Machine Learning Workflow

1. What Is a Machine Learning Workflow?

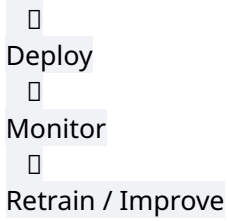
A machine learning workflow is the **complete process of turning a real-world problem into a working ML system**.

It is not simply:

`Dataset → Train Model → Get Prediction`

A realistic workflow is closer to:

Problem
↓
Define Success
↓
Collect Data
↓
Understand Data
↓
Clean & Prepare Data
↓
Split Data
↓
Build Baseline
↓
Train Models
↓
Evaluate
↓
Improve
↓
Final Test



The important idea is that **machine learning is a process, not just an algorithm**.

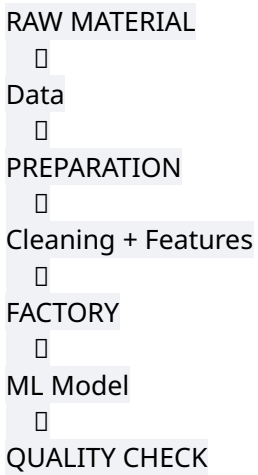
A good algorithm cannot rescue a badly defined problem or poor-quality data.

Google's ML guidance emphasizes problem framing as an early and important step because it determines what success means and whether ML is appropriate for the problem.

---

2. The Big Mental Model

Think of ML as a factory.



Output:
download = yes/no

This becomes a classification problem.

Example

Suppose we have:

Visits	Resources Viewed	Time on Site	Downloaded
2	1	30	0
5	4	180	1
1	1	20	0
8	6	300	1

The target is:

Downloaded

The features are:

Visits

Resources Viewed
Time on Site

So:

X = features
y = target

---

4. Define Success Before Training

"High accuracy" is not always enough.

You need a measurable definition of success.

For example:

Goal:
Predict whether a user will download a resource.

Success criteria:
Recall > 80%
Precision > 70%
Prediction latency < 100 ms

Different problems require different metrics.

For example:

Classification

Possible metrics:

Accuracy
Precision
Recall
F1-score
ROC-AUC

Regression

Possible metrics:

MAE
MSE
RMSE
R^2

The metric should reflect the actual business or product objective.

---

5. Step 2: Collect Data

Machine learning learns patterns from data.

Possible sources include:

Databases
CSV files
APIs
Sensors
Application logs
User activity
Surveys
Images
Text

Audio
Existing datasets

For a website:

Website

Events

Database

Dataset

ML Model

Example:

user_id
pages_viewed
resources_viewed
session_duration
downloads
device
country

---

6. Data Quantity vs Data Quality

More data is not automatically better.

Imagine:

1,000,000 incorrect records

versus:

```
100,000 reliable records
```

The second dataset may be much more useful.

Check:

```
Missing values
Duplicates
Incorrect types
Impossible values
Outliers
Incorrect labels
Data leakage
Class imbalance
Inconsistent formats
```

---

7. Step 3: Understand the Dataset

Before training a model, inspect the data.

With pandas:

```
import pandas as pd

df = pd.read_csv("users.csv")

print(df.head())
print(df.shape)
print(df.info())
```

```
print(df.describe())
print(df.isna().sum())
```

Ask:

How many rows?
How many columns?
Which columns are numeric?
Which are categorical?
Which values are missing?
What is the target?
Are there duplicates?
Are there suspicious values?

This stage is often called **Exploratory Data Analysis (EDA)**.

---

8. Step 4: Clean the Data

Raw data rarely arrives ready for a model.

You may need to:

Remove duplicates
Handle missing values
Fix data types
Normalize formats
Handle invalid records
Treat outliers
Encode categories

Example:

```
df = df.drop_duplicates()

df["age"] = pd.to_numeric(
    df["age"],
    errors="coerce"
)

df["age"] = df["age"].fillna(
    df["age"].median()
)
```

The exact cleaning strategy depends on the dataset.

