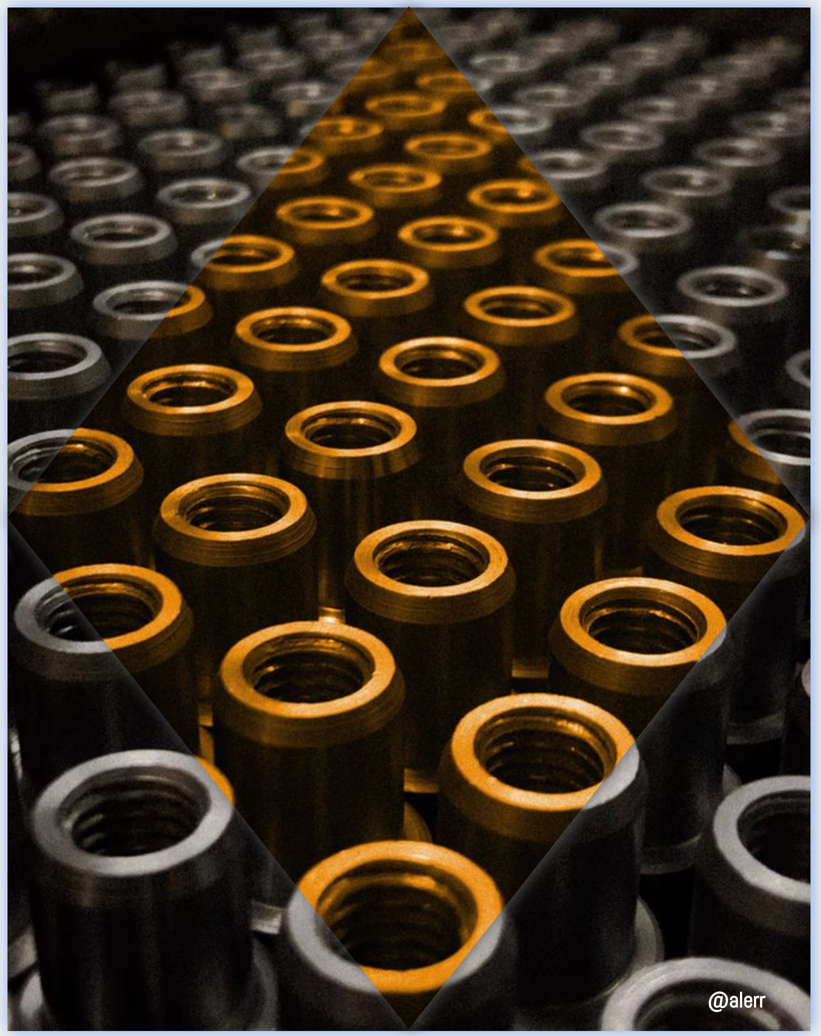


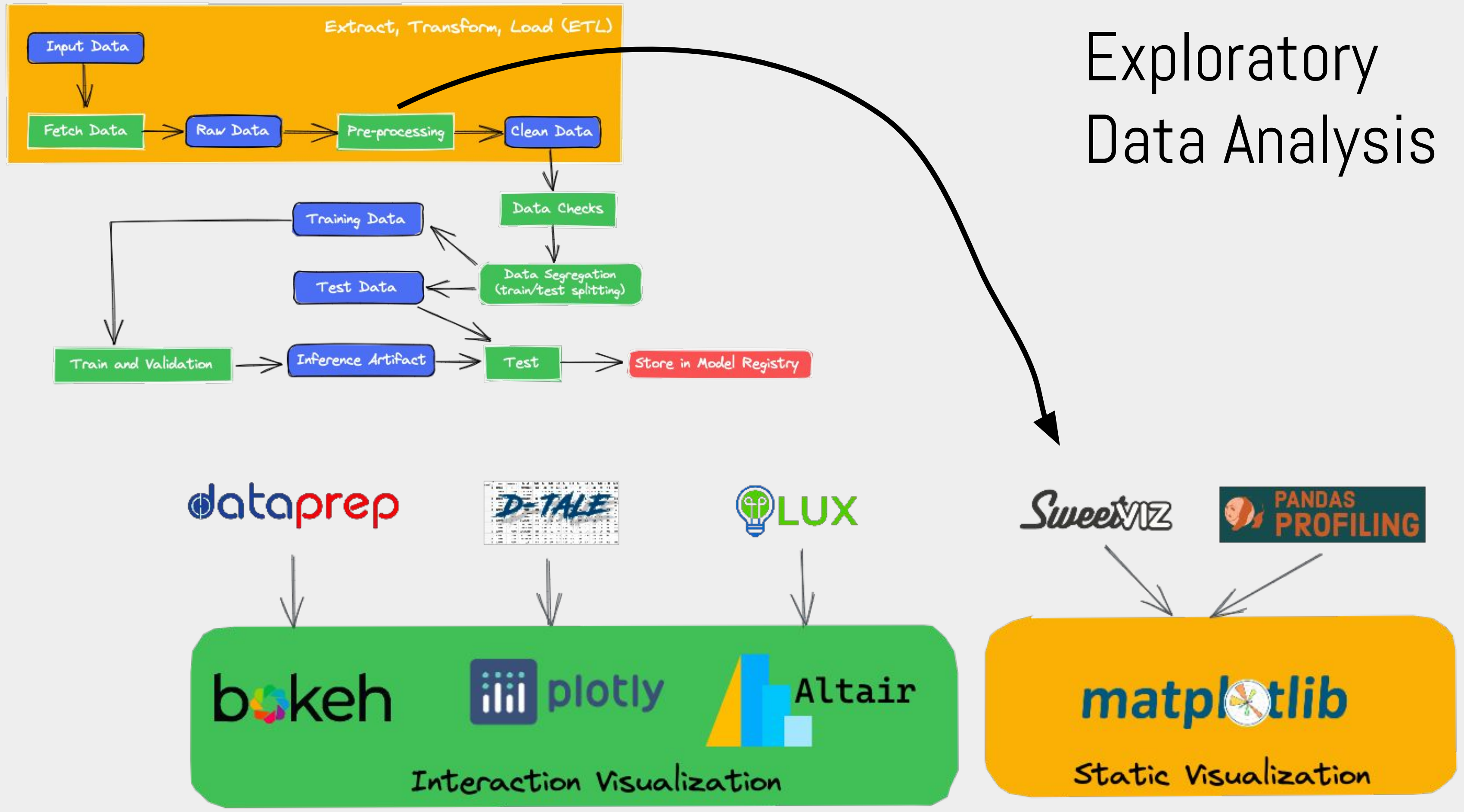
# Building a Reproducible Model Workflow Cont.

