Python meeting
03: Requests, explode, split, Levenshtein

29th April 2026

The Python Meeting will start soon

COURSE PYTHON SCHEDULE

01: 20250114, 9am: Initiation – Case-study: P02_15: Above average prices

- Set-up: Python, Visual Studio Code
- Project creation: Folder, .py, Virtual Environment, CMD
- Libraries: Import, requirements file
- Naming conventions
- Basic objects: Dataframe (table), Series (column), Variable, Function, Group_by
- Basic functions: Import, Rename, Filter, Join, Create columns, Group, Aggregate, Export

02: 20250121,10am: Melt, split, for – Case-studies: D01: Trial balance, P01_19: Supplier debtors, O01_19: Customer creditors

- Using melt, split
- For loop

03: 20250128, 9am: Web-download – Case-study: P01_10: Suppliers in OFAC

- Request for downloading sanctions
- IO for encoding
- Regex for comparison
- Split, explode for row generation
- Levenshtein distance scoring

04: 20250204, 9am: JSON, expressions, rolling Window – Case-study: P02_19: Split POs

- Import JSON, Expression object for adding labels,
- cum_sum() for rolling window

05: 20250211, 9AM: zip, dynamic fields – Case-study: P02_03: PO whilst blocked

- Zip to group lists together
- Using dynamic fields in a for loop

09:00 – 09:05

COURSE PYTHON SCHEDULE

06: 20250218, 10am: Cumulative sum, dates – Case-study: P02_18: PO slow rotation

- cum_sum for calculating future or past inventory movements

07: 20250225, 9am: Isolation forest for unusual transactions – Case-study: Unusual bank transfers

- Using isolation forest library to score for unusual transactions

08: 20250304: 9am: SpaCy – Case-study: P01_11/ O01_11: Suppliers/ customers that are people

- Using SpaCy NLP object to categorize names

09: 20250311: 9am: Contract review – Case-study: Scoring of contracts

- Using Gemini model to check for paragraphs of similar meaning

10: 20250318: 10am: Open AI API – Case-study: Generating audit reports

- Using API to OpenAI to generate text for audit reports

11: 20250325: 9am: Regex and Sklearn matching – Case-study: HR03_12/ H403_13: Unusual T&E

- Using Regex and Sklearn to check for unexpected travel and expenses

12: 20250401: 9am: OCR: Image recognition – Case-study: HR03_14: Expenses for others

- Using OCR python library to read travel and expense receipts

09:00 – 09:05

TODAY'S SCHEDULE

01

Case-study

→ P01_10: check for suppliers on OFAC list

02

Calling another python script

How to call another python script?

03

Python structures

Explode

Group_by

09:00 – 09:05

01

Case study: Suppliers in OFAC list

09:00 – 09:05

Pre-requisites -> run programme

Pre-requisites:

1/ Set-up files

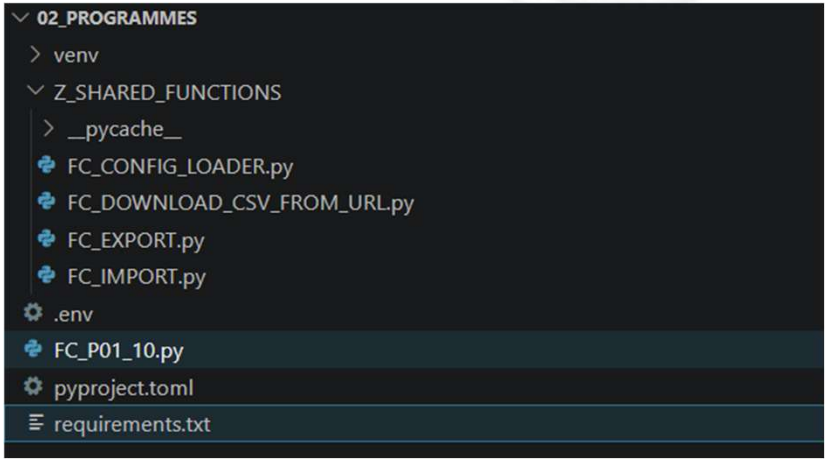
- Follow python cheat sheet for obtaining project from gitHub or organizing project structure as python cheat sheet
- Ensure that you have a Windows folder (we shall refer to it as ***root***) containing sub-folders:
- **/01_SOURCES**: should contain: 20251007_172757_1_LFA1.csv, A_AM_EXCLUSION_WORD.csv, AM_VARIABLES.txt
- **/02_PROGRAMMES**: should contain: **P01_10.py**, **pyproject.toml** and **requirements.txt**
- **/02_PROGRAMMES**: Create a **.env** file with the variable **ZV_ST_ROOT_FOLDER=C:\...\... (change to address of your root folder)**
- **/03_RESULTS**: This is where the results of the program will go

