'id': ['1', '2', '6', '7', '8'], 'Feature1': [‘A', ‘C', 'O', 'Q', 'S’],
        'Feature2': [‘B', ‘D', 'P', 'R', 'T’]})
```

- The *concat* function concatenates the dataframes allowing repetition

```
union_df = pd.concat([dd1, dd2])                # concatenate row-wise (default)
union_df = pd.concat([dd1, dd2], axis = 1)      # concatenate column-wise
```

Utrecht University

22

Join Operation on Dataframes

- The *merge* function joins dataframes on selected attribute

```
df_merge_col = pd.merge(dd1, dd2, on='id')
```

- If the joining attribute has different names in both dataframes

```
df_merge_col = pd.merge(dd1, dd2, left_on='att_dd1’, right_on = 'att_dd2')
```

Utrecht University

23

- Summarize what you learned today in 2-minutes

24