## ETL, Data Checks and Data Segregation

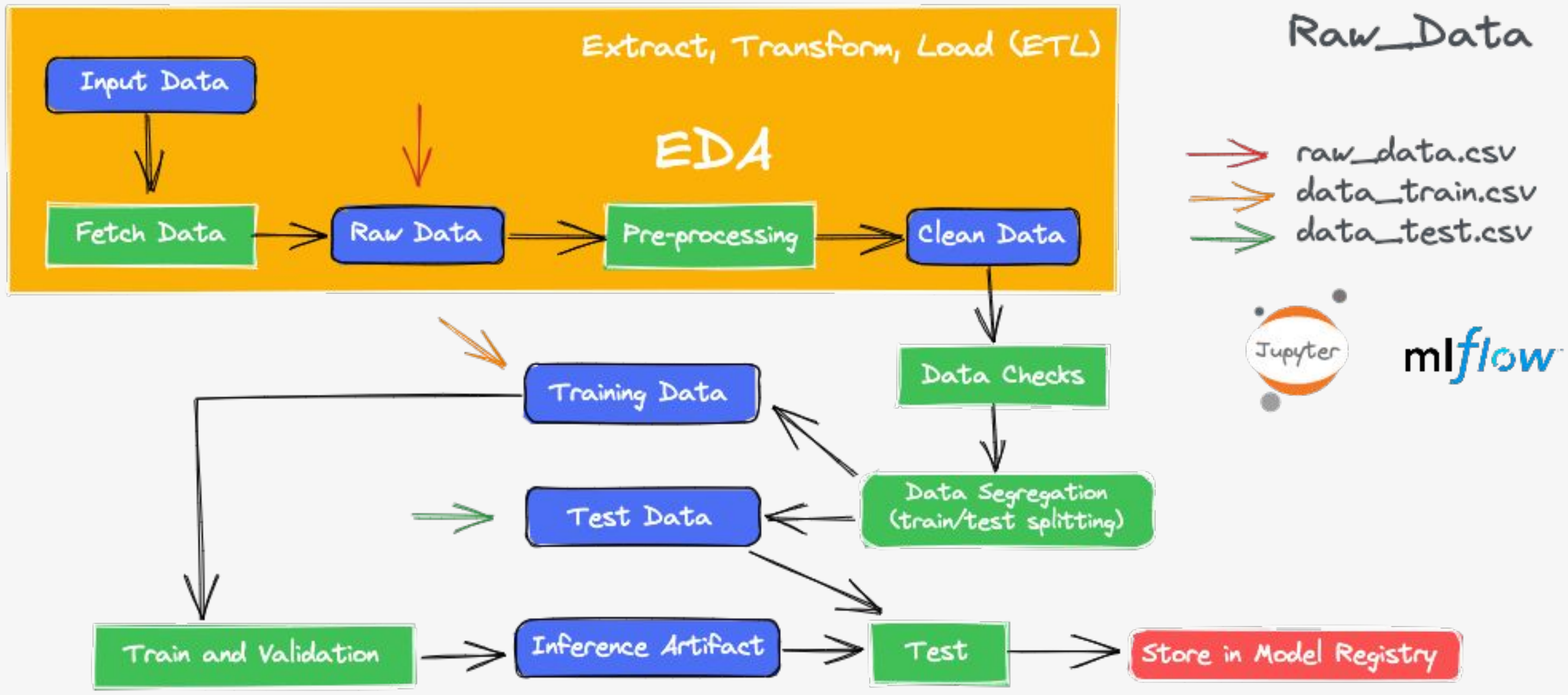
## Table of Contents

"We want to avoid that bad data gets into your ML pipeline."