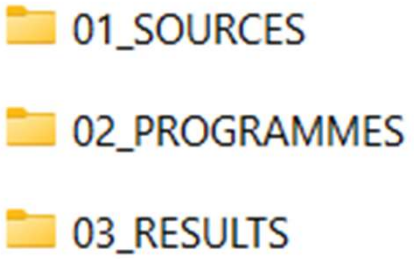
2/ Set-up environment:

- Navigate to the /02_PROGRAMMES folder and open Visual Studio Code there (**CMD** in Windows search bar, then **code .**)
- Terminal: (View-> Terminal or CTRL+SHIFT+`)
- Virtual environment: **py -3.14 -m venv venv**
- Activate virtual environment: **.\venv\Scripts\Activate.ps1**
- Run requirements: **pip install -r requirements.txt**

3/ Run

- From VSC opened at /02_PROGRAMMES, click on the script FC_P01_10.py and then execute.
- If the programme runs successfuly, you will see a new Excel file in /03_RESULTS





09:05 – 09:10

Tip for running the program

As mentioned in the meeting: (thanks Markus!)

if you cannot get python to use the venv inside your project, when you click on the execute button, you can type the following in the terminal, to force python to use the correct venv:

```
.\venv\Scripts\python.exe P01_10_SUPPLIERS_IN_OFAC.py
```

Did you do that already?

20260429_01_DidYouRunTheCode?

1. **Were you able to run the code?** (Single choice)

- Yes
- No

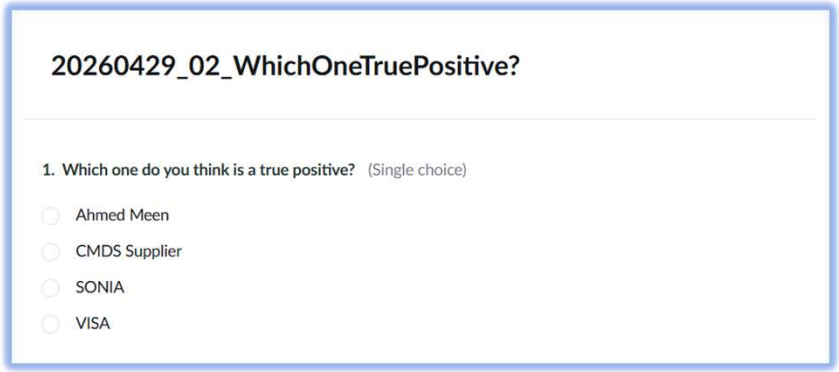
2. **What prevented you from running the code?** (Single choice)

- No free time before the meeting
- I got stuck on a bug

09:10 – 09:15

What should the result be?

