Python for AI and Machine Learning

1. What Is Python for AI and Machine Learning?

Python is one of the main programming languages used to build AI and machine learning systems.

But AI and machine learning are not simply:

```
import sklearn
```

and training a model.

A real machine learning workflow combines:

```
Python
  
Data
  
Data Cleaning
  
Data Preparation
  
Features
  
Machine Learning Model
  
Training
  
Evaluation
  
Prediction
  
Deployment
```

Python provides the programming layer that connects these steps.

The ecosystem includes libraries such as:

- **NumPy** for numerical arrays and mathematical operations
- **pandas** for tabular data
- **Matplotlib** for visualization
- **scikit-learn** for traditional machine learning
- **PyTorch** for deep learning
- Other specialized libraries for NLP, computer vision, generative AI, and deployment

NumPy provides multidimensional arrays and numerical operations that form a major part of the scientific Python ecosystem.

scikit-learn provides tools for predictive data analysis, including classification, regression, clustering, preprocessing, model selection, and evaluation.

PyTorch provides a Python-based deep learning framework with tensors, datasets, neural networks, automatic differentiation, optimization, and model saving/loading workflows.

---

2. AI vs Machine Learning vs Deep Learning

These terms are related but not identical.

```
Artificial Intelligence
□
□□□ Machine Learning
□   □
□   □□□ Supervised Learning
□   □□□ Unsupervised Learning
□   □□□ Reinforcement Learning
```

□
□□□ Deep Learning
  □
  □□□ Neural Networks
  □□□ Computer Vision
  □□□ NLP
  □□□ Generative AI

Artificial Intelligence

The broader field of building systems that perform tasks associated with intelligent behavior.

Machine Learning

A subset of AI where models learn patterns from data.

Deep Learning

A subset of machine learning based primarily on neural networks with multiple layers.

---

3. Why Python Is Popular for AI

Python is useful because it combines:

Simple syntax

```
prediction = model.predict(data)
```

Numerical computing

```
import numpy as np
```

Data analysis

```
import pandas as pd
```

Machine learning

```
from sklearn.ensemble import RandomForestClassifier
```

Deep learning

```
import torch
```

Visualization

```
import matplotlib.pyplot as plt
```

The important advantage is the ecosystem.

You can move from:

```
Raw CSV
```

to:

```
pandas
```

to:

```
NumPy
```

to:

```
scikit-learn
```

or:

`PyTorch`

without leaving Python.

---

4. The Machine Learning Mental Model

Suppose you want to predict whether a customer will buy a product.

You might have:

```
age
previous_purchases
website_visits
average_order_value
```

and want to predict:

```
will_buy
```

The model receives examples:

```
Features                Label
-------------------------------------------------
25, 2, 5, 300            0
32, 8, 15, 1200            1
21, 1, 2, 150            0
40, 10, 20, 2000            1
```

The model learns patterns connecting the features to the target.

Then a new customer arrives:

```
30, 6, 12, 900
```

The trained model produces a prediction.

```
New Data
  ↓
Features
  ↓
Trained Model
  ↓
Prediction
```

---

5. AI Does Not Replace Programming

A common beginner misconception is:

> "Machine learning means I don't need normal programming."

You still need Python for:

- Loading data
- Cleaning data
- Validating input
- Creating features
- Controlling workflows
- Calling APIs
- Saving models
- Building applications
- Handling errors

- Logging
- Testing
- Deploying models

Machine learning is usually one component inside a larger software system.

---

6. The Python AI Ecosystem

A useful mental map is:

Task	Common Python Tool
Numerical computing	NumPy
Data analysis	pandas
Visualization	Matplotlib
Traditional ML	scikit-learn
Deep learning	PyTorch
Image processing	Pillow / OpenCV
NLP	Transformers / spaCy

Experiment notebooks	Jupyter
API serving	FastAPI
Model deployment	Cloud/container platforms

Do not try to learn all of these simultaneously.

Learn the core workflow first.

---

7. NumPy's Role

NumPy provides multidimensional arrays and mathematical operations that are fundamental to scientific Python computing.

Example:

```
import numpy as np

features = np.array([
    [25, 2, 5],
    [32, 8, 15],
    [21, 1, 2]
])

print(features.shape)
```

Output:

```
(3, 3)
```

This means:

```
3 rows
3 features
```

Machine learning libraries frequently work with numerical arrays like these.

---

8. pandas' Role

pandas is useful when your data looks like a spreadsheet.

```
import pandas as pd

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

print(df.head())
```

Example:

```
age  purchases  visits  spending
25   2          5       300
32   8          15      1200
21   1          2       150
```

You can:

- Clean data
- Handle missing values
- Filter rows

- Create columns
- Group data
- Convert types
- Prepare features

This is why data analysis and machine learning are closely connected.

