# SAP data analytics master class
# Regex and Sklearn for unexpected travel and expense

**24th June 2026 – 9AM EST**

# Regex and Sklearn matching – case study: Unusual travel and expense

# WHAT WILL WE COVER TODAY?

01

*Obtain list of invoices and purchases – recap*
***Import***
***Concatenate***
***Join***
***Create fields***
***Group by - .agg – make a list***
***Filter***
***List from a file***

02

*HR03_12: Expenses entered as purchases*
***Using regex***

03

*HR03_13: Misclassified expenses*
***Using Cosine***

01

# Obtain a list of invoices - recap

# Have you already tried?

## 20260624_01: Have you already tried?

1. Have you already tried making the accounting schemes cube? (Single choice)

- ○ Yes
- ○ No

# What is it for?

## 20260624_02: What is it for?

**1. What is it for?** (Matching)

| | | |
|---|---|---|
| A. Group_by() | A. | Select an option |
| B. join() | B. | Select an option |
| C. concat() | C. | Select an option |
| D. select() | D. | Select an option |
| E. .agg() | E. | Select an option |
| F. .with_columns() | F. | Select an option |
| G. .str.join() | G. | Select an option |

# Obtain a list of invoices - recap

Import the functions that we put in Z_SHARED_FUNCTIONS:

```
# Create the venv: python -m venv venv  ... (Yes)
# Activate the venv: .\venv\Scripts\activate
# Run the requirements: pip install -r requirements.txt
# Set the variables (remember…C:\... Becomes …C:/ …  or use r'C:\.
# Run the program
# Check the output

# 1/ Import python libraries
import polars as PI_POLARS
import regex as PI_RE
import numpy as PI_NUMPY

# 2/ Import shared programs
from Z_SHARED_FUNCTIONS.FC_IMPORT import FC_IMPORT_TEXT
from Z_SHARED_FUNCTIONS.FC_EXPORT import FC_EXPORT_EXCEL
from Z_SHARED_FUNCTIONS.FC_BUILD_TFIDF_AND_COSINE_BEST_MATCH impor
from Z_SHARED_FUNCTIONS.FC_PREPROCESS_TEXT import FC_PREPROCESS_TE
```

# Obtain a list of invoices - recap

Now we have quite a lot of files!

```
# 3/ Set variables
ZV_ST_SOURCES_FOLDER = r'C:\300FMASTERCLASS\06_PYTHON_MEETING_MC2
ZV_ST_RESULTS_FOLDER = r'C:\300FMASTERCLASS\06_PYTHON_MEETING_MC2
ZV_ST_NAME_BSIK_FILE = '20251110_153943_1_BSIK.csv'
ZV_ST_NAME_BSAK_FILE = '20251110_153943_1_BSAK.csv'
ZV_ST_NAME_T001_FILE = '20251025_170112_1_T001.csv'
ZV_ST_NAME_BKPF_FILE = '20250911_172412_1_BKPF.csv'
ZV_ST_NAME_BSEG_FILE = '20250910_113032_1_BSEG.csv'
ZV_ST_NAME_SKAT_FILE = '20240513_152943_1_SKAT.csv'
ZV_ST_NAME_SKA1_FILE = '20240513_152943_1_SKA1.csv'
ZV_ST_NAME_SKB1_FILE = '20240513_152943_1_SKB1.csv'
ZV_ST_NAME_T003T_FILE = '20240510_152615_1_T003T.csv'
ZV_ST_NAME_KEYWORD_FILE = 'AM_KEY_WORDS.txt'
ZV_ST_NAME_T706B5_FILE = '20260325_103759_1_T706B5.csv'
```

# Obtain a list of invoices - recap

We can quickly do the import by calling our function

```
def HR12_MISSCLASSIFIED_EXPENSES():

    # 4/ Import the files
    ZV_LI_NUM_COLS = [
        'BSAIK_DMBTR'
    ]
    ZV_DF_BSIK = FC_IMPORT_TEXT(ZV_ST_NAME_BSIK_FILE, ZV_ST_SOUR

    ZV_LI_NUM_COLS = [
        'BSAIK_DMBTR'
    ]
    ZV_DF_BSAK = FC_IMPORT_TEXT(ZV_ST_NAME_BSAK_FILE, ZV_ST_SOUR

    ZV_LI_NUM_COLS = []
    ZV_DF_T001 = FC_IMPORT_TEXT(ZV_ST_NAME_T001_FILE, ZV_ST_SOUR
```

# Obtain a list of invoices - recap

Concatenate BSIK and BSAK into one table

```
ZV_DF_BSAIK = (
    PI_POLARS.concat(
        [
            ZV_DF_BSIK,
            ZV_DF_BSAK
        ]
    )
)
```

