Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING

Lecture 04:

Data Cleaning and EDA, RegEx: Part II

Markus Hohle

University California, Berkeley

Data Science for Scientific Computing

MSSE 277A, 3 Units

Data Cleaning and EDA, Regex: Part II

Outline

Motivation

Messy Files: The Worst-Case Scenario

RegEx

Data Cleaning and EDA, Regex: Part II

Outline

Motivation

Messy Files: The Worst-Case Scenario

RegEx

Data Cleaning and EDA, Regex: Part II:

Pandas, Spark, Dask, Polars etc work perfectly with **well structured data**

	A	B	C	D	E
1	date	location_key	new_confirmed	new_tested	cumulative_confirmed
2	01/01/2020	BR	380.0	1154.0	380.0
3	02/01/2020	BR	906.0	2337.0	1286.0
4	03/01/2020	BR	478.0	1307.0	1764.0

```
def StratSpark(FileNamePath: str = "COVID19_dataset.csv",\
               feature_col:  str   = "new_confirmed",\
               group_col:    str   = "location_key",\
               fraction:     float = 0.2):
```

Spark

```
feature_df = df.select(feature_col, group_col)
```

Pandas

```
pd.read_csv("COVID19_dataset.csv")
```

Index	date	location_key	new_confirmed
0	2020-01-01 00:00:00	BR	380
1	2020-01-02 00:00:00	BR	906
2	2020-01-03 00:00:00	BR	478
3	2020-01-04 00:00:00	BR	1848

```
+----------+-----------+-------------+----------+-
|      date|location_key|new_confirmed|new_tested|c
+----------+-----------+-------------+----------+-
|01/01/2020|         BR|        380.0|    1154.0|
|02/01/2020|         BR|        906.0|    2337.0|
|03/01/2020|         BR|        478.0|    1307.0|
|04/01/2020|         BR|       1848.0|    5007.0|
|05/01/2020|         BR|       1095.0|    3459.0|
+----------+-----------+-------------+----------+-
```

Data Cleaning and EDA, Regex: Part II:

Pandas, Spark, Dask, Polars etc work perfectly with **well structured data**

	A	B	C	D	E
1	date	location_key	new_confirmed	new_tested	cumulative_confirmed
2	01/01/2020	BR	380.0	1154.0	380.0
3	02/01/2020	BR	906.0	2337.0	1286.0
4	03/01/2020	BR	478.0	1307.0	1764.0

```
def StratSpark(FileNamePath: str   = "COVID19_dataset.csv",\
               feature_col:  str   = "new_confirmed",\
               group_col:    str   = "location_key",\
               fraction:     float = 0.2):
```

Pandas

```
pd.read_csv("COVID19_dataset.csv")
```

Index	date	location_key	new_confirmed
0	2020-01-01 00:00:00	BR	380
1	2020-01-02 00:00:00	BR	906
2	2020-01-03 00:00:00	BR	478
3	2020-01-04 00:00:00	BR	1848

- column names are meaningful: “date” contains actual dates
- types within columns are consistent: “location_key” are all `str`, “new_confirmed” are all `int` etc
- date strings all have the same format: *"YYYY-MM-DD HH:MM:SS"*

Data Cleaning and EDA, Regex: Part II:

Pandas, Spark, Dask, Polars etc work perfectly with **well structured data**

	A	B	C	D	E
1	date	location_key	new_confirmed	new_tested	cumulative_confirmed
2	01/01/2020	BR	380.0	1154.0	380.0
3	02/01/2020	BR	906.0	2337.0	1286.0
4	03/01/2020	BR	478.0	1307.0	1764.0

from	to	sensor
29/04/2010	10/06/2010	AB-303k
01/06/2010	03/06/2010	ct102x
06/06/2010	27/02/2012	ab-303k
14/06/2010	12/03/2012	ct-102x
14/06/2010		ab-415k
17/06/2010		ab303k
17/06/2010	23/06/2010	AB-415k
18/06/2010		ct-102x
24/06/2010	30/06/2010	ct-102x
29/06/2010	23.01.2018 06:00:30	CT-102x
30/06/2010	unknown	ct102x
30/06/2010	27.01.2018 15:00:29	ct102x
24.06.2020 15:19:07		ab-303k
25.06.2020 13:19:10		ct102x