---

9. Step 5: Define Features and Target

Suppose:

```
age
visits
resources_viewed
session_minutes
downloaded
```

If we want to predict `downloaded`:

```
X = df[
    [
        "age",
        "visits",
        "resources_viewed",
        "session_minutes"
    ]
```

```
]

y = df["downloaded"]
```

Think:

```
X = what the model knows
y = what the model should learn to predict
```

---

10. Step 6: Split the Data

A model needs data for learning and data for testing.

A common basic setup is:

```
Training data
+
Testing data
```

For example:

```
80% → training
20% → testing
```

With scikit-learn:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
```

```
    test_size=0.2,
    random_state=42
)
```

`train_test_split()` creates random training and testing subsets, and `random_state` can make the split reproducible.

---

11. Why Do We Split the Data?

Imagine giving a student the exact questions that will appear on the exam.

The student may memorize them.

That does not prove they understand the subject.

The same problem happens in ML.

```
Training data
    
Model learns

Testing data
    
Model is evaluated on unseen examples
```

We want to know:

> Can the model generalize to data it did not see during training?

---

12. Train / Validation / Test

For more serious projects, three datasets are useful:

Training Set

Learn parameters

Validation Set

Choose models/settings

Test Set

Final unbiased evaluation

Example:

70% Training
15% Validation
15% Test

The exact percentages are not universal.

Another approach is to use **cross-validation** for model selection instead of maintaining a separate validation set.

---

13. Step 7: Build a Baseline

Before building a complicated model, create a simple baseline.

Suppose you want to predict whether a customer will buy.

A baseline might simply predict:

```
Always predict "No"
```

If 90% of customers do not buy, this baseline gets:

```
90% accuracy
```

A complicated model getting 89% accuracy would not automatically be useful.

The baseline gives you something to compare against.

---

14. Step 8: Choose a Model

Model selection depends on the problem.

Classification

Examples:

```
Logistic Regression
Decision Tree
Random Forest
Gradient Boosting
SVM
Neural Network
```

Regression

Examples:

Linear Regression
Decision Tree Regressor
Random Forest Regressor
Gradient Boosting
Neural Network

Clustering

Examples:

K-Means
Hierarchical Clustering
DBSCAN

Do not start with:

> Which algorithm is the most advanced?

Start with:

> Which model is appropriate for this problem and dataset?

---

15. Step 9: Preprocess Features

Models may require data to be transformed.

Common operations include:

Scaling
Encoding
Imputation
Feature extraction
Feature selection
Transformation

For example, suppose:

Age: 25
Annual income: 2,000,000

The scales are very different.

Some algorithms can benefit from standardization.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Notice:

fit_transform → training data
transform → testing data

The scaler learns its parameters from training data and then applies those learned parameters to unseen data.

---

16. Step 10: Avoid Data Leakage

Data leakage happens when information that should not be available during training accidentally enters the model-building process.

Example:

```
Dataset
  ↓
Scale entire dataset
  ↓
Split into train/test
```

This can leak information from the test set.

Instead:

```
Dataset
  ↓
Split
  ↓
Train data → fit preprocessing
Test data  → transform using training preprocessing
```

A scikit-learn `Pipeline` can combine preprocessing and the model while helping prevent this kind of leakage.

---

17. Step 11: Build a Pipeline

Instead of manually managing every transformation:

```
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("model", LogisticRegression())
])
```

Train:

```
pipeline.fit(X_train, y_train)
```

Predict:

```
predictions = pipeline.predict(X_test)
```

The pipeline applies the transformations in sequence and then passes the result to the final estimator.

---

18. Step 12: Train the Model

Now the model learns patterns.

```
model.fit(X_train, y_train)
```

Conceptually:

```
Features
   
Model
   
Learn patterns
```

Parameters

For example:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

model.fit(X_train, y_train)
```

Training does not mean the model has become "correct."

It means the model has learned parameters from the training data.

