# **Data Tidying and Cleaning**

# What's the most efficient way to perform data transformations in pandas: Step-by-Step guide

# **Step 1: Import the necessary libraries**

# The first step is to import the necessary libraries. In this case, you will need to import pandas. You can do this by typing the following command in your Python environment:

# import pandas as pd

# **Step 2: Load your data**

# The next step is to load your data into a pandas DataFrame. You can do this using the pandas read\_csv() function if your data is in a CSV file. For example:

# df = pd.read\_csv('your\_file.csv')

# **Step 3: Inspect your data**

# Before you start transforming your data, it's a good idea to inspect it first to understand its structure and content. You can do this using the head() function, which returns the first n rows of your DataFrame. For example:

# df.head()

# **Step 4: Perform data transformations**

# There are many ways to perform data transformations in pandas, but the most efficient way is usually to use vectorized operations. These are operations that are performed on entire arrays of data at once, rather than on individual elements. This can significantly speed up your data processing.

# Here are a few examples of common data transformations and how to perform them in a vectorized way:

# Adding a new column based on existing columns:

# df['new\_column'] = df['column1'] + df['column2']

# Applying a function to a column:

# df['column'] = df['column'].apply(lambda x: x\*\*2)

# Replacing values in a column:

# df['column'] = df['column'].replace({'old\_value': 'new\_value'})

# **Step 5: Check your transformations**

# After performing your transformations, it's a good idea to check that they have been applied correctly. You can do this by inspecting your DataFrame again using the head() function.

# **Step 6: Save your transformed data**

# Finally, once you're happy with your transformations, you can save your transformed data back to a CSV file using the to\_csv() function. For example:

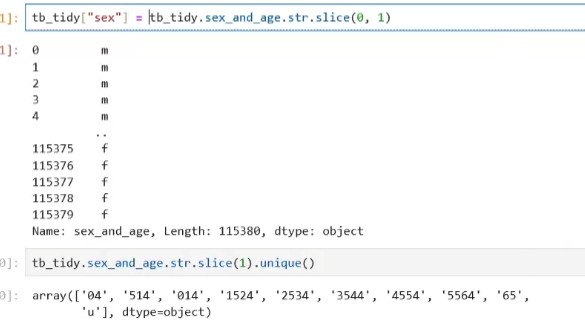
# df.to\_csv('your\_transformed\_file.csv', index=False)

# Remember, the key to efficient data transformations in pandas is to use vectorized operations wherever possible. This will ensure that your transformations are performed as quickly and efficiently as possible.

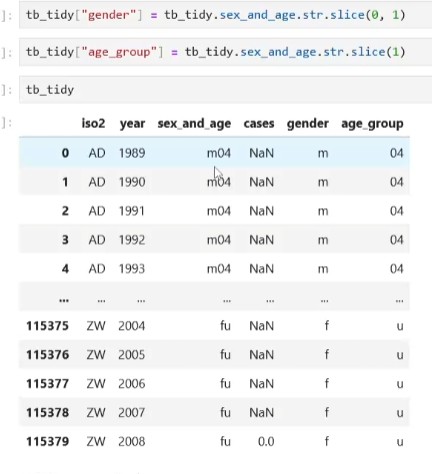
**Melt**

****

**Slice**

****

**Slices**

****

**Drop column**

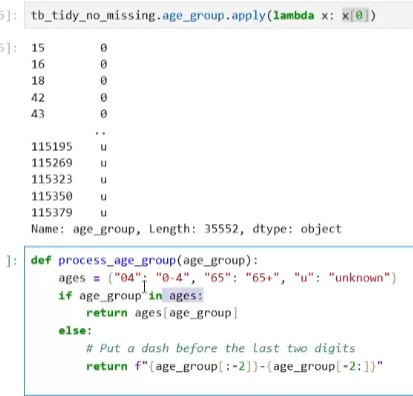
****

**Mean or whatever by condition**

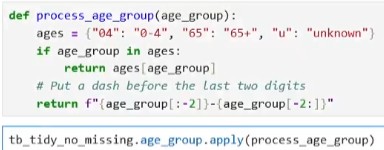
****

**Values count**

**Apply and function**

****

**Apply function**

****

**Split and expand**

**To category**

****

**Rearrange columns**

**Sort by iso2 and then by year**

****

****

**Reset index – rearrange**

**Save to another csv**

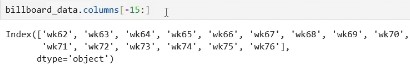
**Loc - save new value “d31” on row 2, 23 column**

**Melt, slice, dropna**

**Pivot**

****

**Which are in week 50? - notna**

**Last 15 columns**

**Melt more examples**

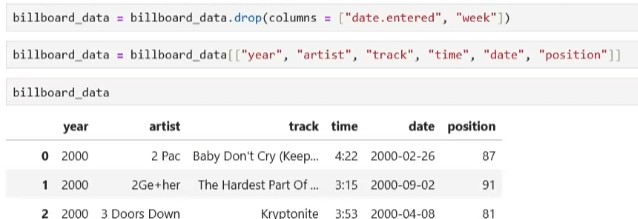
**Slice and astype**

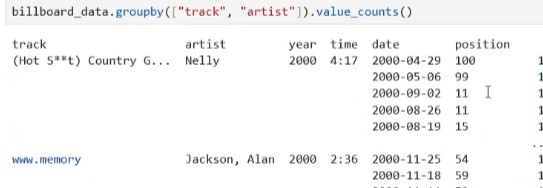
**Data datetime**

**Add week of the year**

****

**Drop and rearange columns**

****

**Group by**

**Condition if**

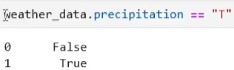
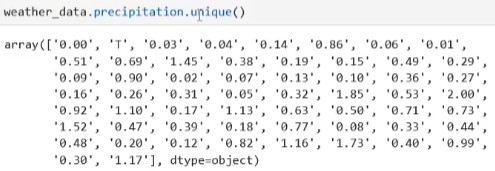
****

