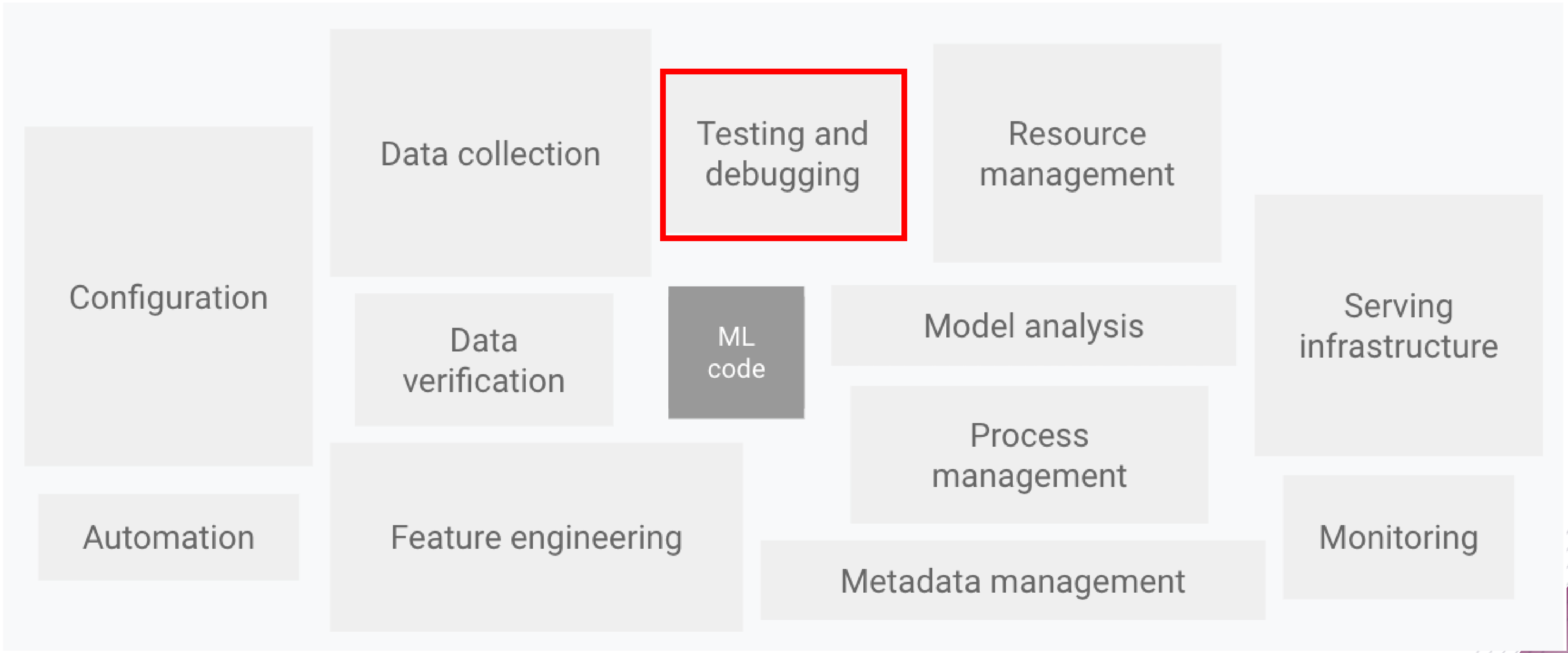
2.4 Testing in MLOps

POSGRAD
MNA

Introduction

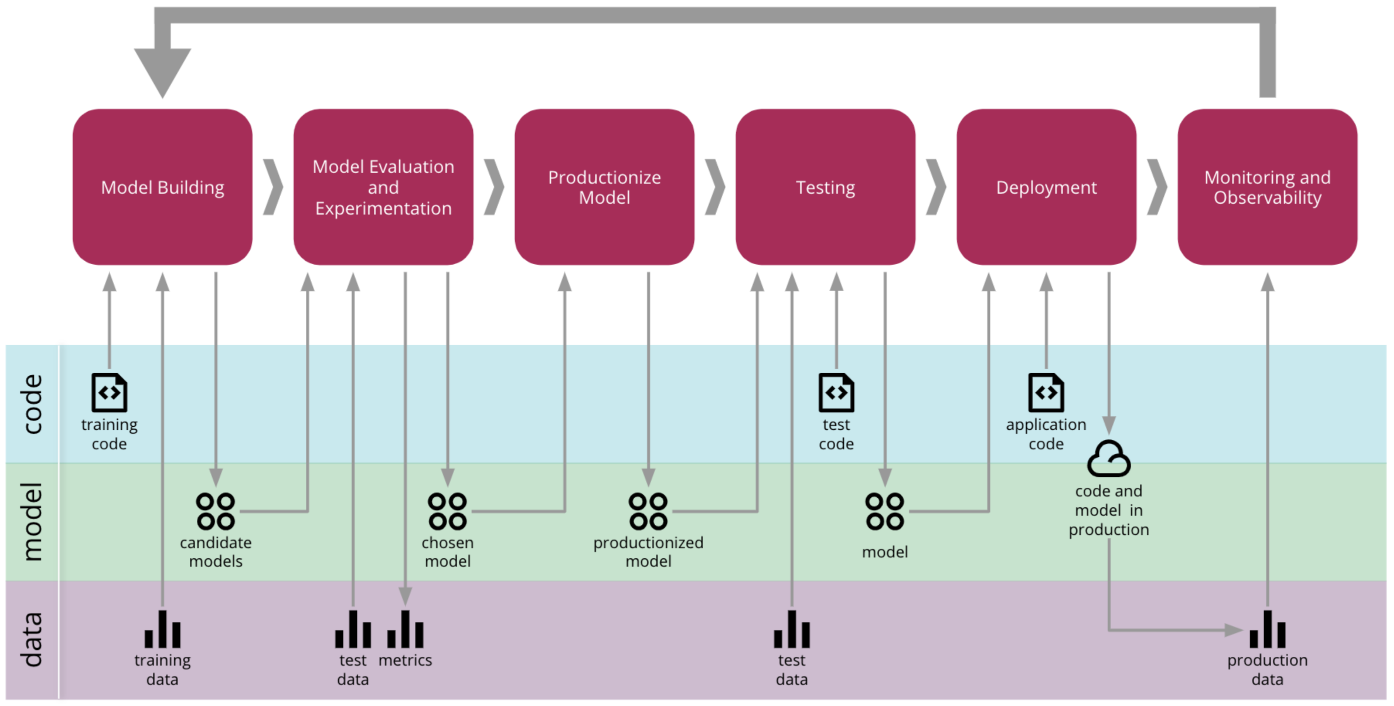
Since we start thinking about moving an ML pipeline into production, we need to consider thinking about the entire system in which our pipeline will be. And the main idea, is that ML systems are difficult to maintain due to the fact that it includes all the pitfalls we encounter in traditional software development, plus the data dependencies, and the infrastructure issues that may arise.

In the next slides, we will be covering the Testing part of this components, and why it is important when developing our ML Pipeline



Our Machine Learning pipelines require testing at different levels because they do not depend only on the code but on the data as well.

In those stages were we crafted our model, we need to include tests to the data and to the model as well. And do not forget when the model is already deployed, because additional tests needs to be added to ensure that the model is working properly.



Key testing principles for Machine Learning pipelines.

What is it, and why do we care?

We need to ensure that our system is behaving in the way we have designed it; even if we make further changes to the system, we need to provide the following:

- Reliability in our system (past and future).
- Awareness of constant change at every stage of the pipeline.
- Functionality

When would it be enough?

It will depend on the reliability we want in our system.

