## Overview (26 weeks)

- Months 1–2: foundations — Python, math, data wrangling, classical ML
- Months 3–4: deep learning fundamentals, CNNs, RNNs, transformers
- Month 5: specialization, production and MLOps basics
- Month 6: capstone project, portfolio, interview prep

**Week 1 — Python & Environment (Expanded Syllabus)**
Goal for Week 1:
You should be fully comfortable writing basic Python programs, manipulating lists and dictionaries, defining functions, understanding simple OOP, and using Jupyter notebooks. You should also have your full ML environment ready.

---

**1. Core Python Topics (Deep List)**
**A. Python Basics (Day 1–2)**

- Variables, types (int, float, string, bool)
- Type casting and type checking
- Input/output, printing formats
- Arithmetic and logical operations
- Conditional statements (if/elif/else)
- Loops (for, while)
- Range(), enumerate(), zip()
- Error handling basics (try/except)

**Mini exercises:**

- Write a script that prints first 20 even numbers.
- Write a function that counts vowels in a string.
- Write a program that checks if a number is prime.

---

**B. Data Structures (Day 2–3)**
Deep understanding of:
**Lists**

- Creating, indexing, slicing
- Appending, inserting, extending
- List comprehension (simple, nested, with conditions)

**Tuples**

- Immutability
- Tuple unpacking

**Dictionaries**

- Keys, values, items
- Adding, updating, deleting entries
- Dictionary comprehension
- Nested dictionaries

**Sets**

- Union, intersection, difference
- Removing duplicates

**Mini exercises:**

- Convert a list of numbers into a list of squared values using list comprehension.
- Count frequency of each character in a string using a dictionary.
- Remove duplicates from a list using a set.

---

**C. Functions & Modules (Day 3–4)**

- Defining functions
- Parameters vs arguments
- Default parameters
- Variable-length arguments (*args, **kwargs)
- Returning multiple values
- Anonymous functions (lambda)
- Writing and importing your own module

**Mini exercises:**

- Write a function that returns mean, median and mode from a list.
- Write a lambda function to sort a list of dictionaries by a key.

---

**D. Object-Oriented Basics (Day 4)**
Just the essentials needed for ML later:

- What classes and objects are
- Attributes and methods
- Constructor (**init**)
- Encapsulation idea
- Creating simple reusable classes
- Basic inheritance (optional this week)

**Mini exercises:**

- Create a Point class with x and y attributes and a distance method.
- Create a simple Calculator class with add/subtract/multiply/divide.

---

**E. Working with Files (Day 5)**

- Opening/reading/writing files
- CSV basics (reading line by line)
- Understanding relative vs absolute paths
- Using Python's os module

**Mini exercises:**

- Write a script that reads a text file, counts words and lines.
- Write a script that reads a CSV and prints the first 5 rows manually.

---

**2. Environment & Tools Setup (Day 1–2)**
Make sure your machine is ready for ML work.
**A. Python Environment Setup**

- Install Python 3.8+
- Install VS Code or PyCharm
- Install Jupyter Notebook or JupyterLab
- Install Anaconda or Miniconda (recommended)
- Create a virtual environment

conda create -n ml python=3.10
conda activate ml

---

**B. Essential Libraries**
Install these:
pip install numpy pandas scikit-learn matplotlib seaborn
pip install jupyterlab
Optional but useful:
pip install ipykernel

---

**C. Git & GitHub Setup**

- Install Git
- Configure name & email
- Create GitHub account
- Learn basic commands:
  - git init
  - git add
  - git commit
  - git push

**Practice:**
Upload your Week 1 exercises to GitHub.

---

**3. Jupyter Notebook Skills (Day 3–4)**
Learn to use Jupyter efficiently:

- Markdown cells
- Running cells
- Using keyboard shortcuts

- Importing libraries
- Inline vs external plots
- Creating clean, readable notebooks

**Practice tasks:**

- Create a notebook with sections using Markdown.
- Load a CSV using pandas and display head/tail.
- Plot a line graph and histogram.

---

**4. Hands-On Mini Projects (Day 5–6)**

These are simple but give you real workflow experience.

**A. Project 1: Explore a CSV**

Pick any dataset (Iris, Titanic, or your own).

Tasks:

1. Load CSV using pandas
2. Check shape, column names
3. Compute mean, median, min, max for numeric columns
4. Handle missing values (drop/replace)
5. Plot:
    - Histogram of one column
    - Scatter plot of two numeric columns
    - Boxplot

Outcome: you learn data loading and basic analysis.

---

**B. Project 2: Build a Simple Script**

Write a Python script that:

- Accepts a number from the user
- Checks if the number is odd/even
- Saves the output to a text file

This teaches input, branching, writing files.

---

**C. Project 3: Basic Class Implementation**

Create a Student class:

- Attributes: name, marks (list)
- Method: average marks
- Method: grade (based on average)

This helps solidify OOP.

---

**5. Week 1 Progress Checklist**

By the end of Week 1, you should be able to:

**Python Skills**

- Write clean Python scripts
- Use lists, dicts, sets, and list comprehensions
- Define and call functions
- Write simple classes
- Read and write files
- Use modules and imports

**Environment Skills**

- Create and use a virtual environment
- Install packages
- Work inside Jupyter notebooks
- Use Git basics
- Push code to GitHub

**Data Handling Skills**

- Load and explore a CSV
- Compute simple statistics
- Create basic plots

**Week 2 — Linear Algebra & Calculus Essentials (Expanded Syllabus)**

Goal for Week 2:

Understand the math behind ML models and be able to implement basic linear algebra operations and gradients using NumPy. You should finish the week with strong intuition for vectors, matrices, derivatives and how they transform data in ML.

---

**1. Linear Algebra for Machine Learning (Days 1–3)**

**A. Scalars, Vectors, Matrices, Tensors**

Understand what each represents in ML:

- **Scalar:** single number (learning rate, loss value)
- **Vector:** 1D array (features of a single sample)
- **Matrix:** 2D array (dataset, weight matrices)
- **Tensor:** N-dimensional arrays (images, deep learning weights)

**Practice:**

```
import numpy as np
a = np.array([1, 2, 3])      # vector
b = np.array([[1, 2], [3, 4]]) # matrix
```

---

**B. Vector Operations**

Learn:

- Vector addition, subtraction
- Scalar multiplication
- Dot product (very important)
- L2 norm, normalization

**Dot product intuition:**

Similarity between vectors.

**Practice:**

- Compute dot product manually and using np.dot.
- Normalize a vector to unit length.

---

**C. Matrix Operations**

Topics to cover:

- Matrix addition, subtraction
- Scalar multiplication
- Matrix multiplication rules
- Identity matrix
- Determinant (intuitive meaning: scaling factor of transformation)
- Inverse (when it exists)
- Transpose

**Why ML needs this:**

- Weight matrices transform inputs.
- Transpose helps align shapes.
- Inverse appears in closed-form solutions like linear regression.

**Practice:**

```
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])

A @ B
A.T
np.linalg.det(A)
np.linalg.inv(A)
```

---

**D. Eigenvalues and Eigenvectors (High-Level Intuition Only)**

Key ideas:

- Eigenvector: direction that doesn't change after transformation
- Eigenvalue: scale factor along that direction
- PCA uses eigenvectors of the covariance matrix to find principal components

Don't memorize formulas. Focus on **intuition**.

**Practice:**

```
w, v = np.linalg.eig(A)
```

Interpret what the values mean.

---

**E. Special ML-Useful Concepts**

1. **Rank:** dimension of vector space; relates to information content
2. **Orthogonality:** vectors perpendicular (zero dot-product)
3. **Span:** set of all possible combinations
4. **Projection:** used in linear regression (geometric view)

---

**2. Calculus for Machine Learning (Days 3–5)**

**A. Functions and Limits**

- Review basic function concepts
- Understand smooth vs non-smooth functions
- Know meaning of derivative as a rate of change

Only the intuition matters.

---

**B. Derivatives**

Learn how to differentiate:

- Polynomial functions
- Exponential functions
- Logarithmic functions
- Simple trigonometric functions (rarely used in ML but good for basics)

**Important ML derivative shapes:**

- $d/dx(x^2) = 2x$
- $d/dx(e^x) = e^x$
- $d/dx(\ln(x)) = 1/x$

---

**C. Partial Derivatives**

Essential because ML models use many variables.

Understand:

- Holding other variables constant
- Gradient = vector of partial derivatives

**Example:**

For $f(x, y) = x^2 + 3y$,

$df/dx = 2x$

$df/dy = 3$

**Practice:**

Compute partial derivatives manually for small functions.

---

**D. Chain Rule Intuition**

This is the heart of backpropagation.

Chain rule:

If $y = f(g(x))$, then $dy/dx = f'(g(x)) * g'(x)$

In neural networks:

- Layers are functions inside functions
- Chain rule passes gradients backward

Avoid formulas; build **intuitive understanding**.

---

**E. Common ML Derivatives**

Learn gradients of:

- Linear function $f(x)=Wx + b$
- Sigmoid
- ReLU
- Softmax (just intuition this week)
- Mean Squared Error

These will help you later in deep learning.

---

**3. Connecting Linear Algebra + Calculus to ML (Day 5–6)**

**A. Linear Transformation in ML**

Understand that:

- Input vector x
- Weight matrix W
- Output = W·x changes direction and magnitude

Visualize 2D transformations if possible.

---

**B. Gradient of a Linear Function**

Example function:

$f(x) = Wx$

The gradient wrt x is W^T.

Do not memorize; understand why:

- Each output depends linearly
- Transpose aligns dimensions

---

**C. Gradient Descent Intuition**

Learn:

- Loss function
- Slope direction
- Learning rate
- Updating parameters:
  **$\theta = \theta - \alpha * \text{gradient}$**

---

**D. Implementing a Simple Gradient in NumPy**

**Example:**

For function

$f(x) = x^2$

Gradient:

$f'(x) = 2x$

**Practice:**

```
x = np.array([1.0, 2.0, 3.0])
grad = 2 * x
```

---

**E. Implementing Gradient Descent on a Linear Function**

Goal:

Fit $y = mx + c$ manually using gradient descent.

Steps:

1. Initialize m, c
2. Predict y_pred = m*x + c
3. Compute loss = mean squared error
4. Compute gradients
5. Update parameters
6. Loop

This builds intuition for training ML models.

---

**4. NumPy Practice Projects (Day 6–7)**

**Project 1: Implement Matrix Multiplication Manually**

- Use nested loops
- Compare with np.dot
- Understand performance benefits of NumPy

---

**Project 2: Create a Linear Transformation Visualizer**

- Generate random 2D points
- Apply matrix transformation
- Plot before vs after (if your system supports matplotlib)

---

**Project 3: Simple Gradient Descent on a Linear Function**

- Use synthetic data
- Train m, c
- Watch loss decrease over iterations

---

**Project 4: Compute Eigenvalues and Use Them for PCA**

- Create a covariance matrix
- Extract eigenvalues/eigenvectors
- Identify principal direction

---

**5. Week 2 Progress Checklist**

By the end of Week 2, you should be able to:

**Linear Algebra**

- Add, subtract, multiply vectors/matrices
- Compute dot products
- Understand concept of matrix rank
- Compute transpose, determinant, inverse
- Explain eigenvalues/eigenvectors intuitively

**Calculus**

- Compute basic derivatives
- Compute partial derivatives
- Understand gradient as a vector
- Use the chain rule intuitively

**Applied**

- Implement a linear function using NumPy
- Compute its gradient
- Run a simple gradient descent loop

**Week 3 — Probability & Statistics Essentials (Expanded Syllabus)**

Goal for Week 3:
Build strong intuition in probability, random variables, distributions, descriptive statistics and ML-relevant concepts like expectation, variance, covariance, correlation, sampling, and basic hypothesis testing. You'll apply everything using NumPy and pandas.

---

**1. Probability Basics (Days 1–2)**
**A. Core Probability Concepts**
Understand:

- Sample space
- Events
- Outcomes
- Independent vs dependent events
- Mutually exclusive events
- Complement of an event

Examples:

- Coin flips
- Dice rolls
- Drawing cards

**Practice:**

- Compute probability of getting at least one head in two tosses.
- Compute probability of rolling > 4 on a die.

---

**B. Conditional Probability**
Understand:

- P(A | B) = Probability of A given B
- Joint probability
- Marginal probability
- When events are independent

**Real ML intuition:**
Spam detection, medical tests, weather forecasting.
**Practice:**

- Compute conditional probabilities from small tables.
- Use pandas crosstabs to estimate conditional probabilities from data.

---

**C. Bayes' Rule (Simple Intuition Only)**
Bayes updates beliefs with new evidence.
Formula:
P(A | B) = [ P(B | A) * P(A) ] / P(B)
Use cases:

- Naive Bayes classifier
- Diagnostic tests
- Recommendations

**Practice:**

- Build a simple Bayes prior–posterior example with disease test accuracy.