**Read csv from link – processing, basic steps**

**Convert column names**

**To datetime**

**Dtypes**

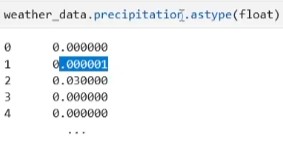
**Object column analysis**

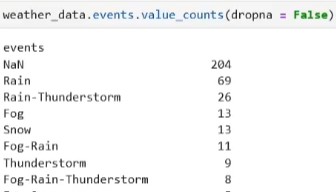
**How many are with ‘T’**

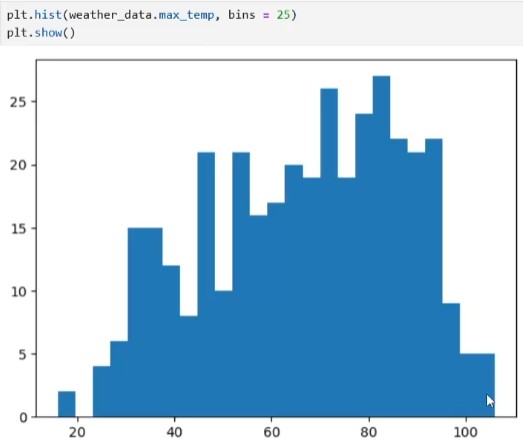
**Replace column value ‘T’ with a very small number**

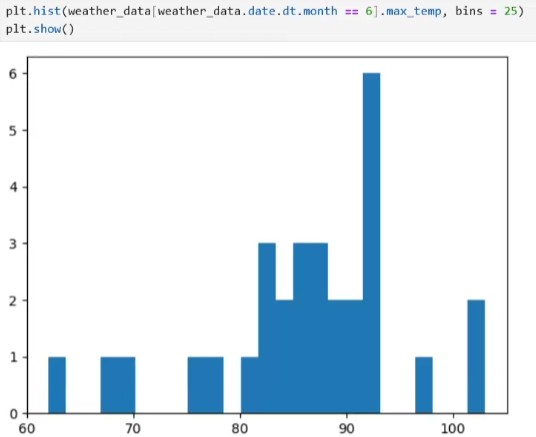
****

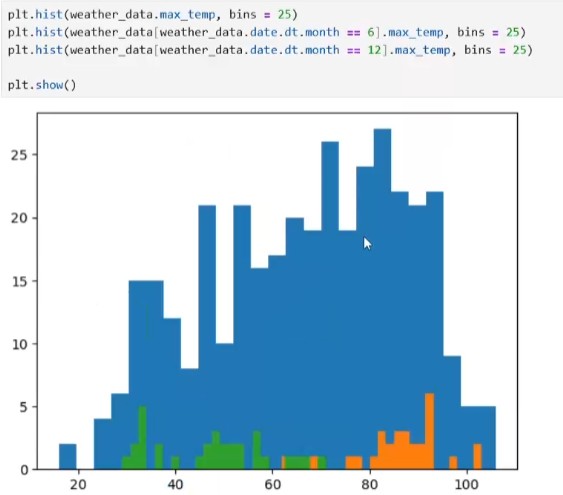
**Astype float**

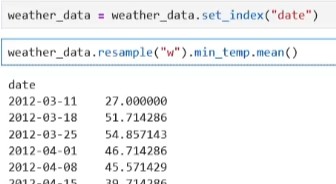
****

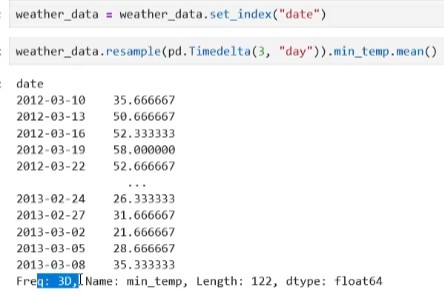
**Count events including NAN**

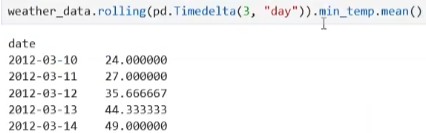
**Hist**

**Hist max temperatures for june – 6**

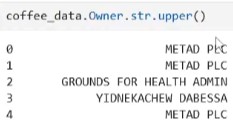
**Plot all together**

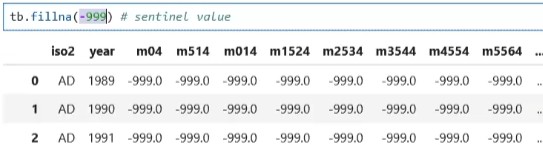
**Average temperature for every week**

**Min temp for every 3 days – by 3 days – 10.03/13.03/16.03**

**Or rolling – there is overlap – every day 10.03/11.03/12.03**

**Str transformations**

****

**We could replace the missing data with a dummy value**

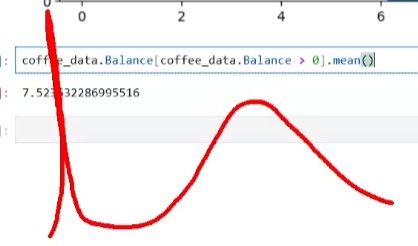
**Median Conclusion:**

* **Use the mean when your data is normally distributed without outliers.**
* **Use the median when your data is skewed or contains outliers.**