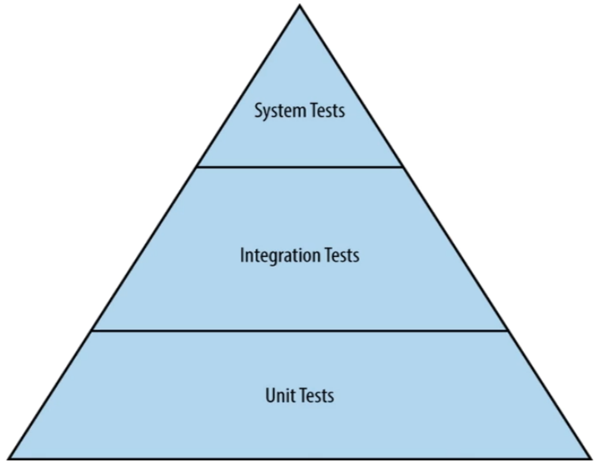
Testing on research environments.

- We try different algorithms.
- We use different settings in our hyperparameters.
- We work with different datasets and use distinct features.

Testing on development environments.

As we move to different environments, we need to use different testing techniques in order to make our pipeline production ready. Some of those test are:

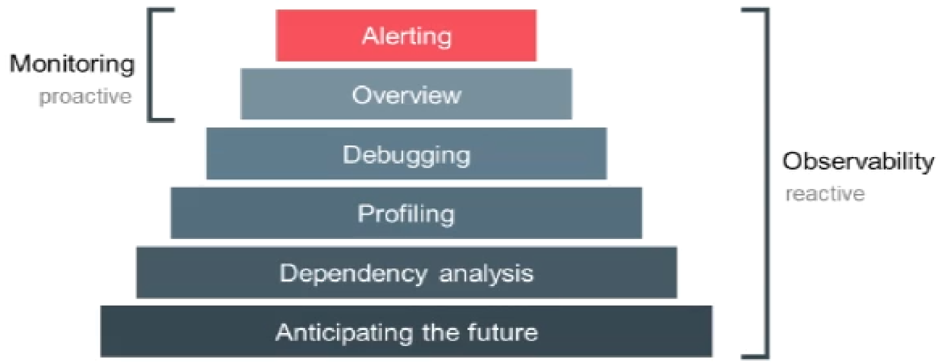
- Unit, integration, and acceptance test.
- Differential tests.
- Benchmark tests.
- Load tests.



Testing on production environments.

Once the model is deployed, we will use the following techniques so we can evaluate them with real data at the same time as we monitor it's behavior on production.

- Shadow mode testing.
- Canary releases.
- Observability and monitoring.
- Logging and tracing.
- Alerting



Credit card analogy

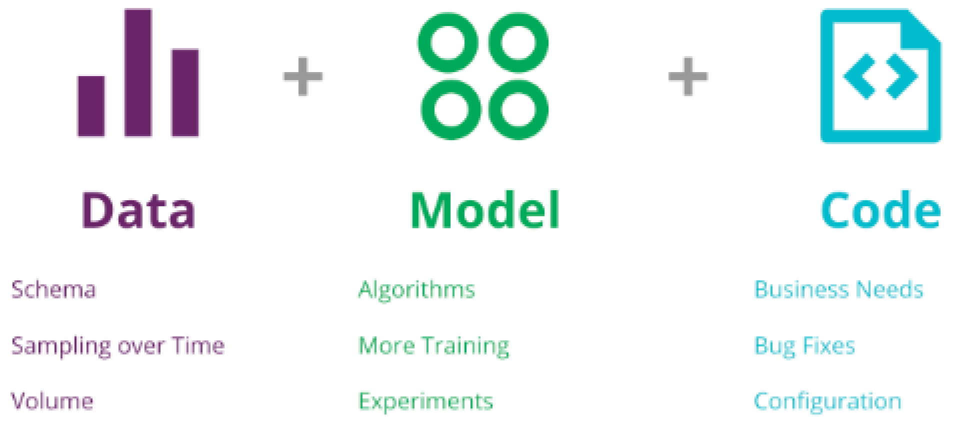
The real value of testing comes with changes, which is why we often skip tests at the start of a project when requirements are simple. At that stage, no new changes or user feedback have arrived yet.