---

**2. Random Variables & Distributions (Days 2–3)**
**A. Random Variables**

- Discrete vs continuous
- Probability mass function (PMF)
- Probability density function (PDF)
- Cumulative distribution function (CDF)

---

**B. Important Distributions**
You need intuition for these, not formulas.
**1. Bernoulli & Binomial**
Used for:

- Binary classification
- Success/failure modeling

**2. Normal (Gaussian)**
Used for:

- Measurement noise
- Many natural phenomena
- ML assumptions (regression residuals)

**3. Uniform**
Used for:

- Random initialization
- Sampling

**4. Poisson**
Used for:

- Rare events
- Counts (clicks, failures)

**5. Exponential**
Used for:

- Time between events

**Practice:**
Using NumPy:
np.random.normal(mean, std, size)
np.random.binomial(n, p, size)
np.random.exponential(scale, size)
Plot histograms and compare shapes.

---

**3. Descriptive Statistics (Day 3)**
**A. Measures of Central Tendency**

- Mean
- Median
- Mode

Understand when each is appropriate:

- Mean sensitive to outliers
- Median robust
- Mode useful for categorical data

---

**B. Measures of Spread**

- Variance
- Standard deviation
- Range
- Interquartile range

Use ML intuition:

- High variance = unstable data
- Outliers influence certain models (like linear regression)

---

**C. Distribution Shape**

- Skewness
- Kurtosis
- Heavy-tailed data intuition (finance, rainfall, traffic)

---

**D. Practice**
Using pandas:
df.describe()
df.skew()
df.kurt()

---

**4. Covariance, Correlation & Relationships (Day 4)**
**A. Covariance**
Tells whether variables move together:

- Positive: move in same direction
- Negative: opposite
- Zero: unrelated

Covariance matrix is key in PCA.

---

**B. Correlation**
Scaled version of covariance.
Pearson correlation:

- Between -1 and +1
- Strong linear relationship indicator

Real ML intuition:

- Feature engineering
- Reducing multicollinearity
- Understanding dataset structure

**Practice:**
correlation matrix from pandas:
df.corr()

---

**5. Sampling, Estimation & The Law of Large Numbers (Day 4)**
**A. Sampling Methods**

- Random sampling
- Stratified sampling
- Bootstrapping (important for ensemble learning)

**B. Estimators**

- Sample mean
- Sample variance
- Unbiased estimators

**C. Central Limit Theorem (High-level intuition)**
Averages of many samples approximate a normal distribution.
This explains:

- Why Gaussian assumptions often work
- Why sample means stabilize

---

**6. Hypothesis Testing Basics (Day 5–6)**
**A. Null and Alternative Hypotheses**
Think:

- Null (H0): nothing changed
- Alternate (H1): something changed

Example:

- Does a new fertilizer increase yield?

---

**B. p-values**
Intuition, not strict definition:

- Small p-value → data unlikely under the null → reject H0

---

**C. Common Tests (Only Intuition This Week)**

- t-test
- chi-square test
- ANOVA

You don't need to compute formulas — just the idea.

---

**D. Confidence Intervals**
Interpretation:

- Interval that likely covers the true value
- Common in regression output

**Practice:**
Run simple t-test using SciPy (optional):
from scipy.stats import ttest_ind

---

**7. Practical NumPy & Pandas Mini Projects (Day 6–7)**
**Project 1 — Simulate and Plot Distributions**

- Generate normal, binomial and exponential datasets
- Plot histograms
- Compare mean and variance

---

**Project 2 — Correlation Analysis**

- Load a real dataset (Iris, rainfall, stock prices, etc.)
- Compute correlation matrix
- Identify highly correlated features
- Interpret relationships

---

**Project 3 — Monte Carlo Simulation**
Examples:

- Estimate probability of rolling > 4 on a die
- Estimate π using random sampling
- Estimate expected rainfall event probability

---

**Project 4 — Simple Hypothesis Test**
Scenario:

- Two sets of crop yields
- Are they significantly different?
- Use t-test or simple confidence intervals

This builds intuition for hypothesis testing.

---

**8. Week 3 Progress Checklist**
By end of Week 3, you should be able to:
**Probability**

- Compute basic and conditional probabilities
- Explain Bayes' rule intuitively
- Understand random variables and distributions

**Statistics**

- Compute mean, median, variance, skewness
- Build and interpret correlation matrices
- Understand covariance and PCA intuition
- Interpret p-values and confidence intervals

**Applied**

- Simulate data using NumPy
- Analyze real datasets using pandas
- Carry out basic statistical tests
- Explain randomness and uncertainty clearly

If all this is comfortable, you're ready for Week 4 (Data Handling + EDA).
**Week 4 — Data Handling & Exploratory Data Analysis (Expanded Syllabus)**
**Goal for Week 4:**
By the end of this week, you should be able to clean, transform, and merge datasets; handle missing values; perform aggregations; create visualizations; detect outliers; and summarize your findings in a short, structured EDA report. You should feel confident exploring any tabular dataset.

---

**1. Data Handling with Pandas (Days 1–3)**
**A. Loading Data**

- Read CSV, Excel, JSON, and text files: pd.read_csv, pd.read_excel, pd.read_json
- Understand index_col, header, dtype parameters
- Check basic info:

df.head()
df.tail()
df.info()
df.describe()
df.shape
df.columns
**Practice:** Load the Titanic or Iris dataset and inspect it.

---

**B. Selecting & Filtering Data**

- Column selection: df['col'] or df[['col1','col2']]
- Row selection: df.iloc[], df.loc[]
- Conditional filtering:

df[df['Age'] > 30]
df[(df['Sex']=='male') & (df['Pclass']==1)]

- Boolean indexing and query:

df.query("Age > 30 & Sex=='male'")

---

**C. Handling Missing Values**

- Detect missing: df.isnull(), df.isnull().sum()
- Remove missing: df.dropna()
- Fill missing: df.fillna(value) or df.fillna(df.mean())
- Interpolation for time series

**Practice:** Handle missing age values in Titanic dataset using mean/median.

---

**D. Renaming, Dropping, and Type Conversion**

- Rename columns: df.rename(columns={'old':'new'})
- Drop columns/rows: df.drop(columns=['col'])
- Convert data types: df['col'].astype(int)
- Strip whitespace, lowercase column names

**E. Aggregation & Grouping**

- groupby basics: sum, mean, count, max, min

df.groupby('Pclass')['Fare'].mean()

- Aggregation with multiple functions:

df.groupby('Pclass').agg({'Fare':['mean','max'], 'Age':'median'})

- Pivot tables:

df.pivot_table(index='Sex', columns='Pclass', values='Fare', aggfunc='mean')

---

**F. Merging and Joining**

- Concatenate rows/columns: pd.concat([df1, df2], axis=0/1)
- Merge datasets (inner, outer, left, right):

pd.merge(df1, df2, on='ID', how='inner')

- Join on index: df1.join(df2, how='outer')

**Practice:** Merge two small datasets (e.g., Iris data with a label mapping).

---

**2. Exploratory Data Analysis (EDA) (Days 3–5)**

**A. Descriptive Statistics**

- df.describe(), df.mean(), df.median(), df.mode()
- Value counts for categorical variables: df['col'].value_counts()
- Quantiles, min/max, std: df.quantile([0.25,0.5,0.75])

---

**B. Detecting Outliers**

- Visual methods: boxplots, scatter plots
- Numeric methods: IQR rule (1.5*IQR beyond Q1/Q3)

**Practice:** Detect outliers in Fare or Age columns in Titanic dataset.

---

**C. Feature Distributions**

- Histograms: df['col'].hist() or plt.hist()
- Density plots: sns.kdeplot()
- Bar plots for categorical: df['col'].value_counts().plot(kind='bar')
- Count plots for categorical: sns.countplot(x='Sex', data=df)

---

**D. Pairwise Relationships**

- Scatter plots: plt.scatter(df['x'], df['y'])
- Pair plots for multiple features: sns.pairplot(df)
- Correlation heatmap:

import seaborn as sns
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

- Helps identify correlated features or potential multicollinearity.

---

**E. Feature Engineering Basics (Optional Intro)**

- Create new features (e.g., Age groups, Fare bins)
- Convert categorical to numeric using pd.get_dummies() or LabelEncoder

---

**3. Visualization Libraries**

- Matplotlib basics: plt.plot(), plt.scatter(), plt.bar(), plt.hist()
- Seaborn for statistical plots: sns.boxplot(), sns.violinplot(), sns.heatmap()
- Customize plots: labels, titles, legends, colors, figure size

**Practice:** Create at least 5 different plots for the dataset you are exploring.

---

**4. Hands-On Mini Projects (Day 5–7)**

**Project 1 — Titanic Dataset EDA**

1. Load dataset
2. Inspect columns, data types, missing values
3. Clean missing values for Age, Embarked
4. Explore distribution of Age, Fare, Pclass, Sex
5. Identify outliers in Fare
6. Visualize correlations using heatmap
7. Summarize findings in a markdown cell

---

**Project 2 — Iris Dataset EDA**

1. Load dataset
2. Compute mean, median, variance for each feature
3. Visualize feature distributions using histograms and boxplots
4. Plot pairwise scatter plots to observe separation by species

---

**Project 3 — Custom Dataset (Optional)**

- Download a Kaggle dataset (e.g., COVID cases, housing prices, rainfall data)
- Apply full cleaning: missing values, renaming, type conversion
- Merge/aggregate if multiple files
- Perform EDA and generate 5–6 visualizations
- Write short conclusions in a notebook

---

**5. Week 4 Progress Checklist**

By the end of Week 4, you should be able to:

**Data Handling**

- Load, inspect, clean datasets
- Handle missing values effectively
- Merge, join, and aggregate data
- Rename columns, convert types, drop unnecessary columns

**EDA Skills**

- Compute descriptive statistics for numerical and categorical features
- Detect outliers and understand their impact
- Visualize distributions, correlations, and pairwise relationships
- Generate insights and summarize observations in notebooks

**Applied**

- Complete at least one mini-project with a clean, documented Jupyter notebook
- Include 5+ plots and markdown cells summarizing findings
- Be confident to explore **any new tabular dataset** before modeling

**Week 5 — Supervised Learning Fundamentals (Expanded Syllabus)**

**Goal for Week 5:**

By the end of the week, you should understand the difference between regression and classification, know common loss functions, understand training and evaluation protocols, implement a linear regression model both from scratch and using scikit-learn, and understand regularization techniques.

---

**1. Introduction to Supervised Learning (Day 1–2)**

**A. Regression vs Classification**

- **Regression:** Predict continuous output (e.g., house price, temperature)
- **Classification:** Predict categorical output (e.g., spam/not spam, disease/no disease)

**Examples:**

| Task | Type | Notes |
|---|---|---|
| Predict salary based on experience | Regression | Continuous target |
| Predict whether a patient has diabetes | Classification | Binary target |
| Predict handwritten digit | Classification | Multiclass target |

**Practice:** Identify the type of supervised problem for a set of example datasets.

---

**B. Loss Functions**

- Quantify how well the model predicts.