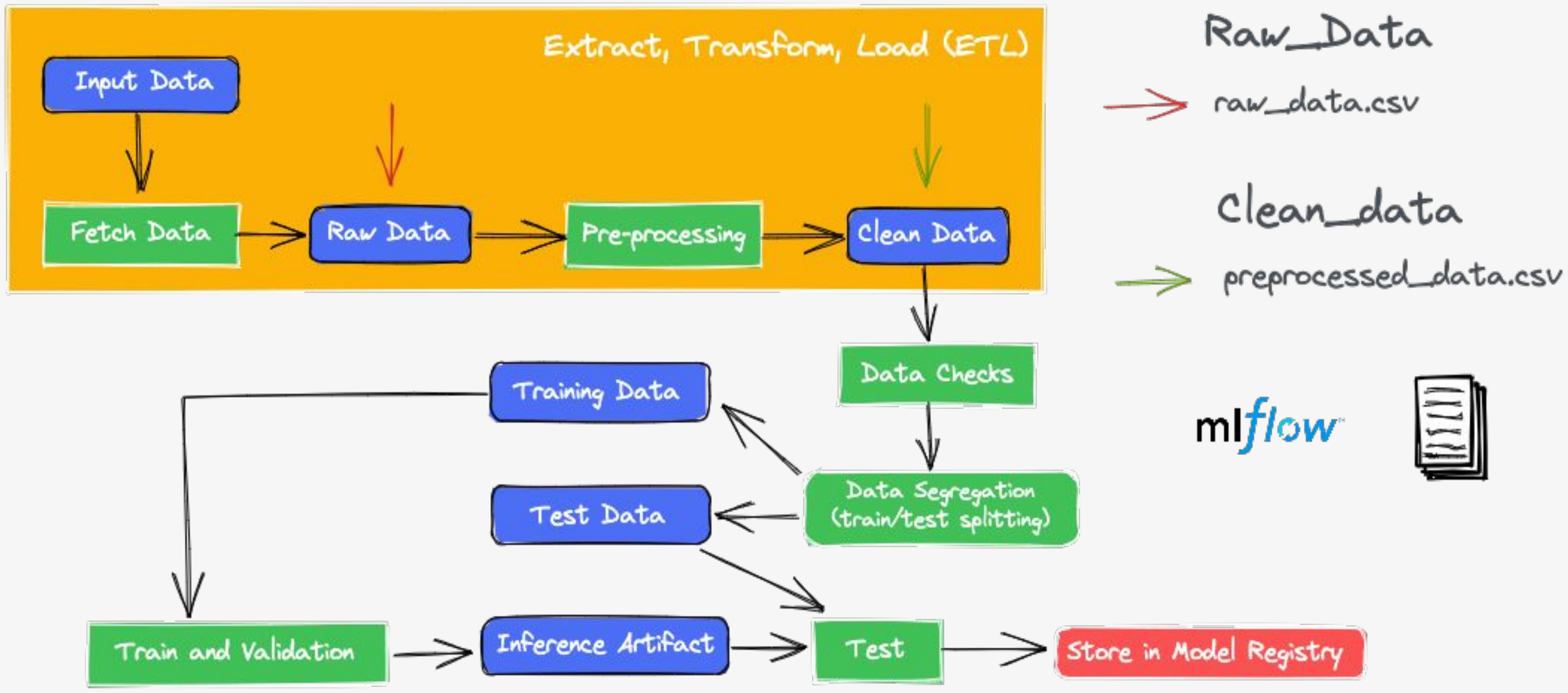
## Exploratory Data Analysis



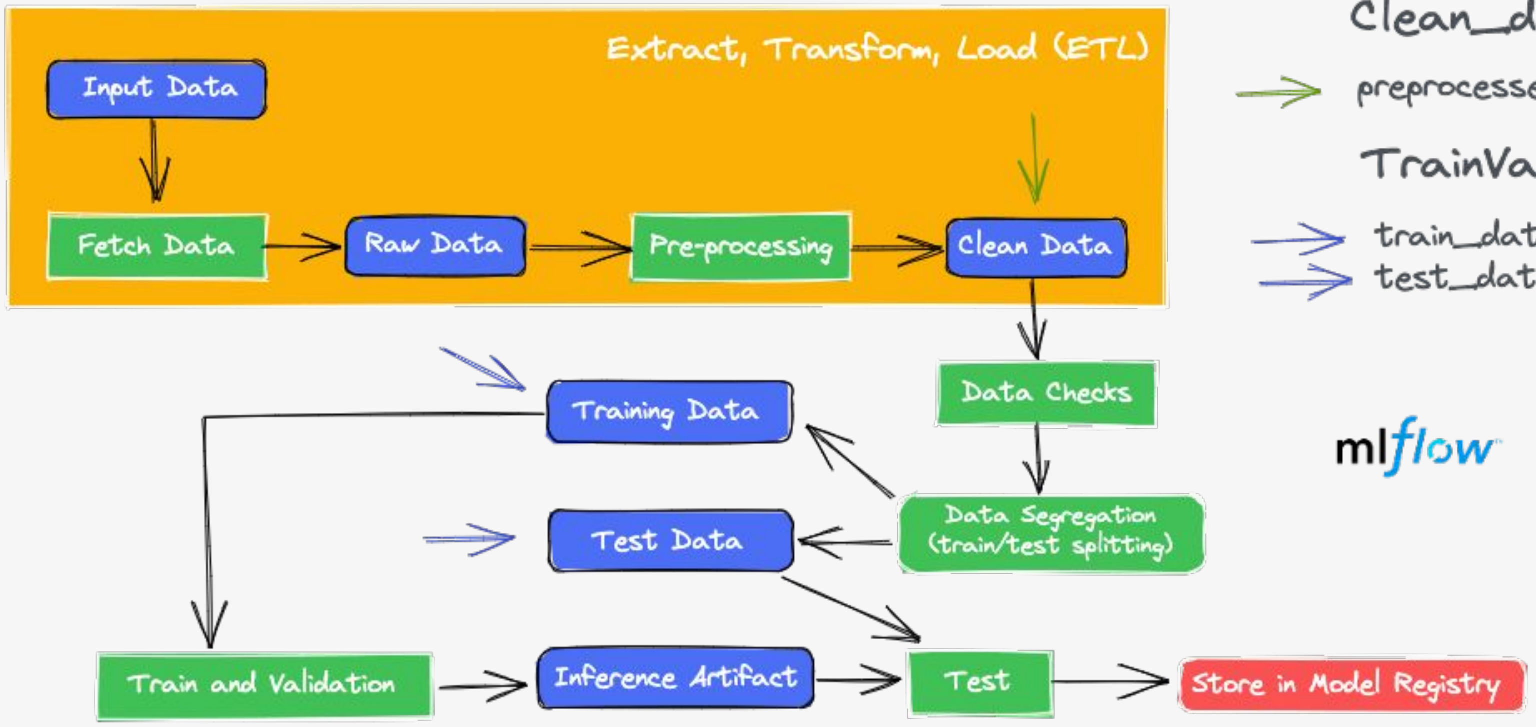
# EDA Example 01

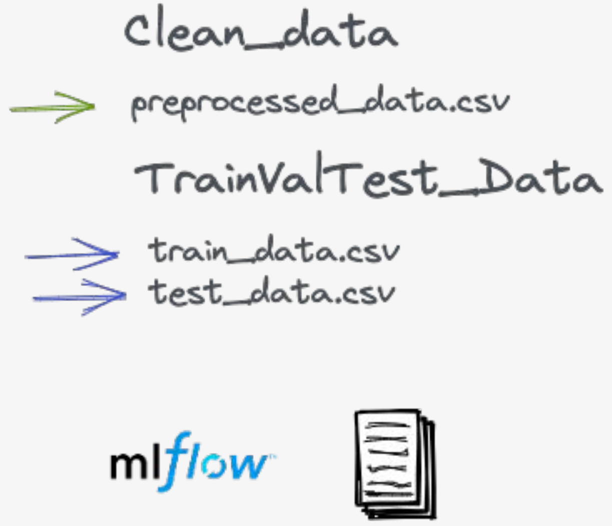


# EDA Example 02



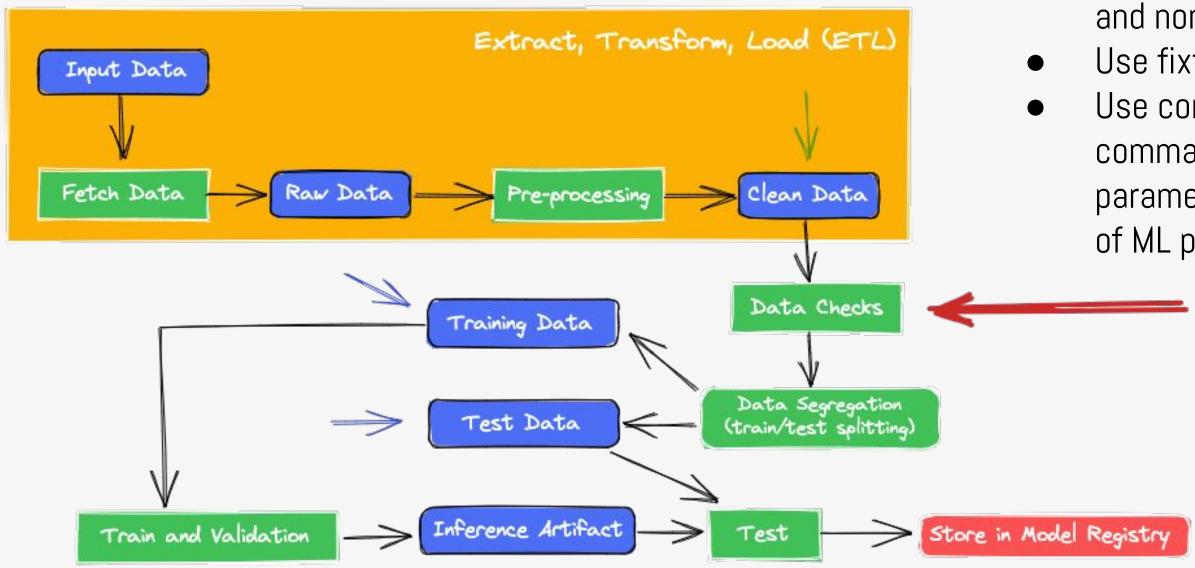
# Data Segregation Example 03





# Data Validation

## We want to avoid that bad data gets into your ML pipeline



- Write tests with pytest, both deterministic and non-deterministic
- Use fixtures to share data between tests
- Use conftest.py to add options to the command line of pytest so you can pass parameters and use it within components of ML pipelines

## Why you need data validation?

A necessary step in a ML pipeline, to **catch changes in the data** that can happen at any time for reasons that might even be external to your system (changes in the world). It is **especially important to allow automatic retraining.**

1. Bugs are introduced upstream (for example in ETL pipelines)
2. Changes in the source of the data are not communicated properly and produce unexpected changes in the input data
3. The world changes and the distribution of the input data changes

## Why you need data validation?

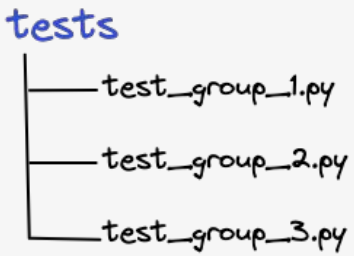
### Let's see some examples of the 3 cases

1. **Due to a bug** in an upstream data ingestion pipeline, the target of a regression model went from **units of million of dollars to just dollars**, so what used to be **1 is now 1,000,000**. This is likely to require a different preprocessing step, for example a scaling stage.
2. The review we were tracking went from a **3 stars to a 10 stars** rating system, and **we weren't alerted** of the change.
3. **Remote working is now more widespread**, consequently people are willing to buy houses much farther from their workplace. **A model that predicts price will have to take that into account**, and probably a new model will be necessary.

