

Business 4720 - Class 6

Data Management in Python using Pandas

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
`jevermann@mun.ca`



Unless otherwise indicated, the copyright in this material is owned by Joerg Evermann. This material is licensed to you under the [Creative Commons by-attribution non-commercial license \(CC BY-NC 4.0\)](https://creativecommons.org/licenses/by-nc/4.0/)

This Class

What You Will Learn:

- ▶ Introduction to Python
- ▶ Introduction to the the Numpy package
- ▶ Introduction to the Pandas package

What is Python?

- ▶ Readability and simplicity
- ▶ Dynamic typing enhancing flexibility
- ▶ Extensive libraries
- ▶ Procedural, object-oriented, and functional programming
- ▶ Widely used in data analysis, AI, scientific computing, etc.
- ▶ Easy to learn
- ▶ Active community support

Intro Tutorial:

<https://python.swaroopch.com/>

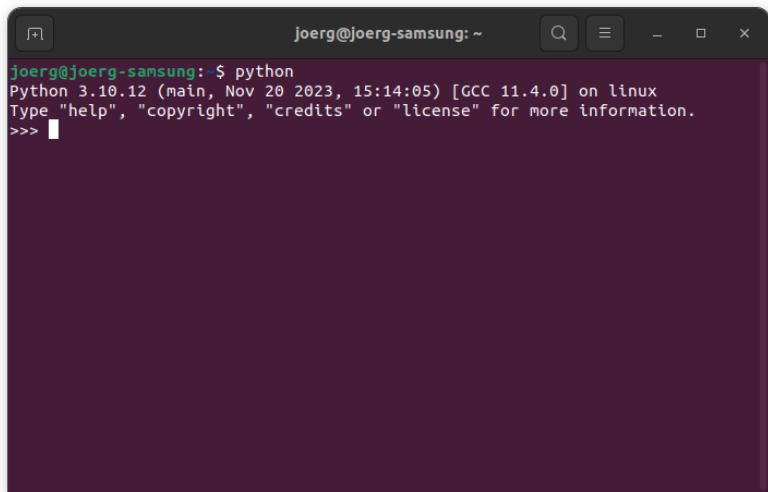
<https://github.com/swaroopch/byte-of-python/releases/>

- 1 Interactive Python Shell (command line)
- 2 Jupyter Notebooks
- 3 PyCharm IDE

Interactive Python Shell

- ▶ Similar to the R shell
- ▶ Type "python" to launch Python interpreter
- ▶ Prompt is "> > >", type **ENTER** to execute a command
- ▶ Use `quit()` to exit
- ▶ Use the **up-arrow** key to retrieve earlier commands.
- ▶ Use the **TAB** key to auto-complete a command.
- ▶ The Ubuntu terminal uses **SHIFT-CTRL-X**, **SHIFT-CTRL-C**, **SHIFT-CTRL-V** for cut/copy/paste.
- ▶ **Tip:** Use a notepad app to assemble commands and to keep results