- date columns are not clearly named as such
- date strings have **different formats**
- **types** are **not consistent** within columns
- entries in “sensor” have different cases and missing characters

Data Cleaning and EDA, Regex: Part II

Outline

Motivation

Messy Files: The Worst-Case Scenario

RegEx

Data Cleaning and EDA, Regex: Part II: Messy Files: The Worst-Case Scenario

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

from	to	sensor
29/04/2010	10/06/2010	AB-303k
01/06/2010	03/06/2010	ct102x
06/06/2010	27/02/2012	ab-303k
14/06/2010	12/03/2012	ct-102x
14/06/2010		ab-415k
17/06/2010		ab303k
17/06/2010	23/06/2010	AB-415k
18/06/2010		ct-102x
24/06/2010	30/06/2010	ct-102x
29/06/2010	23.01.2018 06:00:30	CT-102x
30/06/2010	unknown	ct102x
30/06/2010	27.01.2018 15:00:29	ct102x
24.06.2020 15:19:07		ab-303k
25.06.2020 13:19:10		ct102x

`'MessyFile.xlsx'`

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

for Unix like commands on you OS

MessyFile I.ipynb

iterates over paths r directories d and files f

```
import os

def FindMyFile(filename: str, ServerHardDiscPath: str = r"c:\Users\user\Desktop") -> str:

        for r,d,f in os.walk(ServerHardDiscPath):

                for files in f:
                        if files == filename: #for example "MessyFile.xlsx"
```

Data Cleaning and EDA, Regex: Part II: Messy Files: The Worst-Case Scenario

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

MessyFile I.ipynb

files f

```
['LectureExercise 01 Solution.ipynb', 'LectureExercise 01.ipynb', 'MessyFile.xlsx']
```

directories d

```
['.ipynb_checkpoints', '01 Introduction', '02 Data Sampling and Pandas', '03 EDA I', '04 EDA II', '05 SQL', '07 PCA
Correlation Dimension Reduction', '08 Advanced Visualization Tools', '10 Linear and Logistic Regression', '11
Avoiding Overfitting and Regularization', '13 Gradient Descent', '14 Clustering and Classification']
```

paths r

```
c:\Users\MMH_user\Desktop\Berkeley\MSSE\Chem 273\08 Statistics for Data Science
c:\Users\MMH_user\Desktop\Berkeley\MSSE\Chem 273\08 Statistics for Data Science\.ipynb_checkpoints
```

```
def FindMyFile(filename: str, ServerHardDiscPath: str = r"c:\Users\user\Desktop") -> str:
```

```
for r,d,f in os.walk(ServerHardDiscPath):

        for files in f:
                if files == filename: #for example "MessyFile.xlsx"
```

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

MessyFile I.ipynb

`return` statement within the loop so that the program stops after file has been located

```
In [23]: print(FindMyFile('MessyFile.xlsx'))
c:\Users\MMH_user\Desktop\Berkeley\MSSE\Chem 277A Spring 2026\01 Introduction\Homework\MessyFile.xlsx
```

```
def FindMyFile(filename: str, ServerHardDiscPath: str = r"c:\Users\user\Desktop") -> str:

        for r,d,f in os.walk(ServerHardDiscPath):

                for files in f:
                        if files == filename: #for example "MessyFile.xlsx"

                                file_name_and_path = os.path.join(r,files)
                                return file_name_and_path
```