---

19. Step 13: Make Predictions

After training:

```
predictions = model.predict(X_test)
```

For probabilities:

```
probabilities = model.predict_proba(X_test)
```

Conceptually:

New input

Preprocessing


```
Trained model
   
Prediction
```

---

20. Step 14: Evaluate the Model

Now ask:

> How well does the model perform on unseen data?

For classification:

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(
    y_test,
    predictions
)

print(accuracy)
```

But accuracy is only one possible metric.

For binary classification:

```
Actual
       Positive Negative

Pred   TP       FP
       FN       TN
```

These lead to:

`Precision`
`Recall`
`F1-score`

Choose metrics based on what errors matter.

---

21. Understand Errors

Suppose a fraud detector predicts:

`Fraud`

There are two major types of mistakes:

False Positive

The model says:

`Fraud`

but the transaction is legitimate.

False Negative

The model says:

`Legitimate`

but the transaction is actually fraud.

The cost of these errors may be very different.

Therefore:

`Best metric ≠ always accuracy`

---

22. Step 15: Compare Models

Do not evaluate only one algorithm.

For example:

```
models = {
    "logistic": LogisticRegression(),
    "tree": DecisionTreeClassifier(),
    "forest": RandomForestClassifier()
}
```

You can train and compare them using an appropriate validation strategy.

scikit-learn describes model selection as comparing, validating, and choosing models and parameters, including tools such as cross-validation and grid search.

---

23. Step 16: Cross-Validation

Instead of relying on one validation split, cross-validation repeatedly divides the training data into different training and validation portions.

Conceptually:

```
Fold 1:
Train Train Train Train | Validate

Fold 2:
Train Train Train | Validate | Train

Fold 3:
Train Train | Validate | Train Train
```

Then:

```
Score 1
Score 2
Score 3
...
  
Average performance
```

This can give a more reliable estimate for model comparison.

---

24. Step 17: Tune Hyperparameters

A model has parameters it learns from data.

It may also have **hyperparameters** that you choose.

Example:

```
RandomForestClassifier(
```

```
    n_estimators=200,
    max_depth=10
)
```

Here:

`n_estimators`
`max_depth`

are hyperparameters.

You can search through different combinations.

Example:

```
from sklearn.model_selection import GridSearchCV

search = GridSearchCV(
    model,
    {
        "max_depth": [5, 10, 20]
    },
    cv=5
)

search.fit(X_train, y_train)
```

The goal is not to find the biggest possible model.

The goal is to find a model configuration that performs well on unseen data.

---

25. Step 18: Check Overfitting

Suppose:

```
Training accuracy = 99%
Test accuracy     = 72%
```

The model may have learned the training data too closely.

This is **overfitting**.

Another situation:

```
Training accuracy = 65%
Test accuracy     = 64%
```

This may indicate **underfitting**.

Conceptually:

```
Underfitting
    
Model too simple

Good fit
    
Generalizes well

Overfitting
    
Model memorizes training patterns
```

---

26. Step 19: Improve the Model

If performance is poor, do not immediately switch to a more complicated algorithm.

Investigate:

Is the data correct?
Are labels correct?
Are important features missing?
Are there too many irrelevant features?
Is there class imbalance?
Is there leakage?
Is the model underfitting?
Is the model overfitting?
Is the evaluation metric appropriate?

Possible improvements:

Better data
Better features
Better preprocessing
Better model
Hyperparameter tuning
More training data
Better labels
Regularization
Feature selection

---

27. Step 20: Perform Final Testing

After selecting your approach, evaluate it on the untouched test set.

The test set should represent data the model did not use for training or model selection.

Conceptually:

```
Training data
    ↓
Experiment
    ↓
Validation / Cross-validation
    ↓
Choose final approach
    ↓
Test set
    ↓
Final evaluation
```

Do not repeatedly tune the model based on the test set.

Otherwise, the test set gradually becomes part of the development process.

---

28. Step 21: Save the Model

Once the model is ready, save it.

For scikit-learn models, one common approach is `joblib`.

