Python meeting
06: Python meeting 6

20th May 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

→ P02_18: Purchase orders on slow rotation
→ Assumptions

02

Python script overview

Cum_sum() (reverse), calling a function, group_by(), map_groups()

09:00 – 09:05

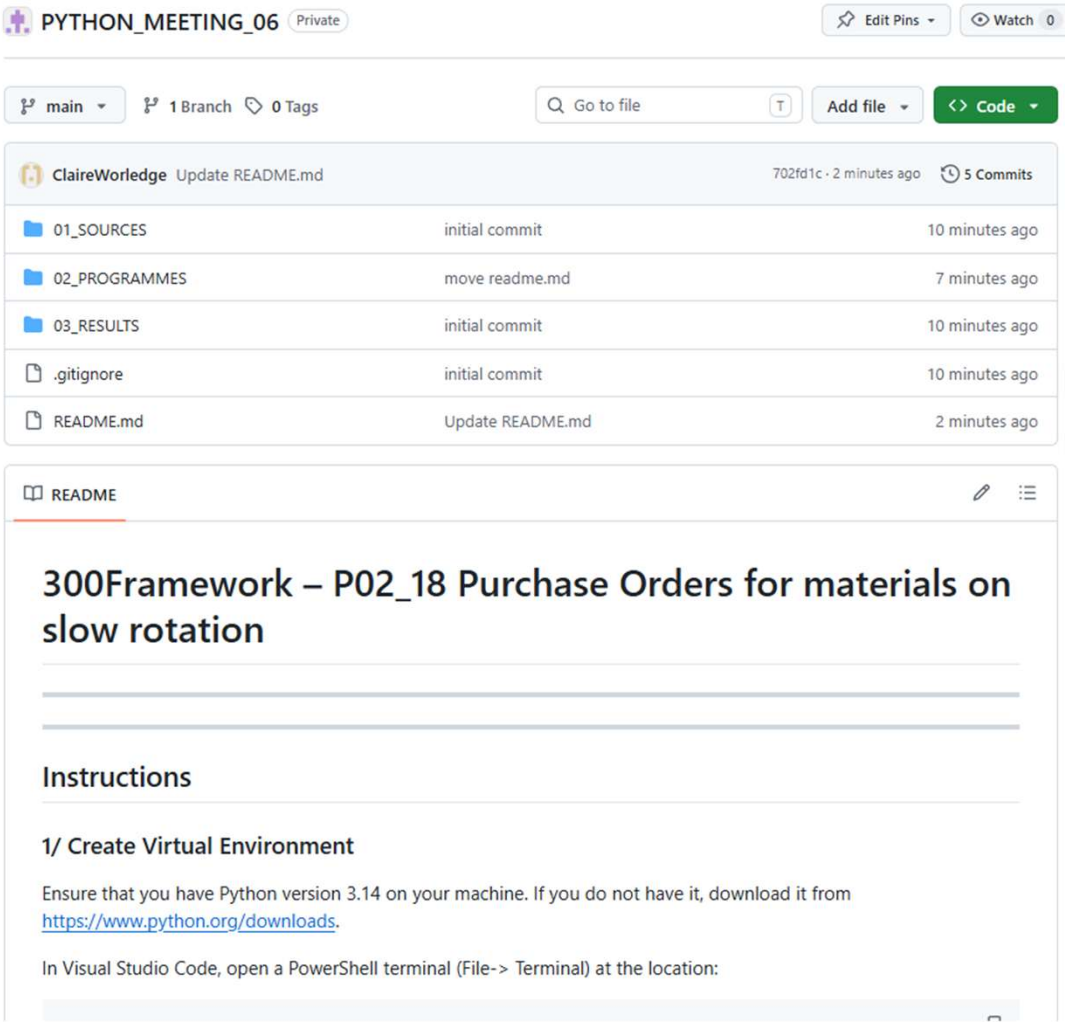
01

Case study: Purchase orders for items on slow rotation

09:00 – 09:05

Before you get started.. README.md

Check out the README.md



New README.md format – instructions at the top.