# Obtain a list of invoices - recap

Join BSEG and BKPF

```
ZV_DF_BSEGBKPF = (
    ZV_DF_BSEG
    .join(
        (
            ZV_DF_BKPF
            .filter(
                PI_POLARS.col('BKPF_BSTAT')==''
            )
        ),
        left_on=[
            'BSEG_BUKRS',
            'BSEG_GJAHR',
            'BSEG_BELNR'
        ],
        right_on = [
            'BKPF_BUKRS',
            'BKPF_GJAHR',
            'BKPF_BELNR'
        ],
        how='inner',
        coalesce=False
    )
)
```

# Obtain a list of invoices - recap

Add other information:

```
ZV_DF_BSEGBKPF = (
    ZV_DF_BSEGBKPF
    .join(
        ZV_DF_T001,
        left_on = 'BKPF_BUKRS',
        right_on = 'T001_BUKRS',
        how = 'inner'
    )
    .join(
        (
            ZV_DF_SKAT
            .filter(
                (PI_POLARS.col('SKAT_SPRAS') == 'E') |
                (PI_POLARS.col('SKAT_SPRAS') == 'EN')
            )
        ),
        left_on = [
            'T001_KTOPL',
            'BSEG_HKONT'
        ],
        right_on = [
            'SKAT_KTOPL',
            'SKAT_SAKNR'
        ],
        how = 'inner'
    )
```

# Obtain a list of invoices - recap

Concatenate text description fields together:

```
ZV_DF_BSEGBKPF_ACC_SCHEMES = (
    ZV_DF_BSEGBKPF
    .with_columns(
        PI_POLARS.concat_str(
            [
                PI_POLARS.col('BSEG_HKONT')
                .str.strip_chars(),

                PI_POLARS.col('SKAT_TXT20')
                .str.strip_chars()
            ],
            separator='-'
        )
        .alias('ZF_BSEGHKONT_SKATTXT20'),
```

# Obtain a list of invoices - recap

Select only certain fields:

```
        .select(
            [
                'BSEG_BUKRS',
                'BSEG_GJAHR',
                'BSEG_BELNR',
                'BSEG_SHKZG',
                'BSEG_DMBTR',
                'ZF_BSEGHKONT_SKATTXT20',
                'ZF_BKPFBLART_T003TLTEXT',
                'SKA1_GVTYP',
                'SKB1_XGKON',
                'SKB1_HBKID',
                'BSEG_KOART',
                'BSEG_KTOSL',
                'BKPF_BKTXT'
            ]
        )
```

# Obtain a list of invoices - recap

Add up the value on debit… journal entry amount

```
    .group_by(
        [
            'BSEG_BUKRS',
            'BSEG_GJAHR',
            'BSEG_BELNR',
            'BKPF_BKTXT'
        ]
    )
    .agg(
        [
            PI_POLARS.when(
                PI_POLARS.col('BSEG_SHKZG') == 'S'
            )
            .then(
                PI_POLARS.col('BSEG_DMBTR')
            )
            .otherwise(
                PI_POLARS.lit(0)
            )
            .sum()
            .alias('ZF_BSEG_DMBTR'),
```

# Obtain a list of invoices - recap

Create list of accounts and account indicators on debit and credit

```
PI_POLARS.col('ZF_BSEGHKONT_SKATTXT20')
.filter(
    PI_POLARS.col('BSEG_SHKZG') == 'S'
)
.sort_by(
    PI_POLARS.col('BSEG_DMBTR')
    .filter(
        PI_POLARS.col('BSEG_SHKZG') == 'S'
    ),
    descending=True
)
.head(10)
.str.join('_')
.alias('ZF_BSEG_HKONTSKAT_DEBIT'),

PI_POLARS.col('ZF_BSEGHKONT_SKATTXT20')
.filter(
    PI_POLARS.col('BSEG_SHKZG') == 'H'
)
.sort_by(
    PI_POLARS.col('BSEG_DMBTR')
    .filter(
        PI_POLARS.col('BSEG_SHKZG') == 'H'
    ),
    descending=True
)
.head(10)
.str.join('_')
.alias('ZF_BSEG_HKONTSKAT_CREDIT'),
```