# Primer on Pytest

## A tool to execute test

Tests are collected in a directory, usually called **tests** and they are grouped into files. These files must have a name starting with `test_`, like `test_data.py` or `test_algorithm.py`.



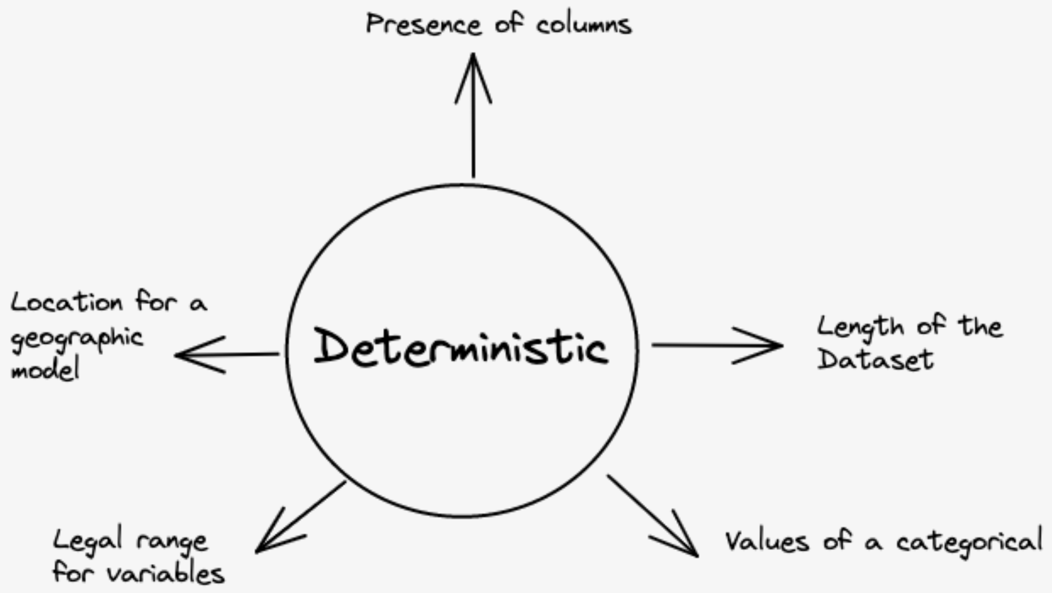
Within each file you can have as many tests as you want. Each test is a function that again must start with `test_`. Within that function you can have one or more **assertions**, where you use the assert command to check that a condition is true.

```
def test_data_length (data):
    """
    Test that we have at least 1000 rows
    """
    assert len(data)> 1000
```

## Primer on Pytest - Fixtures (Example 04)

```
import pytest
import wandb

run = wandb.init()

@pytest.fixture(scope="session")
def data():

    local_path = run.use_artifact("my_project/artifact").file()
    df = pd.read_csv(local_path)

    return df

def test_data_length(data):
    """
    We test that we have enough data to continue
    """
    assert len(data) > 1000
```

## Deterministic Tests (Example 05)

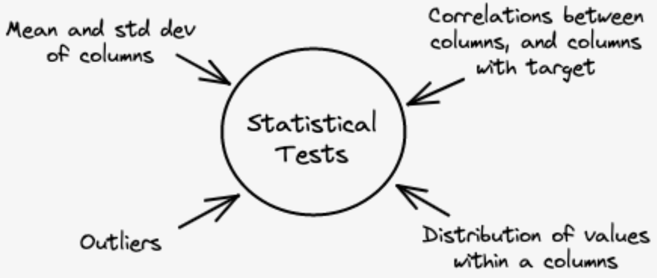


A data test is deterministic when it verifies attributes of the data that can be measured without uncertainty. Some examples are:

- Number of columns in a dataset
- Length of the dataset
- Distinct values in a categorical variable
- Legal range for a numerical variable (for example, length > 0)
- ...and many more!

# Non-Deterministic (Statistical) Tests

A test is non-deterministic when it involves measuring a quantity with intrinsic uncertainty (random variable). It probes for violation of an assumption about the data by using Statistical Hypothesis Testing.


Mean and std dev of columns
Correlations between columns, and columns with target
Statistical Tests
Outliers
Distribution of values within a columns

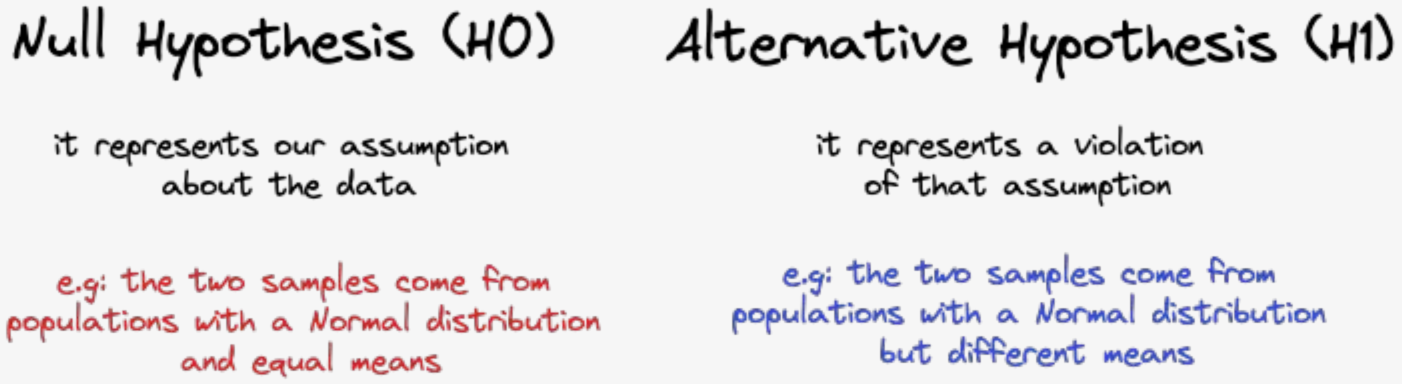
A|B

compare with previous dataset

typically, non-deterministic tests compare the current dataset with the previous one, or a fixed reference one