```
import joblib

joblib.dump(
    pipeline,
    "model.joblib"
```

```
)
```

Later:

```
model = joblib.load("model.joblib")
```

For production systems, model artifacts should be versioned and managed carefully.

---

29. Step 22: Deploy the Model

A trained model is not automatically a product.

You need a way for applications to use it.

For example:

```
Frontend
   
Backend API
   
ML Model
   
Prediction
   
Backend
   
Frontend
```

A FastAPI endpoint could receive:

```
{
```

```
  "visits": 5,
  "resources_viewed": 4,
  "session_minutes": 180
}
```

and return:

```
{
  "will_download": true
}
```

---

30. Step 23: Monitor the Model

Deployment is not the end.

Suppose the model was trained on data from 2026.

Over time, user behavior changes.

```
Training data
     
Old behavior

Production data
     
New behavior
```

The relationship between inputs and predictions may change.

Monitor:

Prediction quality
Input distributions
Missing values
Latency
Error rates
Data drift
Model drift
System failures

---

31. Data Drift vs Model Performance

Data drift

The input data changes.

Example:

Old average session:
4 minutes

New average session:
15 minutes

Model performance degradation

The model's predictions become less accurate.

For example:

Accuracy:
91% □ 76%

A production ML system should therefore have monitoring and retraining strategies.

---

32. Step 24: Retrain

When new reliable data becomes available:

```
New data
  □
Validate
  □
Train
  □
Evaluate
  □
Compare with current model
  □
Deploy if appropriate
```

This creates an ML feedback loop:

```
          □□□□□□□□□□□□□□□□□
          □          □
Data □ Train □ Evaluate □ Deploy
 □                  □
 □□□□□ Monitor □ Production
```

---

33. Complete ML Workflow

The complete process can be remembered as:

1. Define the problem
2. Define success metrics
3. Collect data
4. Understand the data
5. Clean the data
6. Define features and target
7. Split the data
8. Build a baseline
9. Preprocess features
10. Train models
11. Evaluate models
12. Compare models
13. Tune hyperparameters
14. Check overfitting
15. Final test
16. Save the model
17. Deploy
18. Monitor
19. Retrain and improve

---

34. Complete Example: haas.dev Resource Download Prediction

Imagine haas.dev wants to predict:

> Will this visitor download a resource?

Dataset:

visits
resources_viewed

```
session_minutes
device_type
downloaded
```

Step 1: Define target

```
y = df["downloaded"]
```

Step 2: Define features

```
X = df[
    [
        "visits",
        "resources_viewed",
        "session_minutes"
    ]
]
```

Step 3: Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

For classification, `stratify=y` can preserve class proportions across the split.

Step 4: Build model

```
from sklearn.pipeline import make_pipeline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

model = make_pipeline(
    StandardScaler(),
    LogisticRegression()
)
```

Step 5: Train

```
model.fit(X_train, y_train)
```

Step 6: Predict

```
predictions = model.predict(X_test)
```

Step 7: Evaluate

```
from sklearn.metrics import accuracy_score

score = accuracy_score(
    y_test,
    predictions
)

print(score)
```

Step 8: Save

```
import joblib

joblib.dump(
    model,
    "download_predictor.joblib"
)
```

Step 9: Production

Website
  ↓
User activity
  ↓
Backend
  ↓
Feature preparation
  ↓
ML model
  ↓
Prediction

---

35. Why the Pipeline Matters

Without a pipeline, you might write:

```
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

model.fit(X_train_scaled, y_train)
```

This works, but as the project grows, preprocessing can become complicated.

For example:

Missing values
  ↓

Scaling

Categorical encoding

Feature selection

Model

A pipeline makes the sequence explicit:

```
pipeline = Pipeline([
   ("preprocessing", preprocessing),
   ("model", model)
])
```

scikit-learn's pipeline API is specifically designed to chain transformers and a final estimator and allows the entire sequence to be fitted and evaluated together.

---

36. Common ML Workflow Mistakes

Mistake 1: Starting with the algorithm

Bad workflow:

```
"I want to use Random Forest."
      