**Regression Loss**

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Huber Loss (optional)

**Classification Loss**

- 0/1 Loss (simple)
- Cross-Entropy / Log Loss (used in logistic regression, neural networks)

**Practice:** Compute MSE and MAE manually for a small dataset.

---

**C. Training, Validation, and Test Splits**

- **Train set:** Used to train the model
- **Validation set:** Used for tuning hyperparameters and early stopping
- **Test set:** Used for final evaluation

**Practice:** Split a dataset using train_test_split from 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)
```

---

**D. Cross-Validation**

- **K-Fold Cross-Validation:** Split data into K folds, train K times, average performance
- Reduces variance of model evaluation
- Useful for small datasets

**Practice:** Use cross_val_score from scikit-learn:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
```

---

**2. Linear Regression (Day 2–4)**

**A. Model Intuition**

- Predict target yyy as a weighted sum of inputs:

y=w1x1+w2x2+⋯+by = w_1 x_1 + w_2 x_2 + \dots + by=w1x1+w2x2+⋯+b

- Goal: Minimize difference between predicted and true values (MSE)

---

**B. Gradient Descent**

- Iterative optimization algorithm
- Update rule for weights www:

w=w−α⋅∂L∂ww = w - \alpha \cdot \frac{\partial L}{\partial w}w=w−α⋅∂w∂L

where α\alphaα is the learning rate and LLL is the loss function (MSE)

**Practice:** Implement gradient descent for a single variable linear regression in NumPy:

```
# Pseudocode
for epoch in range(epochs):
    y_pred = w*x + b
    dw = (-2/n) * sum(x*(y - y_pred))
    db = (-2/n) * sum(y - y_pred)
    w = w - alpha * dw
    b = b - alpha * db
```

---

**C. Regularization**

- Helps prevent overfitting
- **Ridge (L2):** Penalizes large weights λ∑w2\lambda \sum w^2λ∑w2
- **Lasso (L1):** Penalizes absolute weights λ∑∣w∣\lambda \sum |w|λ∑∣w∣

**Practice:** Visualize effect of regularization by comparing models with different alpha values on the same dataset.

---

**3. Hands-On Linear Regression (Day 4–6)**

**A. Using scikit-learn**

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

- Evaluate with MSE, RMSE, $R^2$ score:

```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

**Practice:**

- Train linear regression on Boston Housing dataset or synthetic data
- Compare performance with Ridge and Lasso

---

**B. Implement Linear Regression from Scratch**

- Use NumPy
- Implement:
  - Forward pass (prediction)
  - Loss computation (MSE)
  - Gradient computation
  - Gradient descent update

**Optional Extensions:**

- Add L1 or L2 regularization manually
- Implement mini-batch gradient descent

---

**C. Visualization**

- Plot predicted vs true values
- Plot loss curve over iterations

```
import matplotlib.pyplot as plt
plt.plot(loss_history)
plt.xlabel("Epochs")
plt.ylabel("MSE Loss")
plt.show()
```

---

**4. Mini Projects / Practice Ideas**

**Project 1 — Predict House Prices**

1. Load Boston Housing or any open regression dataset
2. Perform train-test split
3. Train linear regression model
4. Evaluate and plot predicted vs actual
5. Apply Ridge and Lasso and compare performance

---

**Project 2 — Implement Linear Regression from Scratch**

1. Generate synthetic linear data
2. Implement forward pass and gradient descent in NumPy
3. Plot convergence of loss
4. Compare final weights with scikit-learn implementation

---

**Project 3 — Cross-Validation Exploration**

1. Use K-fold CV on regression dataset
2. Compute mean MSE across folds
3. Observe effect of train-test splits and overfitting

---

**5. Week 5 Progress Checklist**

By the end of Week 5, you should be able to:

**Conceptual**

- Explain regression vs classification
- Understand loss functions and when to use each
- Explain train/validation/test split and cross-validation
- Understand overfitting and regularization techniques

**Applied**

- Implement linear regression using scikit-learn
- Implement linear regression from scratch using NumPy
- Evaluate models using MSE, RMSE, $R^2$
- Apply Ridge and Lasso and interpret effects
- Visualize predictions and loss curves

**Week 6 — Classification Algorithms (Expanded Syllabus)**
**Goal for Week 6:**
By the end of this week, you should understand the main classification algorithms, be able to train them on real datasets, evaluate them with multiple metrics, and compare their strengths and weaknesses.

---

**1. Introduction to Classification (Day 1)**
**A. Concept Recap**

- Classification vs regression: output is categorical instead of continuous
- Binary vs multiclass classification
- Examples:
  - Binary: Spam detection, cancer detection
  - Multiclass: Digit recognition, iris species prediction

**B. Pipeline Overview**

- Feature selection/engineering
- Train-test split
- Model fitting
- Prediction
- Evaluation with metrics

**Practice:** Identify whether given datasets are binary, multiclass, or regression.

---

**2. Classification Algorithms (Day 1–3)**
**A. Logistic Regression**

- Extends linear regression to classification
- Uses **sigmoid function** to map predictions to probabilities
- Loss function: **Binary Cross-Entropy**
- Decision boundary: threshold at 0.5

**Practice:**

- Train logistic regression on a binary dataset (e.g., Titanic survived/not survived)
- Predict probabilities and convert to class labels

---

**B. K-Nearest Neighbors (KNN)**

- Non-parametric, instance-based learning
- Predict class based on majority vote among K nearest neighbors
- Distance metrics: Euclidean, Manhattan
- Hyperparameter K tuning important

**Practice:**

- Implement KNN using sklearn.neighbors.KNeighborsClassifier
- Try different K values and observe effect on accuracy

---

**C. Decision Trees**

- Tree structure with nodes as features and leaves as class labels
- Splitting based on **Gini Impurity** or **Entropy (Information Gain)**
- Can overfit if tree is deep → prune or limit depth

**Practice:**

- Train decision tree classifier on iris dataset
- Visualize tree using plot_tree
- Observe which features are used for splitting

---

**D. Naive Bayes**

- Probabilistic classifier based on **Bayes theorem**
- Assumes features are conditionally independent (naive assumption)
- Variants: Gaussian, Multinomial, Bernoulli
- Fast and effective for text data

**Practice:**

- Train Gaussian Naive Bayes on iris dataset
- Observe predicted probabilities for classes

---

**3. Classification Metrics (Day 3–4)**
**A. Confusion Matrix**

- True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN)

**Practice:**

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

---

**B. Accuracy**

- Fraction of correctly predicted samples
- Not reliable for imbalanced datasets

---

**C. Precision, Recall, F1-Score**

- **Precision:** TP / (TP + FP) → how many predicted positive are actually positive
- **Recall (Sensitivity):** TP / (TP + FN) → how many actual positives were predicted correctly
- **F1-Score:** Harmonic mean of precision & recall

**Practice:**

```
from sklearn.metrics import precision_score, recall_score, f1_score
```

---

**D. ROC Curve & AUC**

- ROC curve: TPR vs FPR at different thresholds
- AUC: area under ROC curve → 1 is perfect, 0.5 is random

**Practice:**

```
from sklearn.metrics import roc_curve, roc_auc_score
```

---

**4. Hands-On Mini Projects (Day 4–6)**
**Project 1 — Binary Classification**

- Dataset: Titanic survival
- Steps:
  1. Load data and clean missing values
  2. Select features
  3. Train logistic regression, KNN, decision tree, naive Bayes
  4. Predict on test set
  5. Compute metrics: accuracy, precision, recall, F1, confusion matrix
  6. Compare model performance

---

**Project 2 — Multiclass Classification**

- Dataset: Iris
- Steps:
  1. Train decision tree, KNN, and naive Bayes
  2. Evaluate using accuracy and macro/micro averaged F1-score
  3. Visualize confusion matrix

---

**Project 3 — Threshold Tuning and ROC-AUC**

- For logistic regression:
  1. Predict probabilities instead of classes
  2. Plot ROC curve
  3. Choose optimal threshold for classification
  4. Evaluate impact on precision, recall, F1

---

**5. Comparing Models**

- Observe:
  - Which model performs better on small data
  - Which model handles nonlinear boundaries
  - Which is faster and simpler
  - Sensitivity to hyperparameters

**Practice:** Document observations in a notebook with tables/plots.

---

**6. Week 6 Progress Checklist**
By the end of Week 6, you should be able to:

**Conceptual**

- Explain differences between logistic regression, KNN, decision tree, and naive Bayes
- Understand pros, cons, and assumptions of each model
- Explain key classification metrics and when to use them

**Applied**

- Train multiple classifiers on the same dataset
- Evaluate using accuracy, precision, recall, F1-score, and ROC-AUC
- Compare models systematically
- Visualize predictions, confusion matrices, and ROC curves

**Week 7 — Ensemble Methods (Expanded Syllabus)**
**Goal for Week 7:**
By the end of this week, you should understand how ensemble methods work, be able to implement bagging and boosting techniques, and use tree-based ensemble libraries like Random Forests, XGBoost, and LightGBM. You should also be able to create a simple Kaggle-style notebook with model comparisons and a leaderboard-style evaluation.

---

**1. Introduction to Ensemble Methods (Day 1)**
**A. Why Ensembles?**

- Combine multiple weak models to create a stronger model
- Reduce variance (overfitting), bias, or improve predictions
- Real-world intuition: "wisdom of the crowd"

**Types of Ensembles:**

1. **Bagging** (Bootstrap Aggregating) – reduces variance
2. **Boosting** – reduces bias
3. **Stacking** (optional advanced) – combine different model types

---

**2. Bagging (Day 1–2)**
**A. Bagging Concept**

- Generate multiple datasets by sampling with replacement (bootstrap)
- Train a separate model on each dataset
- Aggregate predictions (average for regression, majority vote for classification)

**B. Random Forests**

- Bagging applied to decision trees
- Random feature selection at each split
- Reduces overfitting and improves generalization

**Parameters to know:**

- n_estimators – number of trees
- max_depth – maximum depth of each tree
- max_features – number of features to consider at each split

**Practice:**

```
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
rf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

- Evaluate with accuracy, F1, or RMSE
- Visualize feature importance: rf.feature_importances_

---

**3. Boosting (Day 2–4)**
**A. Boosting Concept**

- Sequentially train weak learners
- Each new model focuses on correcting errors of the previous model
- Combines models with weighted votes or sums

---

**B. AdaBoost**

- Adaptive Boosting
- Uses decision stumps (very shallow trees) as weak learners
- Weights misclassified samples more heavily for the next learner

**Practice:**

```
from sklearn.ensemble import AdaBoostClassifier
ab = AdaBoostClassifier(n_estimators=50, learning_rate=1.0, random_state=42)
ab.fit(X_train, y_train)
```

---

**C. Gradient Boosting**

- Generalizes boosting using gradient descent on loss function
- Sequentially adds models that reduce residual errors
- Flexible: works for regression and classification

**Practice:**

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
gb.fit(X_train, y_train)
```

---

**D. Popular Boosting Libraries**

- **XGBoost:** optimized, fast, widely used for Kaggle competitions
- **LightGBM:** faster for large datasets, supports categorical features natively

**Practice:**

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
xgb_model.fit(X_train, y_train)
import lightgbm as lgb
lgb_model = lgb.LGBMClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
lgb_model.fit(X_train, y_train)
```

---

**4. Evaluation for Ensembles (Day 4)**

- Use same metrics as previous weeks:
  - Accuracy, Precision, Recall, F1, ROC-AUC for classification
  - MSE, RMSE, $R^2$ for regression
- Compare multiple models in a table (leaderboard style)

**Practice:** Create a table like:

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC |
|---|---|---|---|---|---|
| Random Forest | 0.87 | 0.85 | 0.88 | 0.86 | 0.92 |
| AdaBoost | 0.85 | 0.84 | 0.85 | 0.84 | 0.90 |
| Gradient Boost | 0.88 | 0.87 | 0.88 | 0.87 | 0.93 |
| XGBoost | 0.89 | 0.88 | 0.89 | 0.88 | 0.94 |
| LightGBM | 0.89 | 0.88 | 0.89 | 0.88 | 0.94 |

---

**5. Hands-On Mini Projects (Day 5–7)**
**Project 1 — Titanic Survival Prediction (Binary Classification)**

1. Load dataset, clean missing values
2. Train Random Forest, AdaBoost, Gradient Boosting
3. Compare metrics (accuracy, F1, ROC-AUC)
4. Rank models like a Kaggle leaderboard
5. Visualize feature importance

---

**Project 2 — Regression Task with Boosting**

1. Dataset: Boston Housing or synthetic regression dataset
2. Train Gradient Boosting Regressor, XGBoost Regressor, LightGBM Regressor
3. Compare RMSE, $R^2$

4. Plot predicted vs actual values

---

**Project 3 — Kaggle Notebook Style**

- Combine all steps:
  - Load and clean dataset
  - Feature engineering if needed
  - Train multiple ensemble models
  - Evaluate using leaderboard table
  - Visualize feature importance and predictions
  - Write markdown explanations for model choices

**Optional Challenge:**

- Tune hyperparameters using GridSearchCV or RandomizedSearchCV

---

**6. Week 7 Progress Checklist**

By the end of Week 7, you should be able to:

**Conceptual**

- Explain bagging, boosting, and their differences
- Describe Random Forests, AdaBoost, Gradient Boosting
- Understand the intuition behind XGBoost and LightGBM
- Know strengths and limitations of tree-based ensembles

**Applied**

- Train Random Forest, AdaBoost, Gradient Boosting, XGBoost, LightGBM
- Evaluate and compare models using classification or regression metrics
- Visualize feature importance and predictions
- Create a mini Kaggle-style notebook with leaderboard and markdown explanations

**Week 8 — Unsupervised Learning & Model Selection (Expanded Syllabus)**

**Goal for Week 8:**

By the end of this week, you should understand clustering algorithms, dimensionality reduction via PCA, hyperparameter tuning methods, and be able to apply these techniques to datasets. You should also prepare a small end-to-end ML notebook (classification or regression) on GitHub as the Month 2 deliverable.

