L2: Data processing & missing value imputations

Andrea Štefancová, Josef Švec,
Thomas Browne

18th of April, 2023

TODAY'S LECTURE

1. Data sources
2. Data exploration
3. Data aggregations, cleaning
4. Visualizing data
5. Scaling & normalization
6. Dealing with missing values

Andrea

Josef

Thomas

czechitas

Data pre-processing

Before you build a model

1. What is the problem/question we are solving?
2. What is your data source(s)?
3. What does your data look like?

Why is the topic of data pre-processing important?

1. Data exploration is always the first step of model building
2. You cannot assume the data you were provided to be flawless
3. Data collection, pre-processing and cleaning is majority of work we do on DS projects
4. Data originates from multiple sources

Version Control Systems (VCS)

Why use version control?

1. A complete long-term change history of every file
2. Collaboration in teams: branching and merging
3. Traceability

git

- git is good
- git is de facto standard
- git is a quality open-source project

czechitas

Repository for version control

Bitbucket

GitLab

czechitas

Author	Commit	Message
G. Sylvie Davies	22b36e57041	TKT-22
G. Sylvie Davies	bd7e94a5447	Revert "petal3.2"
G. Sylvie Davies	a95f4e6c4f2	Revert "petal3.2"
G. Sylvie Davies	6915442ab93	TKT-22
G. Sylvie Davies	a9d62431110	flower
G. Sylvie Davies	a2d78542c8e	petal3.3
~~G. Sylvie Davies~~	~~8efa6b66ade~~	**Reverted:** ~~petal3.2~~
G. Sylvie Davies	9d95af81f49	petal3.1
G. Sylvie Davies	658b04edab9	petal2.3
G. Sylvie Davies	c954c3ed859	petal2.2

Data sources & loading data

What are possible data sources?

Excel (xlsx / xlsm / xls / ...)

```
import pandas as pd

df = pd.read_excel("path_to_folder/file_name.xlsx",
                   sheet_name="your_excel_sheet_name")

print(df)
```

CSV

```
import pandas as pd

df = pd.read_csv("path_to_folder/file_name.csv")

# Preview first 5 rows
df.head()
```

CSV

```
import pandas as pd

df = pd.read_csv("path_to_folder/file_name.csv")

# Specify separator
df = pd.read_csv("path_to_folder/file_name.csv", sep=",")
df = pd.read_csv("path_to_folder/file_name.csv", sep="\t")
df = pd.read_csv("path_to_folder/file_name.csv", sep=";")

# Preview first 5 rows
df.head()
```

csv from URL

```
# Webpage URL
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/iri
s/iris.data"

# Read data from URL
iris_data = pd.read_csv(url)

iris_data.head()
```

csv from URL

```
# Webpage URL
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/iri
s/iris.data"

# Define the column names
col_names = ["sepal_length_in_cm",
             "sepal_width_in_cm",
             "petal_length_in_cm",
             "petal_width_in_cm",
             "class"]

# Read data from URL
iris_data = pd.read_csv(url, names=col_names)

iris_data.head()
```

Configuration files: json / yaml

json

```
# Import the json module
import json

with open('user.json') as user_file:
   file_contents = user_file.read()

print(file_contents)
```

yaml

```
# Import the yaml module
import yaml

# Common file endings: .yml and .yaml
with open('config.yml', 'r') as file:
   prime_service = yaml.safe_load( file)
```

Database

```
# SQLite example
import pandas as pd
import sqlite3 as sql

con = sql.connect("tutorial.db")
cur = con.cursor()

# Create a database table movie
cur.execute("CREATE TABLE movie(title, year, score)" )

# We can verify that the new table has been created
res = cur.execute("SELECT name FROM sqlite_master" )
res.fetchone()

# Turn query directly into dataframe:
df1 = pd.read_sql_query("SELECT * FROM movie", con)
df1.head()

# Close connection to database
con.close()
```

Data summary

(data frame info, column types, units, ...)

Our data example

	active_status	compa_ratio	compa_ratio_range	company	cost_center	country	current_rating	date_of_birth	employee_id	ethnicity	gender	generation
0	True	1.088	Above Compa-Ratio	Super	Global Support - Asia/Pac	Singapore	1 - Unsatisfactory	04/05/1999	1000	Asian	Female	Generation Z (1997 and onwards)
1	True	0.826	Below Compa-Ratio	Super	AMER - United States of America	United States of America	4 - Exceeds Expectations	08/26/1995	1001	Hispanic	Male	Generation Y/Millenials (1981-1996)
2	True	0.998	At Compa-Ratio	Super	Global Support - Asia/Pac	South Korea	2 - Needs Improvement	06/10/1994	1002	Asian	Male	Generation Y/Millenials (1981-1996)
3	True	0.918	Below Compa-Ratio	Super	AMER - Canada	Canada	3 - Meets Expectations	05/09/2001	1003	White	Male	Generation Z (1997 and onwards)
4	True	1.343	Above Compa-Ratio	Super	EMEA - United Kingdom	United Kingdom	5 - Outstanding Performance	06/13/1984	1004	White	Female	Generation Y/Millenials (1981-1996)

Show the first/last *n* rows

```
df.head()
df.tail()
```

	active_status	compa_ratio	compa_ratio_range	company	cost_center	country	current_rating	date_of_birth	employee_id	ethnicity	gender	generation	high_performer	high_potential	hire_da
0	True	1.088	Above Compa-Ratio	Super	Global Support - Asia/Pac	Singapore	1 - Unsatisfactory	04/05/1999	1000	Asian	Female	Generation Z (1997 and onwards)	False	False	08/04/20
1	True	0.826	Below Compa-Ratio	Super	AMER - United States of America	United States of America	4 - Exceeds Expectations	08/26/1995	1001	Hispanic	Male	Generation Y/Millenials (1981-1996)	False	False	01/14/20
2	True	0.998	At Compa-Ratio	Super	Global Support - Asia/Pac	South Korea	2 - Needs Improvement	06/10/1994	1002	Asian	Male	Generation Y/Millenials (1981-1996)	False	False	09/15/20
3	True	0.918	Below Compa-Ratio	Super	AMER - Canada	Canada	3 - Meets Expectations	05/09/2001	1003	White	Male	Generation Z (1997 and onwards)	False	False	01/13/20
4	True	1.343	Above Compa-Ratio	Super	EMEA - United Kingdom	United Kingdom	5 - Outstanding Performance	06/13/1984	1004	White	Female	Generation Y/Millenials (1981-1996)	True	False	11/09/20

czechitas

Discover all column names using attribute columns

```
df.columns
```

```
Index(['active_status', 'compa_ratio', 'compa_ratio_range', 'company',
       'cost_center', 'country', 'current_rating', 'date_of_birth',
       'employee_id', 'ethnicity', 'gender', 'generation', 'high_performer',
       'high_potential', 'hire_date', 'hiring_source', 'is_leader',
       'is_manager', 'job_family_group', 'job_level', 'job_profile',
       'legal_full_name', 'length_of_service', 'location', 'management_level',
       'manager_id', 'org_level_1', 'org_level_2', 'org_level_3',
       'org_level_4', 'org_level_5', 'organization_level', 'region',
       'report_effective_date', 'single_job_family', 'supervisory_org',
       'tenure_category', 'termination_category', 'termination_date',
       'termination_reason', 'total_compensation', 'worker_type'],
      dtype='object')
```

Have a look at data dimensionality

```
df.shape
```

```
(1000, 42)
```

czechitas

```
df.info()
```

print out a concise summary of the dataframe

- index,
- data types,
- columns,
- non-null values
- memory usage

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 42 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   active_status          1000 non-null   bool
 1   compa_ratio            1000 non-null   float64
 2   compa_ratio_range      1000 non-null   object
 3   company                1000 non-null   object
 4   cost_center            1000 non-null   object
 5   country                1000 non-null   object
 6   current_rating         1000 non-null   object
 7   date_of_birth          1000 non-null   object
 8   employee_id            1000 non-null   int64
 9   ethnicity              1000 non-null   object
 10  gender                 1000 non-null   object
 11  generation             1000 non-null   object
 12  high_performer         1000 non-null   bool
 13  high_potential         1000 non-null   bool
 14  hire_date              1000 non-null   object
 15  hiring_source          1000 non-null   object
 16  is_leader              1000 non-null   bool
 17  is_manager             1000 non-null   bool
 18  job_family_group       1000 non-null   object
 19  job_level              1000 non-null   object
 20  job_profile            1000 non-null   object
 21  legal_full_name        1000 non-null   object
 22  length_of_service      1000 non-null   float64
 23  location               1000 non-null   object
 24  management_level       1000 non-null   object
 25  manager_id             1000 non-null   int64
 26  org_level_1            1000 non-null   object
 27  org_level_2            949 non-null    object
 28  org_level_3            853 non-null    object
```

Column types

PersonID	Gender	Income	Years Education	Age	Birth Place
2343	F	50 000	17	35	SK
1213	M	35 000	15	32	LTU
4533	M	40 000	15	53	FR
4563	M	100 000	19	51	CZ
7554	F	50 000	18	28	CZ
6465	F	27 500	13	25	UK
7453	M	34 000	13	32	US
...	...	...	...	...	

Column types

PersonID	Gender	Income	Years Education	Age	Birth Place
2343	F	50 000	17	35	SK
1213	M	35 000	15	32	LTU
4533	M	40 000	15	53	FR
4563	M	100 000	19	51	CZ
7554	F	50 000	18	28	CZ
6465	F	27 500	13	25	UK
7453	M	34 000	13	32	US
...	...	...	...	...	

NUMERICAL

Column types

PersonID	Gender	Income	Years Education	Age	Birth Place
2343	F	50 000	17	35	SK
1213	M	35 000	15	32	LTU
4533	M	40 000	15	53	FR
4563	M	100 000	19	51	CZ
7554	F	50 000	18	28	CZ
6465	F	27 500	13	25	UK
7453	M	34 000	13	32	US
...	...	...	...	...	

STRING/FACTOR

STRING/FACTOR

Column types

PersonID	Gender	Income	Years Education	Age	Birth Place
2343	F	50 000	17	35	SK
1213	M	35 000	15	32	LTU
4533	M	40 000	15	53	FR
4563	M	100 000	19	51	CZ
7554	F	50 000	18	28	CZ
6465	F	27 500	13	25	UK
7453	M	34 000	13	32	US
...	...	...	...	...	

NUMERICAL?
STRING?