Find a dataset
      
Find a problem
```

Better:

Problem

Data

Requirements

Model

Mistake 2: Training before understanding the data

Do not immediately write:

```
model.fit(X, y)
```

First understand:

What is X?
What is y?
Are values valid?
Are labels correct?

Mistake 3: Using the test set repeatedly

Bad:

Train

Test

Change model

Test again
↓
Change features
↓
Test again

The test set is gradually influencing development.

Use validation or cross-validation for experimentation.

Mistake 4: Scaling before splitting

Avoid:

```
scaler.fit_transform(X)
```

before splitting when the scaler learns statistics from the complete dataset.

Prefer fitting preprocessing on training data and applying it to unseen data, often through a pipeline.

Mistake 5: Focusing only on accuracy

A model can have high accuracy while performing badly on an important minority class.

Always ask:

```
What type of mistake matters?
```

Mistake 6: Ignoring the baseline

Without a baseline, you may not know whether your model actually adds value.

Mistake 7: Stopping after deployment

Production data changes.

```
Deployment □ Finish
```

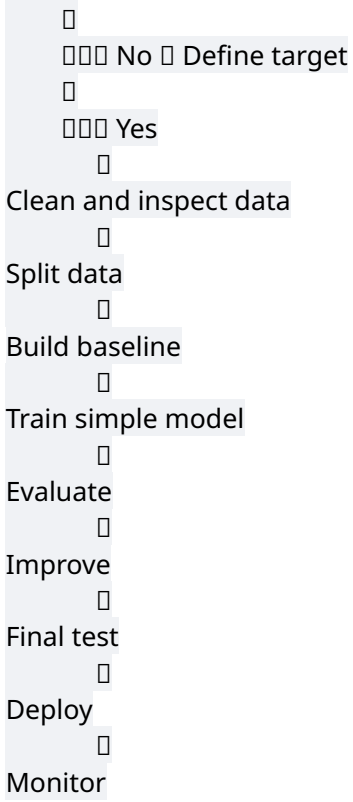
It is the beginning of the monitoring stage.

---

37. A Practical Decision Tree

When starting an ML project, ask:

```
Do I have a clearly defined problem?
    □
    □□□ No □ Define it first
    □
    □□□ Yes
        □
Do I have useful data?
    □
    □□□ No □ Collect data
    □
    □□□ Yes
        □
Is the target clearly defined?
```



---

38. ML Workflow vs ML Algorithm

These are not the same thing.

Algorithm

A mathematical method for learning patterns.

Examples:

```
Linear Regression
Decision Tree
Random Forest
Neural Network
```

Workflow

The entire process around the algorithm.

```
Problem
Data
Cleaning
Features
Splitting
Training
Evaluation
Deployment
Monitoring
```

Therefore:

```
Algorithm = one component

Workflow = complete system
```

---

39. ML Workflow vs Traditional Software Workflow

Traditional software:

```
Requirements
   
Code
```


Test

Deploy

Machine learning:

Problem

Data

Features

Model

Evaluation

Deploy

Monitor data + model

The major difference is that ML systems depend heavily on **data and learned behavior**.

---

40. The 5 Most Important Questions

Before training any ML model, ask:

1. What am I predicting?

Target

2. What information can the model use?

Features

3. How will I measure success?

Metric

4. How will I know the model generalizes?

Validation + Test

5. What happens after deployment?

Monitoring + Retraining

---

41. Five Minute Challenge

Imagine you are building a model that predicts whether a student will pass an exam.

Write down:

Problem:
?

Features:
?

Target:
?

ML task:
?

Metric:

?

Training data:
?

Testing data:
?

Possible model:
?

What could cause data leakage:
?

What should be monitored after deployment:
?

Do not write Python yet.

First design the workflow.

---

42. Exercises

Exercise 1: House Prices

Design an ML workflow that predicts house prices.

Identify:

Features
Target
ML task

Metric
Data source
Possible preprocessing
Possible model

Exercise 2: Spam Detection

Design a workflow that predicts whether an email is spam.

Identify:

Features
Target
ML task
Metric
Possible preprocessing
Possible errors

Exercise 3: Customer Churn

Design a workflow that predicts whether a customer will cancel their subscription.

Think about:

Target
Features
Class imbalance
Precision vs recall
Data leakage
Monitoring

---

43. Mini Project

Build a Customer Purchase Prediction System

Create a dataset containing:

```
age
visits
pages_viewed
session_minutes
previous_purchases
purchased
```

Stage 1

Load the data.

Stage 2

Inspect:

```
df.info()
df.describe()
df.isna().sum()
```

Stage 3

Clean the dataset.

Stage 4

Create:

```
X
y
```

Stage 5

Split the data.

Stage 6

Create a baseline.

Stage 7

Train at least two models.

Stage 8

Compare their performance.

Stage 9

Use cross-validation.

Stage 10

Tune the better candidate.

Stage 11

Evaluate once on the final test set.

Stage 12

Save the model.

Stage 13

Create a simple API that accepts customer information and returns a prediction.

Stage 14

Document the workflow.

The goal is not simply to achieve a high score.

The goal is to practice the **entire ML lifecycle**.

---

44. Interview Questions

Beginner

1. What is an ML workflow?
2. Why should you define the problem before choosing a model?
3. What are features and targets?
4. Why do we split datasets?
5. What is a baseline?
6. What is overfitting?
7. What is underfitting?
8. What is data leakage?

Intermediate

9. What is the difference between training, validation, and testing?
10. Why is cross-validation useful?
11. What is a hyperparameter?
12. Why should preprocessing be fitted only on training data?

13. Why are pipelines useful?
14. When is accuracy a poor metric?
15. What is data drift?
16. What happens after an ML model is deployed?

Advanced

17. How would you design an ML workflow for imbalanced classification?
18. How would you prevent test-set contamination?
19. How would you decide whether to retrain a production model?
20. How would you compare a simple model with a complex model?
21. How would you monitor an ML system in production?
22. How would you detect that the model is degrading?

---

45. Cheat Sheet

ML WORKFLOW

1. Problem
   ↓
2. Success metric
   ↓
3. Data collection
   ↓
4. Data understanding
   ↓
5. Data cleaning
   ↓
6. Features + target
   ↓
7. Train/validation/test
   ↓

8. Baseline

9. Preprocessing

10. Model training

11. Evaluation

12. Model comparison

13. Hyperparameter tuning

14. Final testing

15. Save model

16. Deploy

17. Monitor

18. Retrain

Core rule

Never trust a model simply because it performs well on training data.

Core mental model

Problem
 Data
 Features
 Model
 Evaluation
 Deployment

 Monitoring
 Improvement

---

46. Key Takeaways

- Machine learning is a **workflow**, not just model training.
- Start by defining the problem and success criteria.
- Understand and clean your data before training.
- Separate features from the target.
- Keep training and evaluation data separate.
- Build a simple baseline.
- Use preprocessing carefully.
- Avoid data leakage.
- Pipelines help combine preprocessing and modeling safely.
- Evaluate using metrics appropriate to the problem.
- Use validation or cross-validation for experimentation.
- Keep the final test set protected until final evaluation.
- Deployment is not the end of an ML project.
- Monitor production data and model performance.
- Retrain when the real-world problem or data changes.

The most useful way to think about machine learning is:

Don't ask:

"What model should I use?"

Ask:

"What is the problem,

what data do I have,
how will I measure success,
and how will this system behave in production?"

Visit haas.dev for more resources and guides.

Website:
https://dev-roast-app.vercel.app

Resources:
https://dev-roast-app.vercel.app/resources

Official references:
https://scikit-learn.org/stable/
https://developers.google.com/