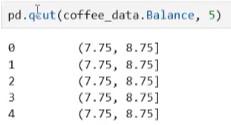
****

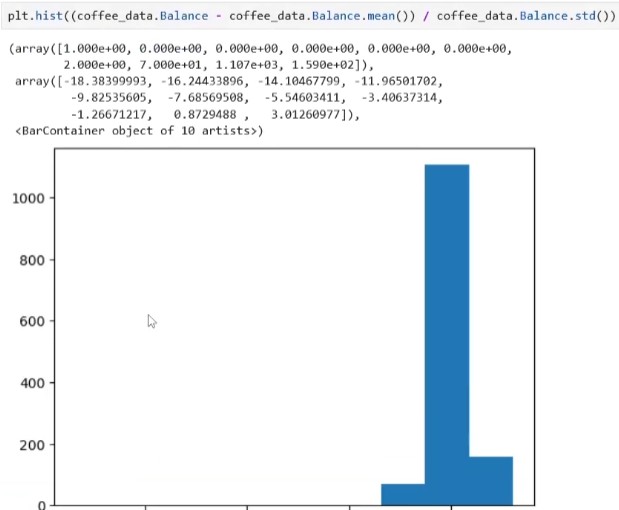
**Mean of only positive example - when we have outliers DATASET TRANSFORMATIONS**

**<https://scikit-learn.org/stable/modules/preprocessing.html#non-linear-transformation>**

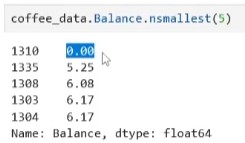
****

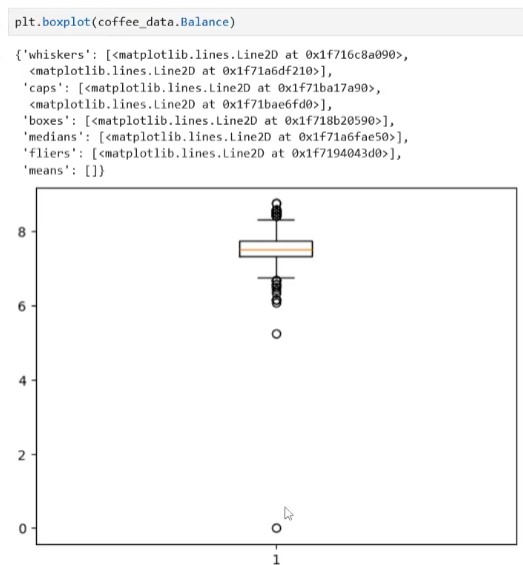
**QCUT**

****

****

**Find smallest 5**

****

**Find outliers**

****

**Median Conclusion:**

* Use the mean when your data is normally distributed without outliers.
* Use the median when your data is skewed or contains outliers.

**Mean of only positive example - when we have outliers DATASET TRANSFORMATIONS**

**<https://scikit-learn.org/stable/modules/preprocessing.html#non-linear-transformation>**

Pandas api request with authentication ChatGPT

To make an API request with authentication and load the data into a pandas DataFrame, you can use Python's requests library. Below is a step-by-step guide for handling API requests that require authentication, such as using an API key or basic authentication.

1. Install Required Libraries

If you haven't installed pandas or requests, you can install them using pip: bash

pip install pandas requests

1. Import Libraries

Start by importing the necessary libraries:

python Копиране на код

import pandas as pd import requests

1. Set Up the Authentication

Depending on the type of authentication the API requires, you can set it up as follows: API Key in Headers

If the API uses an API key in the headers: python

Копиране на код

url = "[https://api.example.com/data"](https://api.example.com/data%22) headers = {

"Authorization": "Bearer YOUR\_API\_KEY", "Content-Type": "application/json"

}

response = requests.get(url, headers=headers) Basic Authentication

If the API uses basic authentication (username and password): python

Копиране на код

url = "[https://api.example.com/data"](https://api.example.com/data%22)

auth = ("your\_username", "your\_password")

response = requests.get(url, auth=auth)

1. Convert the JSON Response to a DataFrame

Assuming the API returns data in JSON format, convert the response to a pandas DataFrame:

python Копиране на код

data = response.json() df = pd.DataFrame(data)

# **Display the DataFrame**

print(df.head()) Example Workflow

Here’s an example that combines all the steps: python

Копиране на код import pandas as pd import requests

# **Example API URL**

url = "[https://api.example.com/data"](https://api.example.com/data%22)

# **Headers for API Key Authentication**

headers = {

"Authorization": "Bearer YOUR\_API\_KEY", "Content-Type": "application/json"

}

# **Make the GET request with headers**

response = requests.get(url, headers=headers)

# **Check if the request was successful**

if response.status\_code == 200:

# Convert the JSON response to a DataFrame data = response.json()

df = pd.DataFrame(data)

# Display the DataFrame print(df.head())

else:

print(f"Failed to retrieve data: {response.status\_code}") Notes:

* Handling Errors: It's good practice to check the response status code before attempting to parse the JSON.
* Pagination: If the API paginates the results, you may need to loop through pages and combine the data.
* Rate Limits: Be mindful of API rate limits, and handle them appropriately (e.g., by adding delays or checking headers for rate limit information).

This method enables you to securely access API data using authentication and load it into a pandas DataFrame for further analysis.

Material Summary: Data Acquisition

1. The Scientific Method

1.1The Scientific Method Steps

Ask a question

Do some research

Form a hypothesis

Test the hypothesis with an experiment

Experiment works Analyze the data

Experiment doesn't work Fix experiment

Results align with hypothesis OK

Results don't align with hypothesis new question, new hypothesis

Communicate the results

2.Getting Data

2.1The Pandas Library

Provides a way to read and work with data

Table (DataFrame)

May have many dimensions

We usually call this a "dataset"

List (Series)

One-dimensional

Usually represents a column of a table

Usage

General requirements

Rows and columns are indexed, columns may have names

Each column has a fixed data type

Python will try to infer the best type according to the data

2.2Data Sources

In order to work with the data, we need to represent it in tabular form

Sometimes our data is tabular – we just need to read it

In other cases, we need to create our tables

Unstructured data: data that doesn't have a model

There is some structure, it's just not very clear

Examples: Images, plain text, audio, web pages

Most common sources

Tables in a text format such as .csv

Spreadsheets (such as Excel or Google Sheets)

Web services

Databases

2.3Reading a Local File

Let's read the file accidents.csv

Copy the file to a data folder

Not required, just makes working with many data files easier

Inspect the file (use a text editor or Excel) just to see what it contains

read\_csv() docs

You'll see that all read\_\*() functions have a lot of optional arguments

They make working with different formats easy, e.g.