---

**1. Introduction to Unsupervised Learning (Day 1)**

**A. Concept Recap**

- No labeled target variable
- Goal: find hidden patterns, structure, or groups in data
- Applications:
  - Customer segmentation
  - Image compression / visualization
  - Anomaly detection

**Practice:** Identify whether example datasets are supervised or unsupervised.

---

**2. Clustering (Day 1–3)**

**A. K-Means Clustering**

- Partition data into K clusters
- Each point assigned to nearest cluster center (centroid)
- Iterative process:
  1. Initialize centroids
  2. Assign points to nearest centroid
  3. Update centroids
  4. Repeat until convergence

**Parameters to know:**

- n_clusters – number of clusters
- init – method for centroid initialization (k-means++, random)
- max_iter – maximum iterations

**Practice:**

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

- Visualize clusters with scatter plots (2D/3D if possible)
- Use elbow method to select optimal K

---

**B. Hierarchical Clustering**

- Builds tree (dendrogram) of nested clusters
- Agglomerative (bottom-up) vs divisive (top-down)
- Linkage criteria: single, complete, average, ward

**Practice:**

```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
Z = linkage(X, method='ward')
dendrogram(Z)
clusters = fcluster(Z, t=3, criterion='maxclust')
```

- Visualize dendrogram to understand cluster hierarchy

---

**3. Dimensionality Reduction (Day 3–4)**

**A. Principal Component Analysis (PCA)**

- Transform data to fewer dimensions while retaining maximum variance
- Components are orthogonal (uncorrelated)
- Useful for:
  - Visualization (2D/3D plots)
  - Reducing noise
  - Preprocessing before clustering

**Practice:**

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```

- Plot 2D/3D PCA results and color points by cluster labels

---

**B. Explained Variance**

- How much information is retained in reduced dimensions

```
pca.explained_variance_ratio_
```

- Choose number of components that explain ~90–95% variance

---

**4. Model Selection & Hyperparameter Tuning (Day 4–6)**

**A. Why Model Selection Matters**

- Different algorithms / hyperparameters produce different results
- Goal: choose best model without overfitting

---

**B. Hyperparameter Tuning**

- **Grid Search:** try all possible combinations of parameters
- **Randomized Search:** try random combinations (faster for large grids)

**Practice:**

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {'n_estimators':[50,100,200], 'max_depth':[3,5,10]}
grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
best_model = grid.best_estimator_
```

- Evaluate performance on validation set

---

**C. Cross-Validation Reminder**

- K-Fold CV ensures model generalizes well
- Combine with hyperparameter tuning to avoid overfitting

---

**5. Hands-On Mini Projects (Day 5–6)**

**Project 1 — Clustering + PCA**

1. Choose dataset (Iris, MNIST subset, or synthetic data)
2. Scale features using StandardScaler

3. Apply PCA to reduce dimensions to 2 or 3
4. Apply K-Means and/or hierarchical clustering
5. Visualize clusters in 2D/3D PCA space
6. Optionally: calculate silhouette score to evaluate clustering quality

---

**Project 2 — Hyperparameter Tuning for Supervised Model**

1. Train a Random Forest or Gradient Boosting model
2. Use Grid Search or Randomized Search to find optimal n_estimators, max_depth, learning_rate
3. Evaluate final model on test set
4. Document results in a small notebook

---

**6. Month 2 Deliverable — Mini End-to-End Project**
**Deliverable:**

- **Notebook:** Classification or Regression task (can use Titanic, Boston Housing, Iris, or Kaggle dataset)
- Include:
  - Data loading & cleaning
  - Exploratory Data Analysis (plots, outliers, feature distributions)
  - Train-test split
  - Model training (one supervised model, optionally ensemble)
  - Evaluation metrics (accuracy, F1, RMSE, etc.)
  - Hyperparameter tuning using GridSearchCV or RandomizedSearchCV
  - Visualizations for predictions, PCA, or clusters if applicable
- **GitHub Submission:**
  - Push notebook to GitHub
  - Include **README.md** with description, dataset source, steps, and observations

---

**7. Week 8 Progress Checklist**
By the end of Week 8, you should be able to:
**Conceptual**

- Explain K-Means and hierarchical clustering
- Understand PCA and dimensionality reduction intuition
- Understand hyperparameter tuning strategies (grid/random search)
- Know how to select the best model using validation

**Applied**

- Apply PCA to reduce data dimensions
- Perform clustering and visualize clusters
- Tune hyperparameters for a supervised model
- Prepare an end-to-end ML notebook with proper documentation on GitHub

**Week 9 — Neural Networks Basics (Expanded Syllabus)**
**Goal for Week 9:**
By the end of this week, you should understand the basic building blocks of neural networks, know how forward and backward propagation works, be familiar with common activation functions, and implement a simple feedforward neural network using a high-level framework like TensorFlow/Keras or PyTorch.

---

**1. Introduction to Neural Networks (Day 1)**
**A. Motivation**

- Inspired by the human brain
- Solve problems where linear models fail (e.g., image classification, nonlinear regression)

**B. Perceptron**

- Basic unit of a neural network
- Input → weighted sum → activation function → output
- Formula:

y=f(∑wixi+b)y = f(\sum w_i x_i + b)y=f(∑wixi+b)

- Can solve **linearly separable** problems

**Practice:**

- Implement a single-layer perceptron for AND, OR logic gates using NumPy

---

**2. Activation Functions (Day 1–2)**
**A. Common Activation Functions**

1. **Sigmoid:** f(x)=11+e−xf(x) = \frac{1}{1+e^{-x}}f(x)=1+e−x1
   - Output in [0,1], used for binary classification
2. **Tanh:** f(x)=tanh⁡(x)f(x) = \tanh(x)f(x)=tanh(x)
   - Output in [-1,1], zero-centered
3. **ReLU (Rectified Linear Unit):** f(x)=max⁡(0,x)f(x) = \max(0, x)f(x)=max(0,x)
   - Solves vanishing gradient problem, widely used in hidden layers
4. **Softmax:** Converts vector of logits into probabilities
   - Used in multiclass classification

**Practice:** Plot each function and its derivative to understand behavior.

---

**3. Feedforward Neural Networks (Day 2–3)**
**A. Structure**

- **Input layer:** receives features
- **Hidden layers:** learn intermediate representations
- **Output layer:** produces prediction
- **Weights and biases:** parameters learned during training

---

**B. Forward Propagation**

- Compute weighted sum → apply activation → pass to next layer
- Formula for layer lll:

al=f(Wlal−1+bl)a^l = f(W^l a^{l-1} + b^l)al=f(Wlal−1+bl)

---

**C. Loss Functions**

- Quantify error between predicted and true output

**Regression:**

- Mean Squared Error (MSE)

**Classification:**

- Binary Cross-Entropy (log loss)
- Categorical Cross-Entropy

**Practice:** Compute MSE and cross-entropy manually for small examples.

---

**4. Backpropagation Intuition (Day 3–4)**
**A. Concept**

- Use chain rule to compute gradients of loss w.r.t weights
- Update weights using gradient descent:

w=w−α∂L∂ww = w - \alpha \frac{\partial L}{\partial w}w=w−α∂w∂L

- Each layer adjusts weights to reduce loss

**B. Key Points**

- Forward pass: compute output
- Backward pass: compute gradients and update weights
- Learning rate α\alphaα controls step size

**Practice:** Use a diagram to trace gradients in a 2-layer network

---

**5. Implementing a Simple Feedforward Network (Day 4–6)**
**A. Using High-Level Frameworks**

- **Keras (TensorFlow)**: easy to build and train networks

Example: Binary classification network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(8, input_dim=4, activation='relu'),
    Dense(4, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=50, batch_size=8, validation_split=0.2)
```

- Evaluate performance on test set:

```
loss, accuracy = model.evaluate(X_test, y_test)
```

---

**B. Using PyTorch (Optional)**

- Build custom forward pass and training loop using nn.Module

```
import torch
import torch.nn as nn
import torch.optim as optim

class SimpleNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(4, 8)
        self.fc2 = nn.Linear(8, 4)
        self.fc3 = nn.Linear(4, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x

model = SimpleNN()
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

- Training loop: forward → loss → backward → optimizer step

---

**6. Hands-On Mini Projects**

**Project 1 — Perceptron Logic Gates**

- Implement AND, OR, XOR using single-layer and multi-layer perceptrons
- Visualize decision boundaries

**Project 2 — Binary Classification**

- Dataset: Titanic survival, Pima Indians diabetes, or synthetic 2D data
- Build 2–3 layer feedforward network
- Train and evaluate accuracy, plot loss curve
- Visualize predictions vs true labels

**Project 3 — Multiclass Classification**

- Dataset: Iris or MNIST subset
- Use softmax activation in output layer
- Evaluate using accuracy and confusion matrix

---

**7. Week 9 Progress Checklist**

By the end of Week 9, you should be able to:

**Conceptual**

- Explain perceptron and feedforward network structure
- Understand activation functions and their role
- Describe forward and backward propagation
- Understand loss functions for regression and classification

**Applied**

- Implement a simple feedforward neural network using Keras or PyTorch
- Train on small datasets and evaluate performance
- Plot training/validation loss and accuracy curves
- Visualize predictions in 2D when possible

**Week 10 — Practical Deep Learning Workflows (Expanded Syllabus)**

**Goal for Week 10:**

By the end of this week, you should understand how to train neural networks efficiently, implement batching, use different optimizers, tune learning rates, and apply regularization techniques. You should be able to build a small MLP for MNIST or a tabular dataset and evaluate its performance.

---

**1. Training Loops & Batching (Day 1–2)**

**A. Mini-Batch Gradient Descent**

- Full-batch vs stochastic vs mini-batch
- Benefits of mini-batch:
  - Faster convergence
  - Better generalization
- Common batch sizes: 16, 32, 64, 128

**Practice:** Split dataset into mini-batches manually (NumPy) or use DataLoader (PyTorch) / batch_size in Keras

---

**B. Epochs and Iterations**

- **Epoch:** one pass through the entire dataset
- **Iteration:** one update per batch
- Observe loss and accuracy over epochs

**Practice:** Plot training and validation loss curves over epochs

---

**2. Optimizers (Day 2–3)**

**A. Gradient Descent Variants**

1. **SGD (Stochastic Gradient Descent)**
   - Basic optimizer
   - Can oscillate, may converge slowly
2. **Momentum**
   - Accelerates updates in relevant direction
3. **Adam**
   - Combines momentum and adaptive learning rate
   - Most commonly used

**Practice:**

- Train same small MLP with SGD vs Adam and compare loss curves

---

**B. Learning Rate**

- Controls step size in gradient descent
- Too high → may diverge, too low → slow training
- Learning rate schedules:
  - Step decay
  - Exponential decay
  - Cosine annealing

**Practice:** Experiment with different learning rates (0.1, 0.01, 0.001)

---

**3. Regularization Techniques (Day 3–4)**

**A. Dropout**

- Randomly drops neurons during training
- Reduces overfitting

```
from tensorflow.keras.layers import Dropout
model.add(Dropout(0.3))
```

**B. Batch Normalization**

- Normalizes layer inputs
- Helps faster convergence

```
from tensorflow.keras.layers import BatchNormalization
model.add(BatchNormalization())
```

**C. L1/L2 Regularization**

- Penalizes large weights
- Helps prevent overfitting

**Practice:** Add dropout and batch norm to MLP and observe effect on training vs validation loss

---

**4. Implementing a Small MLP (Day 4–6)**

**A. Using Keras (TensorFlow)**

**Example: MNIST**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
```

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, batch_size=32, epochs=20,
validation_split=0.2)
```

- Evaluate on test set:

```
loss, accuracy = model.evaluate(X_test, y_test)
```

---

**B. Using PyTorch (Optional)**

- Build nn.Module with dropout and batch norm
- Use torch.utils.data.DataLoader for batching
- Training loop: forward → loss → backward → optimizer step

---

**5. Hands-On Mini Projects**

**Project 1 — MLP on MNIST**

1. Flatten images to 784 features
2. Build 2–3 layer MLP
3. Use batch normalization and dropout
4. Experiment with Adam and SGD optimizers
5. Plot training/validation accuracy and loss curves
6. Visualize some correctly and incorrectly predicted digits

**Project 2 — Tabular Dataset**

- Dataset: Titanic, Pima Indians Diabetes, or similar
- Build MLP with 2–3 hidden layers
- Train using mini-batch gradient descent and Adam
- Apply dropout or L2 regularization
- Evaluate using accuracy, confusion matrix, and F1-score

---

**6. Week 10 Progress Checklist**

By the end of Week 10, you should be able to:

**Conceptual**

- Explain mini-batch, epochs, and iterations
- Describe SGD, Adam, and learning rate schedules
- Understand dropout, batch normalization, and L1/L2 regularization

**Applied**

- Implement a small MLP on MNIST or tabular data
- Train efficiently using mini-batch gradient descent
- Compare optimizers and observe impact of learning rates
- Apply dropout and batch normalization to improve generalization
- Plot and interpret training and validation curves

**Week 11 — Convolutional Neural Networks (Expanded Syllabus)**

**Goal for Week 11:**

By the end of this week, you should understand convolution and pooling operations, the structure of CNNs, common architectures, and how to use transfer learning. You should be able to fine-tune a pre-trained CNN on a small image dataset.

---

**1. Introduction to CNNs (Day 1)**

**A. Motivation**

- Traditional fully connected networks (MLPs) scale poorly for images
- CNNs exploit:
  - **Spatial locality** (neighboring pixels are correlated)
  - **Parameter sharing** (same kernel applied across the image)
  - **Translation invariance**

**B. CNN Components**

- **Input layer:** image tensor (height × width × channels)
- **Convolutional layers:** learn feature maps with kernels/filters
- **Activation functions:** ReLU commonly
- **Pooling layers:** reduce spatial dimensions while retaining key features
- **Fully connected layers:** final classification/regression

---

**2. Convolution Layer (Day 1–2)**

**A. Concept**

- Kernel slides over input image, computes weighted sum → feature map
- Hyperparameters:
  - **Kernel size** (3×3, 5×5)
  - **Stride** (step size of sliding window)
  - **Padding** ('same' or 'valid')

**B. Feature Maps**

- Early layers: edges, corners
- Deeper layers: textures, shapes, high-level features

**Practice:**

- Implement 2D convolution manually on a small grayscale image using NumPy
- Visualize output feature maps

---

**3. Pooling Layer (Day 2)**

**A. Types**

- **Max pooling:** takes maximum value in window
- **Average pooling:** takes average
- Reduces spatial size, computational cost, and overfitting

**B. Practice**

- Apply 2×2 max pooling on a sample feature map manually
- Compare input and output dimensions

---

**4. Common CNN Architectures (Day 3)**

**A. Examples**

- **LeNet-5:** basic CNN for MNIST
- **AlexNet:** introduced ReLU, dropout, large-scale image classification
- **VGG:** deep architecture with small 3×3 kernels
- **ResNet:** residual connections to allow very deep networks

**B. Concepts**

- Depth (more layers → more abstract features)
- Residual connections → solve vanishing gradient problem
- Batch normalization and dropout in CNNs

**Practice:** Look at architecture diagrams and identify convolution, pooling, and fully connected layers

---

**5. Transfer Learning (Day 3–4)**

**A. Concept**

- Use pre-trained CNN (trained on ImageNet or CIFAR)
- Freeze early layers, fine-tune last layers for your dataset
- Advantages:
  - Faster convergence
  - Works with small datasets

**B. Framework Example**

- Keras pre-trained models:

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.layers import Dense, Flatten

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))
for layer in base_model.layers:
    layer.trainable = False

x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
output = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output)
```

- Compile and train on small dataset (cats vs dogs, CIFAR subset, etc.)

---

**6. Hands-On Mini Projects (Day 4–6)**

**Project 1 — Build CNN from Scratch**

1. Dataset: MNIST or Fashion-MNIST
2. Build 2–3 convolutional layers with ReLU and max pooling
3. Flatten → fully connected layer → output
4. Train and evaluate accuracy
5. Visualize feature maps of first convolution layer

**Project 2 — Transfer Learning**

1. Dataset: small subset of CIFAR-10 or Kaggle dataset (50–1000 images per class)
2. Load pre-trained CNN (VGG16, ResNet50, MobileNet)
3. Freeze early layers
4. Fine-tune top layers for your dataset
5. Evaluate performance and compare with training from scratch

**Project 3 — Visualization**

- Plot training/validation accuracy and loss
- Visualize misclassified images

---

**7. Week 11 Progress Checklist**

By the end of Week 11, you should be able to:

**Conceptual**

- Explain convolution and pooling operations
- Understand feature extraction and hierarchical representation in CNNs
- Describe common CNN architectures (LeNet, VGG, ResNet)
- Explain transfer learning and its advantages

**Applied**

- Build a small CNN from scratch and train on MNIST or similar dataset
- Use a pre-trained CNN and fine-tune for a small image classification task
- Evaluate models and visualize predictions and misclassifications
- Understand feature maps and what the network learns at different layers

**Week 12 — Sequence Models (Expanded Syllabus)**

**Goal for Week 12:**

By the end of this week, you should understand the basics of Recurrent Neural Networks (RNNs), LSTMs, and GRUs, know how to preprocess sequence data (text/time series), and be able to implement a simple sequence classification or prediction task. You should also prepare **Month 3 deliverables**: one notebook on an image task (CNN) and one on a sequence/text task.

---

**1. Introduction to Sequence Models (Day 1)**

**A. Motivation**

- Some data has **temporal or sequential structure**: words in a sentence, stock prices, sensor readings
- Traditional MLPs or CNNs are not ideal because they don't retain memory

**B. Recurrent Neural Networks (RNNs)**

- Maintain a **hidden state** that captures previous inputs
- Update rule (simplified):

ht=tanh⁡(Whht−1+Wxxt+b)h_t = \tanh(W_h h_{t-1} + W_x x_t + b)ht=tanh(Whht−1+Wxxt+b)

- Can handle variable-length sequences

**Practice:** Draw the unrolled RNN for a short input sequence

---

**2. RNN Limitations & Solutions (Day 1–2)**

**A. Vanishing/Exploding Gradients**

- Gradient can vanish or explode for long sequences → learning long-term dependencies is hard

**B. LSTM (Long Short-Term Memory)**

- Introduces **gates** to control information flow:
  - Forget gate: what to discard
  - Input gate: what to store
  - Output gate: what to expose
- Retains long-term dependencies

**C. GRU (Gated Recurrent Unit)**

- Simplified version of LSTM
- Combines forget and input gates into a single update gate
- Fewer parameters, faster training

**Practice:** Compare LSTM vs GRU diagrams; note how information flows

---

**3. Sequence-to-Sequence Concept (Day 2)**

**A. Encoder-Decoder Architecture**

- Encoder RNN: compresses input sequence into context vector
- Decoder RNN: generates output sequence from context vector
- Applications:
  - Machine translation
  - Chatbots
  - Text summarization

---

**4. Text Preprocessing (Day 2–3)**

**A. Tokenization**

- Split sentences into words, subwords, or characters
- Map tokens to integers using vocabulary

**B. Padding/Truncating**

- Ensure all sequences have same length for batch processing

**C. Embeddings**

- Convert tokens to dense vectors (learnable or pre-trained like GloVe/Word2Vec)

**Practice:**

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=50)
```

---

**5. Implementing RNNs (Day 3–5)**

**A. Using Keras**

- Simple RNN:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, LSTM, GRU, Dense, Embedding

model = Sequential([
    Embedding(input_dim=5000, output_dim=64, input_length=50),
    LSTM(128, return_sequences=False),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.2)
```

- Evaluate using accuracy, confusion matrix, F1-score

**B. Using PyTorch (Optional)**

- Use nn.RNN, nn.LSTM, nn.GRU layers
- Handle batch-first sequences
- Training loop: forward → loss → backward → optimizer step

---

**6. Hands-On Mini Projects**

**Project 1 — Sentiment Classification**

- Dataset: IMDB, Twitter, or small movie review dataset
- Steps:
  1. Text preprocessing: tokenization, padding, embedding
  2. Train LSTM or GRU for binary sentiment classification
  3. Evaluate metrics: accuracy, precision, recall, F1
  4. Plot training/validation loss and accuracy

**Project 2 — Sequence Prediction (Optional)**

- Dataset: sine wave, stock prices, or synthetic sequences
- Train RNN/LSTM to predict next value(s) in sequence
- Plot predicted vs actual sequences

---

**7. Month 3 Deliverables**

**Deliverable:** 2 Notebooks on GitHub

1. **Image Task (CNN):**
   - Fine-tune pre-trained CNN on small dataset
   - Include training/validation curves, accuracy, and misclassified examples
   - Document dataset, preprocessing, model, and results
2. **Sequence/Text Task (RNN/LSTM/GRU):**
   - Sentiment classification or sequence prediction
   - Include preprocessing steps, model architecture, evaluation metrics, and results
   - Properly comment code and include README.md

---

**8. Week 12 Progress Checklist**

By the end of Week 12, you should be able to:

**Conceptual**

- Explain RNN, LSTM, and GRU intuition
- Understand sequence-to-sequence models
- Understand text preprocessing, tokenization, and embeddings

**Applied**

- Implement LSTM/GRU for text classification or sequence prediction
- Evaluate sequence models with standard metrics
- Fine-tune hyperparameters and visualize training progress
- Prepare well-documented GitHub notebooks for deliverables

**Week 13 — Transformers and Attention (Expanded Syllabus)**

**Goal for Week 13:**

By the end of this week, you should understand the attention mechanism, the high-level structure of transformer encoders and decoders, and pre-trained transformer models like BERT and GPT. You should also be able to apply a pre-trained transformer for a practical NLP task such as text classification or named entity recognition (NER).

---

**1. Introduction to Attention (Day 1)**

**A. Motivation**

- RNNs/LSTMs process sequences sequentially → slow and struggles with long dependencies
- Attention allows models to **focus on relevant parts of input sequence** regardless of distance

**B. Intuition**

- Compute a **weighted sum of all input elements** for each output element
- Key formula:

Attention(Q,K,V)=softmax(QKTdk)V\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)VAttention(Q,K,V)=softmax(dkQKT)V

- Q: Query, K: Key, V: Value
- Intuition: "Which parts of the input should I pay attention to?"

**Practice:** Visualize attention weights for a small sentence (toy example)

---

**2. Transformer Architecture (Day 1–2)**

**A. Encoder**

- Stack of layers:
  - Multi-head self-attention
  - Feedforward network
  - Layer normalization + residual connections

**B. Decoder**

- Stack of layers:
  - Masked multi-head self-attention (prevents looking ahead)
  - Encoder-decoder attention
  - Feedforward network + residual connections

**C. Key Features**

- Parallel processing → faster training than RNNs
- Multi-head attention → capture multiple types of relationships

**Practice:** Draw high-level encoder-decoder diagram and identify attention blocks

---

**3. Pre-trained Transformers (Day 2–3)**

**A. BERT (Bidirectional Encoder Representations from Transformers)**

- Encoder-only model
- Bidirectional attention → understands context from both sides
- Pre-training tasks: masked language modeling, next sentence prediction

**B. GPT (Generative Pre-trained Transformer)**

- Decoder-only model
- Autoregressive → predicts next token
- Pre-trained for text generation

**C. Other Models**

- RoBERTa, DistilBERT, T5 for various tasks

---

**4. Using Pre-trained Transformers (Day 3–5)**

**A. Framework**

- Hugging Face transformers library simplifies using pre-trained models

```
from transformers import AutoTokenizer,
AutoModelForSequenceClassification, pipeline

# Load pre-trained model
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-
uncased")
model =
AutoModelForSequenceClassification.from_pretrained("distilbert-
base-uncased", num_labels=2)

# Example pipeline
nlp = pipeline("sentiment-analysis", model=model,
tokenizer=tokenizer)
result = nlp("I love this course!")
```

---

**B. Text Classification**

1. Load dataset: IMDB, SST-2, or small custom dataset
2. Tokenize and encode sequences using tokenizer
3. Train/Fine-tune pre-trained transformer for classification
4. Evaluate with accuracy, precision, recall, F1-score

---

**C. Named Entity Recognition (NER)**

```
from transformers import pipeline
ner = pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-english")
ner("R. Das works at CRS Pune under IMD.")
```

- Extract entities like person names, organizations, locations

---

**5. Hands-On Mini Projects**

**Project 1 — Text Classification**

- Dataset: IMDB reviews or any sentiment dataset
- Fine-tune DistilBERT/BERT on training data
- Evaluate with F1, accuracy
- Visualize misclassified examples

**Project 2 — Named Entity Recognition**

- Dataset: small subset of CoNLL-2003 or custom sentences
- Use pre-trained transformer for NER
- Extract and visualize entities in text

---

**6. Week 13 Progress Checklist**

By the end of Week 13, you should be able to:

**Conceptual**

- Explain attention mechanism and its role in sequence models
- Understand transformer encoder and decoder structure
- Differentiate between BERT and GPT architectures
- Understand the concept of pre-training and fine-tuning

**Applied**

- Use Hugging Face transformers library for text classification or NER
- Fine-tune pre-trained transformer models on small datasets
- Evaluate transformer-based models using standard metrics
- Visualize predictions and interpret attention outputs

**Week 14 — Computer Vision Advances (Expanded Syllabus)**

**Goal for Week 14:**

By the end of this week, you should understand the basic concepts of object detection and image segmentation, be familiar with modern architectures like YOLO and SSD, and be able to run pre-trained models on sample datasets.