Data Cleaning and EDA, Regex: Part II: Messy Files: The Worst-Case Scenario

MessyFile I.ipynb

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

```
Data = pd.read_excel(FindMyFile('MessyFile.xlsx'))
print(Data.sample(10))
```

	from	to	sensor
1	2010-06-01 00:00:00	2010-06-03 00:00:00	ct102x
9	2010-06-29 00:00:00	23.01.2018 06:00:30	CT-102x
24	2010-09-15 00:00:00	2010-10-13 00:00:00	ct102x
10	2010-06-30 00:00:00	unknown	ct102x
21	2010-08-17 00:00:00	2010-08-18 00:00:00	AB-415k
19	2010-07-26 00:00:00	NaN	AB-415k
37	2011-07-01 00:00:00	NaN	ab-415k
25	2010-10-13 00:00:00	2010-10-14 00:00:00	ab-303k
38	2011-07-05 00:00:00	2011-07-06 00:00:00	CT-102x
8	2010-06-24 00:00:00	2010-06-30 00:00:00	ct-102x

goal: identify date columns as such even though they are not labeled properly

idea: **not all entries** need to be date strings (of any format), but **most of them** should

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

MessyFile I.ipynb

goal: identify date columns as such even though they are not labeled properly

idea: **not all entries** need to be date strings (of any format), but **most of them** should

We print a few date strings using `parse`

```
from dateutil.parser import parse

print(Data['from'][0],  parse(str(Data['from'][0])))
print(Data['from'][23], parse(str(Data['from'][23])))
print(Data['from'][27], parse(str(Data['from'][27])))
print(Data['from'][29], parse(str(Data['from'][29])))
```

```
2010-04-29 00:00:00 2010-04-29 00:00:00
08.06.2023 09:00:10 2023-08-06 09:00:10
12--11--2010 2010-12-11 00:00:00
Dec/03/2011 2011-12-03 00:00:00
```

```
def IsDate(val: str):
        try:
                if pd.isna(val):
                return False
                parse(str(val))
                return True
        except Exception:
                return False
```

Writing a function that checks if input is a date string

Data Cleaning and EDA, Regex: Part II: Messy Files: The Worst-Case Scenario

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

MessyFile I.ipynb

goal: identify date columns as such even though they are not labeled properly

idea: **not all entries** need to be date strings (of any format), but **most of them** should

Writing a function that checks if input is a date string

```
def IsDate(val: str):
        try:
                if pd.isna(val):
                return False
                parse(str(val))
                return True
        except Exception:
                return False
```

run and test **IsDate** for a few strings

```
L = ['17/07/2017', '17-07-2017', '47/47/2017',
'2017', '34435']
```

```
True True False True False
```

note: **default is American date format**!

for international date format:

```
parse("3.2.2025", dayfirst = True)
```

to
10/06/2010
03/06/2010
27/02/2012
12/03/2012
23/06/2010
30/06/2010
23.01.2018 06:00:30
unknown
27.01.2018 15:00:29

idea: running **IsDate** over each column and consider it a date column if it returns True for, say 70%, of the rows

```
def DetectDateColumns(df, Nsample: int = 20, Tolerance: float = 0.7):

    date_cols = []

    for col in df.columns:

        sample_vals = df[col].dropna().astype(str)

        if len(sample_vals) == 0:
            continue

        sample_vals = sample_vals.sample(min(Nsample, len(sample_vals)))

        if sample_vals.apply(IsDate).mean() > Tolerance
            date_cols.append(col)

    return date_cols
```

iterating over all columns

adding those columns that match the criteria

Data Cleaning and EDA, Regex: Part II: Messy Files: The Worst-Case Scenario

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

MessyFile I.ipynb

goal: identify date columns as such even though they are not labeled properly

idea: **not all entries** need to be date strings (of any format), but **most of them** should

```
DateCols = DetectDateColumns(Data)
print(DateCols)
```

```
DateCols = DetectDateColumns(Data)
print(DateCols)
```

```
['from', 'to']
```

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

`MessyFile I.ipynb`

Finally, date strings have to be turned into **numerical values**!
different conventions: counting days/seconds from a reference date:

MJD: **M**odified **J**ulian **D**ate

- most common
- starts Jan 1st, 4712BC
- Physics, Astronomy, Navigation (incl satellites, space craft)

Unix

- Jan 1st, 1970
- Computer Science
- default in most programming languages

```
from astropy.time import Time

mjd = Time(parse(str(d)), format = 'datetime').mjd
```

We don't know

- **where** the file is stored,
- **where the date strings are** stored in the file,
- **if the date columns are labelled** as such,
- **which date format** had been used (maybe several?) and,
- if there are **gaps in the records**.