Instead of True and False, the table contains "Yes" and "No"

The actual table starts at line 30 of the file

There are blank / comment lines which should be skipped

There are no column names in the file

2.4Exploring the Dataset

In Python, we can print the variable

Even better, in Jupyter, a cell outputs its last returned value

This will create a nicer output

We can see that

Rows have numerical indices starting at 0 by default

Columns have names taken from the first line in the .csv file

Column names:

Index values:

Dimensions:

Format: (rows, columns)

2.5 Reading Data from Other Files

The process is very similar

Other text-based formats

pd.read\_table() is the most general function

All others (read\_csv(), read\_fwf(), etc.) just apply some settings

If we come across a file, we can apply our own settings

The point is to match the format in the best possible way

Example: AutoMPG dataset

Excel

Read the green\_tripdata\_2015-09.xls file using pd.read\_excel()

Explore the file dimensions

2.6 Reading Data from Web Services

Web services work over the HTTP protocol and provide data in several formats

Most commonly used: JSON and XML

Some APIs to try

Example: OpenLibrary API

We want information about books with ISBNs

Example: these 4 books

We can put the URL directly, pandas will perform a GET request

Function: pd.read\_json()

We can provide the parameter orient = "index" to arrange the dataset better

Books should be placed by rows, their properties – by columns

More details on this – next time

More complex queries require more pre-processing

2.7 Reading Data from SQL

Relational databases store data in tables

Very similar to the datasets we use

First, install a library to connect to databases

From the command line:

Then, import the library and connect to the database

Note: This is going to vary depending on your server settings

Perform a query

2.8 Web Scraping

Another method for getting data

Sometimes combined with crawling

Traversing a Web page structure recursively

Basic procedure

Read a Web page as HTML

Use the HTML to obtain the data

A webpage is unstructured

We need to create and maintain the structure

We usually need more libraries to do that

Examples

Get all job listings from a website

Get user contact details from a Web page

3.Using Multiple Sources

3.1 Data Guidelines

Some queries will not be simple

E.g., scraping, dealing with "freeform" text, audio data, networks

We need to create a tabular structure from the raw data

How? We'll discuss this later in the course

After we read the data, we have to ensure it's been read

without errors

A very simple first check: check the dimensions (dataframe.shape)

and show the first few rows (dataframe.head())

We may need to rename columns

We may need to perform different manipulations to ensure

the data is in a proper state

We'll do this in the next lectures

3.2 Merging Many Data Sources

Automate the process as much as possible

From reading the raw data to getting the processed dataset

If the dataset changes or updates, you'll just re-run your code

Document the process

Create as few datasets as possible

I.e., merge many sources into one table if you can

We'll talk more about combining relations next time

Ensure the different sources are compatible and consistent

If they aren't, process the raw data

Most common example: Mismatched IDs

Make sure all column types are correct

Check: dataframe.dtypes

Example: str type for a numeric column

# Material Summary: Data Tidying and Cleaning

## Data Tidying

**1.1 Tidy Data**

* Most important rules when creating (or using) datasets
  + Columns – attributes (features, variables)
  + Rows – observations
  + Cells – values (one observation of one feature)
  + All other data is called **messy data**
* Empirical rule for testing whether a dataset is tidy
  + Adding one more observation should create one new row
    - No new columns
    - No multiple rows
    - No partial rows, no changes to other rows
* pandas allows us to read, tidy up and transform datasets
  + Data modelling requires a tidy and clean dataset in order to work well (garbage in – garbage out)

**1.2 Messy Data**

* A group of black arrows

  Description automatically generatedWhat we want
* A screenshot of a computer

  Description automatically generatedWhat we get instead

**1.3 Tidy and Messy Data**

* A very good *[paper](http://vita.had.co.nz/papers/tidy-data.pdf) o*n tidy data
* Example: several datasets
  + Same information, different ease of use