```
df.describe()
```

generates descriptive statistics

- central tendency,
- dispersion,
- shape of the dataset's distribution,
- …

	compa_ratio	employee_id	length_of_service	manager_id	organization_level	total_compensation
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000
mean	0.976774	1499.500000	5.419683	1452.930000	3.89700	4056.791630
std	0.153751	288.819436	5.710193	330.096426	1.10798	3004.498829
min	0.400000	1000.000000	0.005000	1019.000000	1.00000	800.000000
25%	0.898000	1249.750000	1.151750	1114.000000	4.00000	2249.850000
50%	0.971000	1499.500000	3.680500	1428.000000	4.00000	3340.720000
75%	1.055250	1749.250000	7.815000	1823.000000	5.00000	4420.240000
max	1.600000	1999.000000	36.153000	1998.000000	5.00000	20000.000000

```
df.describe(include=["object", "bool"])
```

	active_status	compa_ratio_range	company	cost_center	country	current_rating	date_of_birth	ethnicity	gender	generation
count	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
unique	2	3	1	5	6	5	943	5	2	4
top	True	Below Compa-Ratio	Super	AMER - United States of America	United States of America	2 - Needs Improvement	04/22/1990	White	Female	Generation Y/Millenials (1981-1996)
freq	832	439	1000	256	256	215	4	461	515	570

```
df.describe(include="all")
```

	active_status	compa_ratio	compa_ratio_range	company	cost_center	country	current_rating	date_of_birth	employee_id	ethnicity	gender	generation
count	1000	1000.000000	1000	1000	1000	1000	1000	1000	1000.000000	1000	1000	1000
unique	2	NaN	3	1	5	6	5	943	NaN	5	2	4
top	True	NaN	Below Compa-Ratio	Super	AMER - United States of America	United States of America	2 - Needs Improvement	04/22/1990	NaN	White	Female	Generation Y/Millenials (1981-1996)
freq	832	NaN	439	1000	256	256	215	4	NaN	461	515	570
mean	NaN	0.976774	NaN	NaN	NaN	NaN	NaN	NaN	1499.500000	NaN	NaN	NaN
std	NaN	0.153751	NaN	NaN	NaN	NaN	NaN	NaN	288.819436	NaN	NaN	NaN
min	NaN	0.400000	NaN	NaN	NaN	NaN	NaN	NaN	1000.000000	NaN	NaN	NaN
25%	NaN	0.898000	NaN	NaN	NaN	NaN	NaN	NaN	1249.750000	NaN	NaN	NaN
50%	NaN	0.971000	NaN	NaN	NaN	NaN	NaN	NaN	1499.500000	NaN	NaN	NaN
75%	NaN	1.055250	NaN	NaN	NaN	NaN	NaN	NaN	1749.250000	NaN	NaN	NaN
max	NaN	1.600000	NaN	NaN	NaN	NaN	NaN	NaN	1999.000000	NaN	NaN	NaN

Distribution of boolean features

```
print(df["active_status"].value_counts())
```

```
True     832
False    168
```

Distribution of categorical features

```
print(df["ethnicity"].value_counts())
```

```
Name: active_status, dtype: int64
White                                461
Asian                                291
Hispanic                             123
Black or African American             76
American Indian or Alaska Native      49
Name: ethnicity, dtype: int64
```

```
df.describe()
```

generates descriptive statistics

- central tendency,
- dispersion,
- shape of the dataset's distribution,
- …

	compa_ratio	employee_id	length_of_service	manager_id	organization_level	total_compensation
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000
mean	0.976774	1499.500000	5.419683	1452.930000	3.89700	4056.791630
std	0.153751	288.819436	5.710193	330.096426	1.10798	3004.498829
min	0.400000	1000.000000	0.005000	1019.000000	1.00000	800.000000
25%	0.898000	1249.750000	1.151750	1114.000000	4.00000	2249.850000
50%	0.971000	1499.500000	3.680500	1428.000000	4.00000	3340.720000
75%	1.055250	1749.250000	7.815000	1823.000000	5.00000	4420.240000
max	1.600000	1999.000000	36.153000	1998.000000	5.00000	20000.000000

Summary tables: contingency table

```
pd.crosstab(df["country"],
df["ethnicity"])
```

ethnicity	American Indian or Alaska Native	Asian	Black or African American	Hispanic	White
country					
Canada	23	48	24	40	102
France	0	15	0	15	106
Singapore	0	79	8	0	9
South Korea	0	83	10	0	13
United Kingdom	0	19	0	18	132
United States of America	26	47	34	50	99

Summary tables: contingency table

```
# Normalized contingency table to see ratios
pd.crosstab(df["country"], df["ethnicity"],
normalize=True)
```

ethnicity	American Indian or Alaska Native	Asian	Black or African American	Hispanic	White
country					
Canada	0.023	0.048	0.024	0.040	0.102
France	0.000	0.015	0.000	0.015	0.106
Singapore	0.000	0.079	0.008	0.000	0.009
South Korea	0.000	0.083	0.010	0.000	0.013
United Kingdom	0.000	0.019	0.000	0.018	0.132
United States of America	0.026	0.047	0.034	0.050	0.099

Summary tables: pivot table

```
df.pivot_table(
    ["length_of_service", "total_compensation"],
    ["generation"],
    aggfunc="mean",
)
```

generation	length_of_service	total_compensation
Baby Boomers (1946-1964)	26.274000	6150.880000
Generation X (1965-1980)	13.280067	3823.432164
Generation Y/Millenials (1981-1996)	5.660291	4084.914421
Generation Z (1997 and onwards)	1.143229	4087.364710

Modify feature type

Modify feature type

```python
# ID is categorical, not numeric
df["employee_id"] = df["employee_id"].astype(str)

# Verify
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 42 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   active_status         1000 non-null   bool
 1   compa_ratio           1000 non-null   float64
 2   compa_ratio_range     1000 non-null   object
 3   company               1000 non-null   object
 4   cost_center           1000 non-null   object
 5   country               1000 non-null   object
 6   current_rating        1000 non-null   object
 7   date_of_birth         1000 non-null   object
 8   employee_id           1000 non-null   object
 9   ethnicity             1000 non-null   object
```

Modify feature type

```
# Date format, not categorical
df['date_of_birth'] = pd.to_datetime(df['date_of_birth'])

# Verify
df.head()
```

	active_status	compa_ratio	compa_ratio_range	company	cost_center	country	current_rating	date_of_birth	employee_id
0	True	1.088	Above Compa-Ratio	Super	Global Support - Asia/Pac	Singapore	1 - Unsatisfactory	1999-04-05	1000
1	True	0.826	Below Compa-Ratio	Super	AMER - United States of America	United States of America	4 - Exceeds Expectations	1995-08-26	1001
2	True	0.998	At Compa-Ratio	Super	Global Support - Asia/Pac	South Korea	2 - Needs Improvement	1994-06-10	1002
3	True	0.918	Below Compa-Ratio	Super	AMER - Canada	Canada	3 - Meets Expectations	2001-05-09	1003
4	True	1.343	Above Compa-Ratio	Super	EMEA - United Kingdom	United Kingdom	5 - Outstanding Performance	1984-06-13	1004

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 42 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   active_status         1000 non-null   bool
 1   compa_ratio           1000 non-null   float64
 2   compa_ratio_range     1000 non-null   object
 3   company               1000 non-null   object
 4   cost_center           1000 non-null   object
 5   country               1000 non-null   object
 6   current_rating        1000 non-null   object
 7   date_of_birth         1000 non-null   datetime64[ns]
 8   employee_id           1000 non-null   object
```

czechitas

Select useful columns

Drop columns

```
columns_to_drop = ["active_status", "company", "hire_date",
                   "organization_level",
                   "termination_category", "termination_date",
                   "termination_reason", "worker_type"]
df.drop(columns=columns_to_drop, inplace= True)
```

Select columns

```
columns_to_select = ["employee_id", "gender", "ethnicity",
                     "is_manager", "job_level"]
df[columns_to_select]
```

	employee_id	gender	ethnicity	is_manager	job_level
0	1000	Female	Asian	False	P4
1	1001	Male	Hispanic	False	P1
2	1002	Male	Asian	False	P2
3	1003	Male	White	False	P4
4	1004	Female	White	False	P1
...	...	...	...	...	...
995	1995	Male	Asian	False	P1
996	1996	Female	Black or African American	False	P2
997	1997	Male	White	False	P5
998	1998	Male	Asian	True	M3
999	1999	Female	White	False	P2

1000 rows × 5 columns

Select columns

```
# Add a new column name to the list
columns_to_select += ["location"]
df[columns_to_select]
```

	employee_id	gender	ethnicity	is_manager	job_level	location
0	1000	Female	Asian	False	P4	Singapore
1	1001	Male	Hispanic	False	P1	San Francisco
2	1002	Male	Asian	False	P2	Seoul
3	1003	Male	White	False	P4	Toronto
4	1004	Female	White	False	P1	London
...	...	...	...	...	...	...
995	1995	Male	Asian	False	P1	Singapore
996	1996	Female	Black or African American	False	P2	New York
997	1997	Male	White	False	P5	London
998	1998	Male	Asian	True	M3	San Francisco
999	1999	Female	White	False	P2	London