`MessyFile I.ipynb`

parsing date string, turning it into `float` and saving it

```
def DateStrToMJD(df):

    MJD = [None]*len(df)

    for i, d in enumerate(df):

        try:
            mjd    = Time(parse(str(d)), format = 'datetime').mjd
            MJD[i] = mjd

        except Exception:
            MJD[i] = np.nan

    return MJD
```

Finally, we can put it all together and write a main function that finds the target file and returns the dates as `float`

`MessyFile I.ipynb`

reading the file, replace `pandas` by `dask` or `polars` if performance is important

```
def ReturnDatesFromFile(filename:           str = 'MessyFile.xlsx',
                        ServerHardDiscPath: str = r"c:\Users\Desktop",\
                        Nsample:            int = 20, Tolerance: float = 0.7):

    Data      = pd.read_excel(FindMyFile(filename, ServerHardDiscPath))

    DateCols = DetectDateColumns(Data, Nsample, Tolerance)

    Dates     = [None]*len(DateCols)

    for i, d in enumerate(DateCols):
            Dates[i] = DateStrToMJD(Data[d])

    return np.array(Dates).transpose()
```

date strings to `float`

find date columns

```
NumDates = ReturnDatesFromFile()
```

MessyFile I.ipynb

```
print(NumDates[:25,:])
```

```
[[55315.          55357.        ]
 [55348.          55350.        ]
 [55353.          55984.        ]
 [55361.          55998.        ]
 [55361.                    nan]
 [55364.                    nan]
 [55364.          55370.        ]
 [55365.                    nan]
 [55371.          55377.        ]
 [55376.          58141.25034722]
 [55377.                    nan]
 [55377.          58145.62533565]
 [59024.63827546            nan]
 [59025.55497685            nan]
 [55383.                    nan]
 [55384.                    nan]
 [55384.          56243.        ]
 [55386.                    nan]
 [55392.          55399.        ]
 [55403.                    nan]
 [55412.          58148.0003588 ]
 [55425.          55426.        ]
 [55426.                    nan]
 [60162.37511574            nan]
 [55454.          55482.        ]]
```

Data Cleaning and EDA, Regex: Part II

Outline

Motivation

Messy Files: The Worst-Case Scenario

RegEx

Data Cleaning and EDA, Regex: Part II:

Often, we need to search for specific keywords:

`MessyFile II.ipynb`

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

```
import re #for RegEx
from rapidfuzz import fuzz
```

```
AB-303k
 ct102x
ab-303k
ct-102x
ab-415k
 ab303k
AB-415k
ct-102x
ct-102x
CT-102x
 ct102x
 ct102x
ab-303k
```

```
search_string  = 'ab-415k'
x              = 'this is sensor 415K series ab'
```

fuzz has four modii:

ratio:	returns the ratio of identical sides
partial_ratio:	picks the shorter of the two input strings and calculates the ratio from that
token_sort_ratio:	takes into account the words/letters can be in different order
token_set_ratio:	combines partial_ratio and token_sort_ratio

Data Cleaning and EDA, Regex: Part II:

Often, we need to search for specific keywords:

MessyFile II.ipynb

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

```
x1 = 'this is sensor 415K series ab'
x2 = 'this is sensor 415K ab'
x3 = 'this is sensor ab 415K'
X  = [x1, x2, x3]
```

```
for x in X:
    f1 = fuzz.ratio(           x.lower(), search_string.lower())
    f2 = fuzz.partial_ratio(   x.lower(), search_string.lower())
    f3 = fuzz.token_sort_ratio(x.lower(), search_string.lower())
    f4 = fuzz.token_set_ratio( x.lower(), search_string.lower())

    print(f"fuzz ratio = {f1: .2f}%\nfuzz partial ratio = {f2: .2f}%\n" +
     f"fuzz token sort ratio = {f3: .2f}%\nfuzz token set ratio = {f4: .2f}%\n")
```

doesn't need to be case sensitive in our case

Data Cleaning and EDA, Regex: Part II:

Often, we need to search for specific keywords:

MessyFile II.ipynb

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

```
x1 = 'this is sensor 415K series ab'
x2 = 'this is sensor 415K ab'
x3 = 'this is sensor ab 415K'
X  = [x1, x2, x3]
```

```
search_string = 'ab-415k'
```

```
fuzz ratio =  22.22%
fuzz partial ratio =  57.14%
fuzz token sort ratio =  22.22%
fuzz token set ratio =  22.22%

fuzz ratio =  27.59%
fuzz partial ratio =  57.14%
fuzz token sort ratio =  27.59%
fuzz token set ratio =  27.59%

fuzz ratio =  41.38%
fuzz partial ratio =  85.71%
fuzz token sort ratio =  27.59%
fuzz token set ratio =  27.59%
```

Sort is not automatically the best option if target string is split into different token!

Data Cleaning and EDA, Regex: Part II:

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

MessyFile II.ipynb

write & test code for out messy file:

```
def FuzzyMatch(series, search_string: str, method: str = 'partial_ratio',
               threshold: float = 80):

    M         = getattr(fuzz, method)
    Mfun      = lambda x: M(x.lower(), search_string.lower()) >= threshold

    TrueFalse = series.fillna("").astype(str).apply(Mfun)

    return np.argwhere(TrueFalse == True).reshape(-1)
```

calling the method dynamically

ignoring na by turning them into an empty string ""

Data Cleaning and EDA, Regex: Part II:

RegEx

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- Regular Expression: parts of the sequence are known

`MessyFile II.ipynb`

same results for

'ratio',
'partial_ratio',
'token_sort_ratio',
'token_set_ratio'

	from	to	sensor
0	2010-04-29 00:00:00	2010-06-10 00:00:00	AB-303k
2	2010-06-06 00:00:00	2012-02-27 00:00:00	ab-303k
5	2010-06-17 00:00:00	NaN	ab303k
12	24.06.2020 15:19:07	NaN	ab-303k
15	2010-07-07 00:00:00	unknown	ab303k
18	2010-07-15 00:00:00	2010-07-22 00:00:00	AB-303k
20	2010-08-04 00:00:00	30.01.2018 00:00:31	AB-303k
23	08.06.2023 09:00:10	NaN	AB-303k
25	2010-10-13 00:00:00	2010-10-14 00:00:00	ab-303k
26	2010-10-22 00:00:00	2010-10-25 00:00:00	ab-303k
27	12--11--2010	19--11--2010	AB-303k
29	Dec/03/2011	unknown	AB-303k
31	2011-05-05 00:00:00	unknown	AB-303k
33	2011-06-22 00:00:00	2011-06-27 00:00:00	AB-303k
36	2011-07-01 00:00:00	2011-07-12 00:00:00	AB-303k

	from	to	sensor
0	2010-04-29 00:00:00	2010-06-10 00:00:00	AB-303k
2	2010-06-06 00:00:00	2012-02-27 00:00:00	ab-303k
5	2010-06-17 00:00:00	NaN	ab303k
12	24.06.2020 15:19:07	NaN	ab-303k
15	2010-07-07 00:00:00	unknown	ab303k
18	2010-07-15 00:00:00	2010-07-22 00:00:00	AB-303k
20	2010-08-04 00:00:00	30.01.2018 00:00:31	AB-303k
23	08.06.2023 09:00:10	NaN	AB-303k
25	2010-10-13 00:00:00	2010-10-14 00:00:00	ab-303k
26	2010-10-22 00:00:00	2010-10-25 00:00:00	ab-303k
27	12--11--2010	19--11--2010	AB-303k
29	Dec/03/2011	unknown	AB-303k
31	2011-05-05 00:00:00	unknown	AB-303k
33	2011-06-22 00:00:00	2011-06-27 00:00:00	AB-303k
36	2011-07-01 00:00:00	2011-07-12 00:00:00	AB-303k