Interactive Python Shell



A terminal window titled "joerg@joerg-samsung: ~" with standard window controls. The terminal output shows the command "python" being executed, which starts the Python 3.10.12 interpreter. The prompt changes from "\$" to ">>>".






```
joerg@joerg-samsung:~$ python
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Jupyter Notebooks

- ▶ Interactive computing environment
- ▶ Notebook Interface
- ▶ Combine executable code, text, visualizations
- ▶ Create and share documents with live code, equations, and explanatory text
- ▶ Collaborative editing of notebooks (on web-based services)
- ▶ Popular for Python, but can handle other languages

jupyterlab

Start

-  New notebook...
-  New session...
-  Open File...
-  Open Folder...
-  Connect...

Recent sessions

joerg /home

Jupyter News

[Open Community Call](#)

[And Voici!](#)

[Plug your application into the Jupyter world](#)

[Voilà 0.5.0 : Homecoming](#)

[Bringing Modern JavaScript to the Jupyter Notebook](#)

[Desktop GIS software in the cloud with JupyterHub](#)

[Generative AI in Jupyter](#)

[European Commission Funds Jupyter Bug Bounty Program](#)

[Announcing Jupyter Notebook 7](#)

[JupyterCon 2023 recordings now live on YouTube!](#)

 [Jupyter Blog](#)

JupyterLabs Desktop

The screenshot displays the JupyterLab Desktop environment. The top window title is "Untitled.ipynb - JupyterLab". The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar with icons for file operations. On the left, a file browser shows the current directory as "/ Current / Jupyter /" with a search bar and a table of files. The main area is a code editor for "Untitled.ipynb" using the "Python 3 (ipykernel)" environment. It contains two code cells: the first with the expression `1+1` and the second with `2`. The status bar at the bottom indicates the interface is in "Simple" mode, shows the file name "Untitled.ipynb", and the current cursor position "Ln 1, Col 4".

Untitled.ipynb - JupyterLab

conda: jlab_server

File Edit View Run Kernel Tabs Settings Help

Filter files by name

/ Current / Jupyter /

Name	Last Modified
Untitled.ip...	last month

Untitled.ipynb

Python 3 (ipykernel)

```
[1]: 1+1
```

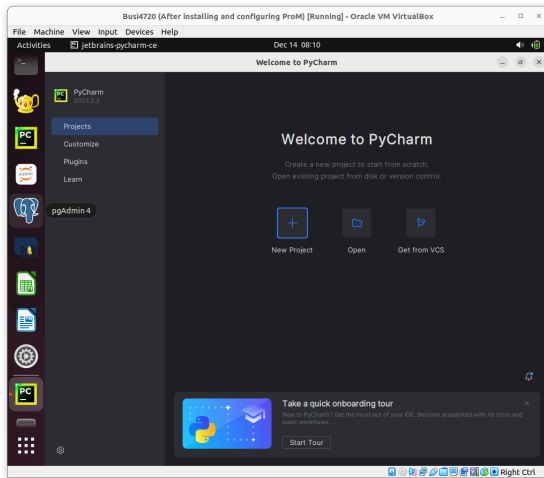
```
[1]: 2
```

Simple 0 3 Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 4 Untitled.ipynb 0

- ▶ "Kernel" is the Python interpreter and environment that runs your code
- ▶ Enter code into empty cell
- ▶ Press `CTRL-ENTER` to execute a cell
- ▶ Merge, split, move, copy, delete cells
- ▶ Save, import, export notebooks

- ▶ When working with multiple Python files in your project
- ▶ Useful for *programming* (defining functions, classes; using control structures, etc.) rather than just *scripting* (executing a few Python commands one after the other)
- ▶ Contains built-in debugging tools

PyCharm IDE



Python knows math:

```
# Addition
2 + 2
# Exponentiation
2**4
# Integer division
13 // 3
-13 // 3
# Modulus (remainder)
13 % 3
-25.5 % 2.25
# Comparisons
3 < 5
3 > 5
3 == 5
# Logical and, or, not operators
(3 < 5) and (4 < 2)
(3 < 5) or not (4 < 2)
```

String formatting methods:

```
# Define some variables
age = 19
name = 'Malina'

# Print them in different ways.
# Pick your favourite and stick with it.
print('{0} is {1} years old'.format(name, age))
print('{name} is {age} years old'.format(name=name, age=age))
print('{} is {} years old'.format(name, age))
print(f'{name} is {age} years old')
print(name+' is '+str(age)+' years old')
```

Backslashes split and continue lines:

```
print('This is a very long \
string and needs a second line')
i = \
5
print(i)
```

Python Strings

Python knows strings:

```
language = 'Innuktitut'

# Check the start of a string
if language.startswith('Innu'):
    print('Yes, the string starts with "Innu"')

# Check if letter contained in string
if 'u' in language:
    print('Yes, it contains the string "u"')

# Find the index of a string in another string
# Returns -1 if not found
if language.find('nuk') != -1:
    print('Yes, it contains the string "nuk"')
```

Note the colon and the indent of exactly 4 significant spaces!

Joining and splitting strings with a delimiter:

```
# Join a list of strings with a delimiter
delimiter = '_*_'
mylist = ['Nain', 'Hopedale', 'Makkovik', 'Rigolet']
mystring = delimiter.join(mylist)
print(mystring)

# Split a string on a delimiter
thelist = mystring.split(delimiter)
print(thelist)
```

Lists

Lists are ordered collections of items:

```
# Define list (Inuit deities)
gods = ['Sedna', 'Nanook', 'Akna', 'Pinga']

# Length of a list
len(gods)

# Iterate over items
for item in gods:
    print(item, end=' ')

# Append to a list
gods.append('Amaguq')

# Sort a list
gods.sort()

# Retrieve items from list
olditem = gods[0]

# Delete item in list
del gods[0]
```

Note the colon and the indent of exactly 4 significant spaces

Tuples are immutable:

```
# Define a tuple (Inuit Nunangat)
regions = ('Inuvialuit', 'Nunavut', 'Nunavik', 'Nunatsiavut')

# Length of a tuple
len(regions)

# Create a tuple of tuples, NOT flattened
more_regions = ('Kalaallit', 'Inupiaq', regions)

# Retrieve element 1 of element 3 in tuple
more_regions[2][1]
```

Dictionaries

- ▶ Key–value pairs
- ▶ Associative arrays
- ▶ Map

```
# Define a dict (largest citities)
c = {
    'Inuvialuit': 'Inuvik',
    'Nunavut': 'Iqaluit',
    'Nunavik': 'Kuujuuaq',
    'Nunatsiavut': 'Nain'
}
# Get the list of keys
list(c.keys())
# Get the list of values
list(c.values())

# Number of entries in dict
len(c)
```

```
# Retrieve a value for a key:
c['Nunavik']

# Delete a key-value pair
del c['Nunavut']

# Add a key-value pair
c['Nunavut'] = 'Iqaluit'

# Check for existence of a key
if 'Nunavut' in c:
    print("\nNunavut's largest city is", c['Nunavut'])
```

Structured Data Types

Important

- ▶ Indexing begins at 0 (different from R!)
- ▶ Can contain any data type

Sequences

- ▶ List, tuples, strings are sequences
- ▶ Membership tests using `in` or `not in`
- ▶ Indexing and slicing

Slicing

```
regions = ('Inuvialuit', 'Nunavut',  
           'Nunavik', 'Nunatsiavut')  
language = 'Inuktitut'
```

```
# Slicing on a tuple
```

```
regions[1:3]  
regions[2:]  
regions[1:-1]  
regions[:]
```

```
# Slicing with step size
```

```
regions[::1]  
regions[::2]  
regions[::3]  
regions[:::-1]
```

Lists

- 1 Create a list containing the numbers 1 to 10. Use list slicing to create a sublist with only the even numbers.
- 2 Using a `for` loop, sum all the items in the list.
- 3 Using a `for` loop, iterate over the list and print each number squared.
- 4 Write a program to append the square of each number in the range `[1:5]` to a new list.

Tuples

- 1 Create a tuple with different data types (string, int, float).
- 2 Demonstrate how tuples are immutable by attempting to change its first element.

Dictionaries

- 1 Create a nested dictionary and demonstrate accessing elements at various levels. A nested dictionary is one in which the values themselves are also dictionaries.

Numerical Data in Python with NumPy

What is Numpy?

- ▶ High-performance scientific computing and data analysis.
- ▶ Multidimensional arrays
- ▶ Comprehensive mathematical function library
- ▶ Foundational package for other scientific libraries like SciPy, Pandas, Matplotlib, scikit-learn, scikit-image, etc.

Intro Tutorials

- ▶ [NumPy Quickstart](#)
- ▶ [NumPy for Absolute Beginners](#)

N-Dimensional Array, type "ndarray"