1000 rows × 6 columns

Filtering observations

Sorting data

```
# Sorting by one column values
df.sort_values(by="length_of_service", ascending=False).head()
```

```
# Sorting by multiple column values
df.sort_values(by=["location", "length_of_service"],
               ascending=[True, False]).head()
```

Filtering observations based on one column

```
# Filter base on one column
df[df['gender'] == 'Female'].head()
```

Boolean operators

Operator	Description
==	True if *a* is equal to *b*, else False
!=	True if *a* is not equal to *b*, else False
>	True if *a* is greater that *b*, else False
<	True if *a* is less that *b*, else False
>=	True if *a* is greater that or equal to *b*, else False
<=	True if *a* is less that or equal to *b*, else False

Logical operators

Python operator	Pandas operator	Description
AND	&	True only if both operands are True
OR	|	True if any operand is True
NOT	~	Returns the opposite of its operand

Filtering observations based on multiple columns

```
# Filter base on multiple columns
df[(df['gender'] == 'Female') & (df['total_compensation'] >=
    10000)].head()
```

Pandas *isin()* method

takes a sequence of values and returns True at the positions within the Series that match the values in the list

```
# select employees with IT - Product Development or Product Management
job family (using the logical operator or)
df[(df['single_job_family'] == 'IT - Product Development') |
    (df['single_job_family'] == 'IT - Product Management')]

# select employees with IT - Product Development or Product Management
job family (using the isin method)
df[df['single_job_family'].isin(['IT - Product Development',
                                 'IT - Product Management'])]
```

czechitas

Pandas *between()* method

takes two scalars separated by a comma which represent the lower and upper boundaries of a range of values and returns True at the positions that lie within that range

```
# employees with a salary higher than or equal to 3000 and less than or
equal to 8000
df[df['total_compensation'].between(3000, 8000)]

# employees with a salary higher than 3000 and less than 8000 euros
df[df['total_compensation'].between(3000, 8000, inclusive=False)]
```

Pandas string methods

str.contains()

str.startswith()

str.endswith()

```
# select all employees with managerial job level, i.e., containig "M"
df[df['job_level'].str.contains('M')]

# select employees whose name starts with 'A'
df[df['legal_full_name'].str.startswith('A')]

# select employees whose name ends with 'ez'
df[df['legal_full_name'].str.endswith('ez')]
```

Combine it all together

Filter only women based in the UK and see distribution of years of service and their salary

1

```
# How is UK logged in the data?
print(df.country.unique())
```

```
['Singapore' 'United States of America' 'South Korea' 'Canada'
 'United Kingdom' 'France']
```

2

```
# Filter data
df_new = df[(df['gender'] == 'Female')
            & (df['country'] == "United Kingdom")]
df_new.head()
```

czechitas

Combine it all together

Filter only women based in the UK and see distribution of years of service and their salary

3

```
# What are the column names for years of service and salary?
print(df_new.columns)
```

```
Index(['compa_ratio', 'compa_ratio_range', 'cost_center', 'country',
       'current_rating', 'date_of_birth', 'employee_id', 'ethnicity', 'gender',
       'generation', 'high_performer', 'high_potential', 'hiring_source',
       'is_leader', 'is_manager', 'job_family_group', 'job_level',
       'job_profile', 'legal_full_name', 'length_of_service', 'location',
       'management_level', 'manager_id', 'org_level_1', 'org_level_2',
       'org_level_3', 'org_level_4', 'org_level_5', 'region',
       'report_effective_date', 'single_job_family', 'supervisory_org',
       'tenure_category', 'total_compensation'],
      dtype='object')
```

Combine it all together

Filter only women based in the UK and see distribution of years of service and their salary

4

```
# Select only needed columns
df_new = df_new[["employee_id", "length_of_service",
"total_compensation"]]
```

5

```
# Summarize years of service and salary
df_new.describe()
```

	length_of_service	total_compensation
count	88.000000	88.000000
mean	6.234682	4260.989659
std	6.737775	3409.375398
min	0.005000	1289.380000
25%	1.039500	2110.940000
50%	3.890500	3502.135000
75%	9.291250	4521.785000
max	27.797000	17811.100000

czechitas

Data format

wide vs. long, one-hot encoding

Data format – wide

PersonID	Gender	Income	Years Education	Age
2343	F	50 000	17	35
1213	M	35 000	15	32
4533	M	40 000	15	53
4563	M	100 000	19	51
7554	F	50 000	18	28
6465	F	27 500	13	25
7453	M	34 000	13	32
…	…	…	…	…

Typically: One column = one variable

→ **1 OBS = All Employee Information**

czechitas

Data format – wide

```
# Wide data format (original data)
df_wide = df[["employee_id", "country", "ethnicity", "gender",
              "hiring_source", "job_family_group", "total_compensation"]]
df_wide.head()
```

	employee_id	country	ethnicity	gender	hiring_source	job_family_group	total_compensation
0	1000	Singapore	Asian	Female	University	Executive	4788.30
1	1001	United States of America	Hispanic	Male	Corporate Website	Services	1238.43
2	1002	South Korea	Asian	Male	Linkedin	Finance	2395.47
3	1003	Canada	White	Male	Recruiter	Sales	4133.05
4	1004	United Kingdom	White	Female	Internal	Sales	1745.30

Typically: One column = one variable

→ **1 OBS = All Employee Information**

czechitas

Data format – long

PersonID	Gender	Income	Years Education	Age
2343	F	50 000	17	35
1213	M	35 000	15	32
4533	M	40 000	15	53
4563	M	100 000	19	51
7554	F	50 000	18	28
6465	F	27 500	13	25
7453	M	34 000	13	32
…	…	…	…	…

PersonID	Variable	Value
2343	Gender	F
2343	Income	50000
2343	Education	17
2343	Age	35
1213	Gender	M
1213	Income	35000
1213	Education	15
…	…	…

Sometimes: Variable names in one column, values in another column

→ **1 OBS = PersonID + Variable**

Usage:

- data processing in bulk (e.g., conversion to numeric),
- visualizations of multiple variables in one plot

czechitas

Data format – long

```
# Long format
df_long = pd.melt(df_wide, id_vars= "employee_id", value_vars=[ "country",
                  "ethnicity", "gender", "hiring_source",
                  "job_family_group", "total_compensation" ])
df_long.head()
```

	employee_id	variable	value
0	1000	country	Singapore
1	1001	country	United States of America
2	1002	country	South Korea
3	1003	country	Canada
4	1004	country	United Kingdom

```
df_long.sort_values(by= "employee_id").head(
)
```

	employee_id	variable	value
0	1000	country	Singapore
2000	1000	gender	Female
1000	1000	ethnicity	Asian
5000	1000	total_compensation	4788.3
4000	1000	job_family_group	Executive

→ **1 OBS = PersonID + Variable**

czechitas

Data format – wide vs. long

PersonID	Gender	Income	Years Education	Age
2343	F	50 000	17	35
1213	M	35 000	15	32
4533	M	40 000	15	53
4563	M	100 000	19	51
7554	F	50 000	18	28
6465	F	27 500	13	25
7453	M	34 000	13	32
…	…	…	…	…

PersonID	Variable	Value
2343	Gender	F
2343	Income	50000
2343	Education	17
2343	Age	35
1213	Gender	M
1213	Income	35000
1213	Education	15
…	…	…

→ Wide data format, **1 OBS = All Employee Information**

→ Long data format, **1 OBS = PersonID + Variable**

czechitas

One-hot encoding of non-numeric data

PersonID	Gender
2343	F
1213	M
4533	M
4563	M
7554	F
6465	F
7453	M
...	...

PersonID	GenderMale
2343	0
1213	1
4533	1
4563	1
7554	0
6465	0
7453	1
...	...

Categorical variables need to be transformed into numeric ones.

Usually, one category is dropped (use only male gender here) to avoid multicollinearity.

One-hot encoding of non-numeric data

```
# Select only ID and gender columns
df_wide2 = df_wide[["employee_id", "gender"]]

# Create dummy for gender
isMale = pd.get_dummies(df_wide2.gender, prefix= "Is", drop_first=True)

# Add to original DataFrame
pd.concat([df_wide2, isMale], axis= 1)
```

	employee_id	gender	Is_Male
0	1000	Female	0
1	1001	Male	1
2	1002	Male	1
3	1003	Male	1
4	1004	Female	0

One-hot encoding of non-numeric data

```
# Select only ID and country columns
df_wide3 = df_wide[["employee_id", "country"]]

# Create dummy for countries
countries = pd.get_dummies(df_wide3.country, prefix= "Country: ",
prefix_sep="")

# Add to original DataFrame
pd.concat([df_wide3, countries], axis= 1)
```

	employee_id	country	Country: Canada	Country: France	Country: Singapore	Country: South Korea	Country: United Kingdom	Country: United States of America
0	1000	Singapore	0	0	1	0	0	0
1	1001	United States of America	0	0	0	0	0	1
2	1002	South Korea	0	0	0	1	0	0
3	1003	Canada	1	0	0	0	0	0
4	1004	United Kingdom	0	0	0	0	1	0

One-hot encoding of non-numeric data

```
# Pass entire DataFrame to get_dummies()
pd.get_dummies(df_wide, columns=[ 'gender', 'country'])
```

	employee_id	ethnicity	hiring_source	job_family_group	total_compensation	gender_Female	gender_Male	country_Canada	country_France
0	1000	Asian	University	Executive	4788.30	1	0	0	0
1	1001	Hispanic	Corporate Website	Services	1238.43	0	1	0	0
2	1002	Asian	Linkedin	Finance	2395.47	0	1	0	0
3	1003	White	Recruiter	Sales	4133.05	0	1	1	0
4	1004	White	Internal	Sales	1745.30	1	0	0	0

```
pd.get_dummies(df_wide, columns=[ 'gender'], drop_first=True)
```

	employee_id	country	ethnicity	hiring_source	job_family_group	total_compensation	gender_Male
0	1000	Singapore	Asian	University	Executive	4788.30	0
1	1001	United States of America	Hispanic	Corporate Website	Services	1238.43	1
2	1002	South Korea	Asian	Linkedin	Finance	2395.47	1

Calculating new column based on other columns

Calculation Method

- Arithmetic operations: addition, subtraction, multiplication, division (+,-,/,*)
- Conditional statements: IF, ELSE, ELSEIF
- Text transformation .str, .split(), .replace(),
- regex operations .findall(), .extract()
- method .apply()

Price	Floor space (m^2)
16 000	50
25 000	40
25 200	50
19 999	60
25 200	50
...	...