09:00 – 09:05

Did you do that already?

20260520_01_DidYouRunTheCode

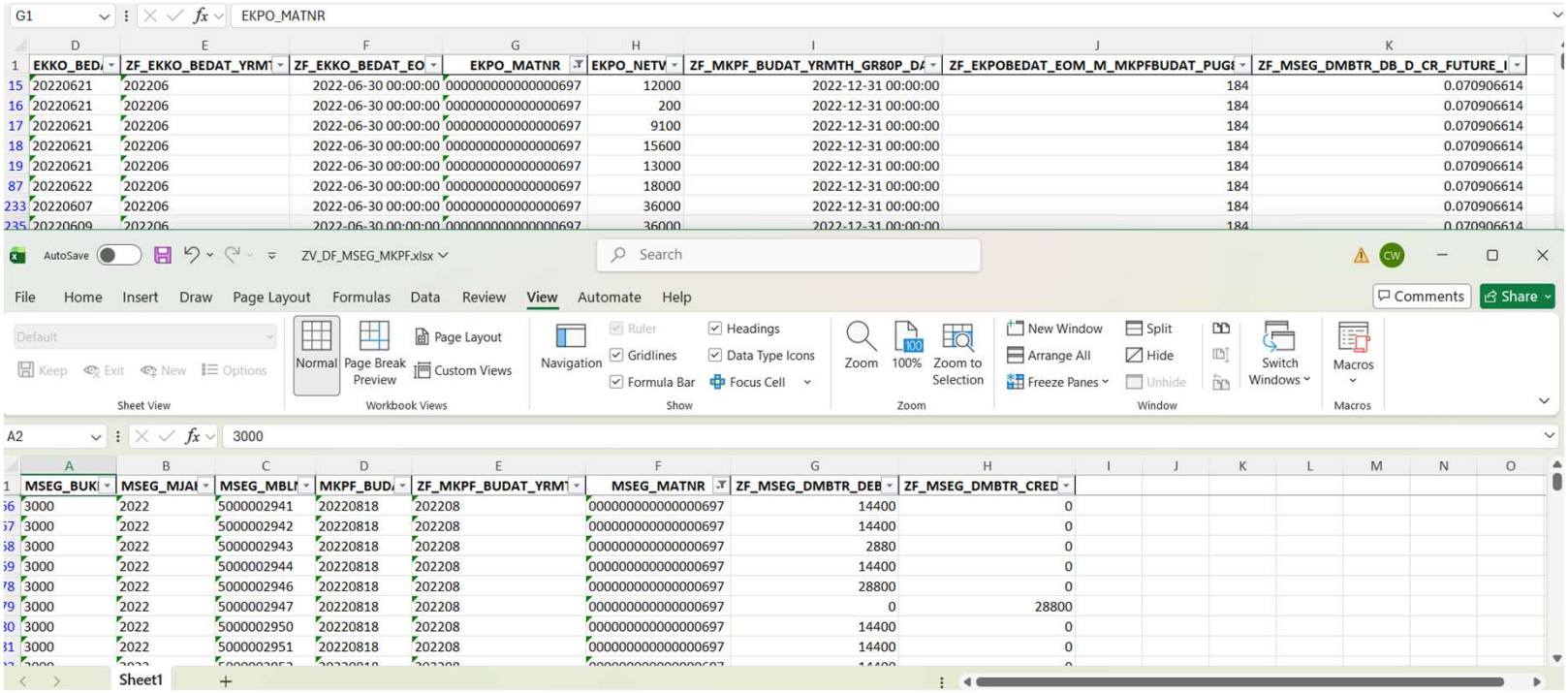
1. Did you manage to run the code already (Single choice)

- ○ Not yet!
- ○ Yes!

09:10 – 09:15

What should the result be?