---

9. Matplotlib's Role

Visualization helps you understand the data before training a model.

```
import matplotlib.pyplot as plt

df["spending"].hist()

plt.title("Customer Spending")
plt.xlabel("Spending")
plt.ylabel("Customers")
plt.show()
```

You might discover:

- Extreme values
- Skewed distributions
- Missing patterns
- Different groups
- Potential relationships

---

10. scikit-learn

scikit-learn is one of the major Python libraries for traditional machine learning.

It supports tasks including:

- Classification
- Regression
- Clustering
- Preprocessing
- Model selection
- Evaluation

Its estimators generally work with numerical feature representations, and pandas DataFrames can be used as inputs when they can be converted appropriately.

Install:

```
pip install scikit-learn
```

Current scikit-learn documentation lists the latest stable release as 1.9.1.

---

11. Your First Machine Learning Model

Let's build a simple classification model.

Suppose:

```
Age
Previous Purchases
Will Buy
```

Dataset:

```
import pandas as pd

df = pd.DataFrame({
    "age": [20, 25, 30, 35, 40, 45],
    "purchases": [1, 2, 3, 5, 7, 8],
    "will_buy": [0, 0, 1, 1, 1, 1]
})
```

---

12. Separate Features and Target

Features are the information the model uses.

```
X = df[
    ["age", "purchases"]
]
```

Target is what we want to predict:

```
y = df["will_buy"]
```

Think:

```
X = input

y = answer
```

---

13. Split the Data

Never evaluate a model only on the same data used to train it.

Use:

```
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
)
```

Conceptually:

```
Dataset
  □
  □□□ Training Data
  □
  □□□ Test Data
```

Training data teaches the model.

Test data checks how it performs on unseen examples.

---

14. Why the Train/Test Split Matters

Imagine you give a student the exact questions that will appear in the exam.

They might memorize the answers.

That does not prove understanding.

Similarly, a model can perform extremely well on data it has already seen while performing poorly on new data.

This is called:

Overfitting

---

15. Train a Model

For a simple classification example:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

model.fit(
    X_train,
    y_train
)
```

The model learns parameters from the training data.

---

16. Make Predictions

```
predictions = model.predict(X_test)

print(predictions)
```

The workflow is:

```
Training data
    
model.fit()
    
Trained model
    
model.predict()
    
Predictions
```

---

17. Evaluate the Model

Accuracy:

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(
    y_test,
    predictions
)

print(accuracy)
```

For classification, accuracy is:

```
Correct predictions
-------------------
Total predictions
```

But accuracy is not appropriate for every problem.

---

18. Classification

Classification predicts categories.

Examples:

```
Spam / Not Spam
Fraud / Not Fraud
Cat / Dog
Approved / Rejected
Disease / No Disease
```

Example:

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(
    random_state=42
)

model.fit(
    X_train,
    y_train
)
```

Predict:

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

---

19. Regression

Regression predicts a continuous numerical value.

Examples:

House price
Sales
Temperature
Revenue
Delivery time

Example:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(
    X_train,
    y_train
)
```

Prediction:

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

---

20. Classification vs Regression

Problem	Output

Spam detection	Category
Customer churn	Category
Image class	Category
House price	Number
Revenue prediction	Number
Temperature prediction	Number

Mental model:

Classification → Which category?

Regression → How much?

---

21. Unsupervised Learning

Sometimes there is no target column.

Suppose you have:

```
customer_id
spending
```

`orders`
`website_visits`

but no:

`customer_segment`

You can use clustering to discover groups.

Example:

```
from sklearn.cluster import KMeans

model = KMeans(
    n_clusters=3,
    random_state=42
)

clusters = model.fit_predict(
    X
)
```

Then:

```
df["cluster"] = clusters
```

Now the model has assigned customers to groups.

---

22. What Is a Feature?

A feature is an input variable used by the model.

For example:

```
age
income
number_of_orders
average_spending
```

can be features.

A feature matrix:

```
X = df[
    [
        "age",
        "number_of_orders",
        "average_spending"
    ]
]
```

---

23. What Is a Target?

The target is what the model is trying to predict.

Example:

```
y = df["churn"]
```

Think:

```
Features → Target
```

X → y

---

24. Feature Engineering

Raw data is not always the best representation for a model.

Suppose you have:

```
order_date
```

You might create:

```
df["order_month"] = (
    df["order_date"].dt.month
)
```

Or:

```
df["customer_age"] = (
    current_year -
    df["birth_year"]
)
```

Or:

```
df["average_order_value"] = (
    df["total_spending"] /
    df["number_of_orders"]
)
```

Creating useful input variables is called:

---

25. Categorical Data

Machine learning models usually require numerical representations of categorical variables.