Price per m^2
320
625
504
333
504
...

```
data['cena/metr'] = data['cena'] / data['plocha']
```

czechitas

Data cleaning

Check variables distributions, outliers, remove such observations or treat them like missing values, visualisation and exploration

Data cleaning takes the most time

Purpose:

- Extract useful information
- Check that you data makes sense
- see missing values

- Check outliers, remove such observations or treat them
- Check variables distributions
- visualisation and exploration

Identifying Outliers

- Z-scores
- IQR method
- Box plots

Z-score

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean

σ = Standard Deviation

Interquartile range

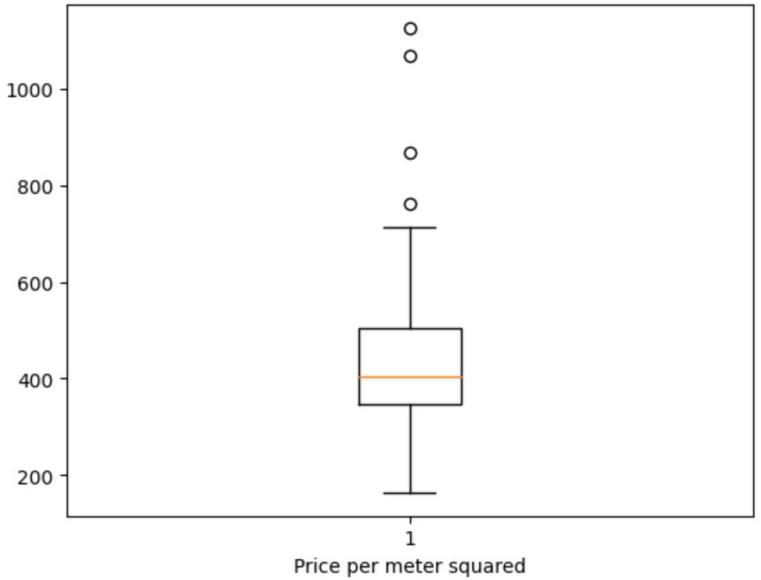
IQR = Q3 - Q1

quartile divides dataset into 4 equal parts when they are sorted from lowest to highest

Box plot

is a graphical representation of five summary statistics:

- minimum,
- first quartile (Q1),
- median (Q2),
- third quartile (Q3),
- maximum.



Handling Outliers

- Remove observations
- Treat as missing values
- Replace with less extreme
- Data transformation (scaling)

Missing Values

- Listwise deletion
- Imputation (mean, median, mode)
- Interpolation
- Predictive models

Univariate Visualization

- Histograms, density plots
- Box plots, violin plots
- Bar charts, pie charts

Bivariate visualizations

- Scatter plots
- Line charts, area charts
- Heatmaps, bubble charts

Data aggregation

group by

Overview

- What is data aggregation?
- Purpose and benefits of data aggregation
- Common use cases: summarizing data, reporting, analytics

Types of Data Aggregations

- Sum, average, count
- Min, max, median
- Standard deviation, variance

Grouping Data

- Categorical variables
- Binning continuous variables
- Time-based variables

Scaling & normalization

Turn columns to the same scales

Different Scales

- Different ranges of values across columns

```
# Track obs w/ too many NAs
nbr_obs = 20
df = pd.DataFrame({})
df["Var1"] = np.random.default_rng().uniform( 0, 1, nbr_obs)
df["Var2"] = np.random.default_rng().uniform( 500, 1000, nbr_obs)
df["Var3"] = np.random.default_rng().uniform( -5000, 10000, nbr_obs)
df["Var4"] = np.random.default_rng().uniform( .0002, .0003, nbr_obs)
```

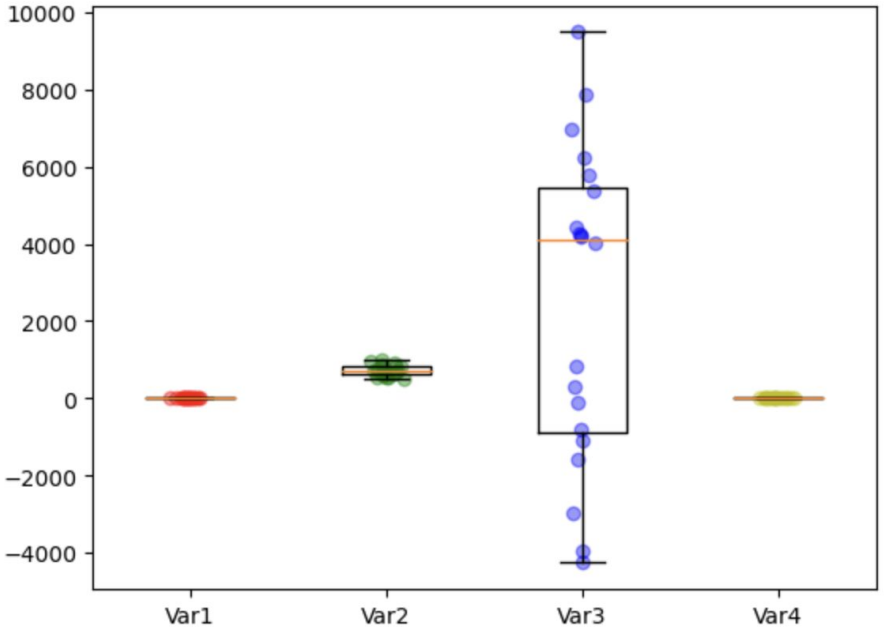
Different Scales

- Different ranges of values across columns

	Var1	Var2	Var3	Var4
0	0.628552	731.190348	4440.828927	0.000237
1	0.144817	828.813006	-3956.805629	0.000209
2	0.585508	660.766838	4225.764669	0.000274
3	0.290634	846.781968	7883.906667	0.000277
4	0.264281	745.271865	820.913692	0.000283

Different Scales

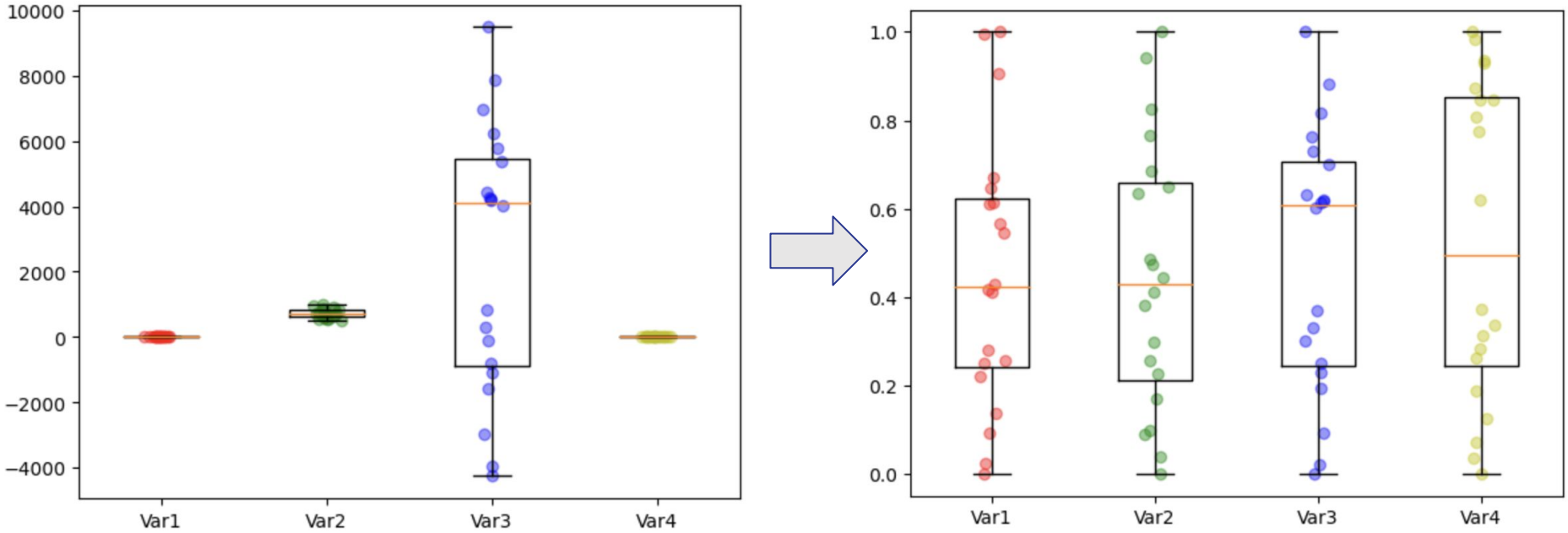
- Different ranges of values across columns



czechitas

Scaling/Normalising = Putting all cols on similar scales

- After scaling…



Why Scaling/Normalising before Algo?