We output two files so that we can check our result against the original material movements data:



	D	E	F	G	H	I	J	K
1	EKKO_BED	ZF_EKKO_BEDAT_YRM	ZF_EKKO_BEDAT_EO	EKPO_MATNR	EKPO_NETV	ZF_MKPF_BUDAT_YRMTH_GR80P_D	ZF_EKPOBEDAT_EOM_M_MKPFBUDAT_PUG	ZF_MSEG_DMBTR_DB_D_CR_FUTURE_I
15	20220621	202206	2022-06-30 00:00:00	000000000000000697	12000	2022-12-31 00:00:00	184	0.070906614
16	20220621	202206	2022-06-30 00:00:00	000000000000000697	200	2022-12-31 00:00:00	184	0.070906614
17	20220621	202206	2022-06-30 00:00:00	000000000000000697	9100	2022-12-31 00:00:00	184	0.070906614
18	20220621	202206	2022-06-30 00:00:00	000000000000000697	15600	2022-12-31 00:00:00	184	0.070906614
19	20220621	202206	2022-06-30 00:00:00	000000000000000697	13000	2022-12-31 00:00:00	184	0.070906614
87	20220622	202206	2022-06-30 00:00:00	000000000000000697	18000	2022-12-31 00:00:00	184	0.070906614
233	20220607	202206	2022-06-30 00:00:00	000000000000000697	36000	2022-12-31 00:00:00	184	0.070906614
235	20220609	202206	2022-06-30 00:00:00	000000000000000697	36000	2022-12-31 00:00:00	184	0.070906614

	A	B	C	D	E	F	G	H
1	MSEG_BUK	MSEG_MJA	MSEG_MBL	MKPF_BUD	ZF_MKPF_BUDAT_YRM	MSEG_MATNR	ZF_MSEG_DMBTR_DEB	ZF_MSEG_DMBTR_CRED
56	3000	2022	5000002941	20220818	202208	000000000000000697	14400	0
57	3000	2022	5000002942	20220818	202208	000000000000000697	14400	0
58	3000	2022	5000002943	20220818	202208	000000000000000697	2880	0
59	3000	2022	5000002944	20220818	202208	000000000000000697	14400	0
78	3000	2022	5000002946	20220818	202208	000000000000000697	28800	0
79	3000	2022	5000002947	20220818	202208	000000000000000697	0	28800
80	3000	2022	5000002950	20220818	202208	000000000000000697	14400	0
81	3000	2022	5000002951	20220818	202208	000000000000000697	14400	0

Maybe slightly different because we added filter on MBEW

09:15 – 09:30

How do we determine slow rotation?

We will determine slow rotation using the following indicators:

Inactivity Burden Index: current value in stock / total future value stock-out (goods issue… or credit movement)

&

Purchase Usage Gap: number of days from purchase order date until 80% of inventory value is issued