## Hypothesis Testing

We are going to consider Frequentist Hypothesis Testing. In this statistical framework we have a null hypothesis and an alternative hypothesis.



Null Hypothesis (H0)

it represents our assumption about the data

e.g: the two samples come from populations with a Normal distribution and equal means

Alternative Hypothesis (H1)

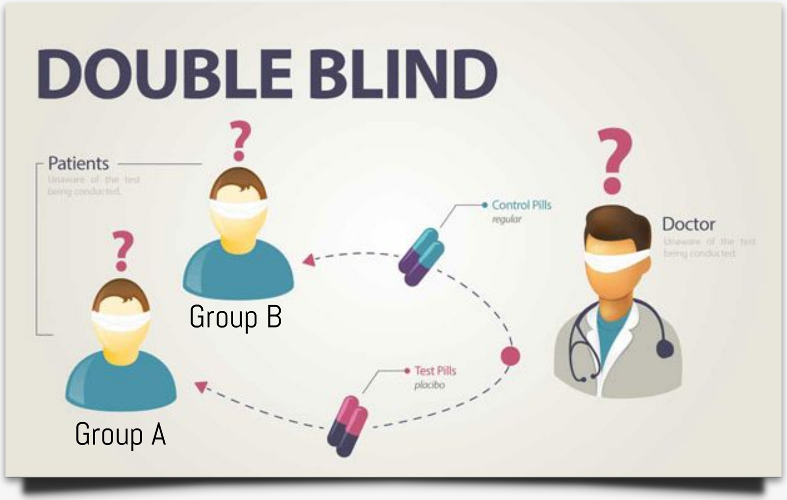
it represents a violation of that assumption

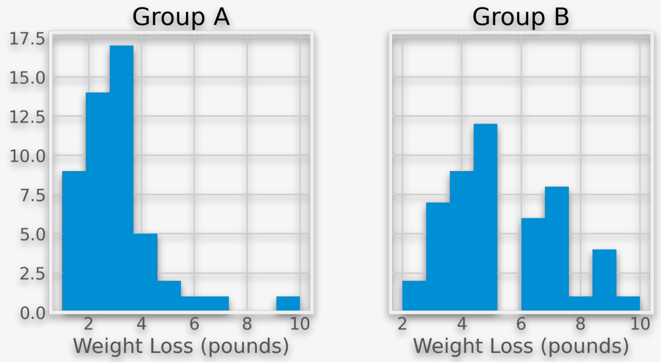
e.g: the two samples come from populations with a Normal distribution but different means

# Hypothesis Testing (Example 06)

A new weight loss pill helped people lose more weight:

$H_0$: patients who went on the weight loss pill lost no more weight than those who didn't.
$H_1$: patients who went on the weight loss pill lost more weight than those who didn't





Group A was given a placebo, or fake, pill and instructed to consume it on a daily basis.

Group B was given the actual weight loss pill and instructed to consume it on a daily basis.

Null hypothesis:
mean(b) - mean(a) = 0

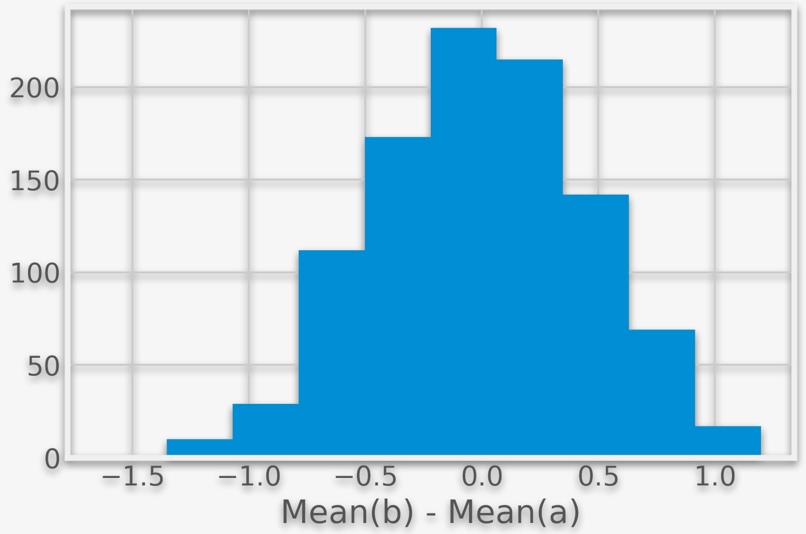
Alternative hypothesis
mean(b) - mean(a) > 0

Observed result
mean(b) - mean(a) = 2.52

# Permutation Test

The goal is to calculate a distribution of the test statistics over these many iterations.
This distribution is called the **sampling distribution** and it approximates the full range of possible test statistics under the null hypothesis.