	from	to	sensor
0	2010-04-29 00:00:00	2010-06-10 00:00:00	AB-303k
2	2010-06-06 00:00:00	2012-02-27 00:00:00	ab-303k
5	2010-06-17 00:00:00	NaN	ab303k
12	24.06.2020 15:19:07	NaN	ab-303k
15	2010-07-07 00:00:00	unknown	ab303k
18	2010-07-15 00:00:00	2010-07-22 00:00:00	AB-303k
20	2010-08-04 00:00:00	30.01.2018 00:00:31	AB-303k
23	08.06.2023 09:00:10	NaN	AB-303k
25	2010-10-13 00:00:00	2010-10-14 00:00:00	ab-303k
26	2010-10-22 00:00:00	2010-10-25 00:00:00	ab-303k
27	12--11--2010	19--11--2010	AB-303k
29	Dec/03/2011	unknown	AB-303k
31	2011-05-05 00:00:00	unknown	AB-303k
33	2011-06-22 00:00:00	2011-06-27 00:00:00	AB-303k
36	2011-07-01 00:00:00	2011-07-12 00:00:00	AB-303k

	from	to	sensor
0	2010-04-29 00:00:00	2010-06-10 00:00:00	AB-303k
2	2010-06-06 00:00:00	2012-02-27 00:00:00	ab-303k
5	2010-06-17 00:00:00	NaN	ab303k
12	24.06.2020 15:19:07	NaN	ab-303k
15	2010-07-07 00:00:00	unknown	ab303k
18	2010-07-15 00:00:00	2010-07-22 00:00:00	AB-303k
20	2010-08-04 00:00:00	30.01.2018 00:00:31	AB-303k
23	08.06.2023 09:00:10	NaN	AB-303k
25	2010-10-13 00:00:00	2010-10-14 00:00:00	ab-303k
26	2010-10-22 00:00:00	2010-10-25 00:00:00	ab-303k
27	12--11--2010	19--11--2010	AB-303k
29	Dec/03/2011	unknown	AB-303k
31	2011-05-05 00:00:00	unknown	AB-303k
33	2011-06-22 00:00:00	2011-06-27 00:00:00	AB-303k
36	2011-07-01 00:00:00	2011-07-12 00:00:00	AB-303k

Data Cleaning and EDA, Regex: Part II:

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

`MessyFile II.ipynb`

three (and more) different level:

- **functions**

- **metacharacters**

- **flags**

Data Cleaning and EDA, Regex: Part II:

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

`MessyFile II.ipynb`

different **functions**:

`re.findall`	returns a list of all matches
`re.search`	returns location of the first(!) character
`re.split`	splits string based on "c1" and returns a list
`re.sub`	searches "c1" and replaces it by "c2"

```
ShowInteractiveTable(RegExFun)
```

Function	What It Does	Example (editable)	Run	Output
findall	returns a list of all matches	`print(re.findall("a", "How many a-s are in Alabama?"))`	▶	`['a', 'a', 'a', 'a', 'a', 'a']`
search	returns location of the first(!) character	`out = re.search("a", "How many a-s are in Alabama?")` `print("first appearance at index", out.span()[0])`	▶	`first appearance at index 5`
split	splits string based on "c1" and returns a list	`print(re.split(" ", "Split this string based on spaces!"))`	▶	`['Split', 'this', 'string', 'based', 'on', 'spaces!']`
sub	searches "c1" and replaces it by "c2"	`print(re.sub("a", "A", "How many a-s are in Alabama?"))`	▶	`How mAny A-s Are in AlAbAmA?`

Data Cleaning and EDA, Regex: Part II:

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

`MessyFile II.ipynb`

different **functions**:

`re.findall`
`re.search`
`re.split`
`re.sub`

can be combined with different **metacharacters**:

`[ ]`	matches all characters (case sensitive) alphabetically between *"c1"* and *"c2"*
`\`	matches all special characters indicated by *""* (*"\d"* all digit, *"\w"* all letters etc)
`.`	match any character except a newline
`^`	starts with

and many more...