# Obtain a list of invoices - recap

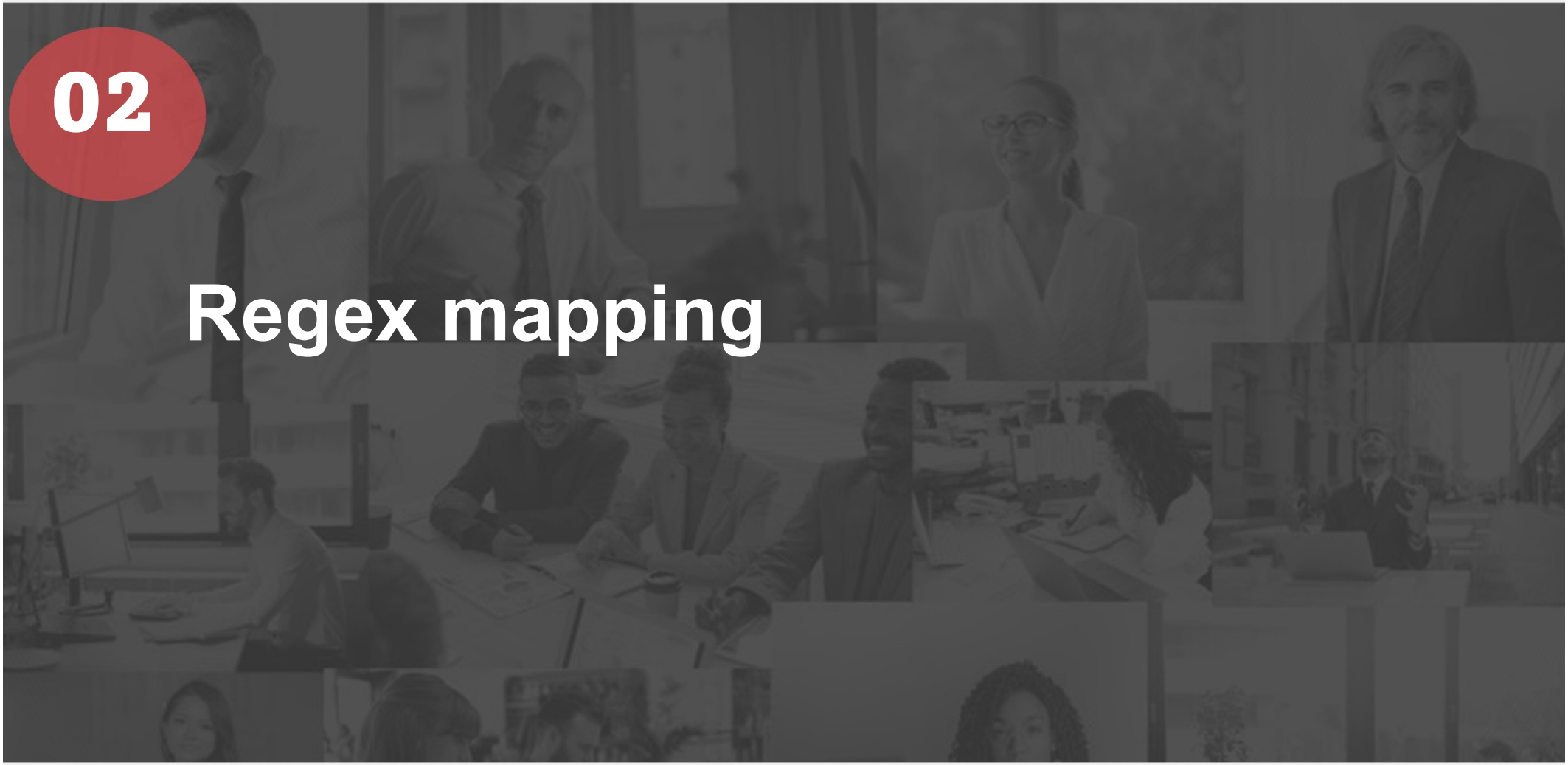
Add the list of accounts back to the general ledger

```
ZV_DF_BSAIK = (
    ZV_DF_BSAIK
    .join(
        ZV_DF_BSEGBKPF_ACC_SCHEMES,
        left_on=[
            'BSAIK_BUKRS',
            'BSAIK_GJAHR',
            'BSAIK_BELNR'
        ],
        right_on=[
            'BSEG_BUKRS',
            'BSEG_GJAHR',
            'BSEG_BELNR'
        ],
        how='inner'
    )
)
```

# Obtain a list of invoices - recap

Create buckets – rules for determining which supplier movements relate to supplier invoices and which relate to supplier payments… and other transactions – for example, foreign exchange adjustments

```
ZV_DF_BSAIK = (
    ZV_DF_BSAIK
    .with_columns(
        [
            PI_POLARS.when(
                (
                    (
                        PI_POLARS.col('ZF_BSEG_KOART_DEBIT')
                        .str.split('_')
                        .list.first()
                    ) == 'S'
                ) &
                (
                    (
                        (
                            PI_POLARS.col('ZF_SKA1_GVTYP_DEBIT')
                            .str.split('_')
                            .list.first()
                            .str.strip_chars()
                        ) != ''
                    ) |
                    (
                        (
                            PI_POLARS.col('ZF_BSEG_KTOSL_DEBIT')
                            .str.split('_')
                            .list.first()
                        ) == 'WRX'
                    )
                ) &
```