```
# Import the numpy package
import numpy as np

# Create an array
a = np.arange(15).reshape(3, 5)

# Examine its properties
a.shape
a.ndim
a.dtype.name
a.size
```

NumPy Basics

```
# Create an array from Python lists and tuples
b = np.array([(1.5, 2., 3),
              (4.0, 5., 6)])

print(b)

# Elementwise operations
3 * b
b + 5
np.sqrt(b)

# NumPy array functions
np.sum(b)
np.max(b)
# Axis 0 is by column
np.max(b, axis=0)
# Axis 1 is by row
np.max(b, axis=1)
np.std(b, axis=0)
# Transpose
np.transpose(b)
# Cov default by row
np.cov(b)
np.cov(np.transpose(b))
```

NumPy Basics [cont'd]

```
# Create an array of zeros with shape (3,4)  
x = np.zeros((3,4))  
print(x)  
  
# Create an array of ones with shape (2,3,4)  
y = np.ones((2,3,4))  
print(y)
```

Array Slicing

- Each axis can be sliced using `[:]` or `[::]`

```
b = np.array([[ 0,  1,  2,  3],
              [10, 11, 12, 13],
              [20, 21, 22, 23],
              [30, 31, 32, 33],
              [40, 41, 42, 43]])

# One element
b[2, 3]
# Multiple rows, one column
b[0:5, 1]
# Every other row up to 4, one column
b[0:5:2, 1]
# All rows, columns 1 and then every other
b[:, 1::2]
# Two rows, all columns
b[1:3, :]
# Last row
b[-1]
# Last column
b[:, -1]
```

Array Reshaping

```
# Create 3x4 array of random numbers
a = np.floor(10 * np.random.random((3, 4)))

a.shape
a.flatten()
a.reshape(6, 2)
a.T
a.T.shape

# Create another 3x4 array of random numbers
b = np.floor(5 * np.random.random((3, 4)))
# Vertical stacking
np.vstack((a, b))
# Horizontal stacking
np.hstack((b, a))

# Iterate over rows
for row in b:
    print(row)

# Iterate over all elements
for element in b.flat:
    print(element)
```

Array Indexing with Boolean Arrays

```
a = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])

# Are entries less than 5?
a < 5
# Entries that are less than 5
a[a < 5]

# Are entries even?
a%2 == 0
# Entries that are even
a[a%2 == 0]
```


Hands-On Exercises

- 1 Create a four-dimensional array with random numbers in the shape indicated by the last four digits of your student number (if your student number contains a 0, use a 1 instead)
- 2 Construct a new array by swapping the first half of rows (axis 0) with the second half of rows (axis 0)
- 3 Calculate all covariance matrices formed by the last two axes of your array. *Tip:* Iterate over the first two axes/dimensions with a `for` loop
- 4 Subtract the mean of the array from each element in the array (mean normalization)
- 5 Select all elements that are greater than the overall mean
- 6 Sort the selected elements from the previous step in ascending order

What is Pandas?

- ▶ Open-source library for data analysis
- ▶ High-performance, easy-to-use data structures and data analysis tools
- ▶ Can handle tabular data, time series, matrix data, etc.
- ▶ Tools for data cleaning, transformation, and preparation
- ▶ Importing data from CSV, Excel, SQL databases, etc.
- ▶ Functions for aggregating, pivoting, joining, and sorting data

Intro Tutorial: [10 Minutes to Pandas](#)

Pandas Dataframe

- ▶ 2-dimensional
- ▶ Row labels are called *index*
- ▶ Columns may have different data types

```
# Create a dict of two Series
d = {
    "col1": pd.Series([1.0, 2.0, 3.0],
                      index=['a', 'b', 'c']),
    "col2": pd.Series([1.0, 2.0, 3.0, 4.0],
                      index=['a', 'b', 'c', 'd'])
}

# Create a dataframe from dict
df = pd.DataFrame(d)
```

Pandas Dataframe – Basic Information

```
# Dimensions (rows, columns)
df.shape

# Row labels (index)
list(df.index)
# Column labels
list(df.columns)

# Information about columns and data types
df.info()

# First few rows
df.head()
# Last few rows
df.tail()

# Summary of data
df.describe()
```

Pandas Dataframe – Indexing

```
# Select one column  
df['col1']  
  