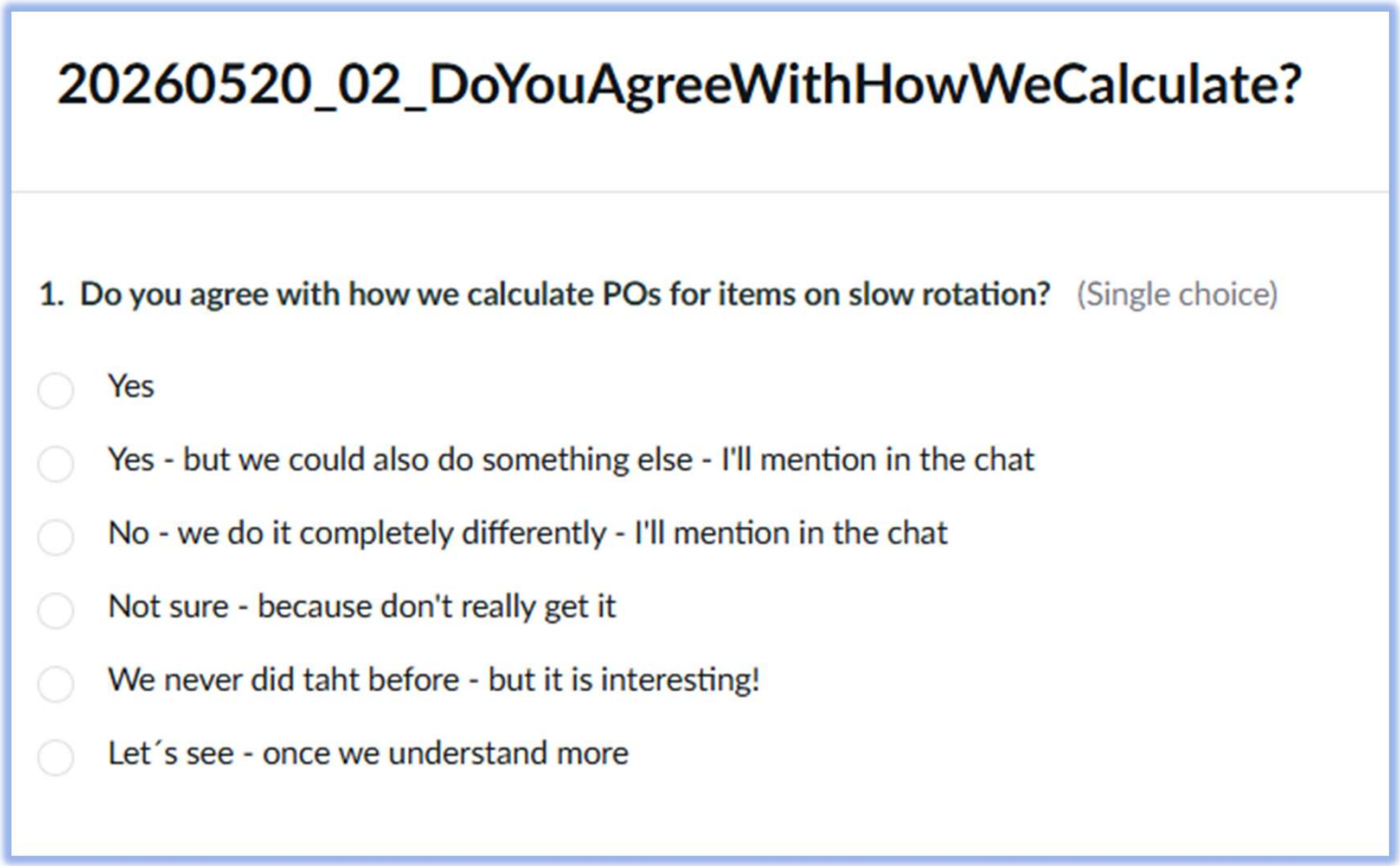
We do the indicators on the material movements and then add them to the POs

```
# 16/ Add the IBI and month next stock out > 80%
ZV_DF_EKPO_EKKO = (
    ZV_DF_EKPO_EKKO
    .join(
        ZV_DF_MSEG_MKPF_TOT_MATNR_MTH,
        left_on=[
            'EKPO_BUKRS',
            'EKPO_MATNR',
            'ZF_EKKO_BEDAT_YRMTH'
        ],
        right_on=[
            'MSEG_BUKRS',
            'MSEG_MATNR',
            'ZF_MKPF_BUDAT_YRMTH'
        ],
        how='left'
```

09:15 – 09:30

How do we know it is on slow rotation?

Do you agree with how we determine slow rotation?



09:15 – 09:30

How do we know it is on slow rotation?

Stock and non-stocked items:

This is why we added a filter on MBEW – took only 1 minute to do

We can only do this test on stocked items

Non-stocked items do not have goods issues

Non-stocked items are consumed immediately

We also need to remember to avoid purchase orders for stock in transit or consumption, or those without a material number

09:15 – 09:30

02

Overview of our python script

Purchase orders for stock on slow rotation

09:30 – 09:45

Filter the purchase orders

After importing our purchase orders we need to filter them:

Non-stocked items:

Those that do not have an account assignment (EKPO_KNTTP is blank).