Suppose:

```
city
-----
Lahore
Karachi
Islamabad
```

One common approach is one-hot encoding.

```
pd.get_dummies(
    df["city"]
)
```

You may get:

```
Lahore  Karachi  Islamabad
1       0        0
0       1        0
0       0        1
```

---

26. Scaling Features

Suppose you have:

```
age:      20–60
income:   20,000–500,000
```

The numerical ranges are very different.

Some algorithms benefit from feature scaling.

Using scikit-learn:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

The scaler learns statistics from the data and transforms values accordingly.

---

27. Avoid Data Leakage

Data leakage happens when information that should not be available to the model during prediction accidentally enters training.

Example:

You want to predict whether a customer will cancel.

But your feature dataset contains:

```
cancellation_date
```

That information only exists after cancellation.

The model may appear extremely accurate, but it has effectively been given the answer.

Mental model:

```
Information available at prediction time
                ↓
            Allowed

Information created after prediction time
                ↓
          Potential leakage
```

---

28. Pipelines

Instead of manually performing:

```
clean
↓
encode
↓
scale
↓
train
```

scikit-learn can combine preprocessing and modeling into a pipeline.

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

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

pipeline.fit(
    X_train,
    y_train
)
```

Then:

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

This makes the workflow more consistent.

---

29. Training vs Testing vs Validation

For larger projects, you may use three datasets:

```
Dataset
□
□□□ Training
□
□□□ Validation
□
□□□ Test
```

Training

Used to learn model parameters.

Validation

Used during development to compare models or tune settings.

Test

Used for final evaluation on unseen data.

Do not repeatedly tune your model against the test set.

---

30. Cross Validation

Instead of relying on one train/validation split, cross-validation evaluates a model across multiple partitions.

Example:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(
    model,
    X,
    y,
    cv=5
)

print(scores)
print(scores.mean())
```

Conceptually:

```
Fold 1 → validation
Fold 2 → validation
Fold 3 → validation
Fold 4 → validation
Fold 5 → validation
```

This gives a more robust estimate of model performance than a single split in many situations.

---

31. Model Evaluation

Different problems need different metrics.

Classification

Common metrics:

```
Accuracy
Precision
Recall
F1-score
ROC-AUC
```

Regression

Common metrics:

```
MAE
MSE
RMSE
R²
```

Never choose a metric simply because it gives a larger number.

Choose one that matches the problem.

---

32. Confusion Matrix

For classification:

```
from sklearn.metrics import confusion_matrix

matrix = confusion_matrix(
    y_test,
    predictions
)

print(matrix)
```

It helps distinguish:

```
True Positive
True Negative
False Positive
False Negative
```

This becomes especially important when the cost of different mistakes is not equal.

---

33. Precision and Recall

Suppose you are detecting fraud.

A model can make two types of important mistakes:

False positive

A legitimate transaction is classified as fraud.

False negative

A fraudulent transaction is classified as legitimate.

Precision and recall help you understand these different error types.

This is why:

```
Accuracy = 95%
```

does not tell the complete story.

---

34. Overfitting

Overfitting happens when a model learns the training data too closely and performs poorly on new data.

```
Training performance  Very high
Test performance      Poor
```

Possible causes:

- Model too complex
- Too little training data
- Noise
- Too many features
- Excessive tuning

---

35. Underfitting

Underfitting occurs when a model is too simple to capture important patterns.

```
Training performance  Poor
Test performance      Poor
```

Possible causes:

- Model too simple
- Weak features
- Insufficient training
- Excessive regularization

Mental model:

```
Underfitting  Good Generalization  Overfitting
```

---

36. Model Selection

Do not assume one algorithm is always best.

For classification, you might compare:

```
LogisticRegression
RandomForestClassifier
KNeighborsClassifier
SVC
```

For regression:

`LinearRegression`
`RandomForestRegressor`
`GradientBoostingRegressor`

The right choice depends on:

- Dataset
- Feature types
- Dataset size
- Interpretability
- Performance requirements
- Training cost
- Deployment requirements

---

37. Hyperparameters

A hyperparameter is a configuration chosen before or around training rather than learned directly from the training examples.

Example:

```
RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    random_state=42
)
```

Here:

`n_estimators`
`max_depth`

are hyperparameters.