---

**1. Introduction to Object Detection (Day 1–2)**

**A. Motivation**

- Unlike classification, object detection identifies:
  - **What** objects are in the image
  - **Where** they are located (bounding boxes)

**B. Key Concepts**

- **Bounding box:** (x_min, y_min, x_max, y_max) coordinates
- **IoU (Intersection over Union):** metric for detection overlap
- **Anchor boxes / priors:** predefined boxes of various shapes and sizes
- **Non-Maximum Suppression (NMS):** remove overlapping predictions

---

**C. Common Architectures**

1. **YOLO (You Only Look Once)**
   - Single-stage detector → fast and accurate
   - Predicts bounding boxes and class probabilities simultaneously
2. **SSD (Single Shot MultiBox Detector)**
   - Similar to YOLO, uses feature maps at multiple scales
   - Handles varying object sizes efficiently

**Practice:** Compare speed vs accuracy of YOLO vs SSD on example images

---

**2. Image Segmentation (Day 2–3)**

**A. Types of Segmentation**

1. **Semantic Segmentation**
   - Classify each pixel to a category (e.g., car, road, tree)
2. **Instance Segmentation**
   - Distinguish individual objects of the same class (e.g., Mask R-CNN)

**B. Key Architectures**

- **U-Net:** Encoder-decoder for biomedical and general segmentation
- **Mask R-CNN:** Adds segmentation masks on top of Faster R-CNN detection

**Practice:** Visualize output masks on example images

---

**3. Hands-On Using Pre-Trained Models (Day 3–5)**

**A. Object Detection with YOLO/SSD**

```
# Using YOLOv5 from ultralytics
from ultralytics import YOLO

model = YOLO("yolov5s.pt")  # pre-trained on COCO
results = model("sample_image.jpg")
results.show()
```

- Detect objects in images
- Visualize bounding boxes and confidence scores
- Evaluate basic detection metrics (IoU, confidence thresholds)

---

**B. Segmentation with U-Net or Mask R-CNN**

```
# Example with segmentation_models.pytorch
import segmentation_models_pytorch as smp
import torch

model = smp.Unet(encoder_name="resnet34", encoder_weights="imagenet", classes=1, activation="sigmoid")
# Load sample image and predict mask
```

- Apply pre-trained segmentation model on sample images
- Visualize predicted masks vs original images

---

**4. Mini Projects (Day 5–6)**

**Project 1 — Object Detection**

1. Dataset: small subset of COCO, Pascal VOC, or custom images
2. Load YOLO or SSD pre-trained model
3. Detect objects and visualize bounding boxes with class labels
4. Experiment with confidence threshold to see effect on predictions

**Project 2 — Image Segmentation**

1. Dataset: small sample images (medical images, street scenes, etc.)
2. Load U-Net or Mask R-CNN pre-trained model
3. Predict masks and overlay on original images
4. Evaluate mask qualitatively or quantitatively (IoU for simple examples)

---

**5. Week 14 Progress Checklist**

By the end of Week 14, you should be able to:

**Conceptual**

- Explain object detection vs classification vs segmentation
- Understand YOLO and SSD principles for detection
- Understand U-Net and Mask R-CNN for segmentation
- Know metrics like IoU and confidence thresholds

**Applied**

- Run pre-trained YOLO or SSD models for object detection on sample images
- Run pre-trained segmentation models (U-Net/Mask R-CNN) on sample images
- Visualize predictions clearly and compare with ground truth (if available)
- Understand basic trade-offs between speed and accuracy for detection models

**Week 15 — Generative Models (Expanded Syllabus)**
**Goal for Week 15:**
By the end of this week, you should understand the basic concepts of generative models, including autoencoders, VAEs, and GANs. You should be able to implement and train a simple autoencoder or VAE for anomaly detection or reconstruction tasks.

---

**1. Introduction to Generative Models (Day 1)**
**A. Motivation**

- Generative models learn the underlying **distribution of data**
- Applications:
  - Image generation (GANs)
  - Data denoising (autoencoders)
  - Anomaly detection
  - Dimensionality reduction

---

**2. Autoencoders (Day 1–2)**
**A. Concept**

- Encoder compresses input into **latent representation**
- Decoder reconstructs input from latent vector
- Objective: minimize reconstruction loss (e.g., MSE)

**Architecture:**

- Input → Encoder → Latent space → Decoder → Reconstructed output

**Practice:**

- Implement a simple autoencoder on MNIST using Keras or PyTorch

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

input_dim = 784
encoding_dim = 64

input_img = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(X_train, X_train, epochs=20, batch_size=32, validation_split=0.2)
```

---

**B. Applications**

- Dimensionality reduction
- Denoising images
- Anomaly detection (high reconstruction error indicates anomaly)

---

**3. Variational Autoencoders (VAE) (Day 2–3)**
**A. Concept**

- Probabilistic autoencoder
- Latent space encodes **mean** and **variance** → allows sampling new data
- Loss = Reconstruction loss + KL Divergence

**Practice:** Implement VAE on MNIST or Fashion-MNIST

- Visualize reconstructed images
- Sample from latent space to generate new images

---

**4. Generative Adversarial Networks (GANs) (Day 3–4)**
**A. Concept**

- Two networks: **Generator** and **Discriminator**
- Generator tries to produce realistic data
- Discriminator tries to distinguish real vs fake data
- Trained adversarially → improves generation quality

**Practice:** Visualize simple GAN training on MNIST

- Generator outputs digits
- Observe how generated digits improve over epochs

---

**5. Hands-On Mini Projects**
**Project 1 — Autoencoder for Anomaly Detection**

1. Dataset: MNIST or a tabular dataset with anomalies
2. Train autoencoder on normal data
3. Compute reconstruction error for test data
4. Identify anomalies based on threshold

**Project 2 — Variational Autoencoder**

1. Dataset: MNIST/Fashion-MNIST
2. Implement encoder to output mean and variance
3. Reparameterization trick: sample latent vector
4. Reconstruct images and visualize sampling from latent space

**Project 3 (Optional) — Simple GAN**

1. Dataset: MNIST
2. Build generator and discriminator networks
3. Train adversarially and visualize generated images
4. Observe learning progression over epochs

---

**6. Week 15 Progress Checklist**
By the end of Week 15, you should be able to:
**Conceptual**

- Explain the difference between autoencoders, VAEs, and GANs
- Understand latent space representation and its sampling in VAEs
- Understand adversarial training in GANs

**Applied**

- Implement a basic autoencoder in Keras or PyTorch
- Train a VAE for image reconstruction and sampling
- Optionally experiment with a simple GAN for image generation
- Apply autoencoder for anomaly detection on simple datasets

**Week 16 — Explainability & Fairness Basics (Expanded Syllabus)**
**Goal for Week 16:**
By the end of this week, you should understand key concepts of model interpretability and fairness, be familiar with methods like feature importance, LIME, and SHAP, and be able to apply them to a real model. You should also prepare the **Month 4 deliverable**: a medium-level project using a transformer or CV model along with an explainability analysis.

---

**1. Introduction to Explainability (Day 1–2)**
**A. Motivation**

- Deep learning models and ensemble methods are often **black boxes**
- Understanding predictions is critical for:
  - Trust in AI systems
  - Debugging models
  - Ethical and regulatory compliance

---

**B. Key Concepts**

1. **Global interpretability**
   - Understand how the model works overall
   - Feature importance, partial dependence plots (PDP)
2. **Local interpretability**
   - Explain a single prediction

- LIME (Local Interpretable Model-agnostic Explanations)
- SHAP (SHapley Additive exPlanations)

---

**2. Feature Importance (Day 2)**
**A. Concept**

- Measures contribution of each feature to the model's predictions
- For tree-based models: built-in feature importance available (e.g., Random Forest, XGBoost)
- For other models: use permutation importance

**Practice:**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance

rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Permutation importance
result = permutation_importance(rf, X_test, y_test, n_repeats=10)
importance = result.importances_mean
```

- Visualize feature importance with a bar chart

---

**3. LIME & SHAP (Day 3)**
**A. LIME**

- Perturbs input around a data point
- Fits simple interpretable model locally
- Explains which features contributed most for that prediction

```
import lime
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(X_train,
feature_names=feature_names, class_names=class_names,
mode='classification')
exp = explainer.explain_instance(X_test[0], rf.predict_proba)
exp.show_in_notebook()
```

**B. SHAP**

- Based on game theory (Shapley values)
- Computes contribution of each feature fairly

```
import shap

explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test,
feature_names=feature_names)
```

**Practice:** Apply LIME and SHAP to a small dataset for both global and local explanations

---

**4. Fairness & Bias Basics (Day 3–4)**
**A. Motivation**

- AI models may reflect historical biases in data
- Fairness ensures **equitable outcomes across groups**

**B. Concepts**

- **Protected attributes:** race, gender, age, etc.
- **Bias detection metrics:**
  - Statistical parity
  - Equalized odds
  - Predictive parity

**C. Tools**

- AIF360 and fairlearn for fairness evaluation and mitigation

**Practice:**

- Evaluate a model for potential bias across subgroups
- Plot differences in accuracy or error rates across groups

---

**5. Hands-On Mini Project**
**Project — Explainability Analysis**

1. Dataset: any dataset you've used with transformer or CV model
2. Train or load a pre-trained model
3. Apply feature importance (for tabular or CNN features) or SHAP/LIME (for transformer predictions)
4. Analyze top contributing features for predictions
5. Document findings in a short report:
   - Highlight which features drive decisions
   - Discuss potential biases or limitations

---

**6. Month 4 Deliverable**
**Medium Project:**

- Build a **realistic model** (transformer for NLP or CV model for images)
- Apply **explainability method** (feature importance, SHAP, LIME)
- Write **explainability note**:
  - Model description and task
  - Key findings from explanation
  - Potential biases or fairness considerations

---

**7. Week 16 Progress Checklist**
By the end of Week 16, you should be able to:
**Conceptual**

- Explain why model interpretability is important
- Describe feature importance, LIME, and SHAP methods
- Understand fairness concepts and how to evaluate bias in AI models

**Applied**

- Apply feature importance, LIME, or SHAP to a trained model
- Analyze predictions and highlight key contributing features
- Detect potential biases in model predictions
- Prepare a short explainability report for a real-world model

**Week 17 — Model Evaluation & Improvement (Expanded Syllabus)**
**Goal for Week 17:**
By the end of this week, you should be able to evaluate models systematically, perform error analysis, calibrate predictions, understand learning curves, and apply techniques such as data augmentation to improve model performance.

---

**1. Error Analysis (Day 1–2)**
**A. Motivation**

- Understanding **where models fail** is key to improving them
- Helps identify patterns in mispredictions and sources of error

**B. Techniques**

1. **Confusion matrix analysis**
   - Identify classes with high misclassification rates
2. **Error breakdown by feature or group**
   - Evaluate errors across different segments of data
3. **Visual inspection**
   - Examine misclassified images or mispredicted sequences

**Practice:**

- For a classification project, compute confusion matrix:

```
from sklearn.metrics import confusion_matrix,
classification_report

y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

- Group errors by category and document findings

**2. Calibration (Day 2)**
**A. Concept**
- Predicted probabilities may not reflect true likelihood
- Well-calibrated models are important for decision-making

**B. Techniques**
- **Platt scaling**
- **Isotonic regression**

**Practice:**
- Plot calibration curve using sklearn:

```
from sklearn.calibration import calibration_curve
import matplotlib.pyplot as plt

prob_true, prob_pred = calibration_curve(y_test, y_probs, n_bins=10)
plt.plot(prob_pred, prob_true, marker='o')
plt.plot([0,1], [0,1], linestyle='--')
plt.show()
```

---

**3. Learning Curves & Overfitting/Underfitting (Day 2–3)**
**A. Learning Curves**
- Plot **training vs validation performance** vs number of training examples
- Diagnose:
  - High bias → underfitting
  - High variance → overfitting

**Practice:**
- Generate learning curves for a model:

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, val_scores = learning_curve(model, X, y, cv=5)
```

- Visualize trends and interpret

---

**4. Data Augmentation Strategies (Day 3–4)**
**A. Motivation**
- Increase effective dataset size → reduce overfitting
- Especially useful for CV and NLP tasks

**B. Image Augmentation**
- Techniques: rotation, flipping, scaling, cropping, brightness/contrast adjustment
- Libraries: imgaug, albumentations, Keras ImageDataGenerator