They do not have an account assignment because the cost account is determined at sale through T030 configuration.

Instead of a cost account at goods receipt, the inventory account is debited.

```
# 5/ Add purchase order detail to header, filter on non-stocked items
ZV_DF_EKPO_EKKO = (
    ZV_DF_EKPO
    .filter(
        (
            PI_POLARS.col('EKPO_KNTTP') # No account assignment: normally stocked
            .str.strip_chars()==''
        ) &
        (
            PI_POLARS.col('EKPO_PSTYP') # Standard purchase order: not consignment, transit, etc.
            .str.strip_chars()=='0'
        ) &
        (
            PI_POLARS.col('EKPO_KZVBR') # Not for consumption
            .str.strip_chars()==''
        ) &
        (
            PI_POLARS.col('EKPO_MATNR')
            .str
            .strip_chars() != ''
        )
    )
    .join(
        ZV_DF_EKKO,
        left_on='EKPO_EBELN',
        right_on='EKKO_EBELN',
        how='inner',
        suffix='_2'
```

09:30 – 09:45

Why not purchase orders for non-stocked items?

20260520_03_WhyDoWeNotIncludeNonStockedItems?

1. **Why do we not include non-stocked items?** (Single choice)

- ○ Because non-stocked items are theoretically consumed immediately or input to fixed assets and we do not expect them to be issued later as a goods issue
- ○ Because non-stocked items do not have any value

09:30 – 09:45

Create columns: year and month

Slice(0,6) – means take the characters from position 0 for 6.

We could also write slice(:6) which would mean the same – because 0 is the default.

```
    .with_columns(
        [
            PI_POLARS.col('EKKO_BEDAT')
            .str
            .slice(0,6)
            .alias('ZF_EKKO_BEDAT_YRMTH'),

            PI_POLARS.col('EKKO_BEDAT')
            .str
            .slice(0,4)
            .alias('ZF_EKKO_BEDAT_YR'),
        ]
    )
)
```

09:30 – 09:45

Unique list of materials for filtering

Here we make a unique list that we will use later for filtering the material movements.

```
# 6/ List of material numbers
ZV_DF_EKPO_EKKO_LIST_MATNR = (
    ZV_DF_EKPO_EKKO
    .select(
            'EKPO_MATNR'
    )
    .unique()
)
```

Filter on the list of materials

Here we filter the material movements on the materials that are found in our purchase orders that are for stocked materials.

We also want to make sure that we do not include material movements that do not mention the material.

```
ZV_DF_MSEG_MKPF = (
    ZV_DF_MSEG_MKPF
    .filter(
        (
            PI_POLARS.col('MSEG_MATNR')
            .str
            .strip_chars() != ''
        ) &
        (
            PI_POLARS.col('MSEG_MATNR')
            .is_in(
                ZV_DF_EKPO_EKKO_LIST_MATNR[
                    'EKPO_MATNR'
                ]
            )
        )
    )
```

09:30 – 09:45

Why consider cases for which there are no movements?

20260520_04_WhatAreWeMissingIfWeOnlyConsiderMove

1. What are we missing if we only consider movements for a certain period? (Multiple choice)

- [ ] The materials that are in a different plant
- [ ] The materials that did not have a purchase order
- [ ] The materials that had a purchase order but that did not have a goods receipt
- [ ] The materials that had a purchase order but no goods receipt and also have a high-value on stock

09:30 – 09:45

Conditional fields

Here we create conditional fields using .when().then.otherwise()

```
            PI_POLARS.when(
                PI_POLARS.col('MSEG_SHKZG') == 'S'
            )
            .then(
                PI_POLARS.col('MSEG_MENGE')
            )
            .when(
                PI_POLARS.col('MSEG_SHKZG') == 'H'
            )
            .then(
                PI_POLARS.col('MSEG_MENGE')*-1
            )
            .otherwise(
                PI_POLARS.lit(0)
            )
            .alias('ZF_MSEG_MENGE_SIGNED'),
        ]
```

Group_by() with agg()

A simple group_by().