**A screenshot of a computer

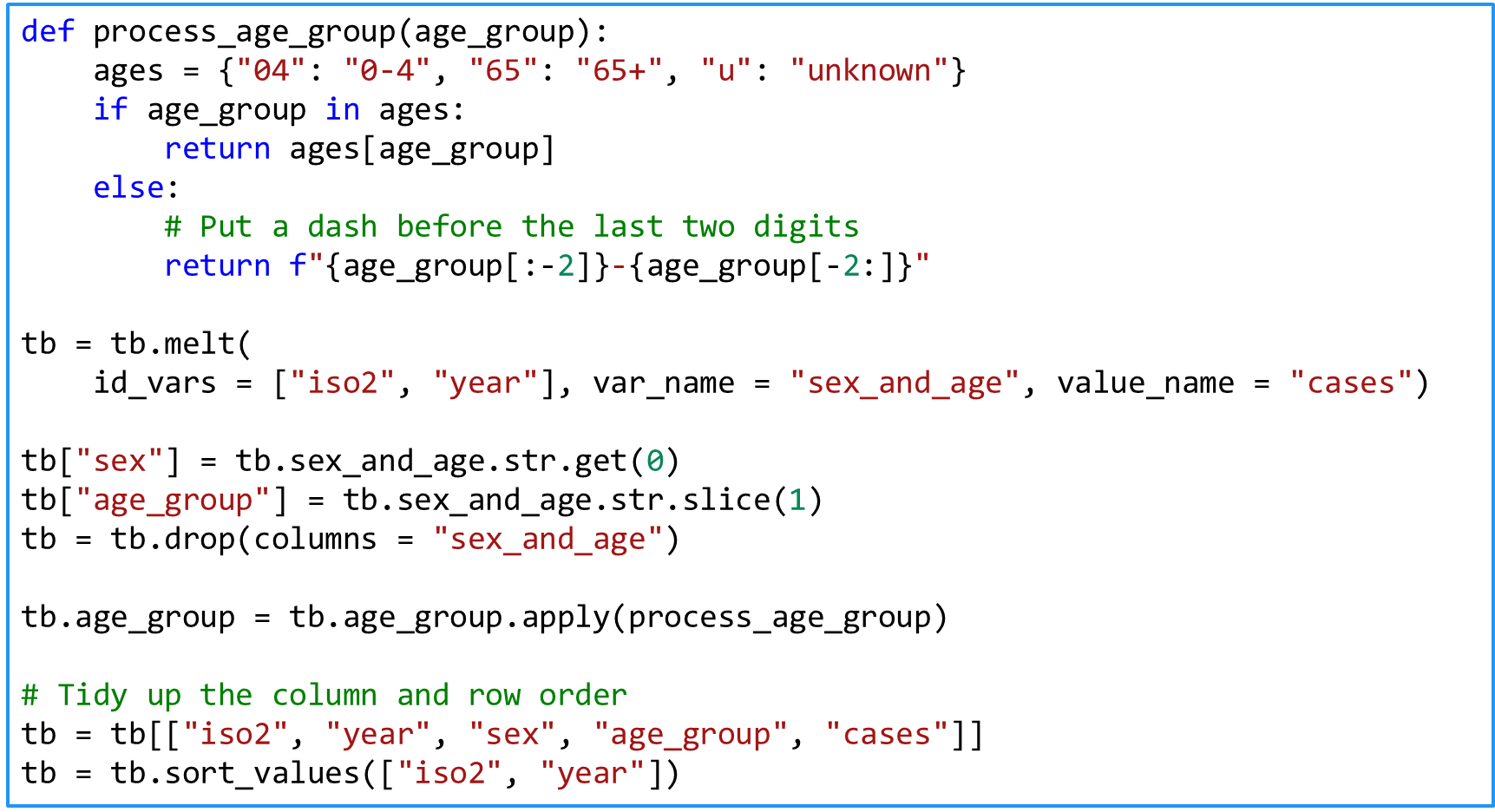
Description automatically generated1.4 Messy to Tidy Data**

1. The table header contains values
   * Identify the variables and distribute (unpivot) the values

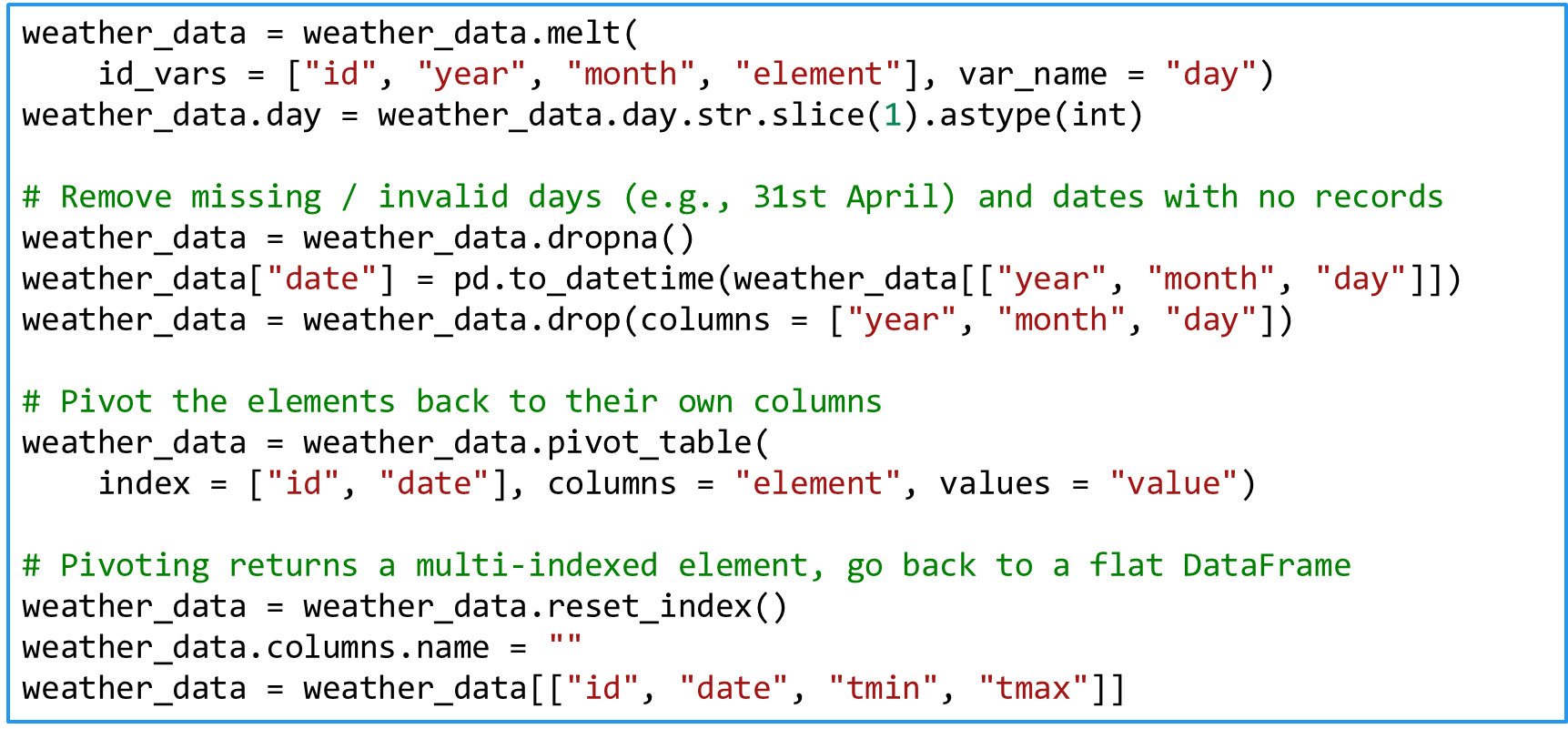
* Read the pew.csv dataset
  + Distribution of income by religion
* Show the first 5 values (use the head() function)
  + Also see the number of variables and observations (shape)
  + This will also ensure that you've read the dataset correctly
  + **Variables:** religion, income, frequency
* A white rectangular sign with green text