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rotation_range=20, horizontal_flip=True)
datagen.fit(X_train)
```

**C. Text/Data Augmentation**
- Techniques: synonym replacement, back-translation, random insertion
- Libraries: nlpaug, textattack

---

**5. Hands-On Mini Projects**
**Project 1 — Error Analysis**
1. Choose a previous classification or sequence model
2. Identify misclassified instances or mispredicted sequences
3. Visualize and document patterns of failure
4. Suggest possible improvements based on analysis

**Project 2 — Model Improvement**
1. Apply **data augmentation** on training data
2. Retrain model and evaluate performance improvement
3. Compare metrics (accuracy, F1, IoU for CV, or loss curves)
4. Plot learning curves before and after augmentation

---

**6. Week 17 Progress Checklist**
By the end of Week 17, you should be able to:
**Conceptual**
- Understand the role of error analysis in model improvement
- Interpret learning curves to detect overfitting or underfitting
- Understand the importance of probability calibration
- Know common data augmentation strategies for CV and NLP

**Applied**
- Perform detailed error analysis on a previous model
- Apply calibration and evaluate predicted probabilities
- Generate learning curves and diagnose model performance issues
- Use data augmentation to improve model accuracy or robustness
- Document improvements and analyze results

**Week 18 — MLOps Basics (Expanded Syllabus)**
**Goal for Week 18:**
By the end of this week, you should understand the basic principles of MLOps, including model versioning, experiment tracking, CI/CD for ML models, and containerization using Docker. You should be able to track experiments, manage model versions, and containerize a simple model.

---

**1. Introduction to MLOps (Day 1)**
**A. Motivation**
- ML workflows are complex: development, training, evaluation, deployment, and monitoring
- MLOps ensures reproducibility, scalability, and maintainability

**B. Key Concepts**
1. **Model versioning**
   - Track changes to models, hyperparameters, and code
   - Keep track of production vs experimental models
2. **Experiment tracking**
   - Compare multiple training runs and hyperparameter settings
   - Tools: MLflow, Weights & Biases, TensorBoard

---

**2. Experiment Tracking with MLflow (Day 1–2)**
**A. Setting Up**

```
import mlflow
import mlflow.sklearn

mlflow.set_experiment("my_experiment")
with mlflow.start_run():
  mlflow.log_param("learning_rate", 0.01)
  mlflow.log_metric("accuracy", 0.95)
  mlflow.sklearn.log_model(model, "model")
```

**B. Key Features**
- **Log parameters, metrics, and artifacts**
- **Visualize experiments** in MLflow UI
- **Compare runs** to choose the best model

**Practice:**
- Track 2–3 model training runs on a dataset and compare metrics in MLflow

---

**3. Model Versioning (Day 2)**
**A. Concept**
- Maintain **versioned models** for production, testing, and rollback
- Versioning strategies:
  - Semantic versioning (v1.0, v1.1)
  - Git-based model storage

- MLflow model registry

**B. Practice**

- Register a model in MLflow registry
- Promote model from staging → production
- Roll back to a previous version if needed

---

**4. CI/CD Concepts for ML Models (Day 2–3)**

**A. Continuous Integration (CI)**

- Automate testing of ML code, scripts, and data preprocessing pipelines

**B. Continuous Deployment (CD)**

- Automate deployment of model into production
- Ensures reproducibility and minimizes human error

**C. Example Workflow**

1. Train model locally → push code to GitHub
2. CI pipeline: run tests, linting, basic evaluation
3. CD pipeline: deploy model as a REST API

**Practice:**

- Sketch a simple CI/CD pipeline for a model using GitHub Actions

---

**5. Containerization with Docker (Day 3–5)**

**A. Motivation**

- Package model, dependencies, and environment for **portable deployment**

**B. Docker Basics**

- Docker image → blueprint of environment
- Docker container → running instance of the image

**C. Practice**

1. Create a Dockerfile for a simple model API:

```
FROM python:3.11-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .

CMD ["python", "app.py"]
```

2. Build image: docker build -t my_model .
3. Run container: docker run -p 5000:5000 my_model

**D. Optional Hands-On**

- Expose a trained model as a REST API using Flask or FastAPI inside Docker

---

**6. Hands-On Mini Project**

**Project — Experiment Tracking + Containerization**

1. Select one of your earlier trained models (CV or NLP)
2. Track experiments with MLflow: parameters, metrics, and model artifacts
3. Register and version your best model
4. Create a Docker container to deploy the model as a REST API
5. Test the container locally and ensure it serves predictions

---

**7. Week 18 Progress Checklist**

By the end of Week 18, you should be able to:

**Conceptual**

- Explain the purpose of MLOps in ML lifecycle
- Understand model versioning, experiment tracking, and CI/CD principles
- Understand the role of containerization in deployment

**Applied**

- Track experiments using MLflow
- Register and manage model versions
- Design a basic CI/CD workflow for ML
- Build a Docker container for a trained model
- Run a model API locally inside a Docker container

**Week 19 — Serving Models (Expanded Syllabus)**

**Goal for Week 19:**

By the end of this week, you should be able to export models in standard formats, create a simple REST API to serve predictions, and understand latency and throughput trade-offs. You should also be able to serve a small model locally and query it programmatically.

---

**1. Model Export Formats (Day 1–2)**

**A. Motivation**

- Trained models need to be saved in formats that are portable, interoperable, and deployable

**B. Common Formats**

1. **TensorFlow SavedModel**
   - Native TensorFlow format, includes architecture, weights, and optimizer state
2. model.save("saved_model/my_model")
3. **ONNX (Open Neural Network Exchange)**
   - Interoperable across frameworks (PyTorch, TensorFlow, etc.)
   - Allows serving on multiple platforms
4. import torch
5. dummy_input = torch.randn(1, 3, 224, 224)
6. torch.onnx.export(model, dummy_input, "model.onnx")

**C. Practice**

- Save a trained model in both SavedModel and ONNX formats
- Reload model and verify predictions match original

---

**2. Serving via REST API (Day 2–4)**

**A. Framework Options**

1. **Flask**
   - Lightweight Python web framework
2. **FastAPI**
   - Modern, high-performance, supports async requests, automatic docs

**B. Basic API Design**

- Endpoints:
  - /predict: accept input data, return prediction
- Input formats: JSON, CSV
- Output formats: JSON

**Example with Flask:**

```
from flask import Flask, request, jsonify
import tensorflow as tf

app = Flask(__name__)
model = tf.keras.models.load_model("saved_model/my_model")

@app.route("/predict", methods=["POST"])
def predict():
    data = request.json['inputs']
    predictions = model.predict(data)
    return jsonify(predictions.tolist())

if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

**Example with FastAPI:**

```
from fastapi import FastAPI
from pydantic import BaseModel
import tensorflow as tf

app = FastAPI()
model = tf.keras.models.load_model("saved_model/my_model")

class InputData(BaseModel):
    inputs: list

@app.post("/predict")
def predict(data: InputData):
    predictions = model.predict(data.inputs)
    return {"predictions": predictions.tolist()}
```

**3. Performance Considerations (Day 4)**

**A. Latency vs Throughput**

- **Latency:** time to process one request
- **Throughput:** number of requests processed per second
- Trade-offs:
  - Batch processing → higher throughput, slightly higher latency
  - Single requests → lower latency, lower throughput

**B. Simple Optimization Tips**

- Load model once at startup
- Use efficient data preprocessing
- Consider multi-threading or async endpoints

---

**4. Hands-On Mini Project**

**Project — Serve a Small Model Locally**

1. Choose a small trained model (e.g., MNIST classifier, sentiment classifier)
2. Export it in SavedModel and/or ONNX format
3. Create a REST API using Flask or FastAPI
4. Query the API locally using Python requests or curl:

```
import requests
data = {"inputs": [[0.0, 0.1, 0.2, ...]]}  # example input
response = requests.post("http://127.0.0.1:5000/predict", json=data)
print(response.json())
```

5. Measure latency for single request and small batch of requests
6. Document API endpoints, input/output format, and performance

---

**5. Week 19 Progress Checklist**

By the end of Week 19, you should be able to:

**Conceptual**

- Understand different model export formats (SavedModel, ONNX)
- Explain latency and throughput trade-offs in serving
- Know basic API design principles for ML model serving

**Applied**

- Export a model in a deployable format
- Build a simple REST API to serve predictions
- Query the API locally and validate responses
- Measure simple performance metrics (latency, throughput)

**Week 20 — Scalable Data Pipelines (Expanded Syllabus)**

**Goal for Week 20:**

By the end of this week, you should understand the basics of ETL (Extract, Transform, Load) pipelines, data validation, and batching vs streaming data workflows. You should also be able to implement a simple data pipeline script and connect it to an inference API for deployment.

---

**1. Introduction to Data Pipelines (Day 1)**

**A. Motivation**

- ML models need **clean, consistent, and timely data**
- Pipelines automate ingestion, transformation, and delivery of data for training or inference

**B. Key Concepts**

1. **ETL (Extract, Transform, Load)**
   - Extract: gather data from sources (CSV, databases, APIs)
   - Transform: clean, normalize, encode, aggregate data
   - Load: deliver processed data to storage or model input
2. **Batch vs Streaming**
   - **Batch:** process data in chunks (daily, hourly)
   - **Streaming:** process data in real-time or near real-time

---

**2. Data Validation (Day 1–2)**

**A. Importance**

- Prevent bad or missing data from affecting model predictions
- Detect schema mismatches, missing values, outliers, and anomalies

**B. Techniques**

- Schema validation (column types, ranges)
- Missing value checks and imputations
- Value distribution comparison to historical data

**Practice:**

```
import pandas as pd

df = pd.read_csv("sample_data.csv")
assert df['age'].min() >= 0, "Age cannot be negative"
assert 'income' in df.columns, "Missing column"
df.fillna(df.mean(), inplace=True)
```

---

**3. Simple ETL Pipeline (Day 2–3)**

**A. Pipeline Steps**

1. Extract: read CSV, database, or API
2. Transform: normalize, scale, encode, filter anomalies
3. Load: save processed data to file, database, or pass to model

**B. Example Script**

```
import pandas as pd

def extract(file_path):
    return pd.read_csv(file_path)

def transform(df):
    df = df.dropna()
    df['feature_scaled'] = df['feature'] / df['feature'].max()
    return df

def load(df, output_path):
    df.to_csv(output_path, index=False)

if __name__ == "__main__":
    df_raw = extract("raw_data.csv")
    df_processed = transform(df_raw)
    load(df_processed, "processed_data.csv")
```

---

**4. Batching vs Streaming (Day 3)**

**A. Batch Processing**

- Process fixed-size data chunks
- Pros: simple, efficient for large data
- Cons: higher latency

**B. Streaming Processing**

- Process data as it arrives (Kafka, Spark Streaming, etc.)
- Pros: real-time, low latency
- Cons: more complex infrastructure

**Practice:**

- Simulate streaming by reading rows one-by-one and applying the transform function
- Compare latency vs batch processing

---

**5. Hands-On Mini Project**

**Project — Data Pipeline + Inference API**

1. Choose a small model trained previously (e.g., MNIST classifier, sentiment classifier)
2. Build a **data pipeline** script:
   - Extract: read new sample input data
   - Transform: normalize/encode features
   - Load: feed to the model for predictions
3. Connect pipeline to the **REST API** created in Week 19

4. Test end-to-end flow locally or on a free cloud platform (e.g., Hugging Face Spaces, Render, or Streamlit Cloud)
5. Document:
    - Data sources and transformations
    - Input/output format
    - Pipeline execution steps

---

**6. Deliverable — End of Month 5**
**Toy Deployed Service:**

- A working inference API connected to a data pipeline
- Accepts raw input data, applies transformations, and returns predictions
- Local deployment or on a free cloud service
- Include documentation (README) with:
    - How to run the service
    - Example inputs and outputs
    - Pipeline steps

---

**7. Week 20 Progress Checklist**
By the end of Week 20, you should be able to:
**Conceptual**

- Explain ETL and its importance in ML workflows
- Differentiate between batch and streaming processing
- Understand data validation and quality checks

**Applied**

- Implement a simple ETL pipeline in Python
- Connect pipeline to an inference API
- Test end-to-end model predictions
- Deploy a toy service locally or on a free cloud environment
- Document pipeline and API usage clearly

**Weeks 21–22 — Capstone Planning & Data Collection (Expanded Syllabus)**
**Goal for Weeks 21–22:**
By the end of Week 22, you should have a clearly defined capstone project, a cleaned and preprocessed dataset, and a baseline model implemented. These weeks focus on planning, data collection, and preliminary experimentation.