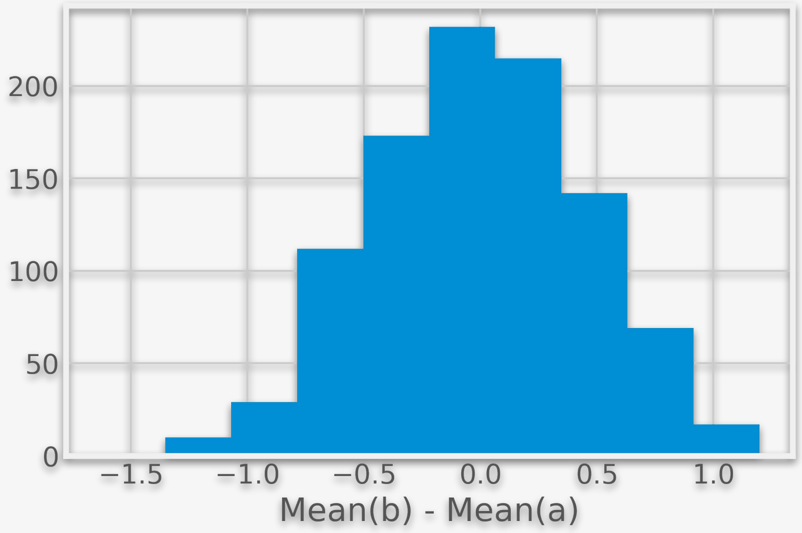
```
mean_difference = 2.52
mean_differences = []
for i in range(1000):
   group_a = []
   group_b = []
   for value in all_values:
       assignment_chance = np.random.rand()
       if assignment_chance >= 0.5:
           group_a.append(value)
       else:
           group_b.append(value)
   iter_diff = np.mean(group_b) - np.mean(group_a)
   mean_differences.append(iter_diff)
plt.hist(mean_differences)
```



## Sampling Distribution and P-Value

Top 10 values from 1000 rounds

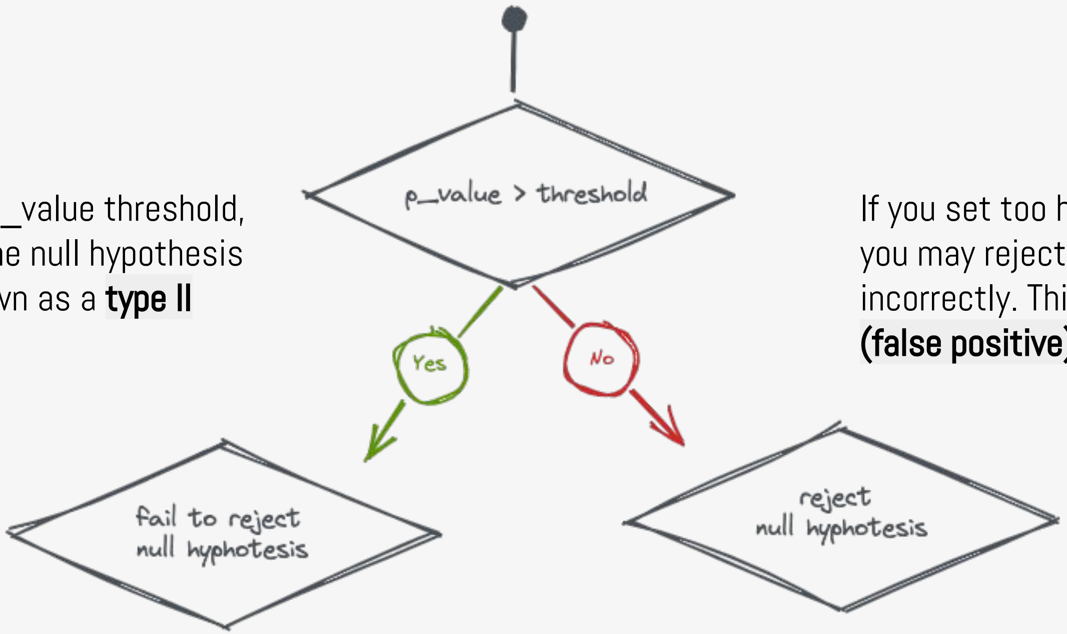
```
[(0.15686274509803955, 8),
 (0.07371794871794801, 7),
 (0.4800000000000004, 7),
 (0.323293317326935, 6),
 (0.08000000000000007, 6),
 (0.20673076923076916, 6),
 (0.19999999999999973, 6),
 (-0.049779205138499094, 5),
 (-0.08000000000000007, 5),
 (0.3115214773183461, 5)]
```



```
frequencies = []
for sp in sampling_distribution.keys():
   if sp >= 2.52:
       frequencies.append(sampling_distribution[sp])
p_value = np.sum(frequencies) / 1000
```

In general, it's good practice to set the p_value threshold before conducting the study

If you set too low of a p_value threshold, you may fail to reject the null hypothesis incorrectly. This is known as a **type II error (false negative).**



If you set too high of a p_value threshold, you may reject the null hypothesis incorrectly. This is known as a **type I error (false positive).**

## Multiple Hypothesis Testing

Multiple hypothesis testing occurs often in practice when searching for anomalies in multiple datasets.

Roughly speaking, a statistical test is something capable of either **rejecting** or **not the null hypothesis** with a given **type I** error probability $\alpha$ (probability of a false positive) and a **type II** error probability $\beta$ (probability of a false negative).

# Multiple Hypothesis Testing

## Toy Problem (Example 07)

Let's consider the problem of determining whether two populations have the same average or not.

- The null hypothesis is that the average is the same.
- The alternative hypothesis is that it is not.