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	LIFNR	NAME1	STRAS	ZF_NAME1_CLEAN	ZF_STR/	LFA1_LI	LFA1_FI	OFAC_E	OFAC_FIELD	ZF_OFA	ZF_INV_	ZF_LFA1	ZF_NUN	ZF_SCO	
2	VMIETER0	AHMED, Ai	hallerstr 3	AHMED AMEEN	HALLERST	VMIETER0	AHMED, Ai	44442	AHMED, Ameen	Names co	100	Yes	2	102	
3	00000031	CMDS Sup	Main Stree	CMDS SUPPLIER	MAIN STRE	00000031	Main Stree	34953	Main Street	Address c	100	Yes	2	102	
4	00002000	MAKO	24, HIGH S	MAKO	24 HIGH S	00002000	24, HIGH S	19709	74 High Street	Address c	50	No	1	51	
5	00001000	Mr Michae	35 Main St	MR MICHAEL HIRE	35 MAIN S	00001000	35 Main St	735	32 Main Street	Address c	50	No	2	52	
6	00001002	Alan Aguil	35 Main St	ALAN AGUILAR	35 MAIN S	00001002	35 Main St	735	32 Main Street	Address c	50	No	2	52	
7	K0103	Sonia	motorstr 1	SONIA	MOTORSTI	K0103	Sonia	15085	SONIA I	Names co	33.33	No	1	34.33	
8	00000031	CMDS Sup	Main Stree	CMDS SUPPLIER	MAIN STRE	00000031	Main Stree	34953	Taiz Street	Address c	33.33	No	1	34.33	
9	00000033	Material S	123 Main S	MATERIAL SUPPLIER	123 MAIN S	00000033	123 Main S	735	32 Main Street	Address c	33.33	No	2	35.33	
10	00000039	VISA	7800 Broa	VISA	7800 BRO/	00000039	7800 Broa	15732	80 Broad Street	Address c	33.33	No	2	35.33	
11	00000039	VISA	7800 Broa	VISA	7800 BRO/	00000039	7800 Broa	15734	80 Broad Street	Address c	33.33	No	2	35.33	
12	00000039	VISA	7800 Broa	VISA	7800 BRO/	00000039	7800 Broa	15736	80 Broad Street	Address c	33.33	No	2	35.33	
13	00000039	VISA	7800 Broa	VISA	7800 BRO/	00000039	7800 Broa	15737	80 Broad Street	Address c	33.33	No	2	35.33	
14	00000039	VISA	7800 Broa	VISA	7800 BRO/	00000039	7800 Broa	26654	80 Broad Street	Address c	33.33	No	2	35.33	
15	00000039	VISA	7800 Broa	VISA	7800 BRO/	00000039	7800 Broa	26689	80 Broad Street	Address c	33.33	No	2	35.33	
16	00000039	VISA	7800 Broa	VISA	7800 BRO/	00000039	7800 Broa	26705	80 Broad Street	Address c	33.33	No	2	35.33	



09:15 – 09:30

02

Calling another python script

Organizing scripts

09:30 – 09:45

Calling another python script

All python scripts are organized into functions, in order to avoid code repetition.

```
# 1.2/ Custom functions
from Z_SHARED_FUNCTIONS.FC_DOWNLOAD_CSV_FROM_URL import FC_DOWNLOAD_CSV_FROM_URL
from Z_SHARED_FUNCTIONS.FC_CONFIG_LOADER import ZV_DI_VARIABLES
from Z_SHARED_FUNCTIONS.FC_EXPORT import FC_EXPORT_EXCEL_POLARS
from Z_SHARED_FUNCTIONS.FC_IMPORT import FC_IMPORT_TEXT
```

```
# Step 3/ Import data
# From URL
P01_10_00_TT_DF_OFAC_ADD = (
    PI_POLARS.read_csv(
        FC_DOWNLOAD_CSV_FROM_URL('ADD.csv'),
        separator=',',
        has_header=False,
        infer_schema=False,
        new_columns=[
            'ADD_ENT_NUM',
            'ADD_ADD_NUM',
            'ADD_ADDRESS',
            'ADD_CITY_STATE_PROVINCE_POSTCODE',
            'ADD_COUNTRY',
            'ADD_REMARKS',
        ],
    )
)
```

Passing information to our script

We pass a variable to FC_DOWNLOAD_CSV_FROM_URL()

```
# Step 3/ Import data
# From URL
P01_10_00_TT_DF_OFAC_ADD = (
    PI_POLARS.read_csv(
        FC_DOWNLOAD_CSV_FROM_URL('ADD.csv'))
```

What are we passing to FC_DOWNLOAD_CSV_FROM_URL?

```
import requests as PI_REQUESTS
import io as PI_IO

def FC_DOWNLOAD_CSV_FROM_URL(ZVFCI_FILENAME):
    ZV_URL = f"https://sanctionslistservice.ofac.treas.gov/api/download/{ZVFCI_FILENAME}"
    ZV_RESPONSE = PI_REQUESTS.get(ZV_URL)
    ZV_RESPONSE.raise_for_status()
    return PI_IO.StringIO(ZV_RESPONSE.content.decode('iso-8859-1'))
```

What is coming back from FC_DOWNLOAD_CSV_FROM_URL?

Passing information between functions

What is passed to .read_csv()?