---

**1. Capstone Project Selection (Day 1–3, Week 21)**
**A. Choosing a Problem**

- Pick a **real-world problem** or domain you are interested in
- Consider available data, feasibility, and learning goals

**Examples of Capstone Projects:**

1. **Crop yield prediction** (tabular + time series)
2. **Medical image classification** (X-ray, MRI, skin lesions)
3. **Recommendation system** (movies, products, articles)
4. **Fraud detection** (transactional data)
5. **Time series forecasting** (stock prices, weather, sales)
6. **NLP summarizer** (news, reports, legal documents)

**B. Define Scope**

- Clearly outline:
    - Objective and target variable
    - Input features or modalities (tabular, image, text)
    - Performance metrics (accuracy, F1, RMSE, BLEU, etc.)
- Decide whether to focus on **proof-of-concept** or **fully deployable solution**

---

**2. Data Collection (Day 3–5, Week 21)**
**A. Identify Sources**

- Public datasets: Kaggle, UCI Machine Learning Repository, government portals, APIs
- Proprietary or personal data: domain-specific CSVs, images, logs

**B. Data Acquisition**

- Download, scrape, or query datasets
- Ensure **data volume and diversity** are sufficient for the model

**C. Data Storage & Organization**

- Organize raw data in structured folders
- Maintain metadata: source, description, date, fields

---

**3. Data Cleaning & Preprocessing (Week 22)**
**A. Tabular Data**

- Handle missing values: drop or impute
- Remove duplicates or irrelevant columns
- Encode categorical features: one-hot, label encoding
- Normalize/scale numerical features

**B. Image Data**

- Resize/crop images to standard dimensions
- Normalize pixel values
- Data augmentation (optional at this stage)

**C. Text Data**

- Remove punctuation, lowercasing, tokenization
- Remove stopwords (if appropriate)
- Optional: stemming or lemmatization

**D. Exploratory Data Analysis**

- Visualize feature distributions and correlations
- Detect anomalies or outliers
- Document insights and preprocessing steps

**Practice Example:**

```
import pandas as pd

df = pd.read_csv("data.csv")
df.dropna(inplace=True)
df = pd.get_dummies(df, columns=['category'])
df['feature_scaled'] = (df['feature'] - df['feature'].mean()) /
df['feature'].std()
```

---

**4. Baseline Model (Week 22)**
**A. Goal**

- Establish a simple model to benchmark future improvements

**B. Approach**

- Choose a model suitable for your task:
    - Regression: linear regression, decision tree
    - Classification: logistic regression, random forest
    - Time series: ARIMA, simple LSTM
    - Image: small CNN
    - NLP: TF-IDF + logistic regression or simple transformer

**C. Evaluation**

- Compute baseline metrics (accuracy, RMSE, F1, etc.)
- Document assumptions, limitations, and data splits

**Practice Example:**

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Baseline Accuracy:", accuracy_score(y_test, y_pred))
```

---

**5. Hands-On Deliverables for Weeks 21–22**

1. **Capstone Proposal Document**
    - Problem statement, motivation, and objectives
    - Input features, target variable, data sources
    - Success metrics and evaluation criteria
2. **Dataset Preparedness**

- Cleaned and preprocessed dataset ready for modeling
- Organized folder structure, metadata, and sample visualizations

3. **Baseline Model Notebook**
   - Simple model trained and evaluated
   - Metrics logged and documented for comparison with future models

---

**6. Weeks 21–22 Progress Checklist**
**Conceptual**

- Clearly define the problem scope and goals
- Identify and acquire suitable datasets
- Understand preprocessing requirements for tabular, image, or text data

**Applied**

- Collect and organize dataset for your project
- Perform data cleaning and preprocessing
- Conduct exploratory data analysis to understand the data
- Train a baseline model and evaluate initial performance
- Document all decisions, assumptions, and baseline results

**Weeks 23–24 — Capstone Modeling and Iteration (Expanded Syllabus)**
**Goal for Weeks 23–24:**
By the end of Week 24, you should have a fully trained, iteratively optimized capstone model, with proper handling of data challenges (imbalances, missing values), added explainability, and documented results.

---

**1. Model Training & Iteration (Week 23)**
**A. Model Selection**

- Choose appropriate model types based on task:
  - **Tabular data:** Random Forest, XGBoost, LightGBM, simple neural networks
  - **Images:** CNNs, transfer learning (ResNet, EfficientNet, etc.)
  - **Text/NLP:** Transformers (BERT, DistilBERT), RNN/LSTM for sequences
  - **Time series:** LSTM, GRU, Prophet, ARIMA

**B. Iterative Approach**

- Train an initial model (baseline from Weeks 21–22)
- Evaluate performance using validation set
- Identify weaknesses (low accuracy classes, poor regression errors)
- Iteratively improve:
  - Add new features or engineered features
  - Tune hyperparameters (learning rate, depth, regularization, batch size)
  - Experiment with different architectures for deep learning models

---

**2. Handling Data Challenges (Week 23)**
**A. Imbalanced Data**

- Techniques for classification:
  - Oversampling (SMOTE), undersampling
  - Class weighting in loss function
  - Ensemble methods focusing on minority class

**B. Missing or Noisy Data**

- Impute missing values or remove problematic rows/columns
- Apply noise reduction for image or sensor data
- Feature scaling and normalization if needed

---

**3. Optimization & Performance Improvement (Week 23–24)**
**A. Hyperparameter Tuning**

- Methods:
  - Grid search
  - Random search
  - Bayesian optimization (optional)

**B. Model Evaluation**

- Compute metrics relevant to task:
  - Classification: Accuracy, F1, Precision/Recall, ROC-AUC
  - Regression: RMSE, MAE, $R^2$
  - Image segmentation/detection: IoU, mAP
- Visualize learning curves, confusion matrices, error distribution

**C. Model Ensemble (Optional)**

- Combine multiple models to boost performance (bagging, stacking)

---

**4. Model Explainability (Week 24)**
**A. Global Explainability**

- Feature importance (tree-based models)
- SHAP summary plots
- Partial dependence plots

**B. Local Explainability**

- SHAP values for individual predictions
- LIME explanations for key instances

**Practice:**

```
import shap

explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test,
feature_names=feature_names)
```

---

**5. Documentation & Reporting (Week 24)**
**A. Documenting Iterations**

- Record:
  - Feature additions and preprocessing steps
  - Hyperparameter changes and rationale
  - Metric improvements over iterations

**B. Visualizations**

- Include:
  - Learning curves
  - Confusion matrices or prediction vs true plots
  - SHAP/LIME plots for explainability

**C. Final Report Structure**

1. Problem statement and objective
2. Data sources and preprocessing steps
3. Model architecture and training details
4. Iterative improvements and results
5. Explainability analysis
6. Limitations and next steps

---

**6. Hands-On Deliverables for Weeks 23–24**

1. **Capstone Model Notebook**
   - Full training, iteration, and evaluation workflow
   - Well-commented code with explanations
2. **Explainability Notebook/Section**
   - Visualizations showing feature importance and prediction explanations
3. **Documentation**
   - Short report or README covering model decisions, improvements, and insights

---

**7. Weeks 23–24 Progress Checklist**
**Conceptual**

- Understand iterative modeling and how to incrementally improve models
- Know techniques for handling data imbalance and noise
- Understand the importance of explainability in model evaluation

**Applied**

- Train and iteratively improve your capstone model
- Apply hyperparameter tuning and feature engineering
- Evaluate performance with appropriate metrics and visualizations
- Add model explainability using SHAP or LIME
- Document all iterations, insights, and results

**Weeks 25–26 — Deployment, Writeup, Portfolio & Interview Prep (Expanded Syllabus)**

**Goal for Weeks 25–26:**

By the end of Week 26, you should have a polished, deployable capstone project with full documentation, a demo, and a GitHub portfolio. You will also prepare for ML-related interviews and coding challenges.

---

**1. Model Deployment (Day 1–5, Week 25)**

**A. Deployment Options**

1. **Local Deployment**
   - REST API using Flask or FastAPI
   - Notebook-based demo (Jupyter or Colab)
2. **Web App Deployment**
   - Streamlit or Gradio apps for interactive demos
   - Quick deployment to free platforms: Hugging Face Spaces, Streamlit Cloud, or Render
3. **Cloud Deployment (Optional)**
   - AWS SageMaker, Google AI Platform, or Azure ML

**B. Deployment Steps**

1. Export final trained model in portable format (SavedModel, ONNX, pickle)
2. Create inference API or web app:
   - Input form or file upload for test data
   - Display predictions and optionally visualizations
3. Test API/app with sample inputs for correctness and stability
4. Optional: containerize using Docker

**Practice Example (Streamlit):**

```
import streamlit as st
import pickle

model = pickle.load(open("model.pkl", "rb"))

st.title("Capstone Demo")
user_input = st.text_input("Enter features separated by commas")
if st.button("Predict"):
    features = [float(x) for x in user_input.split(",")]
    prediction = model.predict([features])
    st.write(f"Prediction: {prediction[0]}")
```

---

**2. Documentation & Portfolio (Day 5–7, Week 25)**

**A. GitHub Repo**

- Include:
  - All notebooks and scripts
  - Cleaned dataset or instructions to obtain it
  - Requirements.txt or environment.yml for reproducibility

**B. README**

- Problem description
- Data sources and preprocessing steps
- Model architecture and training summary
- Demo instructions (link to deployed app or instructions to run locally)
- Key results and insights

**C. Demo Video or GIF**

- Short 1–3 minute recording showing:
  - App in action or notebook workflow
  - Sample inputs and predictions
  - Explainability outputs (if available)

---

**3. Capstone Polishing (Week 26)**

**A. Code Review**

- Ensure clean, well-commented code
- Remove redundant or exploratory cells/scripts

**B. Results Summary**

- Include:
  - Performance metrics
  - Explainability plots (SHAP, LIME)
  - Error analysis highlights
- Prepare a concise **summary table** comparing baseline vs final model

**C. Demo Preparation**

- Make demo self-explanatory for viewers unfamiliar with your project
- Ensure smooth execution in one go

---

**4. Interview Preparation (Week 26)**

**A. System Design for ML**

- How to design end-to-end ML pipelines
- Considerations for data collection, preprocessing, model training, serving, monitoring
- Discuss trade-offs: batch vs streaming, latency vs throughput

**B. ML Fundamentals Review**

- Key algorithms: regression, classification, clustering, deep learning
- Metrics and evaluation: accuracy, F1, RMSE, ROC-AUC, IoU
- Model interpretability: SHAP, LIME, feature importance
- Bias & fairness concepts

**C. Coding Practice**

- Data munging: cleaning, aggregating, joining datasets
- Model evaluation: writing custom metric functions, splitting data
- Platforms: LeetCode, HackerRank, Kaggle kernels for practice

---

**5. Hands-On Deliverables for Weeks 25–26**

1. **Deployed Capstone Demo**
   - REST API, Streamlit/Gradio app, or notebook-based interactive demo
2. **GitHub Portfolio**
   - Fully organized repo with code, environment setup, README, and demo link
3. **Short Video/GIF**
   - Demonstrates project workflow, prediction examples, and results
4. **Interview Prep Checklist**
   - 10–15 system design practice scenarios for ML pipelines
   - 20–30 coding problems on data handling and evaluation
   - Quick review of key ML concepts and metrics

---

**6. Final Deliverable**

**Polished Capstone Project**

- Code: clean, modular, and reproducible
- README: clear, professional, and comprehensive
- Demo: interactive, user-friendly, and visual
- Repository ready for portfolio presentation or job application

---

**7. Weeks 25–26 Progress Checklist**

**Conceptual**

- Understand deployment options and trade-offs
- Explain end-to-end ML pipeline design
- Recall ML fundamentals for interviews

**Applied**

- Deploy capstone model with a working demo
- Document everything clearly on GitHub
- Record demo video or GIF showing results
- Prepare for interviews with system design and coding practice

**Suggested learning resources (stable classics)**

- Intro ML course: Andrew Ng's ML (Coursera) for conceptual grounding.
- Deep learning: practical course like fast.ai or official TensorFlow/PyTorch tutorials.
- Book: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.
- Book: Deep Learning by Goodfellow (for theory).
- Practice: Kaggle competitions and datasets, GitHub for portfolio.

---

**How to practice effectively**

- Code every day even if short. Prioritize short, frequent hands-on practice.
- Start projects early. Theory without projects is brittle.
- Keep a learning diary. Note mistakes and lessons.
- Use experiment tracking (even a simple CSV) to record runs and hyperparameters.
- Focus on clarity of communication in README and comments. Recruiters and collaborators read those first.

---

**Sample weekly time plan (if 15 hours/week)**

- 6 hours learning theory/readings/videos
- 6 hours hands-on coding/notebooks
- 3 hours reading papers/blogs or polishing documentation and GitHub

---

**Capstone ideas (pick one)**

- End-to-end image classifier with deployment and explainability.
- Time series forecasting dashboard for a real dataset.
- NLP pipeline: classification plus inference API (use transformer).
- Recommender system using implicit feedback.

---

**Quick checklist to measure progress**

- Can implement and train models with scikit-learn end-to-end.
- Can build a neural network and train it on real data with a DL framework.
- Have 2–3 public projects on GitHub with clear READMEs.
- Can explain bias/variance and choose appropriate metrics.
- Can deploy a simple model and demonstrate inference.