---

38. Hyperparameter Tuning

scikit-learn provides tools such as `GridSearchCV`.

```
from sklearn.model_selection import GridSearchCV

params = {
    "n_estimators": [100, 200],
    "max_depth": [5, 10, None]
}

search = GridSearchCV(
    RandomForestClassifier(
        random_state=42
    ),
    params,
    cv=5,
    scoring="accuracy"
)

search.fit(
    X_train,
    y_train
)
```

Then:

```
print(search.best_params_)
```

Hyperparameter tuning should be performed without contaminating the final test evaluation.

---

39. Save a Trained Model

A trained model can be saved so you do not need to retrain it every time.

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

```
pip install joblib
```

Then:

```
import joblib

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

Load it:

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

Only load model files from trusted sources. Serialized model formats can have security implications.

---

40. Prediction Application

A trained model can become part of a Python application.

For example:

```
User
 
Web App
 
FastAPI
 
Preprocessing
 
ML Model
 
Prediction
 
JSON Response
```

Example:

```
prediction = model.predict(
    [[30, 6]]
)

print(prediction)
```

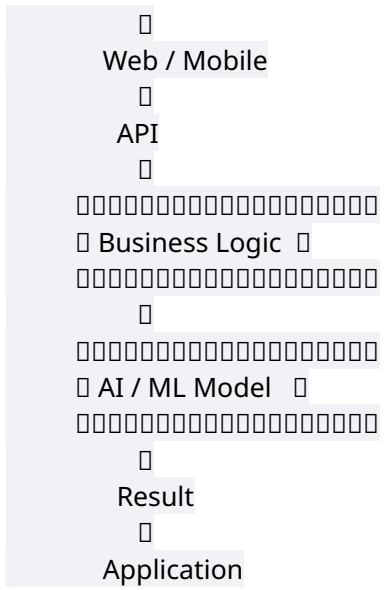
This is where machine learning becomes software engineering.

---

41. AI Application Architecture

A realistic system might look like:

```
            USER
```



The model is only one part of the application.

---

42. Deep Learning With PyTorch

For deep learning, PyTorch provides:

- Tensors
- Datasets
- DataLoaders
- Neural network modules
- Automatic differentiation
- Optimization
- Model saving/loading

Its official beginner workflow explicitly walks through tensors, datasets/data loaders, transforms, model construction, autograd, optimization, and saving/loading models.

Install:

```
pip install torch
```

---

43. PyTorch Tensors

A tensor is a multidimensional numerical data structure.

```
import torch

x = torch.tensor([
    [1, 2],
    [3, 4]
])

print(x)
```

You can inspect:

```
print(x.shape)
```

PyTorch's documentation describes it as an optimized tensor library for deep learning on CPUs and GPUs.

---

44. Tensor Operations

```
a = torch.tensor([1, 2, 3])
```

```
b = torch.tensor([4, 5, 6])

result = a + b

print(result)
```

Output:

```
tensor([5, 7, 9])
```

The syntax resembles NumPy because both libraries use array/tensor-oriented numerical operations.

---

45. A Simple Neural Network

```
import torch
from torch import nn

model = nn.Sequential(
    nn.Linear(4, 8),
    nn.ReLU(),
    nn.Linear(8, 1)
)

print(model)
```

A PyTorch model is commonly built from `torch.nn.Module` components, with learnable parameters registered as `Parameter` objects.

---

46. What Does a Neural Network Do?

Conceptually:

Input Features
  ↓
Linear Layer
  ↓
Activation
  ↓
Linear Layer
  ↓
Output

For example:

Age
Income
Orders
Visits
  ↓
Neural Network
  ↓
Probability / Prediction

---

47. Automatic Differentiation

Training neural networks requires gradients.

PyTorch's `autograd` automatically computes derivatives needed for gradient-based optimization.

Example:

```
x = torch.tensor(
  2.0,
```

```
    requires_grad=True
)

y = x ** 2

y.backward()

print(x.grad)
```

The gradient tells the optimizer how the model parameters should change to reduce the loss.

---

48. The Deep Learning Training Loop

A simplified training process is:

```
Input Data
    
Model
    
Prediction
    
Loss
    
Gradients
    
Optimizer
    
Updated Parameters
    
Repeat
```

In PyTorch:

```
for epoch in range(epochs):

    predictions = model(X)

    loss = loss_function(
        predictions,
        y
    )

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

This loop is one of the most important concepts in deep learning.

---

49. Loss Function

The model needs a way to measure how wrong its predictions are.

That measurement is the **loss**.

Conceptually:

```
Prediction
    +
Actual Answer
    
Loss
```

Low loss generally means the predictions are closer to the desired outputs according to that loss function.

Example:

```
loss_function = nn.MSELoss()
```

for a regression problem.

---

50. Optimizer

An optimizer updates model parameters based on gradients.