Skipping tests is often compared to accumulating credit card debt—it becomes more complicated over time. It's easy to collect but harder to resolve.

How many tests should we write? It depends on the reliability of our system's requirements. The goal is to reduce uncertainty caused by system changes.

Why is it difficult to make tests in ML systems?

When we are developing software, the code is the element that controls the behavior of the system, but, when talking about ML pipelines, we are including two new components into the equation that have a huge impact on the behavior of our system, the data, and the model.



Validation checks for deployment.

Using schemas for the features:

Encoding the metadata about the data in a schema is useful so it can be automatically checked.

Create specification tests for the models:

Important configuration changes in the model or in its hyperparameters should have a code review and unit tests in certain cases; the reason for this is to identify unexpected value ranges or values for categorical features.

Test the input code for the variables:

All input code needs to be tested, it may seem that it is simple enough to avoid unit tests, but it’s crucial for the correct behavior of our model

Validation checks for deployment.

Validate the model's quality:

Before pushing our model into production, we need to test two types of quality degradation: sudden degradation and slow degradation.

Ensure the model's reproducibility:

Ideally, training twice on the same data, should produce two identical models, when this happens, the reasoning about the whole system is simplified and it helps us un auditability and debugging. To avoid this, you can look for steps in the pipeline that are not deterministic, like avoiding to set the random seed or not ensuring that the keys are sorted in json inputs for certain types of optimization algorithms.

Integration tests in the pipeline:

A complete pipeline includes assembling training data, feature generation, model training, model verification, and deployment to a serving system, and each stage can introduce errors that could affect the other parts of the pipelines

Tests to consider for deployment.

The fundamental set of tests that should be considered are the following:

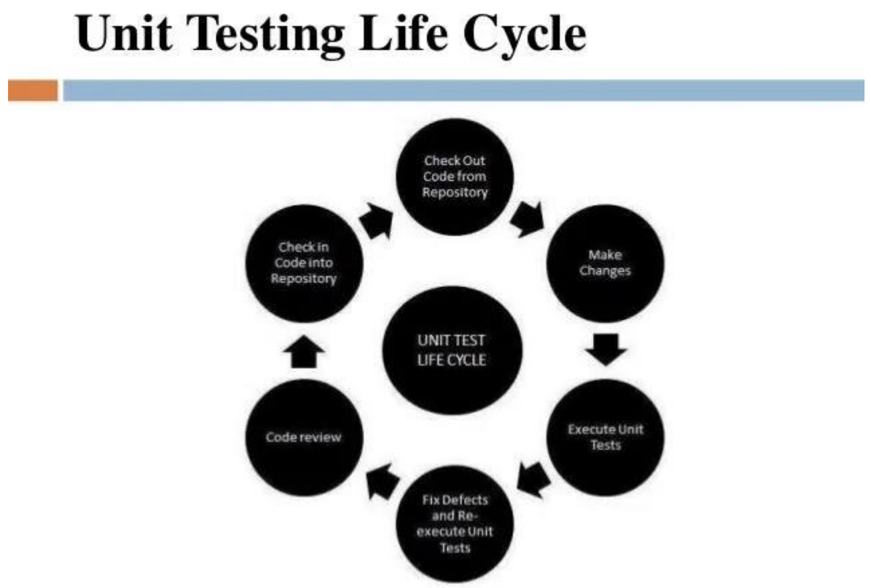
- Unit tests.
- Integration tests.
- Differential tests.
- Shadow mode tests.

Unit test in ML.