02
Regex mapping

# Import the list of key words

Now we can obtain a list of key words from our manual list of key words

```
ZV_LI_KEY_WORDS = [
    str(k)
    .strip()
    for k in ZV_DF_AM_KEYWORDS['AMKW_KEY_WORD']
    if k is not None and str(k).strip()!=''
]
```

# Next we make a Regex pattern match

Make a regex pattern match from the list of key words (HR03_12):

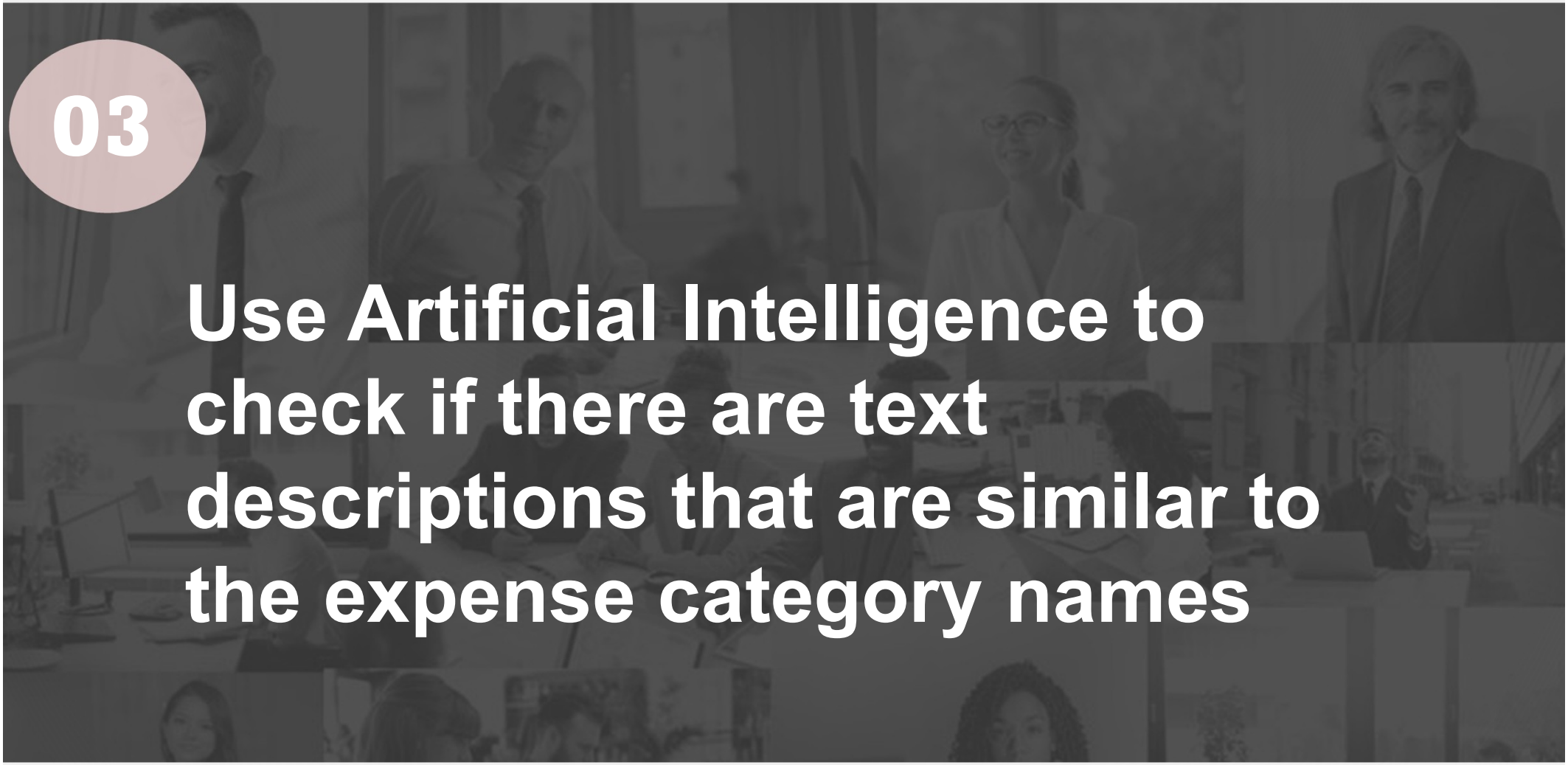
```
if not ZV_LI_KEY_WORDS:
    ZV_RE_FILTER_PATTERN = r'(?!)'
else:
    ZV_RE_KEY_WORDS = '|'.join(PI_RE.escape(k) for k in ZV_LI_KEY_WORDS)
    ZV_RE_FILTER_PATTERN = f'(?i)(?:^|[^A-Za-z0-9_])(?:{ZV_RE_KEY_WORDS})(?:$|[^A-Za
```