When we do .agg() the group_by() materializes into a dataframe.

```
ZV_DF_MSEG_MKPF_TOT_MATNR_MTH = (
    ZV_DF_MSEG_MKPF
    .group_by(
        [
            'MSEG_BUKRS',
            'MSEG_MATNR',
            'ZF_MKPF_BUDAT_YRMTH'
        ]
    )
    .agg(
        [
            PI_POLARS.col('ZF_MSEG_DMBTR_DEBIT')
            .sum()
            .alias('ZF_MSEG_DMBTR_DEBIT'),

            PI_POLARS.col('ZF_MSEG_DMBTR_CREDIT')
            .sum()
            .alias('ZF_MSEG_DMBTR_CREDIT'),
```

Balance at the end of the month

Within the same group_by()…

A group_by() can be considered as a dataframe that is divided up into many mini dataframes.

Here we see that we do a sort on these mini dataframes.

We get the last stock-on-hand before movement for our sort criteria and we add it to the last movement quantity.

```
            (
                (
                    PI_POLARS.col('MSEG_LBKUM')
                    .sort_by(
                        [
                            'MKPF_BUDAT',
                            'MKPF_CPUDT',
                            'MKPF_CPUTM',
                            'MSEG_MBLNR',
                            'MSEG_ZEILE'
                        ]
                    ).last()
                ) +
                (
                    PI_POLARS.col('ZF_MSEG_MENGE_SIGNED')
                    .sort_by(
                        [
                            'MKPF_BUDAT',
                            'MKPF_CPUDT',
                            'MKPF_CPUTM',
                            'MSEG_MBLNR',
                            'MSEG_ZEILE'
                        ]
                    ).last()
                )
            )
            .alias('ZF_MSEG_LBKUMMENGE_EOM'),
```

09:30 – 09:45

Calling another python script

20260520_05_WhatDoYouThinkItMeans?

1. What do you think ZF_MSEG_SALK3DMBTR_EOM means? (Single choice)

- ○ It is the total value received for the material to-date
- ○ It is the total goods receipts for the material in the current month
- ○ It is the total value in stock of the material at the end of the month
- ○ It is the total of the stock movements for the material for the current month

Reverse cum_sum(): what is the value of goods issues after the current month?

Normal cum_sum:

Month	Credit	Cum_sum
202401	10	10
202402	20	30
202403	30	60
202404	40	100

Reverse cum_sum():

Month	Credit	Cum_sum
202401	10	100
202402	20	90
202403	30	70
202404	40	40

```
ZV_DF_MSEG_MKPF_TOT_MATNR_MTH = (
    ZV_DF_MSEG_MKPF_TOT_MATNR_MTH
    .sort(
        by = [
            'MSEG_BUKRS',
            'MSEG_MATNR',
            'ZF_MKPF_BUDAT_YRMTH'
        ]
    )
    .with_columns(
        [
            PI_POLARS.col('ZF_MSEG_DMBTR_CREDIT')
            .cum_sum(reverse=True)
            .over(
                [
                    'MSEG_BUKRS',
                    'MSEG_MATNR'
                ]
            )
            .alias('ZF_MSEG_DMBTR_CREDIT_FUTURE')
        ]
    )
)
```

09:30 – 09:45

Function to find the month at which we have reached >80% value as goods issue

Here we create a function.

Sort on year month (we will run on mini dataframes per company and material)

Convert columns to lists

Create a list for storing the month we find

For each month, walk along the coming months… cumulating the credit movements as you go… until the total credit movements is at least 80% the value that was in stock

Remember the month at which you got to at last 80%.. Or if not found the month of the date at which the data was extracted.