Unit testing in ML Pipelines

The set of unit tests that need to be considered for machine learning pipelines includes the following:

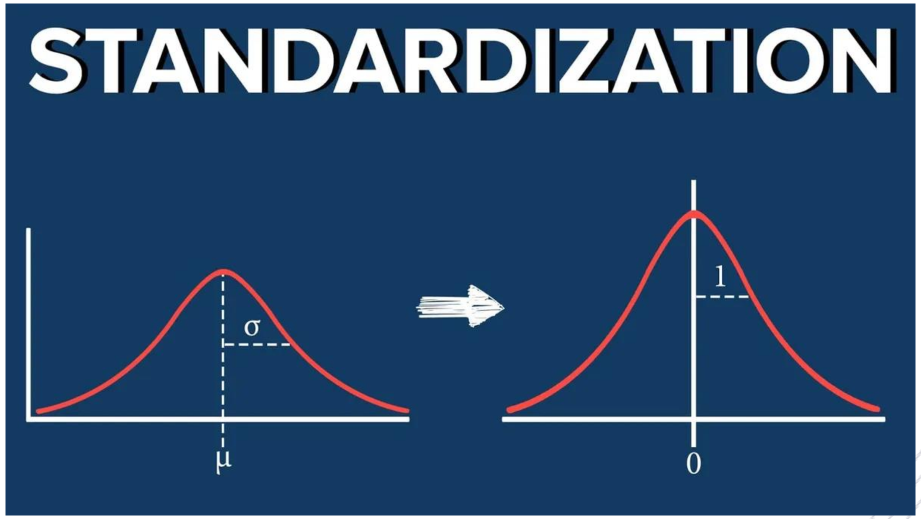
- Preprocessing and Feature Engineering.
- Model Settings.
- Input data.
- Model quality.


Unit Testing Life Cycle
Check Out Code from Repository
Make Changes
Execute Unit Tests
Fix Defects and Re-execute Unit Tests
Code review
Check in Code into Repository
UNIT TEST LIFE CYCLE

Preprocessing and Feature Engineering Tests.

Bugs in features may be impossible to detect once they have entered into the model development process, especially if these data are represented in both the test and training sets.

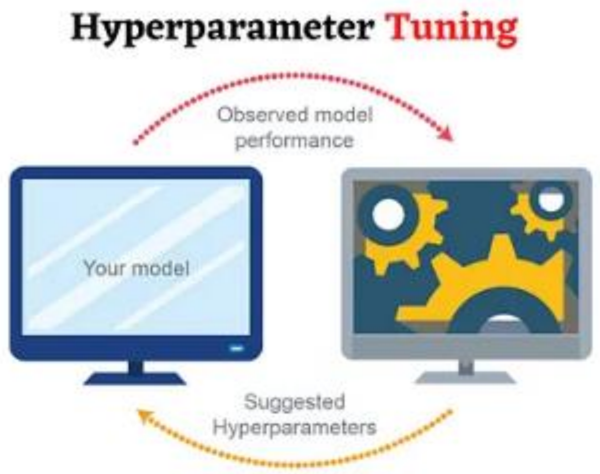
- Numerical values are scaled and normalized.
- Those feature engineering steps that involve calculations
- Handling of missing data
- Handling of outliers



Model Setting Tests.

If we know that a particular hyperparameter setting creates a poor performance in our model, it makes sense not to include them in the config file. Also, web service configurations arise for a particular configuration file that has some configuration tests looking for the settings that are available in production to see how particular code is configured and report discrepancies. The most subtle configuration change can modify the behavior of our entire system.