A variable to say that there is no header

A list of field names

A delimiter

A .csv file

A variable to import all as text

```
# Step 3/ Import data
# From URL
P01_10_00_TT_DF_OFAC_ADD = (
    PI_POLARS.read_csv(
        FC_DOWNLOAD_CSV_FROM_URL('ADD.csv'),
        separator=',',
        has_header=False,
        infer_schema=False,
        new_columns=[
            'ADD_ENT_NUM',
            'ADD_ADD_NUM',
            'ADD_ADDRESS',
            'ADD_CITY_STATE_PROVINCE_POSTCODE',
            'ADD_COUNTRY',
            'ADD_REMARKS',
        ],
    )
)
```

What does read_csv() give back to us?

A polars dataframe

Which library does read_csv() come from?

polars

09:30 – 09:45


03
Python structures

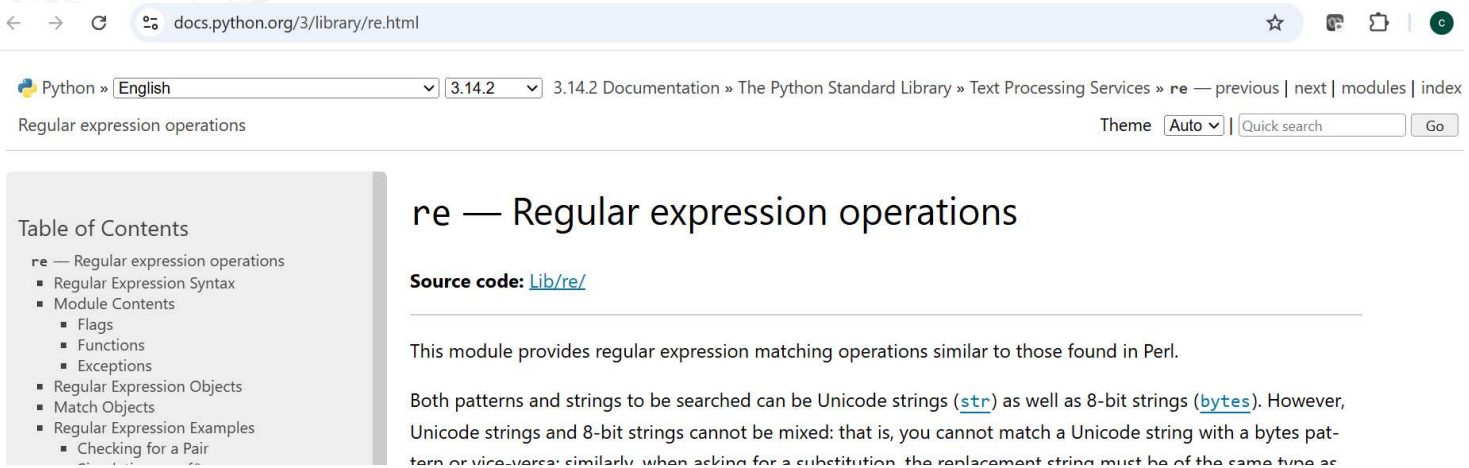
Regex

Replace any non-alpha-numeric or spaces with ‘’
Replace multiple spaces with one space

```python
# Step 5.1/ Clean the OFAC Addresses
P01_10_00_TT_DF_OFAC_ADD = (
    P01_10_00_TT_DF_OFAC_ADD
    .filter(
        PI_POLARS.col('ADD_ADDRESS')
        .str.strip_chars()
        .str.len_chars() > 0
    )
    .with_columns(
        PI_POLARS.col('ADD_ADDRESS')
        .str.replace_all(r'[^0-9a-zA-Z ]', '')
        .str.replace_all(r'\s+', ' ')
        .str.to_uppercase()
        .alias('ZF_ADD_ADDRESS_CLEAN')
    )
)
```

Recommended reading:

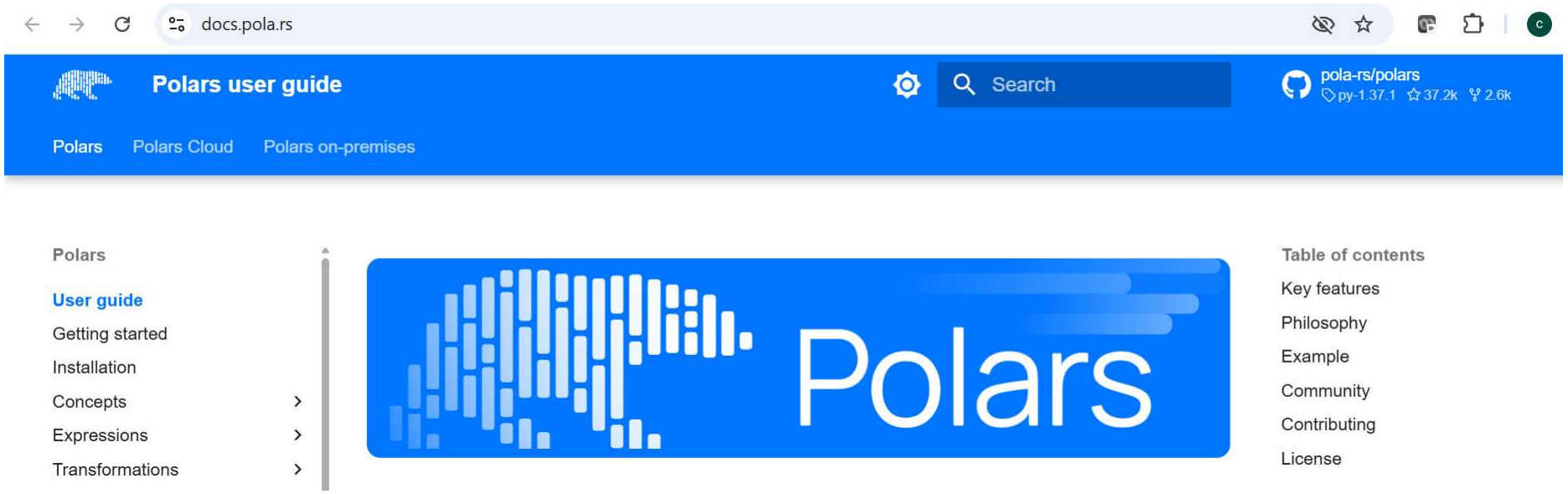
docs.python.org/3/library/re.html



09:45 – 09:50

Polars

Also recommended reading: docs.pola.rs



09:45 – 09:50

.str.split(‘ ’)-> Explode

Add rows for each word

Code example: Obtain one row per word

```
A_LFA1 = (
    A_LFA1
    .filter(
        PI_POLARS.col(
            'LFA1_STRAS'
        ).str.len_chars() > 2)
    .select(
        [
            'LFA1_LIFNR',
            'LFA1_STRAS',
            'ZF_LFA1_STRAS_CLEAN'
        ]
    )
    .with_columns([
        PI_POLARS.col('ZF_LFA1_STRAS_CLEAN')
        .str.split(' ')
        .alias('ZF_LFA1_STRAS_CLEAN_WORD')
    ])
    .explode('ZF_LFA1_STRAS_CLEAN_WORD')
)
```

ZF_LFA1_STRAS_CLEAN	ZF_LFA1_STRAS_CLEAN_WORD
"12 MAIN STREET"	["12", "MAIN", "STREET"]
"AV DA LIBERDADE 45"	["AV", "DA", "LIBERDADE", "45"]

LFA1_LIFNR	ZF_LFA1_STRAS_CLEAN	ZF_LFA1_STRAS_CLEAN_WORD
100001	12 MAIN STREET	12
100001	12 MAIN STREET	MAIN
100001	12 MAIN STREET	STREET
100002	AV DA LIBERDADE 45	AV
100002	AV DA LIBERDADE 45	DA

Notice how explode() is run outside .with_columns()

Join

We do explode so that we can join word-by-word:

Code example: Inner join to filter

```
A_LFA1=(
    A_LFA1
    .join(
        A_OFAC,
        left_on = 'ZF_LFA1_NAME1_WORD',
        right_on = 'ZF_OFAC_SDN_WORD',
        how = 'inner'
    )
)
```

And later we can count matching words

Levenshtein on the full name

We do Levenshtein on full name … we could use rapidfuzz

Rapidfuzz is more intelligent

Levenshtein is easy to explain and predictable

Dr. Levenshtein

Vladimir I. Levenshtein

Biography

A pioneer in the theory of error correcting codes, Dr. Vladimir I. Levenshtein is known as the father of coding theory in Russia. A research professor at the Keldysh Institute for Applied Mathematics at the Russian Academy of Sciences in Moscow, Levenshtein's contributions are present in consumers' everyday lives. His "Levenshtein distance," or "edit distance," is the root of today's spell-checking computer applications; and he has also contributed to the basic technology found in third generation wired cellular telephony.