Conver the list of months found to a new column

```python
# 13 Function: Find the next month when stock out (80% or greater current )
def FC_P02_18_FIND_NEXT_MONTH_CREDIT(ZVFCI_DF_GROUP):

    ZVFCI_DF_GROUP = (
        ZVFCI_DF_GROUP
        .sort('ZF_MKPF_BUDAT_YRMTH')
    )

    ZV_LI_CREDIT = ZVFCI_DF_GROUP['ZF_MSEG_DMBTR_CREDIT'].to_list()
    ZV_LI_SALK3DMBTREOM = ZVFCI_DF_GROUP['ZF_MSEG_SALK3DMBTR_EOM'].to_list()
    ZV_LI_MONTHS = ZVFCI_DF_GROUP['ZF_MKPF_BUDAT_YRMTH'].to_list()

    ZV_LI_NEXT_MONTHS = []

    for ZV_IDX_01 in range(len(ZV_LI_MONTHS)):
        ZV_ST_MONTH_FOUND = None
        ZV_NU_CREDIT_CUMUL_IDX02 = 0
        for ZV_IDX_02 in range(ZV_IDX_01, len(ZV_LI_MONTHS)):
            ZV_NU_CREDIT_CUMUL_IDX02= ZV_NU_CREDIT_CUMUL_IDX02 + ZV_LI_CREDIT[ZV_IDX_02]
            if ZV_NU_CREDIT_CUMUL_IDX02 > (0.8 * ZV_LI_SALK3DMBTREOM[ZV_IDX_01]):
                ZV_ST_MONTH_FOUND = ZV_LI_MONTHS[ZV_IDX_02]
                break

        ZV_LI_NEXT_MONTHS.append(
            ZV_ST_MONTH_FOUND
            if ZV_ST_MONTH_FOUND
            else ZV_ST_EXTRACTION_DATE[:6]
        )

    return ZVFCI_DF_GROUP.with_columns(
        PI_POLARS.Series(
            name='ZF_MKPF_BUDAT_YRMTH_GR80P',
            values=ZV_LI_NEXT_MONTHS
        )
    )
```

09:30 – 09:45

Which approach would you take?

20260520_DoYouDisagree?

1. **Do you disagree with any of these?** (Multiple choice)

- [ ] Tick to disagree: We should compare the pruchase order date tot he next goods issue date
- [ ] Tick to disagree: We should find the month by which at least 80% of the value of the goods receipt was issued (cumulative goods issues going forward)
- [ ] Tick to agree: We should compare the goods receipt date to the month where the goods issued for that month is 80% the goods receipt value
- [ ] Tick to agree: We should compare to the purchase order value rather than the goods receipt value

09:30 – 09:45

Using map_groups() to apply a function to a group

Use .map_groups() to call the function.

Just as .agg() is a function that runs on each mini dataframe in the group_by()… map_groups() runs any custom function on the mini-dataframes.

map_groups() concatenates the results of all of the mini dataframes back into one dataframe.

As with agg(), map_groups() materializes the group_by() into a dataframe.

```
ZV_DF_MSEG_MKPF_TOT_MATNR_MTH = (
    ZV_DF_MSEG_MKPF_TOT_MATNR_MTH
    .group_by(
        'MSEG_BUKRS',
        'MSEG_MATNR'
    )
    .map_groups(
        FC_P02_18_FIND_NEXT_MONTH_CREDIT
    )
)
```

Inventory Burden Index (IBI)