- Visualise
- Some algos get **biased** when not scaled/norm...
- Some algos **require** it
- Sometimes numerical requirements to run

Normalising = Spreading cols in [0,1]

	Var1	Var2	Var3	Var4
0	0.628552	731.190348	4440.828927	0.000237
1	0.144817	828.813006	-3956.805629	0.000209
2	0.585508	660.766838	4225.764669	0.000274
3	0.290634	846.781968	7883.906667	0.000277
4	0.264281	745.271865	820.913692	0.000283

For each Var1, …, Var4:

- get col_max and col_min
- for each old value x, do:

  $y = (x - col\_min) / (col\_max - col\_min)$

czechitas

Normalising = Spreading cols in [0,1]

```
# Use MinMaxScaler function to normalise
from sklearn.preprocessing import MinMaxScaler
trans = MinMaxScaler()
df_sc = trans.fit_transform(df)
df_sc = pd.DataFrame(df_sc, columns=[ 'Var1','Var2','Var3','Var4'])
```

	Var1	Var2	Var3	Var4
0	0.613351	0.445308	0.633013	0.373502
1	0.093926	0.649611	0.021581	0.073951
2	0.567131	0.297926	0.617354	0.776647
3	0.250502	0.687216	0.883704	0.806947
4	0.222205	0.474777	0.369447	0.872548

Scaling = Transform cols to get mean = 0, std = 1

	Var1	Var2	Var3	Var4
0	0.628552	731.190348	4440.828927	0.000237
1	0.144817	828.813006	-3956.805629	0.000209
2	0.585508	660.766838	4225.764669	0.000274
3	0.290634	846.781968	7883.906667	0.000277
4	0.264281	745.271865	820.913692	0.000283

For each Var1, …, Var4:

- get col_mean and col_std
- for each old value x, do:

$$y = (x - \text{col\_mean}) / \text{col\_std}$$

czechitas

Scaling = Transform cols to get mean = 0, std = 1

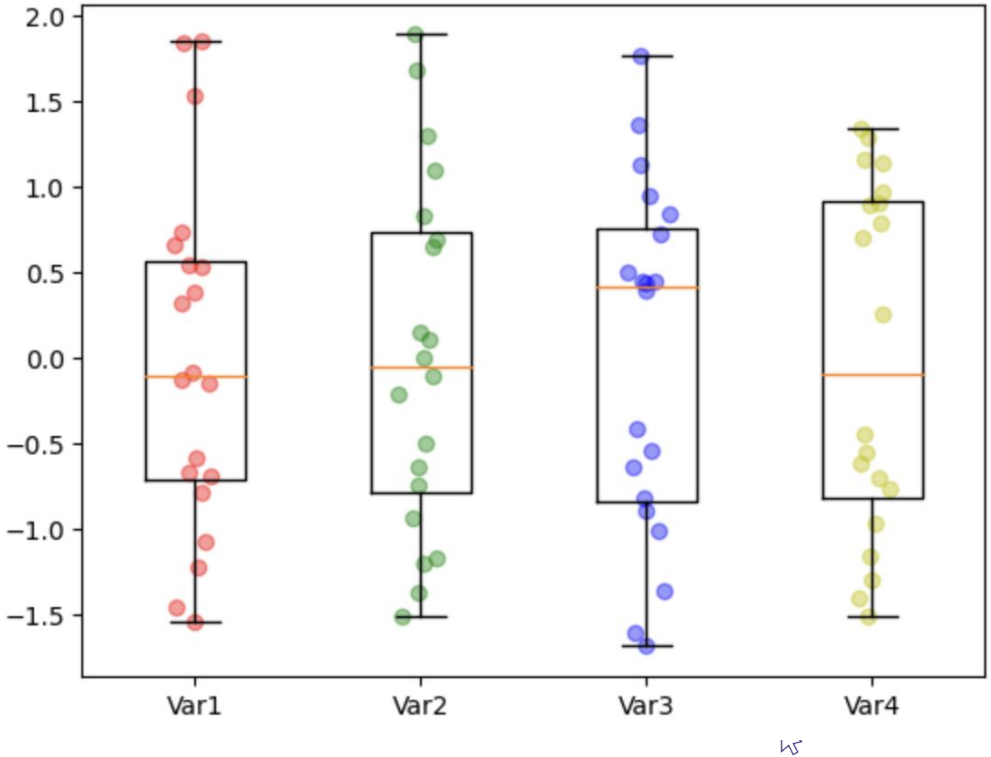
```
# Use MinMaxScaler function to normalise
from sklearn.preprocessing import  StandardScaler

# define standard scaler
scaler = StandardScaler()
# transform data
scaled = pd.DataFrame(scaler.fit_transform(df),
columns=['Var1','Var2','Var3','Var4'])
```

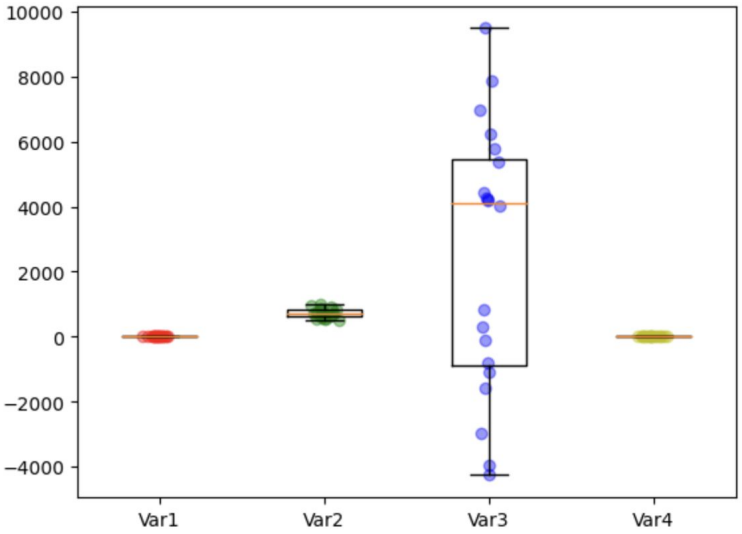
	Var1	Var2	Var3	Var4
0	0.538194	0.002226	0.496616	-0.448265
1	-1.222892	0.696970	-1.610933	-1.301488
2	0.381488	-0.498952	0.442642	0.700030
3	-0.692028	0.824848	1.360724	0.786334
4	-0.787969	0.102439	-0.411872	0.973188

Scaling = Transform cols to get mean = 0, std = 1

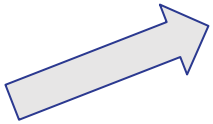
	Var1	Var2	Var3	Var4
0	0.538194	0.002226	0.496616	-0.448265
1	-1.222892	0.696970	-1.610933	-1.301488
2	0.381488	-0.498952	0.442642	0.700030
3	-0.692028	0.824848	1.360724	0.786334
4	-0.787969	0.102439	-0.411872	0.973188

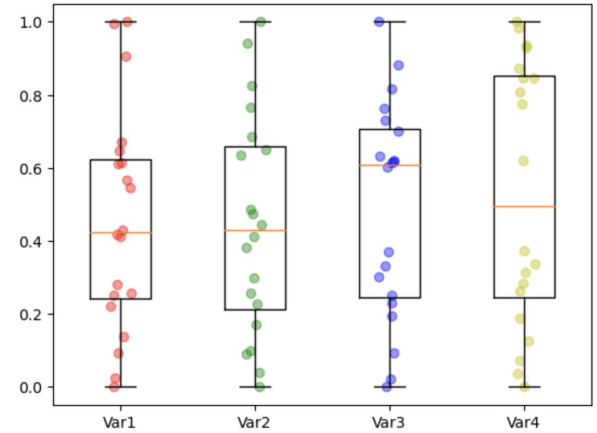


Scaling/Normalising comparison

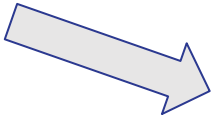


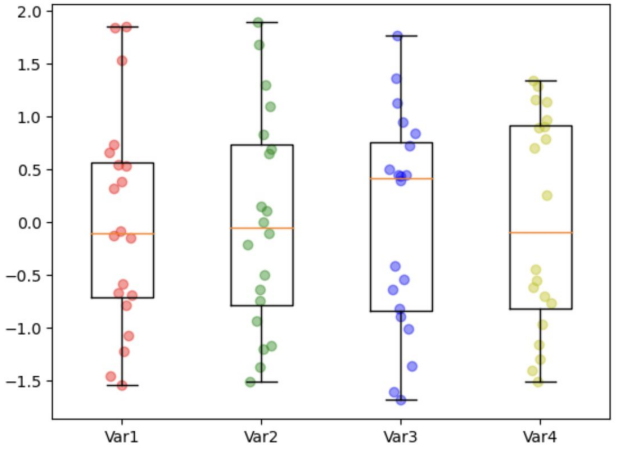
Normalising





Scaling





Scaling/Normalising comparison

- Some algos do not need either (Random forests...)
- Never harms to scale/normalise
- No clear guideline to choose scale/normalise
- Can run both, run models and compare performance...