Dr. Levenshtein has provided the best-known universal bounds to optimal sizes of codes and designs in metric spaces, including the Hamming space and the Euclidean sphere. In particular, they led to the discovery of the long-sought kissing numbers for n=8 and n=24. Dr. Levenshtein authored optimal constructions for several error correcting problems, including: codes that correct a quarter or more of the errors present; codes with a given comma free index; perfect codes able to correct single deletions and single peak shifts; and binary codes with a given probability of undetected error. His work on the universal efficient coding of integers has led to algorithms that offer promising applications in data compression.

The Levenshtein Distance and his designs and bounds are widely used in many engineering, statistics and bioinformatics applications. His recent study into the efficient decoding of information based on the observation of several corrupted copies is expected to have applications in areas as

Vladimir I. Levenshtein

Vladimir I. Levenshtein

Associated organizations

Russian Academy of Sciences

Fields of study

Coding

09:45 – 09:50

Levenshtein on the full name

We import the library and then we use the function from the library –so that we don't need to write the code ourselves about how many characters different the names are

Two different approaches – same result (using zip or map_elements()

Code example: Scoring with Levenshtein

```
import Levenshtein as PI_LEVENSHTEIN

ZV_LI_INV_LEVENSHTEIN = [
        round((1 / (PI_LEVENSHTEIN.distance(ZV_ST_ADDR1, ZV_ST_ADDR2) + 1)) * 100,
2)
        for ZV_ST_ADDR1, ZV_ST_ADDR2 in zip(
            P01_10_05_TT_DF_OFAC_LFA1['ZF_OFAC_ADD_CLEAN'].to_list(),
            P01_10_05_TT_DF_OFAC_LFA1['ZF_LFA1_STRAS_CLEAN'].to_list()
        )
    ]
    P01_10_05_TT_DF_OFAC_LFA1 = (
        P01_10_05_TT_DF_OFAC_LFA1
        .with_columns(
            [
                PI_POLARS.Series('ZF_INV_LEVENSHTEINDIST', ZV_LI_INV_LEVENSHTEIN)
            ]
        )
    )
```

```
464  # Define a function to calculate levenshtein distance between two values
465  def FC_CALCULATE_INV_LEVENSHTEIN_DISTANCE(ZVFCI_VALUE1, ZVFCI_VALUE2):
466      return round((1 / (PI_LEVENSHTEIN.distance(ZVFCI_VALUE1, ZVFCI_VALUE2) + 1)) * 100, 2)
467
468  # Define a function to pass row items to levenshtein distance calculation
469  def FC_MAP_INV_LEVENSHTEIN_DISTANCE(ZVFDI_DI_STRUCT):
470      ZV_LI_STRUCT = list(ZVFDI_DI_STRUCT.values())
471
472      return FC_CALCULATE_INV_LEVENSHTEIN_DISTANCE(
473          ZV_LI_STRUCT[0] or '',
474          ZV_LI_STRUCT[1] or ''
475      )
476
477  # Step 9.1/ Addresses
478  P01_10_15_TT_DF_LFA1ADDR_OFACADD = (
479      P01_10_15_TT_DF_LFA1ADDR_OFACADD
480      .with_columns(
481          [
482              PI_POLARS.lit('Address comparison')
483              .alias('ZF_OFAC_LFA1_COMP_DESC'),
484
485              PI_POLARS.struct(
486                  [
487                      'ZF_ADD_ADDRESS_CLEAN',
488                      'ZF_LFA1_STRAS_CLEAN'
489                  ]
490              )
491              .map_elements(
492                  FC_MAP_INV_LEVENSHTEIN_DISTANCE,
493                  return_dtype=PI_POLARS.Float64
494              )
495              .alias('ZF_INV_LEVENSHTEINDIST'),
```

09:45 – 09:50

Group by to consolidate the results

We use group_by() very often

In the agg() we can use many options.

```
P01_10_15_TT_DF_LFA1ADDR_OFACADD = (
    P01_10_15_TT_DF_LFA1ADDR_OFACADD
    .group_by(
        [
            'LFA1_LIFNR',
            'LFA1_STRAS',
            'ZF_LFA1_STRAS_CLEAN',
            'ADD_ENT_NUM',
            'ADD_ADDRESS',
            'ZF_ADD_ADDRESS_CLEAN'
        ]
    )
    .agg(
        PI_POLARS.col('ZF_OFAC_LFA1_COMP_DESC')
        .first(),

        PI_POLARS.col('ZF_INV_LEVENSHTEINDIST')
        .first(),

        PI_POLARS.col('ZF_LFA1_OFAC_CLEAN_EXACT_MATCH')
        .first(),

        PI_POLARS.col('ZF_LFA1_STRAS_CLEAN_WORD')
        .n_unique()
        .alias('ZF_NUM_WORDS')
    )
    .with_columns(
```