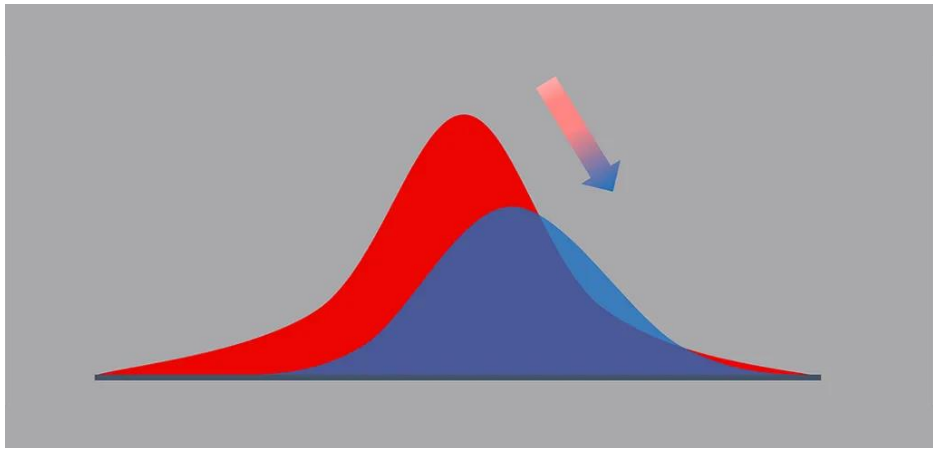
- Hyperparameter setting.
- Web service setting.



Input Data Tests.

In this case we are referring to the raw data or raw input from a client. Testing the data against the schema may let us to find anomalies, unexpected values for categorical data. Catching these errors on time saves much time and it's easier to debug; if we do not managed them before, these errors could make a mess once we have to work with a large number of features that are generated from complex input data.

- Understanding the range and distributions
- Considering values as potential categorical variables.
- Testing our data against the schema.



Model Quality Tests.

In model quality, the quality of our model is validated after training but before being used in real life with accurate data. If we need random numbers, ensure that we have set the seed for reproducibility.

Our tests need to be as simple as possible, there is no need to create tests that checks against a validation set, or that will look for convergence to a specific distribution, and this tests will help us to avoid certain degradation problems.

- Deterministic Tests.
- Short Tests.

Degradation types in Machine Learning.

ML pipelines usually suffer from two types of degradation:

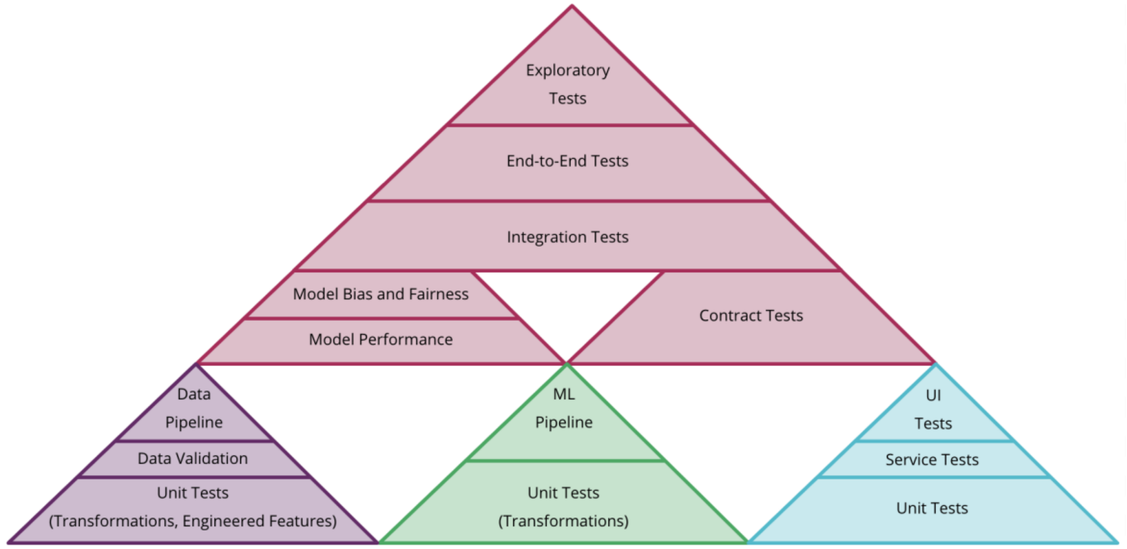
- **Sudden degradation**: Usually caused by a bug in the new version that results in lower accuracy, which can be validated against previous versions of the model.
- **Gradual degradation**: Harder to detect, it requires that the training and test sets be updated to prevent it from impacting model development.