- Ignore case (?i)
- If it does not have a word character before it ... ^|[^A-Za-z0-9_] ... and it does not have a word character after it $|[^A-Za-z0-9_]
- .. or in other words, find me all the cases where the word is found and it is not part of another word.
- ... for example, if my word is "tool", find it where you see " tool " but not where you see "machinetool"

# Use the regex pattern…

Then we use .str.contains … with literal = False to search for our key words… and filter out supplier invoices that may be travel and expenses

```
ZV_DF_BSAIK_INVOICES_TANDE = (
    ZV_DF_BSAIK_INVOICES
    .filter(
        (
            (
                PI_POLARS.col('BSAIK_SGTXT')
                .str.contains(ZV_RE_FILTER_PATTERN, literal=False)
            )
            (
                PI_POLARS.col('BKPF_BKTXT')
                .str.contains(ZV_RE_FILTER_PATTERN, literal=False)
            )
        )
    )
)
```



03

# Use Artificial Intelligence to check if there are text descriptions that are similar to the expense category names

# Obtain a list of travel and expense category names SAP

Better than having to invent our manual file!

```
ZV_LI_T706B5_SPTXT = (
    ZV_DF_T706B5
    .filter(
        (PI_POLARS.col('T706B5_SPRAS') == 'E') |
        (PI_POLARS.col('T706B5_SPRAS') == 'EN')
    )
    .unique(subset='T706B5_SPTXT')
    .get_column('T706B5_SPTXT')
    .to_list()
)
```

# Obtain a list of travel and expense category names SAP

Obtain a list of words from our text descriptions:

```
ZV_LI_BSAIKSGTXT_BKPFBKTXT = (
    (
        ZV_DF_BSAIK_INVOICES
        .select(
            [
                'ZF_BSAIKSGTXT_BKPFBKTXT'
            ]
        )
        .filter(
            (
                (
                    PI_POLARS.col('ZF_BSAIKSGTXT_BKPFBKTXT')
                    .fill_null('')
                    .str.strip_chars() != ''
                )
            )
        )
        .unique()
        .get_column('ZF_BSAIKSGTXT_BKPFBKTXT')
        .to_list()
    )
)
```

# Obtain a list of travel and expense category names SAP

Use a function to clean-up the list of words for both lists:

```
# 14/ Process text to new lists
ZV_LI_T706B5_SPTXT_PREP = []
for ZV_ST_TXT in ZV_LI_T706B5_SPTXT:
    ZV_LI_T706B5_SPTXT_PREP.append(
        FC_PREPROCESS_TEXT(ZV_ST_TXT)
    )

ZV_LI_BSAIKSGTXT_BKPFBKTXT_PREP = []
for ZV_ST_TXT in ZV_LI_BSAIKSGTXT_BKPFBKTXT:
    ZV_LI_BSAIKSGTXT_BKPFBKTXT_PREP.append(
        FC_PREPROCESS_TEXT(ZV_ST_TXT)
    )
```

```
def FC_PREPROCESS_TEXT(ZVFCI_ST_TXT: str) -> str:

    if ZVFCI_ST_TXT is None:
        return ''

    ZV_ST_TXT = str(ZVFCI_ST_TXT).strip().lower()

    ZV_ST_TXT = ZV_ST_TXT.replace('/', ' ').replace('-', ' ').repl

    ZV_ST_TXT = PI_RE.sub(r'[^a-z0-9\s]', ' ', ZV_ST_TXT)

    ZV_ST_TXT = PI_RE.sub(r'\s+', ' ', ZV_ST_TXT).strip()

    return ZV_ST_TXT
```

# Obtain a list of travel and expense category names SAP

Use a function to compute the cosine for the word in the expense descriptions that best matches the words in the supplier invoice text descriptions: -- ZV_AR_BEST_IDX puts the index from the list of expense category words that best matched the supplier invoice text.

```
ZV_AR_BEST_COS = PI_NUMPY.array([], dtype=float)
ZV_AR_BEST_IDX = PI_NUMPY.array([], dtype=int)

if (
    len(ZV_LI_T706B5_SPTXT_PREP) > 0 and
    len(ZV_LI_BSAIKSGTXT_BKPFBKTXT_PREP) > 0
):
    ZV_AR_BEST_COS, ZV_AR_BEST_IDX = FC_BUILD_TFIDF_AND_COSINE_BEST_MATCH(
        PI_LI_CATEGORIES=ZV_LI_T706B5_SPTXT_PREP,
        PI_LI_TEXTS=ZV_LI_BSAIKSGTXT_BKPFBKTXT_PREP,
        PI_ST_NGRAM='1,2',
        PI_I_MIN_DF=1
    )
```