  Description automatically generatedTransform the dataset to make it tidy (*[docs](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.melt.html)*)

1. Multiple variables stored in one column
   * Identify and split the variables into separate columns

* Read the tb.csv dataset
  + Tuberculosis cases
  + m**04**, m**514**, m**1524**, etc. contain two variables (gender and age)
    - male, 0-4 years old; male, 5-14 years old, etc.
    - There's also a problem with missing values (NaN)
* Tidying process
  + First, melt all columns (they are values and should not be)
  + Next, split the column names and extract the gender and age information
  + Add the new info to the dataset
  + Remove all missing values

1. Variables are stored in both rows and columns
   * Identify and split the variables

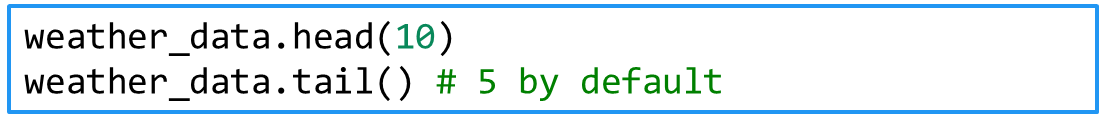
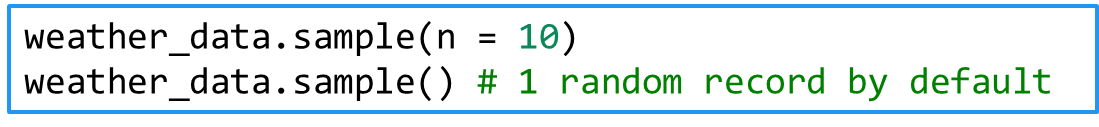
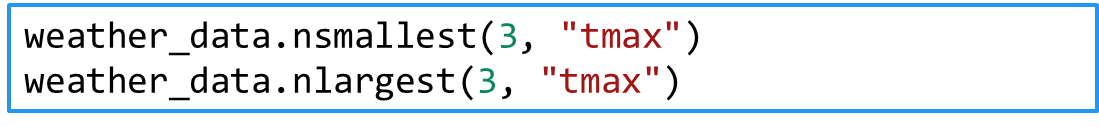
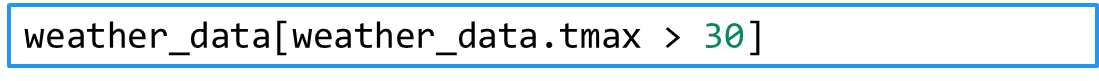
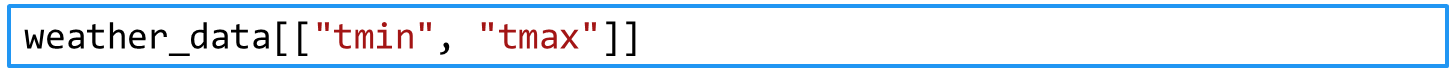
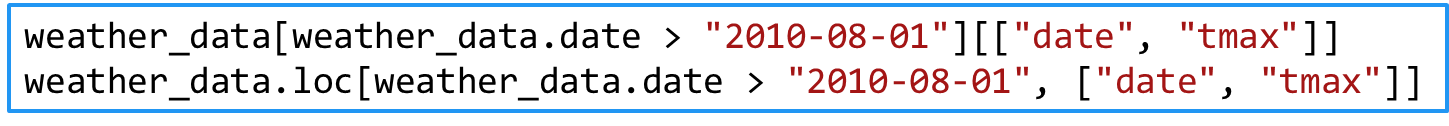
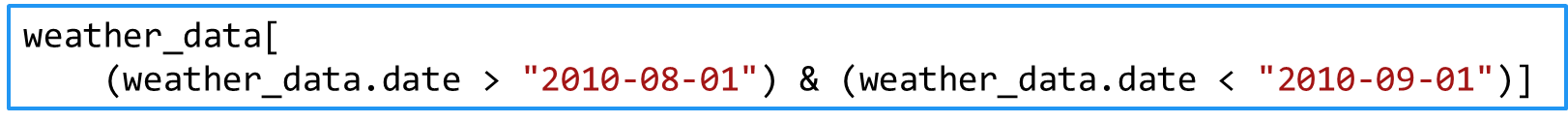
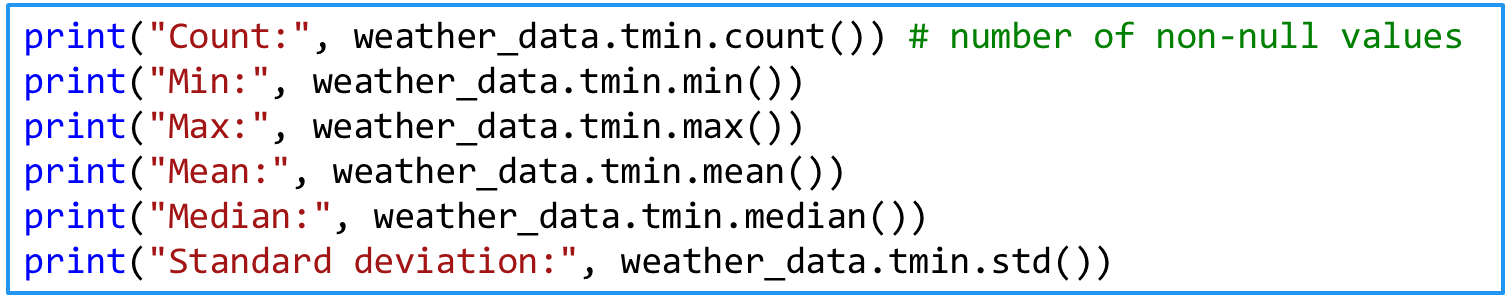
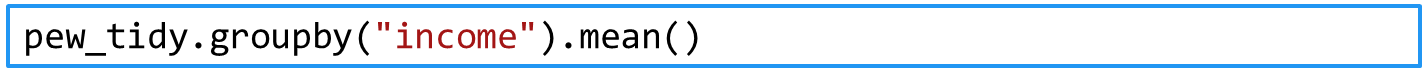
* Read the weather.csv dataset
  + Daily weather records in Mexico in 2010
  + d1, d2, etc. are the days of a month; tmin and tmax should   
    be columns
    - Make a new column with the date: [date, tmin, tmax]
* Tidying process
  + Melt all days
  + Create days based on date, month and year
  + Pivot the tmin and tmax columns