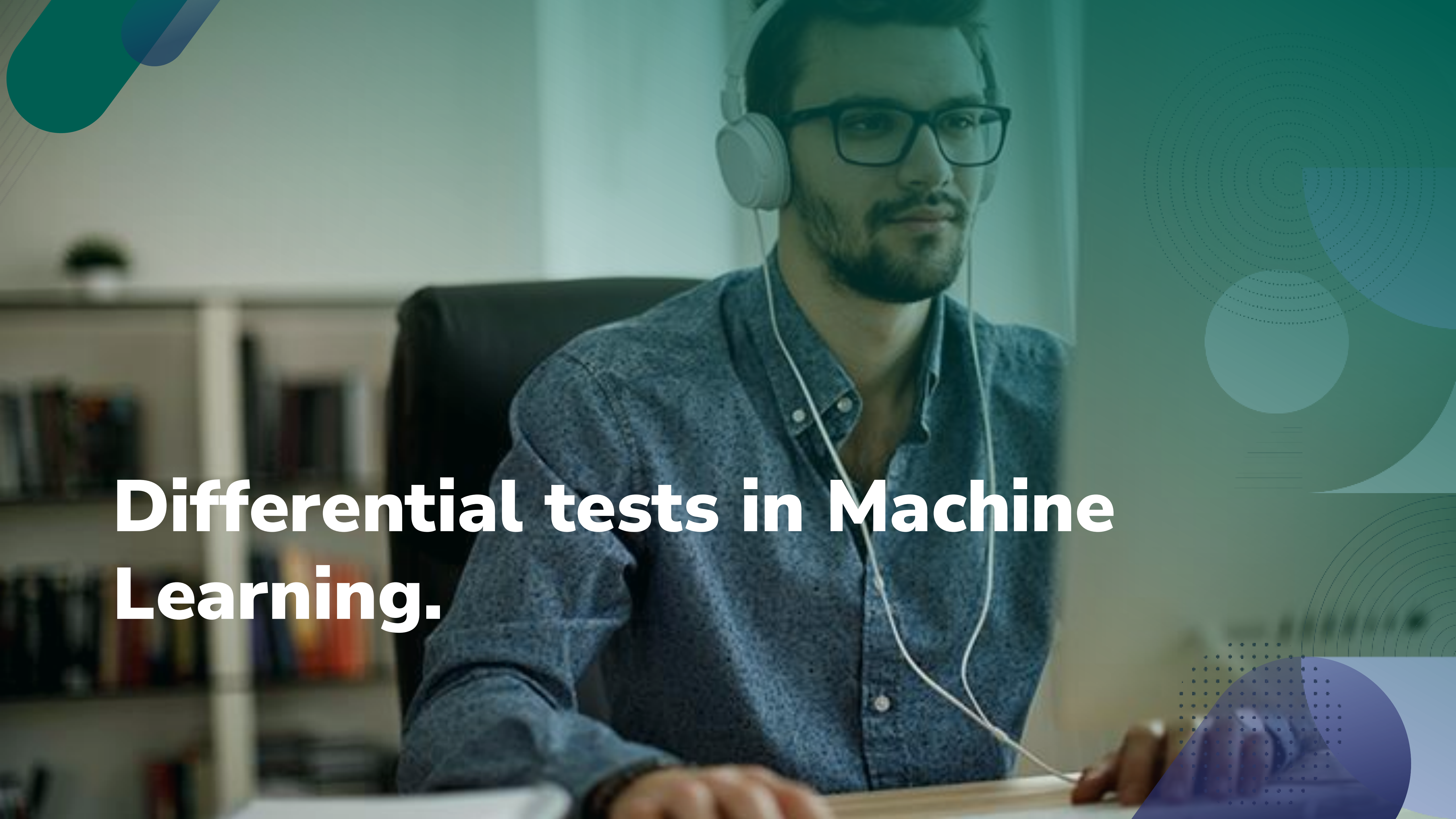
Integration tests in Machine Learning.