# Select multiple columns (list of columns)  
df[['col1', 'col2']]  
  
# Select rows by label, returns Series  
df.loc['a']  
  
# Select single row by number  
df.iloc[2]  
  
# Select single column by number  
df.iloc[:,1]  
  
# Select rows 0 to 3, columns 0 to 1  
df.iloc[0:4:2, 0:2]  
  
# Select every other row 0 to 3  
df[0:4:2]  
  
# Select rows by boolean array  
df[df['col1'] > 2]
```

Pandas Dataframe – Operations

```
# Elementwise operators
df * 5 + 2
1/df
df**4

# Transpose
df.T

# Using Numpy functions on Pandas data frames
np.exp(df)
np.sum(df[['col1', 'col2']], axis=1)
```

Pandas Dataframe – Selection with Query

```
df = pd.DataFrame(np.random.rand(10, 3),
                  columns=['a', 'b', 'c'])

# Pure python
df[(df['a'] < df['b']) & (df['b'] < df['c'])]

# Shorter with Query
df.query('(a < b) & (b < c)')
df.query('a < b & b < c')
df.query('a < b and b < c')
df.query('a < b < c')
```

Easy Pandas – Example Dataset

- ▶ Government of Canada, Open Government Portal
- ▶ Fuel Consumption Ratings – Battery-electric vehicles – 2012–2023
- ▶ <https://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64>

Column	Data Type
Make	Categorical (string)
Model	Categorical (string)
Year	Numeric
Category	Categorical (string)
City	Numeric
Hwy	Numeric
Comb	Numeric
Range	Numeric

Easy Pandas – Reading CSV Files

```
# Import pandas
import pandas as pd

# Read CSV into a Pandas data frame
data = pd.read_csv('https://evermann.ca/busi4720/fuel.csv')

# Basic information about data
data.shape
list(data.columns)
data.info()
data.describe()
```

Easy Pandas – Filtering

```
# Filter values  
data.query('Make=="Ford" & Year==2023')
```

Equivalent in R:

```
data |>  
  filter(Make=='Ford',  
         Year==2023) |>  
  print()
```

Equivalent in SQL:

```
SELECT *  
FROM data  
WHERE Make=='Ford' AND  
       Year==2023;
```

Easy Pandas – Selecting Columns

```
# Filter values and select columns
data.query('Make=="Ford" & Year==2023') \
[['Model', 'Category', 'Range']]
```

Equivalent in R

```
data |>
  filter(Make=='Ford',
         Year==2023) |>
  select(Model, Category,
         Range) |>
  print()
```

Equivalent in SQL:

```
SELECT Model, Category, Range
FROM data
WHERE Make=='Ford' AND
       Year==2023;
```

Easy Pandas – Create New Columns

```
# Filter values, create new calculated column and select cols  
data.query('Make=="Ford" & Year==2023') \  
    .assign(HwyRange = data['Range']*data['Comb']/data['Hwy']) \  
    [['Model', 'Category', 'Range', 'HwyRange']]
```

Equivalent in R:

```
data |>  
  filter(Make=='Ford',  
         Year==2023) |>  
  mutate(HwyRange=  
         Range*Comb/Hwy) |>  
  select(Model, Category,  
         Range, HwyRange) |>  
  print()
```

Equivalent in SQL:

```
SELECT Model, Category, Range,  
        (Range*Comb)/Hwy AS HwyRange  
FROM data  
WHERE Make=='Ford' AND  
        Year==2023;
```

Easy Pandas – Renaming Columns

```
# Filter values, create two new calculated columns,  
# rename a column, and select columns  
data.query('Make=="Ford" & Year==2023') \  
  .assign(HwyRange = data['Range']*data['Comb']/data['Hwy']) \  
  .assign(CityRange = data['Range']*data['Comb']/data['City']) \  
  .rename(columns={'Range': 'CombRange'}) \  
  [['Model', 'Category', 'CombRange', 'CityRange', 'HwyRange']]
```

Equivalent in R:

```
data |>  
  filter(Make=='Ford',  
         Year==2023) |>  
  mutate(HwyRange =  
         Range * Comb / Hwy) |>  
  mutate(CityRange =  
         Range * Comb / City) |>  
  rename(CombRange = Range) |>  
  select(Model, Category,  
         CombRange, CityRange,  
         HwyRange) |>  
  print()
```

Equivalent in SQL:

```
SELECT Model, Category,  
       Range AS CombRange,  
       (Range * Comb) / Hwy  
       AS HwyRange,  
       (Range * Comb) / City  
       As CityRange  
FROM data  
WHERE Make=='Ford' AND  
       Year==2023;
```

Easy Pandas – Distinct Values

```
# Find distinct values  
data[['Make', 'Model']].drop_duplicates()
```

Equivalent in R:

```
data |>  
  distinct(Make, Model) |>  
  print()
```

Equivalent in SQL:

```
SELECT DISTINCT Make, Model  
FROM data;
```

Easy Pandas – Ordering

```
# Filter values, order by values of two columns
# and select columns
data.query('Make=="Ford" & Year==2023') \
    .sort_values(['Category', 'Range'], ascending=[True, False]) \
    [['Model', 'Category', 'Range']]
```

Equivalent in R:

```
data |>
  filter(Make=='Ford',
         Year==2023) |>
  select(Model, Category,
         Range) |>
  arrange(Category,
          desc(Range)) |>
  print()
```

Equivalent in SQL:

```
SELECT Model, Category, Range
FROM data
WHERE Make=='Ford' AND
      Year==2023
ORDER BY Category ASC,
      Range DESC;
```

Easy Pandas – Grouping and Summarizing

```
# Filter values, group the data,  
# calculate aggregates of multiple columns  
# filter on aggregate data, order by value  
# and select certain columns  
data.query('Year==2023') \  
    .groupby(['Make', 'Category']) \  
    .agg(meanCity = ('City', 'mean'),  
         meanHwy = ('Hwy', 'mean'),  
         meanComb = ('Comb', 'mean'),  
         maxRange = ('Range', 'max'),  
         nVehicle = ('Model', 'count')) \  
    .query('nVehicle > 1') \  
    .sort_values(['Category', 'meanComb']) \  
    .reset_index() \  
    [['Category', 'meanComb', 'Make', 'meanCity', \  
      'meanHwy', 'maxRange', 'nVehicle']]
```


Grouping and Summarizing [cont'd]

Equivalent in R:

```
data |>
  filter(Year==2023) |>
  group_by(Make, Category) |>
  summarize(
    meanCity = mean(City),
    meanHwy = mean(Hwy),
    meanComb = mean(Comb),
    maxRange = max(Range),
    nVehicle = n()) |>
  filter(nVehicle > 1) |>
  arrange(Category, meanComb) |>
  relocate(Category, meanComb) |>
  print()
```

Equivalent in SQL:

```
SELECT Category,
        AVG(Comb) AS meanComb,
        Make,
        AVG(City) AS meanCity,
        AVG(Hwy) AS meanHwy,
        MAX(Range) AS maxRange,
        COUNT(*) AS nVehicle
FROM data
WHERE Year==2023
GROUP BY Make, Category
HAVING COUNT(*) > 1
ORDER BY Category ASC,
          meanComb ASC;
```

Advanced Pandas with the Pagila Database

```
rentals = pd.read_csv(  
    'http://evermann.ca/busi4720/rentals.csv')  
  
actors = pd.read_csv(  
    'https://evermann.ca/busi4720/actors.categories.csv')  
  
addresses = pd.read_csv(  
    'https://evermann.ca/busi4720/addresses.csv')
```

Find all films and the actors that appeared in them, ordered by film category and year, for those films that are rated PG:

```
data = pd.merge(rentals, actors, on='title',
                suffixes=('_customer', '_actor'), how='outer')
data.query('rating == "PG"') \
    .assign(actor = data['last_name_actor'] + \
                ', ' + data['first_name_actor']) \
    .rename(columns={'release_year': 'year'}) \
    [['actor', 'title', 'category', 'year']] \
    .drop_duplicates(['actor', 'title', 'category', 'year']) \
    .groupby(['category', 'year', 'title']) \
    ['actor'].apply(list) \
    .reset_index() \
    .sort_values(['category', 'year']) \
```

Find the most popular actors in the rentals in each city:

```
full_data = pd.merge(rentals, addresses,
                     left_on='customer_address',
                     right_on='address_id')
full_data = pd.merge(full_data, actors, on='title',
                     suffixes=('_customer', '_actor'))
```

```
full_data \
    .assign(actor=full_data['last_name_actor'] + ', ' +
            full_data['first_name_actor'] ) \
    .groupby(['city', 'actor']) \
    .agg(count = ('title', 'count')) \
    .reset_index() \
    .assign(ranking=lambda df:
            df.groupby('city')['count']
              .rank(method='min', ascending=False)) \
    .query('ranking <= 3') \
    .sort_values(by=['city', 'ranking', 'actor'])
```

Find the customers who spend the most on rentals, and the number of rentals with the highest total rental payments for each category grouped by rental duration.

```
full_data \
    .assign(customer=full_data['last_name_customer'] + ', ' +
            full_data['first_name_customer'] ) \
    [['customer', 'amount', 'rental_duration', \
      'category', 'phone', 'city']] \
    .groupby(['category', 'rental_duration', 'customer']) \
    .agg(payments=('amount', 'sum'),
         num_rentals=('amount', 'count')) \
    .reset_index() \
    .assign(ranking=lambda df: \
            df.groupby(['category', 'rental_duration'])['payments'] \
              .rank(method='min', ascending=False)) \
    .loc[lambda df: \
         df.groupby(['category', 'rental_duration'])
         ['ranking'].idxmin() ]
```

Advanced Pandas with the Pagila Database [cont'd]

Get the top 5 and the bottom 5 grossing customers for each quarter.

```
full_data \
    .assign(customer=full_data['last_name_customer'] + ', ' +
            full_data['first_name_customer'],
            q=pd.to_datetime(full_data['rental_date'], utc=True)
                .dt.to_period("Q")) \
    [['customer', 'q', 'amount', 'rental_date']] \
    .groupby(['q', 'customer']) \
    .agg(payments=('amount', 'sum')) \
    .reset_index() \
    .drop_duplicates(['customer', 'q', 'payments']) \
    .assign(rank_top = lambda df :
            df.groupby('q')['payments']
                .rank(method='min', ascending=False),
            rank_bot = lambda df :
            df.groupby('q')['payments']
                .rank(method='min', ascending=True)) \
    .reset_index() \
    .query('rank_top <= 5 or rank_bot <= 5') \
    .sort_values(by=['q', 'payments'], ascending=[True, False])
```

Find the set of film titles by rental customer and the total number rentals for each customer

```
full_data \
    .assign(customer=full_data['last_name_customer'] + ', ' +
            full_data['first_name_customer']) \
    [['customer', 'title']] \
    .groupby('customer') \
    ['title'].apply(list) \
    .reset_index(name='titles') \
    .assign(rentals = lambda df :
            df['titles'].apply(len) ,
            unique_titles = lambda df :
            df['titles'].apply(lambda x: list(set(x)))) \
    .drop(columns=['titles']) \
    .sort_values(by='customer')
```

Hands-On Exercises

- 1 Find all films with a rating of 'PG'
- 2 List all customers who live in Canada (with their address)
- 3 Find the average *actual* rental duration for all films
 - ▶ This requires date arithmetic
- 4 Find the average overdue time for each customer
 - ▶ This requires date arithmetic
- 5 List all films that have never been rented
- 6 List the names of actors who have played in more than 15 films