Example:

```
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=0.001
)
```

PyTorch's optimization workflow includes calculating gradients, updating parameters, and repeating this process during training.

---

51. Machine Learning vs Deep Learning

Traditional ML	Deep Learning
Often works well with structured/tabular data	Particularly powerful for complex unstructured data

Usually smaller models	Often larger neural networks
Feature engineering can be important	Representation learning can reduce manual feature engineering
scikit-learn commonly used	PyTorch commonly used
Often easier to interpret	Can be more complex to interpret

Neither approach is universally better.

Choose based on the problem.

---

52. AI With Text

Python can also process text.

Example:

```
text = "Python is useful for AI."

words = text.split()

print(words)
```

For modern NLP systems, specialized libraries can provide:

- Tokenization

- Embeddings
- Transformers
- Text classification
- Summarization
- Question answering
- Generative AI

A future AI curriculum can go much deeper into these topics.

---

53. AI With Images

Images can be loaded as arrays/tensors.

Conceptually:

```
Image
 
Pixels
 
Numerical Representation
 
Model
 
Prediction
```

A color image may be represented as:

```
height × width × channels
```

Deep learning frameworks can transform these numerical representations into model inputs.

---

54. Generative AI

Generative AI systems produce new content.

Examples include:

- Text
- Images
- Audio
- Code
- Video

Python is frequently used around these systems for:

- Calling model APIs
- Preparing prompts/data
- Building RAG systems
- Processing documents
- Running local models
- Building evaluation pipelines
- Serving AI applications

But using an AI API is different from training a machine learning model from scratch.

---

55. AI API vs Machine Learning Model

Calling an AI API

Your Python application sends:

```
Prompt / Input
     
External AI Service
     
Response
```

Example conceptually:

```
response = client.generate(
   prompt="Explain Python"
)
```

Training a model

Your system performs:

```
Dataset
 
Training
 
Parameters
 
Evaluation
 
Model
```

These are different workflows.

---

56. AI Project Lifecycle

A realistic AI project can follow:

1. Define Problem

2. Collect Data

3. Understand Data

4. Clean Data

5. Create Features

6. Split Data

7. Train Baseline

8. Evaluate

9. Improve

10. Test

11. Save Model

12. Deploy

13. Monitor

14. Retrain / Update

This is much more important than memorizing individual algorithms.

---

57. Baseline Models

Before building a complicated model, create a simple baseline.

For example:

```
Simple model
    
Measure performance
    
Complex model
    
Compare
```

If a complex model barely improves the baseline, the extra complexity may not be justified.

---

58. Reproducibility

Machine learning experiments should be reproducible.

Use fixed random seeds when appropriate:

```
train_test_split(
    X,
    y,
    random_state=42
)
```

For NumPy:

```
np.random.seed(42)
```

For PyTorch:

```
torch.manual_seed(42)
```

Reproducibility does not guarantee identical results across every environment and hardware configuration, but controlled randomness makes experiments easier to compare.

---

59. Project Structure for an ML Application

A more maintainable project might look like:

```
ml_project/
│
├── data/
│   ├── raw/
│   └── processed/
│
├── notebooks/
│
├── src/
│   ├── data.py
│   ├── features.py
│   ├── train.py
│   ├── evaluate.py
│   └── predict.py
│
├── models/
│
├── tests/
│
├── requirements.txt
│
└── README.md
```

This separates:

Data
Features
Training
Evaluation
Prediction
Tests
Models

---

60. Example ML Project

Build a **Customer Purchase Prediction System**.

Goal:

> Predict whether a customer is likely to make a purchase.

Dataset:

customer_id
age
website_visits
previous_orders
average_spending
discount_used
will_buy

---

61. Load the Dataset

```
import pandas as pd

df = pd.read_csv(
    "data/customers.csv"
)
```

Inspect:

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

---

62. Clean the Data

```
df = df.drop_duplicates()

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

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

df = df.dropna()
```

Then validate:

```
assert df["age"].between(
    0,
    120
).all()
```

---

63. Select Features

```
features = [
    "age",
    "website_visits",
    "previous_orders",
    "average_spending",
    "discount_used"
]

X = df[features]
y = df["will_buy"]
```

---

64. Split the Data

```
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, stratification can help preserve class proportions across splits when appropriate.

---

65. Train the Model

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(
    n_estimators=200,
    random_state=42
)

model.fit(
    X_train,
    y_train
)
```

---

66. Evaluate

```
from sklearn.metrics import (
    accuracy_score,
    classification_report
)

predictions = model.predict(
    X_test
)

print(
    accuracy_score(
        y_test,
        predictions
    )
)
```

```
print(
    classification_report(
        y_test,
        predictions
    )
)
```

---

67. Predict a New Customer

```
new_customer = [[
    30,
    12,
    5,
    850,
    1
]]

prediction = model.predict(
    new_customer
)

print(prediction)
```

The model produces the predicted class.