# Obtain a list of travel and expense category names SAP

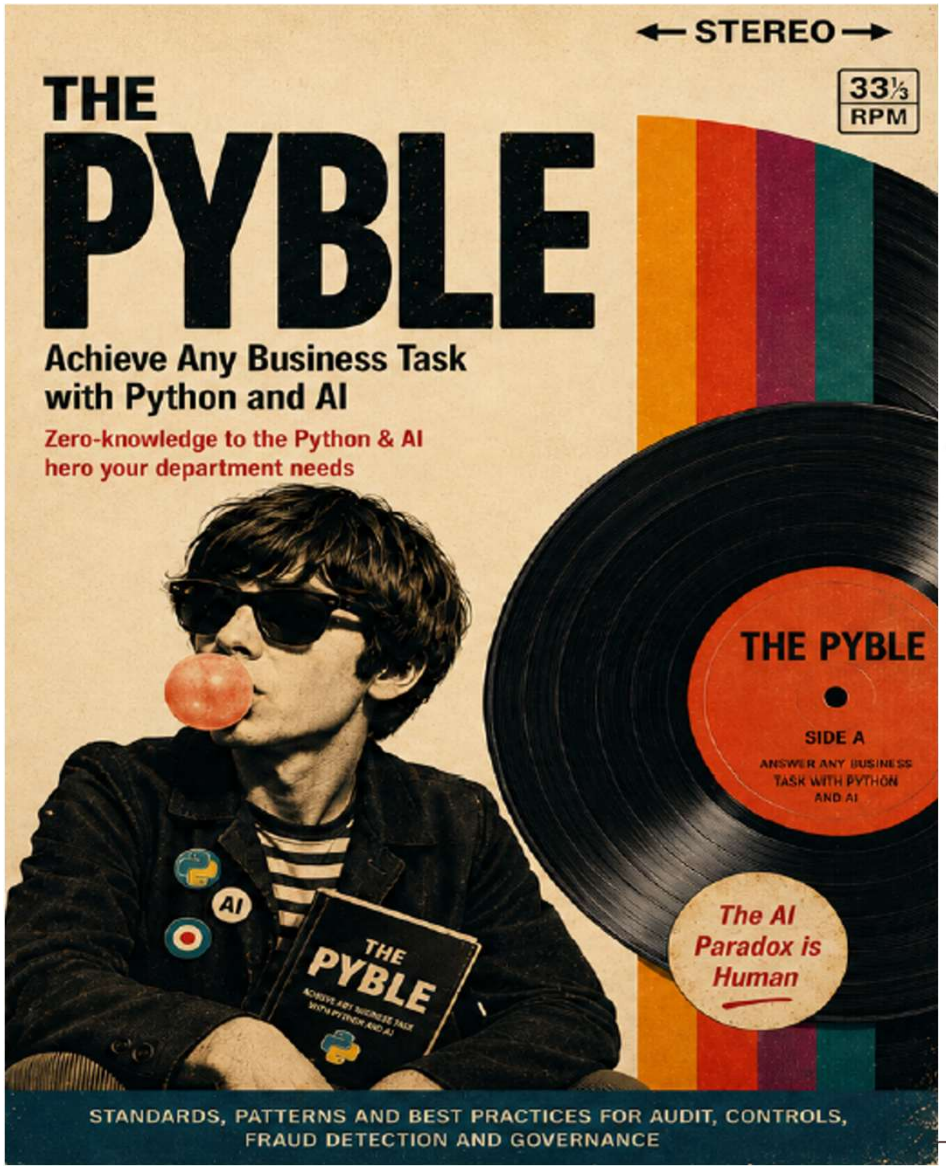
Our function uses tfidVectorizer

```
Z_SHARED_FUNCTIONS > FC_BUILD_TFIDF_AND_COSINE_BEST_MATCH.py > FC_BUILD_TFIDF_AND_COSINE_BEST_MA
1   from typing import List as PI_LIST, Tuple as PI_TUPLE
2
3   import polars as PI_POLARS
4   import numpy as PI_NUMPY
5
6   from sklearn.feature_extraction.text import TfidfVectorizer as PI_TFIDFVECTORIZER
7   from sklearn.preprocessing import normalize as PI_NORMALIZE
8
9
10  def FC_BUILD_TFIDF_AND_COSINE_BEST_MATCH(
11      PI_LI_CATEGORIES: PI_LIST[str],
12      PI_LI_TEXTS: PI_LIST[str],
13      PI_ST_NGRAM: str = '1,2',
14      PI_I_MIN_DF: int = 1
15  ) -> PI_TUPLE[PI_NUMPY.ndarray, PI_NUMPY.ndarray]:
```