1. One type in multiple tables
   * Merge the tables into one
     + Read all tables, add the new columns
     + Often the filename should be in its own column (if it's important)
     + Melt and tidy if necessary
2. Multiple types in one table
   * Split into more tables
     + If necessary, introduce relations (similar to a relational database)

* Each table should be responsible for one type of measurement
* \* Read the billboard.csv dataset and apply those transformations

## Operations on Datasets

**2.1 Subsetting Rows**

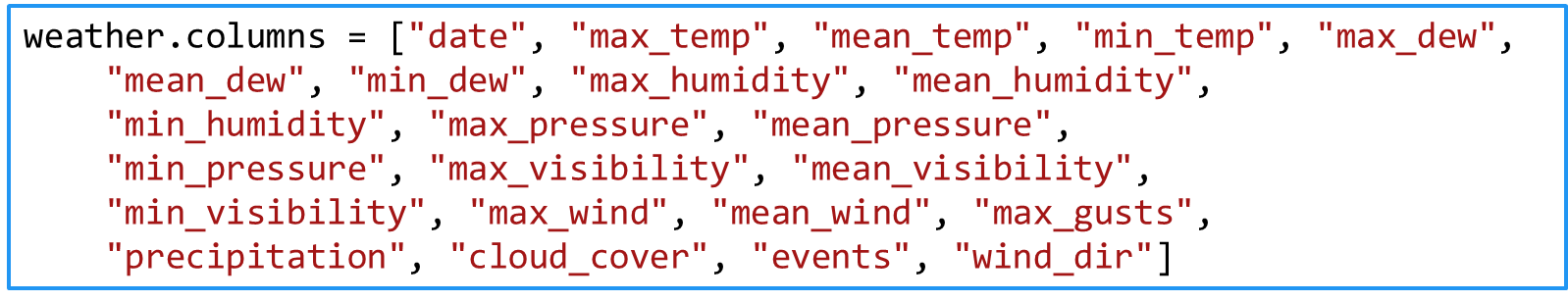
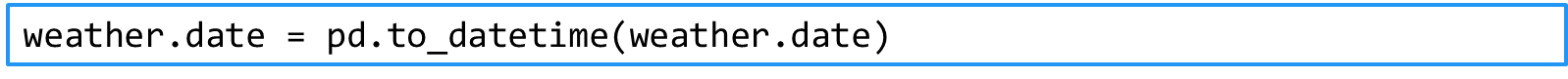
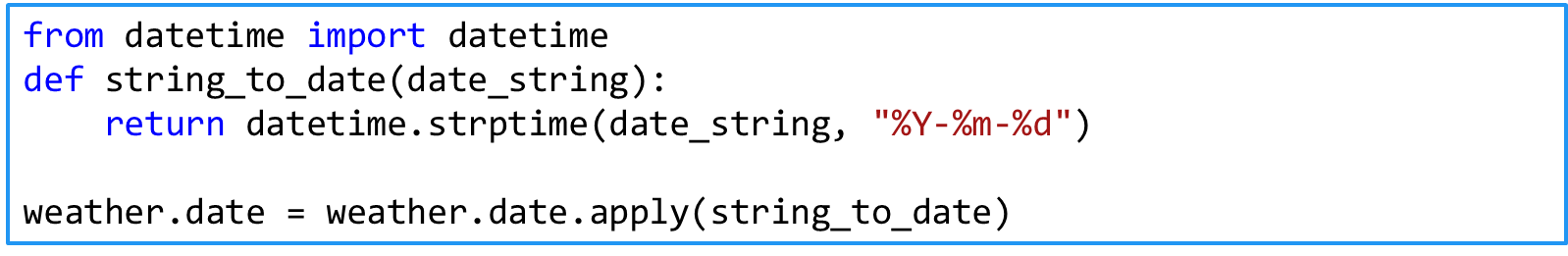
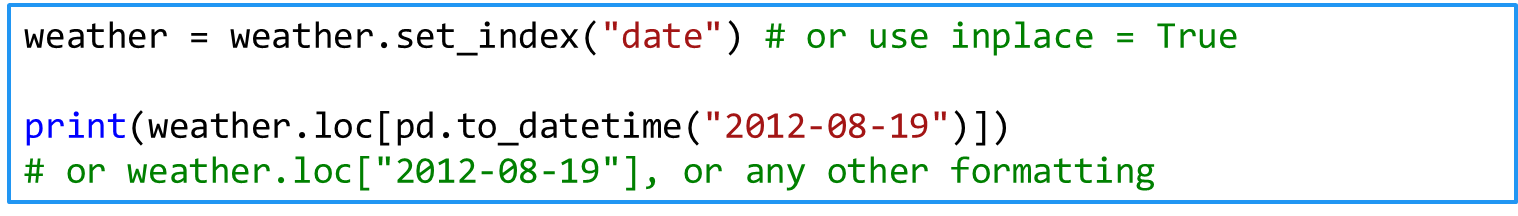
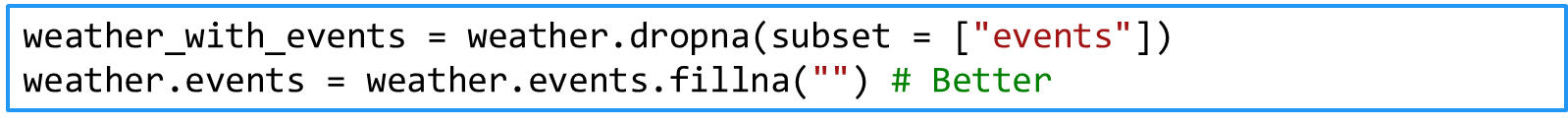
* Selecting only some rows (aka **selection**)
* First / last n records (observations)
* Random n records
* Smallest / largest n records in each column
* Subsetting by a Boolean expression (predicate)
  + Returns only rows where the expression returns True
* Selecting only some columns (aka **projection**)
* Single column (returns a Series object)
* More than one column (returns a DataFrame object)
* Combining filters
* A note on Boolean expressions
  + and, or, not are &, |, ~
  + **Always** put parentheses around the individual expressions
* These methods work by columns
  + If multiple columns are passed, they are applied to each   
    column individually
* Grouping
  + Splits the data into several groups based on the values of a column
  + We have to apply a method after grouping
    - Or iterate over the groups (using a for-loop)
  + Example: Average number of people for each income group