---

68. Save the Model

```
import joblib
```

```
joblib.dump(
    model,
    "models/customer_purchase_model.joblib"
)
```

Later:

```
model = joblib.load(
    "models/customer_purchase_model.joblib"
)
```

---

69. Turn the Model Into an API

Once the model works, FastAPI can expose it to other applications.

Conceptually:

```
Mobile/Web App
      
POST /predict
      
FastAPI
      
Load Model
      
Validate Input
      
Predict
      
JSON Response
```

Example:

```
from fastapi import FastAPI

app = FastAPI()

@app.post("/predict")
def predict(data: dict):

    result = model.predict([[
        data["age"],
        data["website_visits"],
        data["previous_orders"],
        data["average_spending"],
        data["discount_used"]
    ]])

    return {
        "prediction": int(result[0])
    }
```

This connects the machine learning model to normal software engineering.

---

70. AI Project Testing

Test the surrounding application, not only the model.

Examples:

```
Valid input
Invalid input
Missing feature
Wrong data type
Extreme value
```

```
Empty request
Model unavailable
```

You can test the prediction function:

```
def predict_customer(data):
    return model.predict([[
        data["age"],
        data["website_visits"],
        data["previous_orders"],
        data["average_spending"],
        data["discount_used"]
    ]])
```

Then write tests around expected behavior.

---

71. Model Monitoring

Deployment is not the end.

After deployment, monitor:

```
Prediction volume
Prediction errors
Input distributions
Latency
Failures
Model performance
Data drift
```

A model trained on historical data may become less useful when real-world patterns change.

---

72. Data Drift

Suppose your model was trained when customers typically spent:

```
$100–$500
```

Later, the business changes and customers typically spend:

```
$1,000–$5,000
```

The model's input distribution has changed.

This can affect performance.

Mental model:

```
Training Data
    
Model
    
Real World Changes
    
Input Distribution Changes
    
Model May Degrade
```

---

73. Model Drift

Model performance can also change over time.

For example:

January → 91%
February → 90%
March → 87%
April → 80%

A production ML system therefore needs monitoring and potentially retraining.

---

74. AI Engineering vs Data Science

These roles overlap but are not identical.

Data Scientist

Often focuses on:

- Data analysis
- Experimentation
- Statistical reasoning
- Modeling
- Insights

Machine Learning Engineer

Often focuses on:

- Training pipelines
- Model systems
- Deployment
- Infrastructure

- Monitoring
- Production reliability

AI Engineer

Often works on:

- AI applications
- Model/API integration
- LLM systems
- RAG
- Agents
- Evaluation
- Production AI workflows

Python can be central to all three.

---

75. Common Beginner Mistakes

Mistake 1: Starting with deep learning immediately

Learn:

Python
↓
NumPy
↓
pandas
↓
Data Cleaning
↓

Statistics
□
Machine Learning
□
Deep Learning

Mistake 2: Training without understanding the data

A model cannot compensate for fundamentally bad input data.

Mistake 3: Measuring only accuracy

Choose metrics according to the actual problem.

Mistake 4: Testing on training data

This can give an overly optimistic performance estimate.

Mistake 5: Data leakage

Never allow future or target-derived information into features.

Mistake 6: Using complex models unnecessarily

Start with a baseline.

Mistake 7: Treating model output as automatically correct

A model produces predictions, not guaranteed truth.

Mistake 8: Ignoring deployment

A model that works in a notebook is not automatically a production system.

---

76. Best Practices

1. Start with the problem

Define what you are trying to predict or discover.

2. Understand the data

Use pandas and visualization before modeling.

3. Build a baseline

Measure something simple first.

4. Separate training and evaluation data

Do not evaluate using training examples alone.

5. Prevent leakage

Only use information legitimately available at prediction time.

6. Use reproducible experiments

Track datasets, parameters, code, and results.

7. Evaluate with appropriate metrics

Different problems require different measurements.

8. Keep preprocessing consistent

The same transformations used during training should be applied during inference.

9. Test the application

ML code still needs normal software engineering practices.

10. Monitor production behavior

Real-world data changes.

---

77. What You Should Learn Next

After understanding Python for AI and machine learning, a sensible progression is:

Python

NumPy

pandas

Data Cleaning

-
Data Analysis
-
Statistics for ML
-
Machine Learning Fundamentals
-
scikit-learn
-
Regression
-
Classification
-
Clustering
-
Feature Engineering
-
Model Evaluation
-
Hyperparameter Tuning
-
ML Pipelines
-
Deep Learning
-
PyTorch
-
Computer Vision / NLP
-
Generative AI
-
AI Application Development
-
Deployment and MLOps

This topic should therefore be treated as a **bridge into the AI/ML track**, not as a replacement for the deeper topics that follow.