Here we compute the simple Inventory Burden Index, using the future goods issue value.

```
ZV_DF_MSEG_MKPF_TOT_MATNR_MTH = (
    ZV_DF_MSEG_MKPF_TOT_MATNR_MTH
    .with_columns(
        (
            PI_POLARS.when(
                (
                    PI_POLARS.col('ZF_MSEG_DMBTR_DEBIT') > 0
                ) &
                (
                    PI_POLARS.col('ZF_MSEG_DMBTR_CREDIT_FUTURE') > 0
                )
            )
            .then(
                PI_POLARS.col('ZF_MSEG_DMBTR_DEBIT')
                / PI_POLARS.col('ZF_MSEG_DMBTR_CREDIT_FUTURE')
            )
            .otherwise(
                PI_POLARS.lit(0)
            )
        )
        .alias('ZF_MSEG_DMBTR_DB_D_CR_FUTURE_IBI')
    )
)
```

Add the indicators back to the purchase orders

We add the indicators to the purchase orders based on the company code, material and month.

```
ZV_DF_EKPO_EKKO = (
    ZV_DF_EKPO_EKKO
    .join(
        ZV_DF_MSEG_MKPF_TOT_MATNR_MTH,
        left_on=[
            'EKPO_BUKRS',
            'EKPO_MATNR',
            'ZF_EKKO_BEDAT_YRMTH'
        ],
        right_on=[
            'MSEG_BUKRS',
            'MSEG_MATNR',
            'ZF_MKPF_BUDAT_YRMTH'
        ],
        how='left'
    )
)
```

Calculate the PUG

We can now calculate the PUG because we know the date at which the cumulated goods issues reached 80% of the value in stock at the time of the purchase order.

```
ZV_DF_EKPO_EKKO = (
    ZV_DF_EKPO_EKKO
    .with_columns([
        (
            PI_POLARS.col('ZF_EKKO_BEDAT_YRMTH')
            .cast(PI_POLARS.Utf8)
            .str.strptime(PI_POLARS.Date, '%Y%m',strict=False)
            .dt.month_end()
        )
        .alias('ZF_EKKO_BEDAT_EOM'),

        (
            PI_POLARS.col('ZF_MKPF_BUDAT_YRMTH_GR80P')
            .cast(PI_POLARS.Utf8)
            .str.strptime(PI_POLARS.Date, '%Y%m',strict=False)
            .dt.month_end()
        )
        .alias('ZF_MKPF_BUDAT_YRMTH_GR80P_DAY')
    ])
    .with_columns([
        (
            PI_POLARS.col('ZF_MKPF_BUDAT_YRMTH_GR80P_DAY')
            - PI_POLARS.col('ZF_EKKO_BEDAT_EOM')
        )
        .dt.total_days()
        .alias('ZF_EKPOBEDAT_EOM_M_MKPFBUDAT_PUG80')
    ])

)
```

Filter on indicators

Finally, we can do a simple filter on the indicators that we created.

The results are filtered based on the parameters that we set at the beginning.

```
ZV_DF_EKPO_EKKO_HIGHPUG_OR_LOWIBI = (
    ZV_DF_EKPO_EKKO
    .filter([
        (
            PI_POLARS.col('ZF_EKPOBEDAT_EOM_M_MKPFBUDAT_PUG80')
            > ZV_NU_P02_18_PURCHASETOUSAGEGAP
        ) |
        (
            PI_POLARS.col('ZF_MSEG_DMBTR_DB_D_CR_FUTURE_IBI')
            > ZV_NU_P02_18_INACTIVITYBURDENINDEX
        )
    ])
)
```

What would influence our parameters?

20260520_07_WhatWouldInfluenceOurParameters?

1. **What would influence our parameters?** (Multiple choice)

- [ ] PUG: It would be influenced by the type of industry (FMCG, manufacturing, etc)
- [ ] PUG: We should compare it to what the internal control guidelines say
- [ ] IBI: It would be influenced by the type of industry
- [ ] IBI: It would be influenced by how expensive the warehouse cost is
- [ ] IBI: We should compare it to what the internal control guidelines say
- [ ] PUG and IBI: We need to be able to filter by material group
- [ ] PUG and IBI: We need to be able to filter by plant code
- [ ] PUG and IBI: We would need to be able to filter by material number

09:30 – 09:45


Questions?