Integration Tests in Machine Learning.

Integration tests should be run continuously and during new deployments of our models to catch any issues before they reach production.

As soon as the integration tests are run with a subset of our test data or on a simple model, we will get faster results.





Differential tests in Machine Learning.

Differential tests compare the differences in executing one system version against another when the inputs are the same. In the case of ML, this refers to the difference between one model and another using the same data.

Their primary use is to detect errors that don't raise exceptions in our traditional tests, such as bugs in our feature engineering tasks, which can lead to poor model performance.

We can check that different components work together to produce the result we expect from our system, for example: If I forget to include a feature which is important for the model, but the pipeline is still functioning.

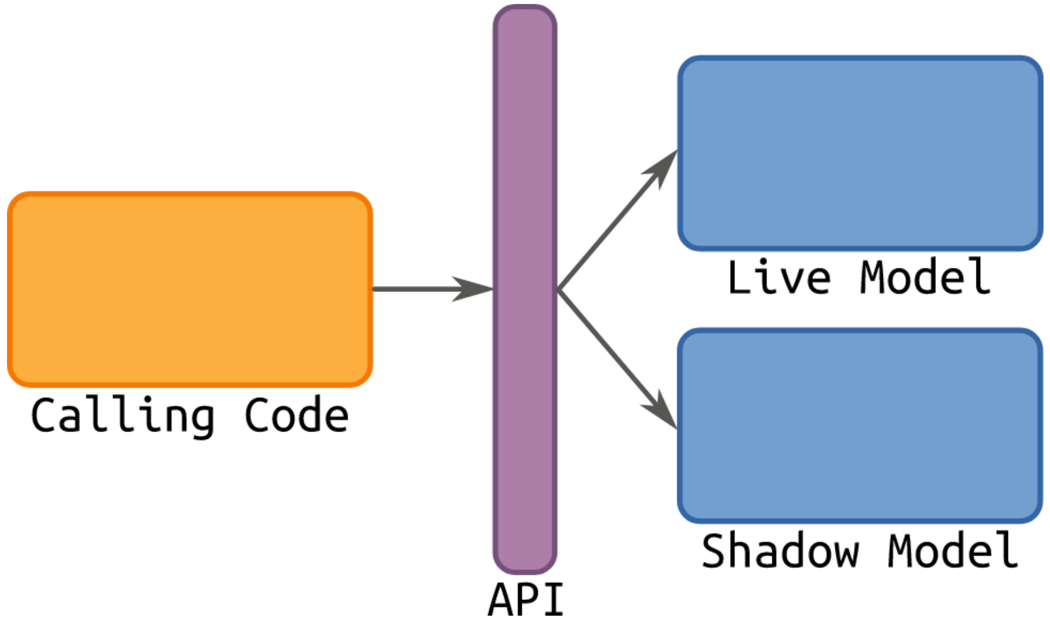
We can consider them an end-to-end test, and they are created to anticipate that our system may go wrong. It is a must for effective ML systems.

Shadow Mode.

Once our code is deployed, the tests we generate change and reach the production environment. The idea is that these tests should not affect the end user.

Data is collected for analysis purposes.

Its implementation requires a high level of maturity within the DevOps team.



References.

For further references, here are the following links.

- The ML test score: A rubric for ML Production Readiness and Technical Debt Reduction.
- Testing ML based Systems.

Tecnológico de Monterrey