---

78. Mini Project

Customer Purchase Prediction

Build an end-to-end machine learning project.

Requirements

Create or obtain a dataset containing:

```
age
income
website_visits
previous_orders
average_order_value
discount_used
will_buy
```

Your program should:

1. Load the dataset.
2. Inspect it.
3. Clean it.
4. Handle missing values.
5. Remove duplicates.
6. Validate values.
7. Create useful features.
8. Split the data.
9. Train at least two models.

10. Compare their evaluation metrics.
11. Select a model based on documented criteria.
12. Save the selected model.
13. Create a prediction function.
14. Expose the prediction through an API.
15. Write tests.
16. Document the project.

---

79. 5 Minute Challenge

Create this dataset:

```
import pandas as pd

df = pd.DataFrame({
    "age": [20, 25, 30, 35, 40, 45],
    "orders": [1, 2, 3, 5, 7, 8],
    "spending": [100, 250, 400, 700, 1200, 1500],
    "will_buy": [0, 0, 1, 1, 1, 1]
})
```

Your challenge:

1. Select the features.
2. Select the target.
3. Split the dataset.
4. Train a classification model.
5. Make predictions.
6. Calculate accuracy.

Then try a second model and compare the results.

---

80. Exercises

Exercise 1

Create a DataFrame containing:

```
age
income
orders
```

Create a target:

```
will_buy
```

Separate `X` and `y`.

Exercise 2

Create a train/test split and explain why the split is necessary.

Exercise 3

Train both:

```
LogisticRegression
RandomForestClassifier
```

Compare their results.

Exercise 4

Create a regression dataset where the target is:

```
house_price
```

Train:

```
LinearRegression()
```

Exercise 5

Create customer groups using:

```
KMeans
```

Experiment with:

```
n_clusters = 2
n_clusters = 3
n_clusters = 4
```

Exercise 6

Create a preprocessing pipeline containing:

```
StandardScaler
+
LogisticRegression
```

Exercise 7

Save a trained model using `joblib`.

Load it again and make a prediction.

Exercise 8

Create a FastAPI endpoint that accepts customer information and returns a prediction.

---

81. Mini Quiz

Question 1

What is a feature?

A. The model's final answer
B. An input variable used by the model
C. A Python package
D. A database

Answer: B

Question 2

What is the target?

A. The value the model tries to predict
B. The training dataset
C. The Python interpreter
D. The feature scaler

Answer: A

Question 3

Which library is commonly used for traditional machine learning in Python?

A. Flask
B. pandas
C. scikit-learn
D. pathlib

Answer: C

Question 4

What is overfitting?

A. Model cannot learn anything
B. Model learns training data too closely and generalizes poorly
C. Dataset contains no columns
D. Python code has a syntax error

Answer: B

Question 5

What is data leakage?

A. Losing a CSV file
B. Accidentally giving the model information it should not have during training
C. Deleting duplicate rows
D. Scaling features

Answer: B

Question 6

What is PyTorch mainly used for?

A. File management
B. Deep learning and tensor-based computation
C. HTML parsing only
D. Git management

Answer: B

---

82. Interview Questions

Beginner

1. Why is Python popular for AI and machine learning?

Because it combines accessible syntax with a large ecosystem for numerical computing, data analysis, machine learning, deep learning, visualization, and application development.

2. What is the difference between AI and machine learning?

AI is the broader field. Machine learning is an approach within AI where systems learn patterns from data.

3. What is the difference between a feature and a target?

Features are inputs used for prediction. The target is the output the model is trying to predict.

4. What is classification?

Predicting a category.

5. What is regression?

Predicting a numerical value.

Intermediate

6. Why do we split data into training and testing sets?

To evaluate how well a model generalizes to unseen data.

7. What is overfitting?

When a model performs very well on training data but poorly on unseen data.

8. What is feature engineering?

Creating or transforming input variables into representations that are more useful for a machine learning model.

9. What is data leakage?

When information unavailable at prediction time enters the training process and gives the model unfair information.

10. Why use pipelines?

To combine preprocessing and modeling steps into a consistent, reusable workflow.

Advanced

11. Why isn't accuracy always enough?

Because different errors can have different costs, and class imbalance can make accuracy misleading.

12. What is cross-validation?

A method that evaluates a model across multiple train/validation partitions.