09:45 – 09:50

Group by to consolidate the results

We use concat() very often to concatenate dataframes together

```
# Step 10/ Concatenate the three sets of results into one table
P01_10_18_XT_DF_LFA1_OFAC_SCORING = (
    PI_POLARS.concat(
        [
            P01_10_15_TT_DF_LFA1ADDR_OFACADD,
            P01_10_16_TT_DF_LFA1NAME1_OFACALT,
            P01_10_17_TT_DF_LFA1NAME1_OFACSDN
        ],
        how='diagonal_relaxed'
    )
)
```

Diagonal_relaxed is for when we are not sure the columns are the same

Python structures

A list:

[1, ‘2’,[‘a’,’b’,’c’], 1]

A list is a list of objects: can be different types

A set:

{1, ‘2’,[‘a’,’b’,’c’]}

A set is a list – but no duplicates allowed

An array:

[1, 2,3,4,5]

An array is like a list, but all objects of the same type

A tuple

(1, ‘2’,[‘a’,’b’,’c’], 1)

A tuple is like a list but it cannot be changed or indexed

A dictionary:

{
‘Suppliers’: (‘a’,’b’,’c’)
‘Number’: (1,2,3)
‘Value’:(100,150,300)
}

A dictionary has names that reference objects.

A dataframe:

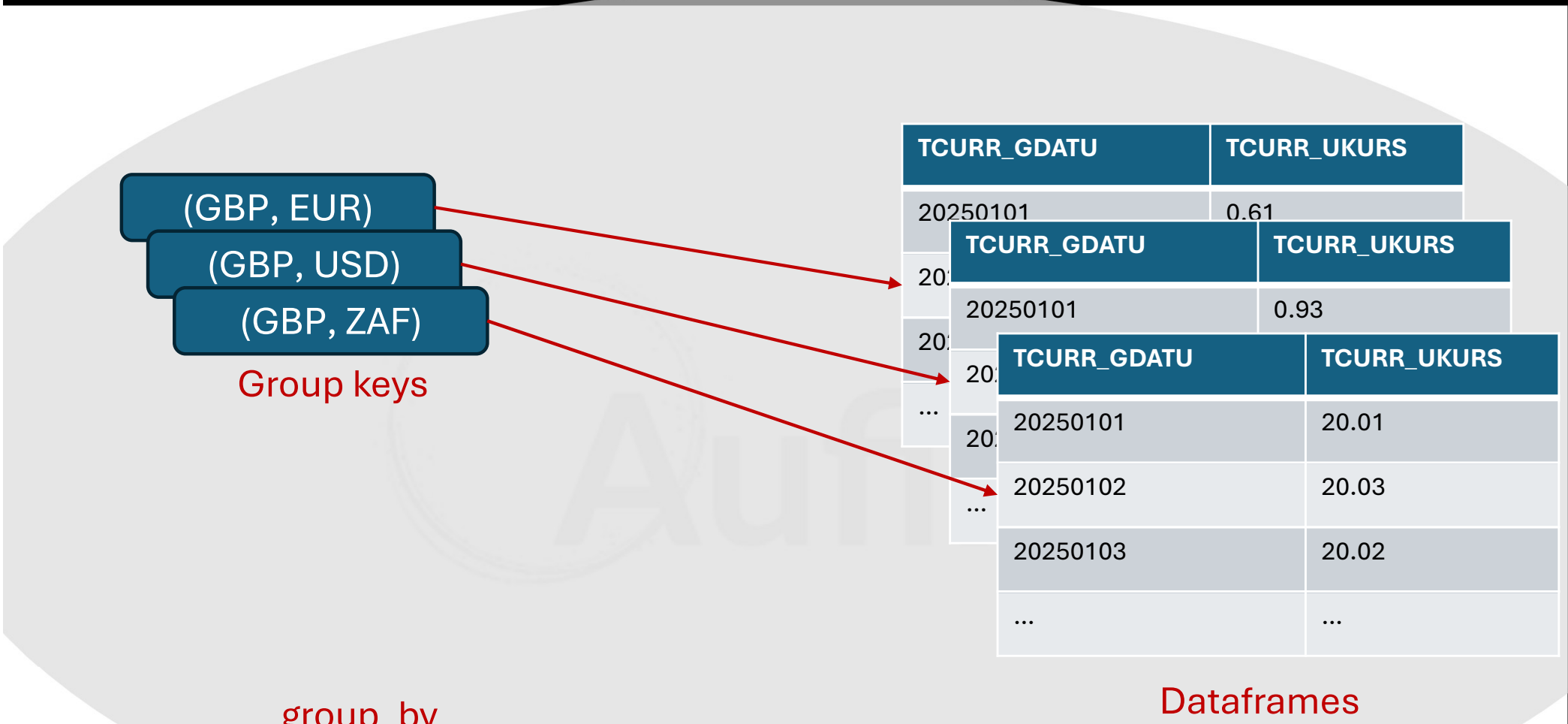
A dataframe (~table) is like a dictionary, where the names are column names and the objects are arrays (have same type within them).