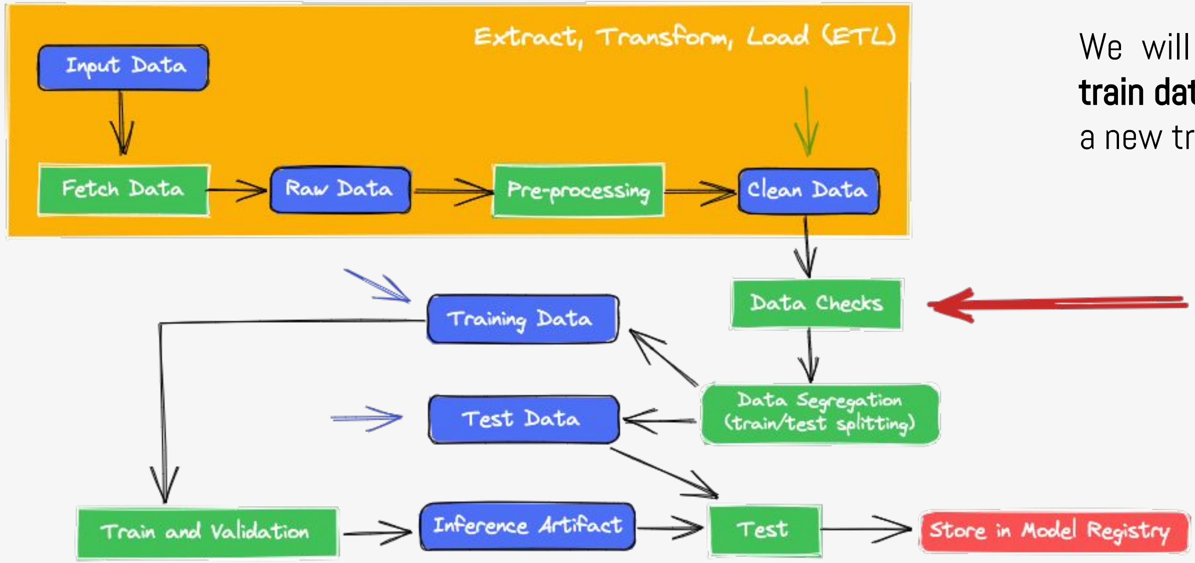
The test appropriate for the problem at hand is the Student's T-test.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample are normally distributed.
- Observations in each sample have the same variance.

# Multiple Hypothesis Testing

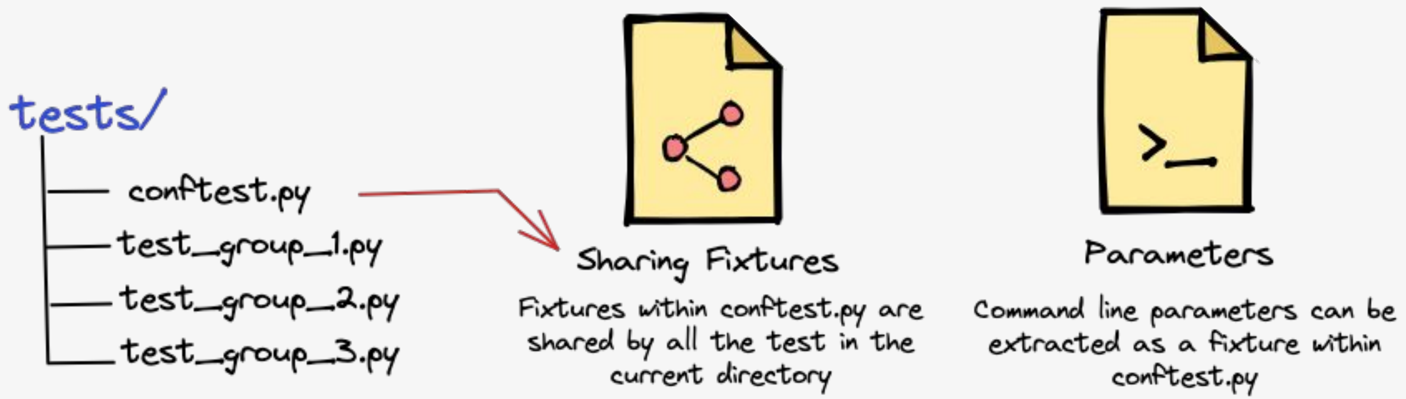
## Example 08



We will compare the **test dataset** against the **train dataset**. This is a useful trick when obtaining a new training dataset right away is not possible.

We will use the Kolmogorov-Smirnov test for goodness of fit. Remember that the 2 sample KS test is used to test whether two vectors come from the same distribution (null hypothesis), or from two different distributions (alternative hypothesis), and it is non-parametric.

# Parameters in PyTest


tests/
conftest.py
test_group_1.py
test_group_2.py
test_group_3.py
Sharing Fixtures
Fixtures within conftest.py are shared by all the test in the current directory
Parameters
Command line parameters can be extracted as a fixture within conftest.py

```python
def pytest_addoption(parser):
    parser.addoption("--input_artifact", action="store")
```

```
> pytest . -vv --input_artifact example/my_artifact:latest
```

# Parameters in PyTest

## Example 09

conftest.py

```python
def pytest_addoption(parser):
    parser.addoption("--input_artifact", action="store")

@pytest.fixture(scope="session")
def data(request):
    input_artifact = request.config.option.input_artifact
    if input_artifact is None:
        pytest.fail("--input_artifact missing on command line")
    local_path = run.use_artifact(input_artifact).file()
    return pd.read_csv(local_path)
```