## Cleaning Data

**3.1 Cleaning Data**

* No common way of doing this
* We need to rely on intuition and some common patterns
  + Tidy up the dataset
    - You must know the dataset documentation first
  + Treat nulls / NaNs: either remove them or replace them
    - Replacing values might be **dangerous**
    - If done properly, it will affect the data in a positive way
  + Identify and fix errors (also **dangerous**)
  + Melt and pivot datasets
  + Merge (join) and separate datasets
  + Subset variables and / or observations
  + Summarize and group variables
  + *[Pandas Cheat Sheet](https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf)*

**3.2 Example: Weather Data**

* Since there's no common way of cleaning, we'll explore  
  and clean a dataset, showing steps and examples as we go
* *[Dataset](https://raw.githubusercontent.com/synesthesiam/blog/master/posts/data/weather_year.csv)* (weather data, courtesy of synesthesiam@github)
* Read the dataset (you don't need to download it)
  + See how many variables and observations are there
  + Display the first and last few rows to get a sense of the data
  + Check the data types (to see if something's wrong with the reading)
    - E.g., numbers recognized as strings
  + See a subset of the columns
  + Summarize (describe) the dataset
* The column names don't look good
  + Make them "pythonic" (lowercase\_with\_underscores)
    - This will make selecting them easier (weather.mean\_temp)
* What are the ranges of data?
  + E. g. temperature, pressure, humidity
  + Use the min() and max() methods
* \* Try to explore the data a bit
  + Plot a few histograms and / or boxplots to see the distributions
* Convert the dates to a datetime object
  + To make performing time-dependent analysis easier
  + If needed, use apply() to perform a function on every row
    - It's even better to use dates as indices (when we need to subset date ranges or perform other time-dependent tasks)
  + Also see why precipitation is not a float and edit it
* Remove or replace missing values
  + In this case, replacing is better because removing takes away an entire row
    - Try to see how variables interact – group the data
  + E.g., by cloud cover and events
  + Print the number of days when each combination of {cover, events} occurred
* Plot data
  + Next time
* If needed, perform transformations
  + Math operations: log, square root, addition, multiplication, etc.
    - Be careful as you'll get results in different dimensions
  + Normalizing scores (such as using Z-scores) is recommended in most cases
    - It's much better for ML algorithms to have data of similar scales
    - You can do that manually or use a library (such as *[sklearn.preprocessing](http://scikit-learn.org/stable/modules/preprocessing.html)*)
  + By convention, calculated columns are added to the dataset
* **Describe all operations as you're doing them**
  + Describe what you're doing and why
    - Useful to check your work later (or allow others to do that)
  + If needed, save the resulting dataset into a file
    - Supply your data transformation log with it
    - Provide a dataset description

**3.3 Outliers and Errors**

* **Outliers** – values which are far from their expected range
  + Or having a very low probability of happening (assuming a model)
* Many possible cases
  + Wrong data entry (e.g. an adult weighing 5kg might be 50kg or something else)
  + Wrong assumptions (the data is correct, our view isn't)
* What to do?
  + Inspect the data point
  + Try to figure out what happened
    - If needed, remove the row or try to replace the value
  + Try a transformation
  + A cartoon of a person drawing a graph

    Description automatically generatedIf possible, perform analysis with and without the outlier(s) and compare your results

**3.4 Transformations on Features**

* The quality of our results depends strongly on the features we use
  + "Garbage in – garbage out"
* **Dimensionality reduction**
  + Reducing the number of variables (features)
  + We can do this manually or use algorithms
  + **Feature selection**
    - Selecting only columns that are useful
  + **Feature extraction**
    - Transforming non-structured to structured data
      * Examples: images, audio, text
    - Getting meaningful features
* **Feature engineering**
  + Using our knowledge of the data to create meaningful features
    - Involves a lot of brainstorming and testing

**3.5 Next Steps (Optional)**

* Have a look at scikit-learn's ["](http://scikit-learn.org/stable/data_transforms.html)*[Dataset Transformations](http://scikit-learn.org/stable/data_transforms.html)*["](http://scikit-learn.org/stable/data_transforms.html) module
  + It describes the most common operations
    - Data cleaning
    - Dimensionality reduction
    - Feature extraction
* There are many algorithms based on
  + Data types (e.g., text or numerical data, labelled vs. not labelled)
  + Model types (how we want to present our data, e.g., linear model)
  + Algorithm types (e.g., finding similar news articles, recommending movies to users, classifying, etc.)
* No "hard and fast rule", use your intuition
  + Knowing more tools / models / algorithms -> better performance