A series:

If we take a column out it is called a series

09:45 – 09:50

Python structures



Python structures

```
group_by([
        Material,
        Month
])
```

3 Sub dataframes

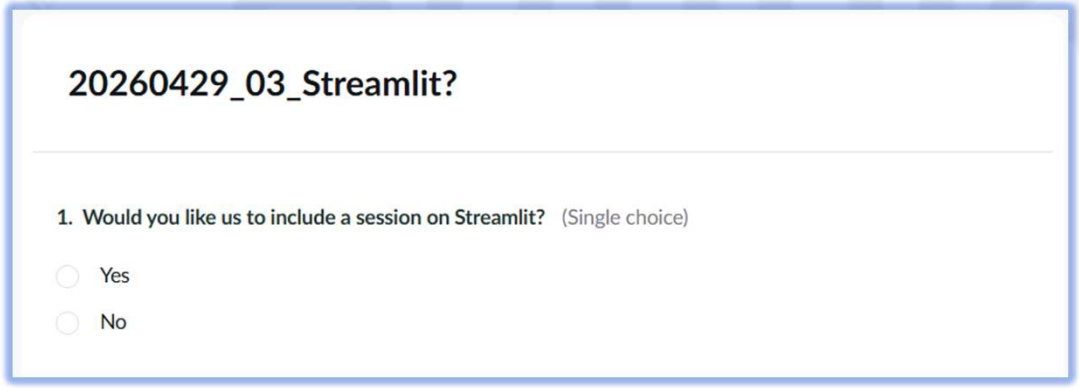
Material	Month	Posting date	Input date	Input time	Document	Item	Quantity	Value
Bolts	Jan 2025	1 Jan 2025	1 Jan 2025	00:01:30	00001234	0001	10	200
Bolts	Jan 2025	2 Jan 2025	2 Jan 2025	00:05:20	00001235	0001	3	60
Bolts	Jan 2025	3 Jan 2025	3 Jan 2025	00:16:10	00001236	0001	2	400
Bolts	Jan 2025	25 Jan 2025	25 Jan 2025	00:01:30	00001237	0001	14	280
Bolts	Feb 2025	1 Feb 2025	1 Jan 2025	00:01:30	00001238	0001	5	1000
Bolts	Feb 2025	2 Feb 2025	1 Jan 2025	00:05:20	00001241	0001	6	1200
Bolts	Feb 2025	3 Feb 2025	1 Jan 2025	00:16:10	00001242	0001	17	3400
Bolts	Feb 2025	25 Feb 2025	1 Jan 2025	00:01:30	00001256	0001	2	400
Bolts	Mar 2025	1 Mar 2025	1 Mar 2025	00:01:30	00001281	0001	21	4200
Bolts	Mar 2025	28 Mar 2025	28 Mar 2025	00:05:20	00001290	0001	32	6400

Streamlit

https://300academy.github.io/STREAMLIT_EXAMPLE/02_PROGRAMMES/

Did you know that you can host your python applications as a web-page for free using Streamlit?

This could be a quick work around for sharing your python analysis with the internal audit team.



09:50 – 09:55


Questions?