Handling Missing Values

Why missing values happen

- System errors
- Bad design of the process
- Incompetence
- Reluctance to report data
- Some data not being collected
- Tendency to hide certain data
- Errors in preparing data for example incorrect joins

! If you are designing a study or a survey, think how to create your process that amount of missing data would be minimized

Why to deal with and think about missing values

Some algorithms will not work if you pass data with missing observations to them

Some algorithms will work if you pass missing observations but will have default behavior which might not necessary the right one for your case

Having missing data will impact results of your analysis

What are the types of missing values

weight ~ f(gender)

MCAR – missing completely at random

Error entering observation to the system

MAR – missing at random

Stereotype – women are more likely to not report their weight

MNAR – missing not at random

Society stigma – people with big weight would not report their weight

Quiz – assign situation to type of missing data

System errors

Not answering complicated question in survey

Being diagnosed with Covid-19 and not reporting all contacts when having many of them

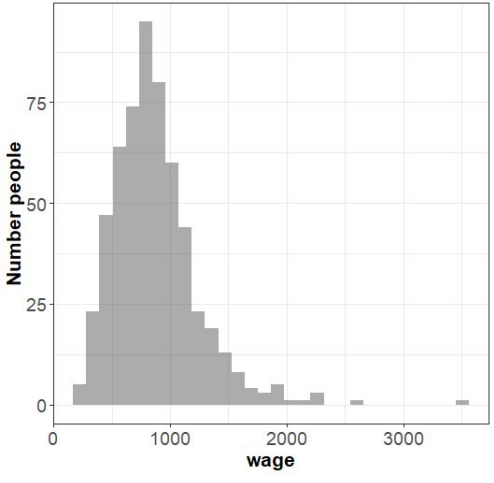
Reluctance to report high income

MCAR – missing completely at random	MAR – missing at random	MNAR – missing not at random
Ex: Error entering observation to the system	Ex: Stereotype – women are more likely to not report their weight	Ex: Society stigma – people with big weight would not report their weight

czechitas

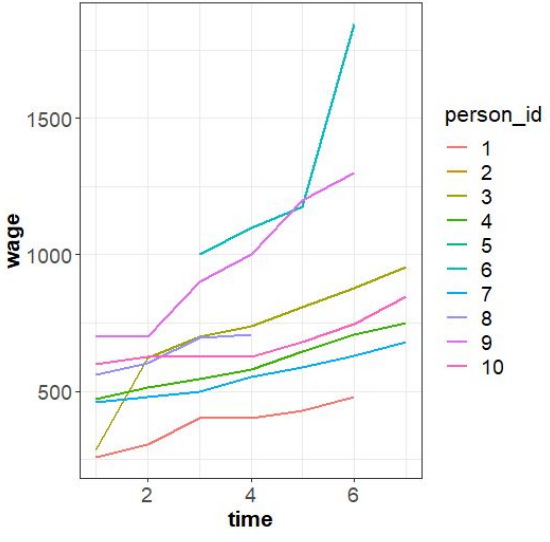
Exploration of missing structure

- Cross-sectional data: hard from data itself to see which type of missing structure you have
- Some patterns still possible to notice



sex	percent_missing_married_status
<fct>	<dbl>
female	20.6
male	10.4

- If you have time series, you can use the time dependent structure to understand



- If you have panel data, you can even get from data which type you have

id	time	wage	n_missing	type
6	1	NA	3	some_missing_observations
6	2	NA	3	some_missing_observations
6	3	1000.0047	3	some_missing_observations
6	4	1100.0050	3	some_missing_observations
6	5	1174.9960	3	some_missing_observations
6	6	1844.9916	3	some_missing_observations
6	7	NA	3	some_missing_observations
7	1	461.0009	0	no_missing_observations
7	2	480.0019	0	no_missing_observations
7	3	499.0019	0	no_missing_observations
7	4	552.0011	0	no_missing_observations
7	5	586.9972	0	no_missing_observations
7	6	630.0001	0	no_missing_observations
7	7	679.9981	0	no_missing_observations

type	mean_salary	median_salary
no_missing_observations	808	744
some_missing_observations	910	846

Handling Missing Values

1. Deduce missing values

Some missing values can be deduced

Age, years of experience can be deduced

id	time	exp	sex	wage
1	1	3	male	259.9996
1	2	4	male	304.9995
1	3	5	male	401.9992
1	4	NA	NA	401.9992
1	5	7	male	429.0013
1	6	8	male	480.0019
1	7	9	male	NA

We don't need any model to deduce missing gender observations

czechitas

Handling Missing Values

2. Remove observations/variables

Removing incomplete data

```
# Track obs w/ too many NAs
threshold = 0.5
nbr_cols = len(df.columns)
obs_to_keep = df.isnull().sum(axis=0) / nbr_cols    <   threshold
df = df[obs_to_keep]
```

var1	var2	var3	var4	var5	var6	var7
X	X	X	X	X	X	X
X	NA	X	NA	NA	NA	NA
X	X	X	X	X	X	NA
X	X	NA	NA	X	X	NA
X	NA	X	X	X	X	X
X	X	X	X	X	X	NA
X	X	X	X	X	X	NA

Row/observation deletion

czechitas

Quiz: When there will be biggest impact when removing rows?

MCAR – missing completely at random

Ex: Error entering observation to the system

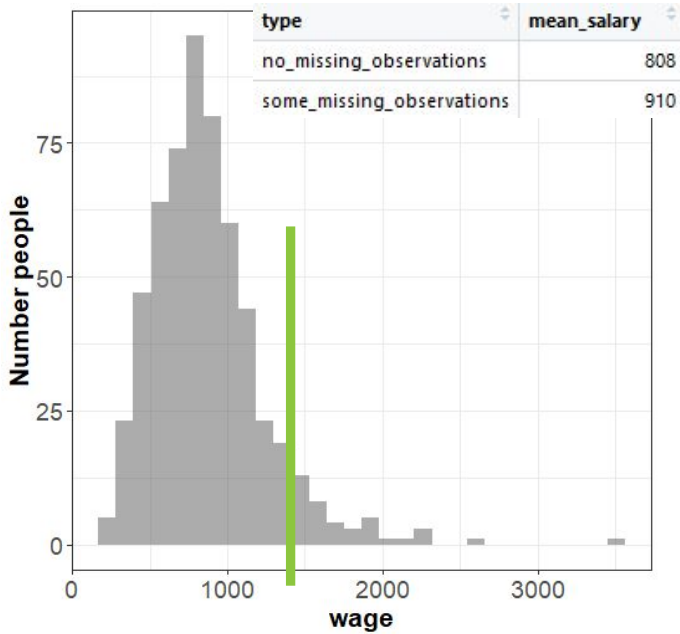
MAR – missing at random

Ex: Stereotype – women are more likely to not report their weight

MNAR – missing not at random

Ex: Society stigma – people with big weight would not report their weight

type	mean_salary	median_salary
no_missing_observations	808	744
some_missing_observations	910	846



czechitas

Removing incomplete data

```
# Track cols w/ too many NAs
threshold = 0.5
nbr_obs = len(df)
col_to_drop = df.columns[df.isnull().sum(axis=1) / nbr_obs   >  threshold]
df = df.drop(col_to_drop, axis=1)
```

var1	var2	var3	var4	var5	var6	var7
X	X	X	X	X	X	X
X	NA	X	NA	NA	NA	NA
X	X	X	X	X	X	NA
X	X	NA	NA	X	X	NA
X	NA	X	X	X	X	X
X	X	X	X	X	X	NA
X	X	X	X	X	X	NA

Column deletion

czechitas

When variable deletion is unavoidable

1) **Variable has way too many values missing and variable does not correlate strongly with anything else**

- losing too many observations when building a model
- imputation is too imprecise

id	time	variable
540	4	7.601
540	5	*NA*
540	6	*NA*
540	7	*NA*
542	1	6.540
542	2	6.537
542	3	*NA*
542	4	*NA*
542	5	*NA*
542	6	*NA*
542	7	7.060
543	1	*NA*
543	2	*NA*
543	3	*NA*
543	4	*NA*
543	5	*NA*

2.) **Variable has too many categories and many values missing**

- Not much additional value for a model and we still lose observations

id	time	serial_number
1	1	PgZWv
1	2	UaiGb
1	3	E5TDF
1	4	6f81e
1	5	XQBTc
1	6	ED4Rz
1	7	ovBOQ
2	1	iV8OK
2	2	IlmF8
2	3	p9c93
2	4	OFwsy
2	5	UGqFB
2	6	JGt6H
2	7	dajNH
3	1	qnohm

Impact: no conclusions about that variable, possible bias to the model

czechitas

Handling Missing Values

3. Simple imputation methods

Start all methodologies from simple steps

ADVANTAGES of simple methods for imputation:

- Short computational time
- Easy to understand
- Good performance if we have enough information about variables

1. Mean or Median

- Suitable when there is no trend and no seasonality.

Two types of data:

a)

id	log_wage
1	5.561
2	6.163
3	5.652
4	6.157
5	6.438
6	*NA*
7	6.133
8	6.332
9	6.551
10	6.397
11	*NA*
12	6.551
13	6.906

b)

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	*NA*
2	1	6.163
2	2	*NA*
2	3	6.263
2	4	*NA*
2	5	*NA*
2	6	*NA*
2	7	6.816

Is it a good solution for both data structures to impute the overall mean of the whole dataset?

czechitas

1. Mean or Median

- Suitable when there is no trend and no seasonality.

a) Cross-sectional data: one row has data for one individual

```
# Get mean and impute it where NAs
avg = df["log_wage"].mean()
obs_w_na = df["log_wage"].isnull()
df.loc[obs_w_na, "log_wage"] = avg
```

id	log_wage
1	5.561
2	6.163
3	5.652
4	6.157
5	6.438
6	NA
7	6.133
8	6.332
9	6.551
10	6.397
11	NA
12	6.551
13	6.906

Use mean of the whole dataset: 6.366

id	log_wage
1	5.561
2	6.163
3	5.652
4	6.157
5	6.438
6	6.366
7	6.133
8	6.332
9	6.551
10	6.397
11	6.366
12	6.551
13	6.906

czechitas

1. Mean or Median

- Suitable when there is no trend and no seasonality.

a) Cross-sectional data: one row has data for one individual

id	log_wage
1	5.561
2	6.163
3	5.652
4	6.157
5	6.438
6	NA
7	6.133
8	6.332
9	6.551
10	6.397
11	NA
12	6.551
13	6.906