Get the list of best matching words – using the array of indexes – look-up the word that matched the best from the list of expense category descriptions

```
ZV_LI_T706B5_SPTXT_BEST_MATCH = []
for ZV_I in range(0, len(ZV_AR_BEST_IDX)):
    ZV_LI_T706B5_SPTXT_BEST_MATCH.append(
        ZV_LI_T706B5_SPTXT[int(ZV_AR_BEST_IDX[ZV_I])]
    )
```

# Obtain a list of travel and expense category names SAP




## Score lists of strings for similar text

The below function would typically be used on two lists of strings that come from two fields of the same table. If the two lists of strings come from two fields of the same table, then we can be sure that the length of the lists is the same.

The function takes the two lists and gives back an array, that has a **COSINE** score for each pair of strings in the two lists. The higher the **COSINE** score, the closer the match between the two strings. ***The match is not about meaning, only about similar text strings.***

A **COSINE** of 0 would mean no similarity, 0.35 weak to moderate similarity, 0.75 high similarity and 1: exact match.

The below code uses **Term Frequency – Inverse Document Frequency**. This approach converts text to numbers, so that mathematics can be applied. The term **ngrams** means the **number of words to consider as a single unit**. For example, consider "bank account" as a single unit and not only "bank". In our naming, **MX** stands for **matrix**, which is similar in concept to pivot table. The rows represent all the text in the lists that are passed, the columns represent all of the words or phrases found. The values are similar to frequency scores.

Note: **TF-IDF** uses **sparce scipy matrices**. Therefore, we need to use the method **multiply()** instead of *.
Note: Below, we call our list of strings "**Corpus**", meaning ***full*** collection of strings.

# Obtain a list of travel and expense category names SAP

```
import polars as PI_POLARS
import numpy as PI_NUMPY
from sklearn.feature_extraction.text import TfidfVectorizer as PI_TFIDFVECTORIZ
from sklearn.preprocessing import normalize as PI_NORMALIZE

def FC_BUILD_TFIDF_AND_COSINE_PER_ROW(
    PI_LI_LEFT,
    PI_LI_RIGHT,
    PI_ST_NGRAM: str = '1,2',
    PI_I_MIN_DF: int = 1
):
    ZV_LI_NGRAM = PI_ST_NGRAM.split(',')
    ZV_I_NGRAM_MIN = int(ZV_LI_NGRAM[0])
    ZV_I_NGRAM_MAX = int(ZV_LI_NGRAM[1])

    ZV_LI_CORPUS = []
    ZV_LI_CORPUS.extend(PI_LI_LEFT)
    ZV_LI_CORPUS.extend(PI_LI_RIGHT)

    ZV_OB_VECT = PI_TFIDFVECTORIZ(
        ngram_range=(ZV_I_NGRAM_MIN, ZV_I_NGRAM_MAX),
        min_df=PI_I_MIN_DF
    )

    ZV_MX_TFIDF_ALL = ZV_OB_VECT.fit_transform(ZV_LI_CORPUS)
    ZV_NU_LEN = len(PI_LI_LEFT)
    ZV_MX_LEFT = ZV_MX_TFIDF_ALL[0: ZV_NU_LEN, :]
    ZV_MX_RIGHT = ZV_MX_TFIDF_ALL[ZV_NU_LEN:(2 * ZV_I_N), :]

    ZV_MX_LEFT_N = PI_NORMALIZE(ZV_MX_LEFT, norm='l2', axis=1)
    ZV_MX_RIGHT_N = PI_NORMALIZE(ZV_MX_RIGHT, norm='l2', axis=1)

    ZV_AR_COS = ZV_MX_LEFT_N.multiply(ZV_MX_RIGHT_N).sum(axis=1)
    ZV_AR_COS = PI_NUMPY.asarray(ZV_AR_COS).reshape(-1)

    return ZV_AR_COS
```

TfidVectorizer does similar text

# Obtain a list of travel and expense category names SAP

## Score lists of strings for similar meaning

The below function would typically be used on two lists of strings that come from two fields of the same table. If the two lists of strings come from two fields of the same table, then we can be sure that the length of the lists is the same.

The function takes the two lists and gives back an **array**, that has a **COSINE** score for each pair of strings in the two lists. The higher the **COSINE** score, the closer match between the two strings.

A **COSINE** of 0 would mean no similarity, 0.50 weak to moderate similarity, 0.70 good match, 0.80 strong match and 1: exact match.

The below code uses the Python library **SentenceTransformer**. This Python library downloads a model from **HuggingFace**. Because the model mentioned below is a **lightweight public model**, we do not need to use a **HuggingFace API** to use it.