Data Cleaning and EDA, Regex: Part II:

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

MessyFile II.ipynb

```
ShowInteractiveTable(RegExMeta)
```

Character	What It Does	Example (editable)	Run	Output
[]	matches all characters (case sensitive) alphabetically between "c1" and "c2"	`print(re.findall("[c-f]", "All Men must die"))`	▶	`['e', 'd', 'e']`
\	matches all special characters indicated by "" ("\d" all digit, "\w" all letters etc)	`print(re.findall("\d", "sensor AB-404"))`	▶	`['4', '0', '4']`
.	match any character except a newline	`print(re.findall(".", "AB-404"))`	▶	`['A', 'B', '-', '4', '0', '4']`
^	starts with	`print(re.findall("^AB", "AB-404"))`	▶	`['AB']`
$	ends with	`print(re.findall("AB$", "sensor 404, version AB"))`	▶	`['AB']`
{}	matches sequence that starts with "c1", followed by excactly n characters, and an "c2"	`print(re.findall("2.{2}k", "sensors: 200k, 234k, 211k, 533k, 277k, 544c"))`	▶	`['200k', '234k', '211k', '277k']`
|	logical or	`print(re.findall("200k|200k", "sensors: 200k, 234k, 211k, 533k, 277k, 544c"))`	▶	`['200k']`

Data Cleaning and EDA, Regex: Part II:

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

`MessyFile II.ipynb`

different **functions**:

```
re.findall
re.search
re.split
re.sub
```

can be combined with different **metacharacters**:

```
[ ]
\
.
^
```

for more precise search we can add **flags**:

- focusing on ASCII (letters: A–Z, a–z digits: 0–9, underscore (_))
- Unicode (umlauts, Greek letters/symbols, math symbols)
- debugging (returns how python **`re`** processes the input, is not an error message!)
- and much more!

Data Cleaning and EDA, Regex: Part II:

- **Fuzzy search**: if the exact sequence is not known or might vary a lot
- **Reg**ular **Ex**pression: parts of the sequence are known

MessyFile II.ipynb

Flag	What It Does	Example (editable)	Run	Output
ASCII or A	returns only ASCII characters	`print(re.findall("\w", "Pößneck is a town in Germany.", re.A))`	▶	`['P', 'n', 'e', 'c', 'k', 'i', 's', 'a', 't', 'o', 'w', 'n', 'i', 'n', 'G', 'e', 'r', 'm', 'a', 'n', 'y']`

DEBUG	debugs information	`print(re.findall("\w", "Pößneck is a town in Germany.", re.DEBUG))`	▶	`IN`
				`  CATEGORY CATEGORY_WORD`
				
				` 0. INFO 7 0b100 1 1 (to 8)`
				`      in`
				` 5.     CATEGORY UNI_WORD`
				` 7.     FAILURE`
				` 8: IN 4 (to 13)`
				`10.   CATEGORY UNI_WORD`
				`12.   FAILURE`
				`13: SUCCESS`
				`['P', 'ö', 'ß', 'n', 'e', 'c', 'k', 'i', 's', 'a', 't', 'o', 'w', 'n', 'i', 'n', 'G', 'e', 'r', 'm', 'a', 'n', 'y']`

DOTALL or S	searches for c1 followed by a symbol (one) and ends with c2	`print(re.findall("o.n", "This town has been known to be my own oxygen", re.S))`	▶	`['own', 'own', 'own']`

Data Cleaning and EDA, Regex: Part II

Thank you very much for your attention!