Use mean of the whole dataset: 6.366

id	log_wage
1	5.561
2	6.163
3	5.652
4	6.157
5	6.438
6	6.366
7	6.133
8	6.332
9	6.551
10	6.397
11	6.366
12	6.551
13	6.906

czechitas

1. Mean or Median

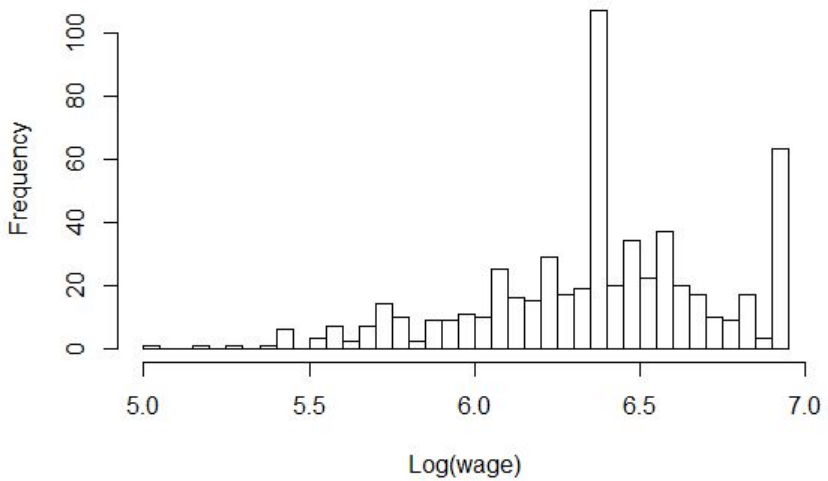
- Suitable when there is no trend and no seasonality.

a) Cross-sectional data

Use mean of the whole dataset: 6.366

id	log_wage
1	5.561
2	6.163
3	5.652
4	6.157
5	6.438
6	6.366
7	6.133
8	6.332
9	6.551
10	6.397
11	6.366
12	6.551
13	6.906

DRAWBACK:
Spike at mean value & low variance



Original variance: 0.21
Variance after imputing mean: 0.12

czechitas

1. Mean or Median

- Suitable when there is no trend and no seasonality.

b) Time series data: one row has data for one individual at certain time point

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	*NA*
2	1	6.163
2	2	*NA*
2	3	6.263
2	4	*NA*
2	5	*NA*
2	6	*NA*
2	7	6.816

Use mean for individual

id	mean_log_wage
1	5.918
2	6.414

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	5.918
2	1	6.163
2	2	6.414
2	3	6.263
2	4	6.414
2	5	6.414
2	6	6.414
2	7	6.816

1. Mean or Median

- Suitable when there is no trend and no seasonality.

b) Time series data: one row has data for one individual at certain time point

```
# Get mean per group and impute it where NAs
mean_p_grp = df.groupby([ "id"], as_index=False).mean()
df = df.fillna(mean_p_grp)
```

1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	NA
2	1	6.163
2	2	NA
2	3	6.263
2	4	NA
2	5	NA
2	6	NA
2	7	6.816

Use mean for individual

id	mean_log_wage
1	5.918
2	6.414

1	4	5.996
1	5	6.061
1	6	6.174
1	7	5.918
2	1	6.163
2	2	6.414
2	3	6.263
2	4	6.414
2	5	6.414
2	6	6.414
2	7	6.816

1. Mean or Median

- Suitable when there is no trend and no seasonality.

b) Time series data

Use mean for individual

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	5.918
2	1	6.163
2	2	6.414
2	3	6.263
2	4	6.414
2	5	6.414
2	6	6.414
2	7	6.816

DRAWBACK: Assumption of no trend may not hold

WAGES: do they increase over lifetime or stagnate/decrease?

czechitas

1. Mean or Median

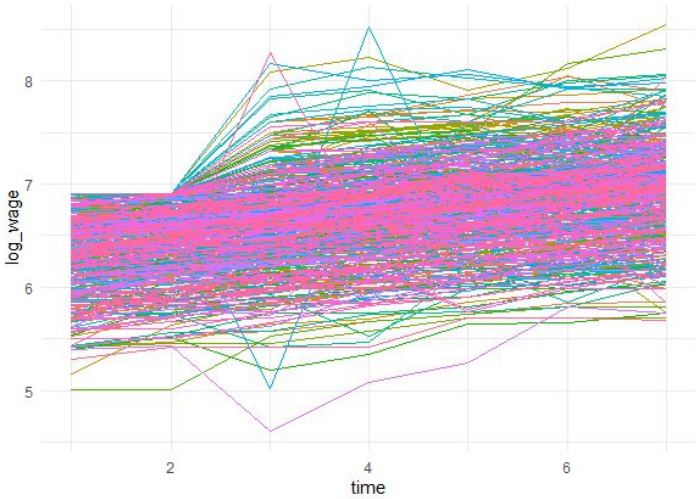
- Suitable when there is no trend and no seasonality.

b) Time series data

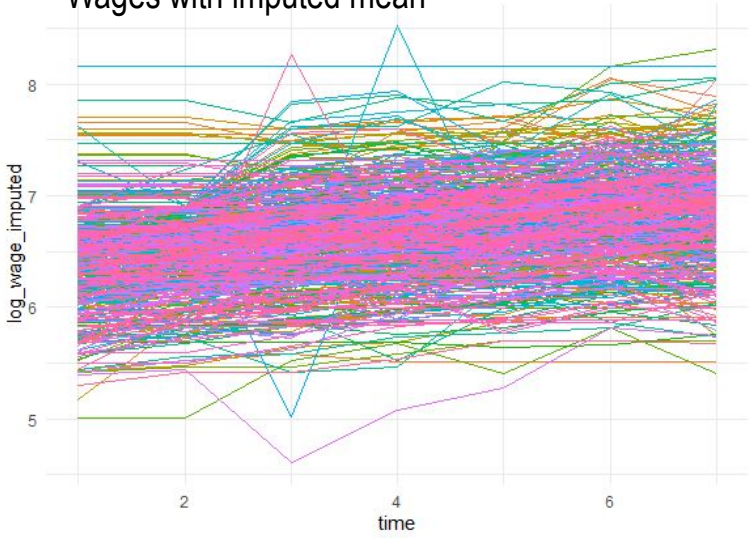
DRAWBACK: Assumption of no trend may not hold

Problematic outcome: truncation of the trend
Solution: exploratory analysis/gaining knowledge about variable before imputing

Original wages



Wages with imputed mean



chitas

2. Linear Interpolation

- Suitable when there is some trend and no seasonality.

id	time	log_wage
2	1	6.163
2	2	*NA*
2	3	6.263
2	4	*NA*
2	5	*NA*
2	6	*NA*
2	7	6.816

Mean between time 1 and time 3

Gradual steps between time 3 and time 7

$$\frac{6.816 - 6.263}{7 - 3} = \mathbf{0.14}$$

id	time	log_wage	
2	1	6.163	
2	2	6.213	
2	3	6.263	
2	4	6.401	+ 0.14
2	5	6.540	+ 0.14
2	6	6.678	+ 0.14
2	7	6.816	+ 0.14

czechitas

2. Linear Interpolation

- Suitable when there is some trend and no seasonality.

```
# Substitute NAs by linear interpolation
df["log_wage"] = df["log_wage"].interpolate()
```

id	time	log_wage
2	1	6.163
2	2	6.213
2	3	6.263
2	4	6.401
2	5	6.540
2	6	6.678
2	7	6.816

} + 0.14
} + 0.14
} + 0.14
} + 0.14

czechitas

2. Linear Interpolation

- Suitable when there is some trend and no seasonality.

id	time	log_wage	
2	1	6.163	
2	2	6.213	
2	3	6.263	+ 0.14
2	4	6.401	+ 0.14
2	5	6.540	+ 0.14
2	6	6.678	+ 0.14
2	7	6.816	

DRAWBACKS:

- Assumes the trend is linear
- At least two values are needed per individual

czechitas

3. Observation carry-over

a) Last Observation Carried Forward

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	6.174
2	1	6.163
2	2	6.163
2	3	6.263
2	4	6.263
2	5	6.263
2	6	6.263
2	7	6.816

b) Next Observation Carried Backward

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	NA
2	1	6.163
2	2	6.263
2	3	6.263
2	4	6.816
2	5	6.816
2	6	6.816
2	7	6.816

czechitas

3. Observation carry-over

Last Observation Carried Forward, Next Observation Carried Backward

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	6.174
2	1	6.163
2	2	6.163
2	3	6.263
2	4	6.263
2	5	6.263
2	6	6.263
2	7	6.816

id	time	log_wage
1	1	5.561
1	2	5.720
1	3	5.996
1	4	5.996
1	5	6.061
1	6	6.174
1	7	NA
2	1	6.163
2	2	6.263
2	3	6.263
2	4	6.816
2	5	6.816
2	6	6.816
2	7	6.816

DRAWBACKS:

- Overestimation/Underestimation of variable value
- At least one value per individual needed to get some imputed values

czechitas

SIMPLE METHODS SUMMARY

ADVANTAGES

- Fast computation
- Easy to understand
- Can have very good performance

DISADVANTAGES

- Can be imprecise
- Can lower the variance of the data
- For time series you need to know at least one data point

Handling Missing Values

4. Advanced imputation methods

Advanced Imputation Methods: leverage all data

Step 1: Use **algorithm** to estimate relationship between selected variable and all other variables.

id	time	log_wage	exp	wks	bluecol	ind	south	smsa	married	sex	union	ed	black
1	1	5.561	3	32	no	0	yes	no	yes	male	no	9	no
1	2	5.720	4	43	no	0	yes	no	yes	male	no	9	no
1	3	5.996	5	40	no	0	yes	no	yes	male	no	9	no
1	4	5.996	6	39	no	0	yes	no	yes	male	no	9	no
1	5	6.061	7	42	no	1	yes	no	yes	male	no	9	no
1	6	6.174	8	35	no	1	yes	no	yes	male	no	9	no
1	7	*NA*	9	32	no	1	yes	no	yes	male	no	9	no
2	1	6.163	30	34	yes	0	no	no	yes	male	no	11	no
2	2	*NA*	31	27	yes	0	no	no	yes	male	no	11	no
2	3	6.263	32	33	yes	1	no	no	yes	male	yes	11	no
2	4	*NA*	33	30	yes	1	no	no	yes	male	no	11	no
2	5	*NA*	34	30	yes	1	no	no	yes	male	no	11	no

Advanced Imputation Methods: Approach

Step 2: **Predict missing values** of selected variable based on the estimated relationship with all other variables.