The model looks at all of the words in the lists and converts them to **high-dimensional representation of meaning**. For example: “taxi” becomes [0.12, -0.44, 0.91, 0.02, ...], “uber” becomes [0.10, -0.40, 0.88, 0.01, ...] When we multiply and then sum these **representations**, we can see if they have a similar direction of meaning.

# Obtain a list of travel and expense category names SAP

```
from sentence_transformers import SentenceTransformer as PI_SENTENCETRANSFORMER
import numpy as PI_NUMPY


def FC_BUILD_EMBEDDING_AND_COSINE_PER_ROW(
    PI_LI_LEFT,
    PI_LI_RIGHT,
    PI_ST_MODEL_NAME='all-MiniLM-L6-v2'
):


    ZV_LI_CORPUS = []
    ZV_LI_CORPUS.extend(PI_LI_LEFT)
    ZV_LI_CORPUS.extend(PI_LI_RIGHT)

    ZV_OB_MODEL = PI_SENTENCETRANSFORMER(PI_ST_MODEL_NAME)

    ZV_MX_EMBEDDINGS_ALL = ZV_OB_MODEL.encode(
        ZV_LI_CORPUS,
        convert_to_numpy=True,
        normalize_embeddings=True
    )

    ZV_I_N = len(PI_LI_LEFT)

    ZV_MX_LEFT_N = ZV_MX_EMBEDDINGS_ALL[0:ZV_I_N, :]
    ZV_MX_RIGHT_N = ZV_MX_EMBEDDINGS_ALL[ZV_I_N:(2 * ZV_I_N), :]

    ZV_AR_COS = (
        ZV_MX_LEFT_N * ZV_MX_RIGHT_N
    ).sum(axis=1)

    ZV_AR_COS = PI_NUMPY.asarray(ZV_AR_COS).reshape(-1)

    return ZV_AR_COS
```

SentenceTransformer does similar meaing

# Obtain a list of travel and expense category names SAP

## 20260624_03: What do you think we should use?

1. For checking if our T&E looks like it belongs to an expected T&E category, as configured in the SAP system, would you use: (Single choice)

- TfidfVectorizer
- SentenceTransformer

# Create a list of scores for each supplier invoice description

Here we put the lists – that are all of the same length – into a dictionary… - which has three nodes …- and then we convert this 2-dimensional dictionary into a table – where the nodes become field names:

```
ZV_DF_BSAIK_TEXTS_MATCH = PI_POLARS.DataFrame(
    {
        'ZF_BSAIKSGTXT_BKPFBKTXT': ZV_LI_BSAIKSGTXT_BKPFBKTXT,
        'ZF_T706B5_SPTXT_BEST_MATCH': ZV_LI_T706B5_SPTXT_BEST_MATCH,
        'ZF_T706B5_BSAIK_COSINE': ZV_AR_BEST_COS.tolist()
    }
).with_columns(
    (
        PI_POLARS.col('ZF_T706B5_BSAIK_COSINE') < 0.35
    )
    .alias('ZF_BO_SGTXTBKTXT_NOTIN_T706B5SPTXT')
)
```

# Create a list of scores for each supplier invoice description

And finally, we add the score back to the final table

```
ZV_DF_BSAIK_INVOICES  = (
    ZV_DF_BSAIK_INVOICES
    .join(
        ZV_DF_BSAIK_TEXTS_MATCH,
        on=[
            'ZF_BSAIKSGTXT_BKPFBKTXT'
        ],
        how='left'
    )
)
```

# Check the results

Where our supplier invoices had a description, we can see the score that was given to the text field:

| | BC | BD | BE | BF |
|---|---|---|---|---|
| | ZF_BSAIKSGTXT_BKPFBKTXT | ZF_T706B5_SPTXT_BEST_MATCH | ZF_T706B5_BSAIK_COSINE | ZF_BO_SGTXTBKTXT_NOTIN_T706B5SPTXT |
| in | FAC ELEC JANV | Payment in Kind for All Meals | 0 | TRUE |
| in | non PO INvoice directly ordered Flight ticket | Flight | 0.222898635 | TRUE |
| in | flight tickets | Flight | 0.488655564 | FALSE |
| in | Blocked for payment - Vendor dispute Header Text | Payment in Kind for All Meals | 0.112912287 | TRUE |

Our cut-off score was .35… meaning some similarity… however, since we have a long text description in line 2 above, the score was only 0.22

# Where are you at in terms of priorities right now?

## 20260624_04_Are you interested in the Pyble?

1. Are you interested in getting a copy of the Pyble (Single choice)

- ○ Yes
- ○ My priority is all the e-learning Lesson 4s with the S4 code
- ○ Not at the moment


Questions?