13. What is a hyperparameter?

A model configuration chosen outside the learned parameters, such as tree depth or learning rate.

14. Why should a model be monitored after deployment?

Because real-world data, behavior, and performance can change over time.

15. What is the difference between machine learning and deep learning?

Deep learning is a subset of machine learning based primarily on neural networks with multiple layers.

---

83. Python AI/ML Cheat Sheet

NumPy

```
import numpy as np

x = np.array([1, 2, 3])

x.mean()
x.max()
x.min()
x.shape
```

pandas

```
import pandas as pd

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

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

Split Data

```
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
)
```

Classification

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

model.fit(
    X_train,
    y_train
)

predictions = model.predict(
    X_test
)
```

Regression

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(
    X_train,
    y_train
```

```
)
```

Evaluation

```
from sklearn.metrics import accuracy_score

accuracy_score(
    y_test,
    predictions
)
```

Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

Pipeline

```
from sklearn.pipeline import Pipeline

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

PyTorch

```
import torch
from torch import nn

model = nn.Sequential(
    nn.Linear(4, 8),
```

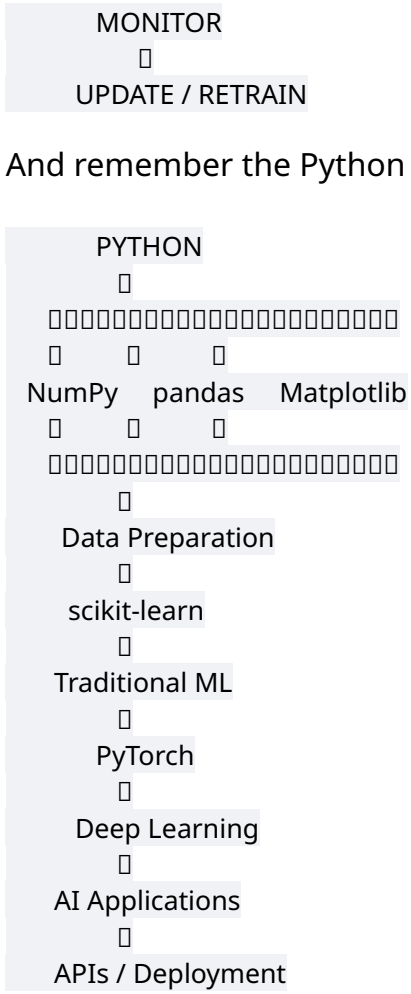
```
    nn.ReLU(),
    nn.Linear(8, 1)
)
```

---

84. The Complete Mental Model

Remember the entire AI/ML workflow as:

```
                PROBLEM
                   
                DATA
                   
             CLEAN DATA
                   
           EXPLORE + VISUALIZE
                   
           CREATE FEATURES
                   
            SPLIT DATA
                   
           TRAIN BASELINE
                   
             EVALUATE
                   
         IMPROVE / TUNE
                   
          FINAL EVALUATION
                   
            SAVE MODEL
                   
             DEPLOY
                   
```



And remember the Python ecosystem:

---

85. Key Takeaways

1. Python is the programming foundation of a large part of the AI/ML ecosystem.
2. NumPy handles numerical arrays and scientific computation.

3. pandas handles structured data preparation and analysis.
4. scikit-learn provides a broad toolkit for traditional machine learning.
5. PyTorch provides a major framework for deep learning.
6. Machine learning requires much more than choosing an algorithm.
7. Data quality directly affects model quality.
8. Features are inputs; the target is what you predict.
9. Training data and evaluation data must be separated.
10. Overfitting means poor generalization despite strong training performance.
11. Data leakage can make model evaluation misleading.
12. Different problems require different evaluation metrics.
13. Feature engineering can strongly affect model performance.
14. Pipelines help keep preprocessing and modeling consistent.
15. A trained model is only one component of an AI application.
16. Production AI requires testing, deployment, monitoring, and maintenance.
17. Deep learning introduces concepts such as tensors, neural networks, gradients, loss, and optimization.
18. The best AI projects connect machine learning with normal software engineering.

Official References

Python documentation:
https://docs.python.org/3/

NumPy documentation:
https://numpy.org/doc/stable/

pandas documentation:
https://pandas.pydata.org/docs/

scikit-learn documentation:
https://scikit-learn.org/stable/

PyTorch documentation:
https://docs.pytorch.org/docs/main/

PyTorch beginner tutorials:
https://docs.pytorch.org/tutorials/beginner/basics/

The official PyTorch beginner sequence currently covers tensors, datasets/data loaders, transforms, model building, automatic differentiation, optimization, and saving/loading models.

Visit haas.dev for more resources and guides.

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

Resources:

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