.

Impute log(wage)

id	time	log_wage	exp	wks	bluecol	ind	south	smsa	married	sex	union	ed	black
1	1	5.561	3	32	no	0	yes	no	yes	male	no	9	no
1	2	5.720	4	43	no	0	yes	no	yes	male	no	9	no
1	3	5.996	5	40	no	0	yes	no	yes	male	no	9	no
1	4	5.996	6	39	no	0	yes	no	yes	male	no	9	no
1	5	6.061	7	42	no	1	yes	no	yes	male	no	9	no
1	6	6.174	8	35	no	1	yes	no	yes	male	no	9	no
1	7	6.331	9	32	no	1	yes	no	yes	male	no	9	no
2	1	6.163	30	34	yes	0	no	no	yes	male	no	11	no
2	2	6.151	31	27	yes	0	no	no	yes	male	no	11	no
2	3	6.263	32	33	yes	1	no	no	yes	male	yes	11	no
2	4	5.912	33	30	yes	1	no	no	yes	male	no	11	no
2	5	5.822	34	30	yes	1	no	no	yes	male	no	11	no

Advanced Imputation Methods

EXAMPLES

Linear Regression
Decision Trees, Random Forest
K-nearest neighbors

DISADVANTAGES

- Computation can take really long time sometimes
- Hard to track down how exactly the given value was calculated

Check the assumptions:

- Most methods work well if data is missing completely at random
- Not all methods can accommodate missingness not at random

czechitas

Handling Missing Values

5. Selection of the best method

Use data to test which method works the best

id	time	lwage	married
1	1	5.56068	yes
1	2	5.72031	yes
1	3	5.99645	yes
1	4	5.99645	yes
1	5	6.06146	NA
1	6	6.17379	yes
1	7	NA	NA
2	1	6.16331	yes
2	2	NA	NA
2	3	6.26340	NA
2	4	NA	yes
2	5	NA	yes
2	6	NA	yes
2	7	6.81564	yes
3	1	5.65249	yes
3	2	6.43615	yes
3	3	6.54822	NA
3	4	6.60259	yes
3	5	6.69580	NA
3	6	6.77878	no
3	7	6.86066	no

1

Find set of data without missing values within your dataset

id	time	lwage
30	2	6.29895
30	3	6.33150
30	4	6.36303
30	5	6.48158
30	6	6.57786
30	7	6.64379
31	1	6.90575
31	2	6.90575
31	3	7.31322
31	4	7.24423
31	5	7.48437
31	6	7.54961
31	7	7.61332
32	1	6.90575
32	2	6.90575
32	3	7.46737
32	4	7.40853
32	5	7.60589
32	6	8.04879
32	7	7.74066

2

Artificially create missing values within your dataset

id	time	lwage	missing_lwage
30	2	6.29895	6.29895
30	3	6.33150	NA
30	4	6.36303	NA
30	5	6.48158	NA
30	6	6.57786	6.57786
30	7	6.64379	6.64379
31	1	6.90575	NA
31	2	6.90575	NA
31	3	7.31322	NA
31	4	7.24423	NA
31	5	7.48437	7.48437
31	6	7.54961	NA
31	7	7.61332	7.61332
32	1	6.90575	NA
32	2	6.90575	NA
32	3	7.46737	7.46737
32	4	7.40853	7.40853
32	5	7.60589	7.60589
32	6	8.04879	8.04879
32	7	7.74066	7.74066

Use data to test which method works the best

3

Use any method you want to impute those missing values

id	time	lwage	missing_lwage	mean_imputed	median_imputed
30	1	6.19044	6.19044	6.190440	6.190440
30	2	6.29895	6.29895	6.298950	6.298950
30	3	6.33150	NA	6.429260	6.438405
30	4	6.36303	NA	6.429260	6.438405
30	5	6.48158	NA	6.429260	6.438405
30	6	6.57786	6.57786	6.577860	6.577860
30	7	6.64379	6.64379	6.643790	6.643790
31	1	6.90575	NA	7.548845	7.548845
31	2	6.90575	NA	7.548845	7.548845
31	3	7.31322	NA	7.548845	7.548845
31	4	7.24423	NA	7.548845	7.548845
31	5	7.48437	7.48437	7.484370	7.484370
31	6	7.54961	NA	7.548845	7.548845
31	7	7.61332	7.61332	7.613320	7.613320
32	1	6.90575	NA	7.654248	7.605890
32	2	6.90575	NA	7.654248	7.605890
32	3	7.46737	7.46737	7.467370	7.467370
32	4	7.40853	7.40853	7.408530	7.408530
32	5	7.60589	7.60589	7.605890	7.605890
32	6	8.04879	8.04879	8.048790	8.048790
32	7	7.74066	7.74066	7.740660	7.740660

4

Compare real vs imputed values MAPE (mean absolute percentage error), preserved correlation

5

Choose the best performing method

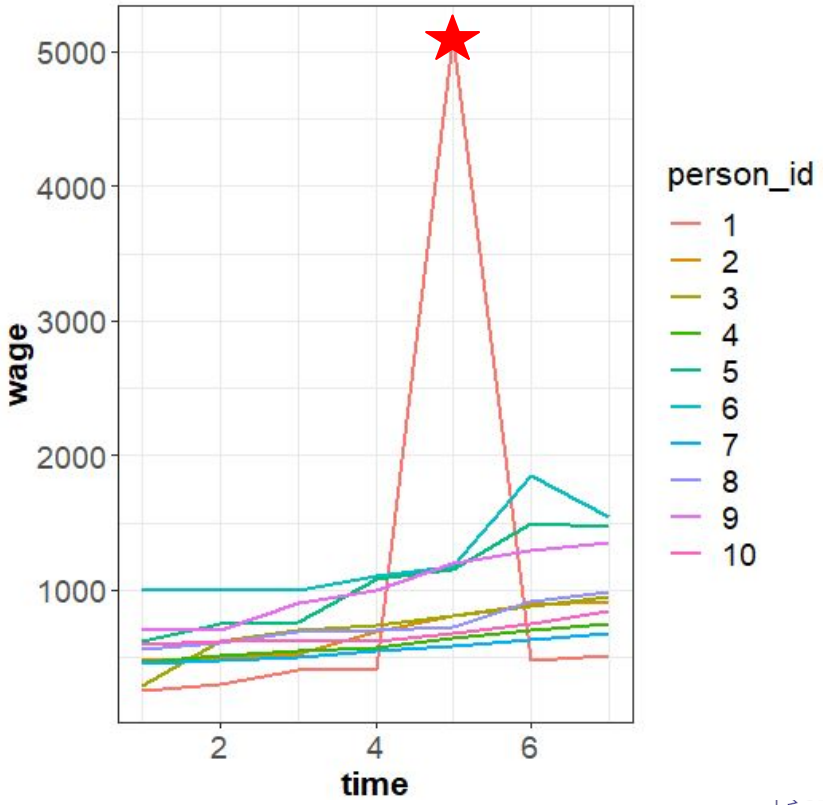
czechitas

Handling Missing Values

5. Selection of the best method

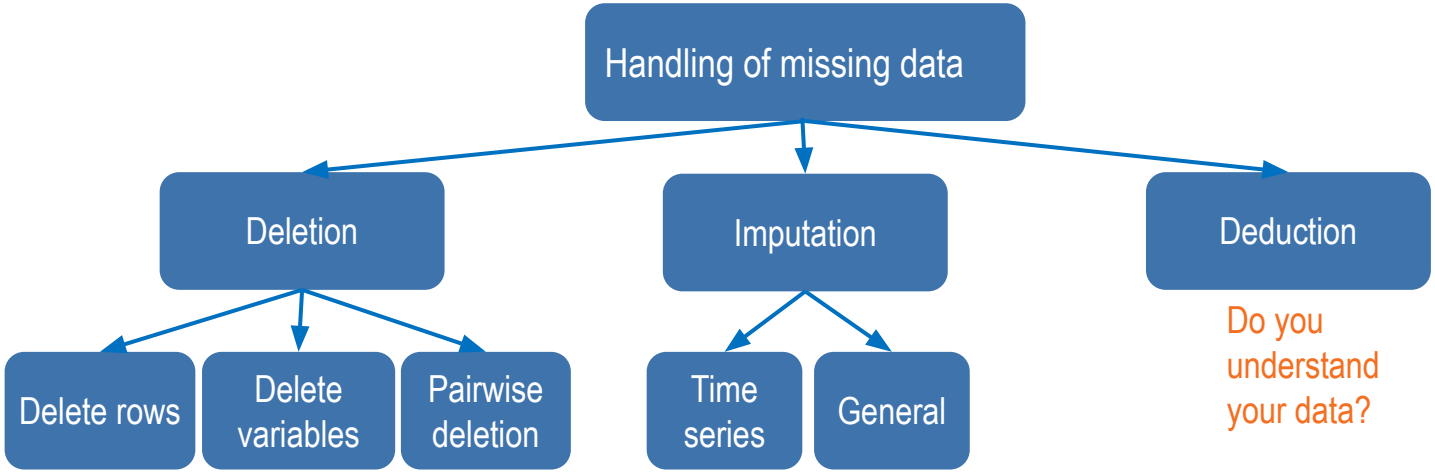
Missing values challenge is a second step in outlier detection problem

1. Use algorithms to find an outlier
2. Remove or impute with new value



echitas

Summary



Tips

- Understand why data is missing
- Understand data before any action
- Explore patterns of missingness
- Use data to validate the best method
- Be cautious of impacts of missing data to your analysis

czechitas