AI and Machine Learning Course: From Beginner to Advanced

Course Overview

This comprehensive course guides learners from foundational Python programming to advanced machine learning (ML) and deep learning (DL) techniques, covering 20 labs as outlined in the provided document. It is designed for self-study, with detailed explanations, code snippets, and real-world projects to ensure practical application. Each module includes:

- **Detailed Content**: In-depth explanations of concepts, algorithms, and techniques with examples.
- **Code Snippets**: Practical Python code for key tasks, using libraries like NumPy, Pandas, Scikit-learn, Keras, and OpenCV.
- **30 Practice Questions**: Hands-on exercises to reinforce learning.
- **Real-World Project**: A detailed project with step-by-step instructions, dataset suggestions, and evaluation metrics.
- **Resources**: References to datasets, documentation, and further reading.

Prerequisites: Basic understanding of programming concepts; no prior ML knowledge required.

Learning Outcomes:

- Master Python programming for data science and ML.
- Understand and implement ML algorithms (regression, classification, clustering).
- Build and deploy deep learning models using Keras.
- Apply advanced techniques like transfer learning and experiment tracking.
- Solve real-world problems with end-to-end ML pipelines.

Module 1: Python Basics (Lab 01 & 02)

Duration: 6 Hours (3 hours per lab)
Objective: Build a strong foundation in Python programming, covering syntax, variables, data types, conditions, loops, and basic error handling.
Content:

- **Hello World & Arithmetic**: Use `print()` for output and perform arithmetic operations (e.g., `+`, `-`, `*`, `/`, `%`, `**`).
- **User Input**: Collect and process user input with `input()`.
- **Conditions**: Implement decision-making with `if`, `elif`, `else`.
- **Loops**: Use `for` and `while` loops for iteration.
- **Error Handling**: Use `try-except` to manage invalid inputs.
- **Key Libraries**: Introduction to `math` for mathematical operations.

Detailed Explanation: Python is a versatile, high-level programming language widely used in AI and ML due to its simplicity and extensive libraries. Variables store data (e.g., integers, floats, strings), and data types determine how data is processed. Conditional statements allow decision-making, while loops enable repetitive tasks. Error handling ensures robust programs by catching exceptions like invalid user inputs.

Code Snippet (Basic Program with Input, Conditions, and Loops):

```
# Simple program to check eligibility and print multiplication table
try:
    name = input("Enter your name: ")
    age = int(input("Enter your age: "))
    if age >= 18:
        print(f"{name}, you are eligible to vote!")
    else:
        print(f"{name}, you are not eligible to vote yet.")

    num = int(input("Enter a number for multiplication table: "))
    for i in range(1, 11):
        print(f"{num} x {i} = {num * i}")
except ValueError:
    print("Please enter a valid number.")
```

Practice Questions:

1. Write a program to print "Welcome to AI and ML".
2. Calculate the sum of two user-input numbers.
3. Check if a number is even or odd.
4. Compute the square of a user-input number.
5. Find the maximum of two numbers.
6. Swap two variables without a temporary variable.
7. Convert Celsius to Fahrenheit.
8. Check if a year is a leap year.
9. Calculate the area of a rectangle.
10. Find the remainder when a number is divided by 5.
11. Print the first 10 natural numbers using a for loop.
12. Calculate the factorial of a user-input number.
13. Generate a multiplication table for a user-input number.
14. Count the digits in a number.
15. Reverse a user-input number.
16. Check if a number is positive, negative, or zero.
17. Print all even numbers between 1 and 50.
18. Sum all numbers from 1 to 100.
19. Check if a string is a palindrome.
20. Print the Fibonacci sequence up to n terms.

21. Find the GCD of two numbers.
22. Check if a number is prime.
23. Calculate the average of five user-input numbers.
24. Convert a decimal number to binary.
25. Find the square root using the `math` module.
26. Check if a string contains only digits.
27. Count vowels in a user-input string.
28. Print a star pattern (e.g., triangle).
29. Calculate the power of a number without `**`.
30. Check if two strings are anagrams.

Real-World Project: *Student Grade Calculator*
Description: Build a Python program to manage student grades, calculate averages, assign grades, and generate a report.
Dataset: Create a simple dataset (e.g., a FacetGrid **Steps**:

1. Prompt the user to input a student's name, roll number, and marks for three subjects (Math, Science, English).
2. Validate inputs using `try-except` to handle non-numeric marks.
3. Calculate total marks, average, and assign a grade (A: ≥80, B: ≥60, C: ≥40, else F).
4. Save the results to a text file in a formatted report.
5. Display the report on the console.

Code Snippet (Core Logic):

```
def calculate_grade(marks):
    avg = sum(marks) / len(marks)
    if avg >= 80:
        return 'A'
    elif avg >= 60:
        return 'B'
    elif avg >= 40:
        return 'C'
    return 'F'

try:
    name = input("Enter student name: ")
    roll = input("Enter roll number: ")
    marks = []
    for subject in ["Math", "Science", "English"]:
        mark = float(input(f"Enter marks for {subject}: "))
        if mark < 0 or mark > 100:
            raise ValueError("Marks must be between 0 and 100")
        marks.append(mark)

    total = sum(marks)
    avg = total / len(marks)
    grade = calculate_grade(marks)

    report = f"Student: {name}\nRoll Number: {roll}\nTotal Marks: {total}\nAverage: {avg:.2f}\nGrade: {grade}"
    print(report)

    with open("grade_report.txt", "w") as f:
        f.write(report)
except ValueError as e:
    print(f"Error: {e}")
```

Evaluation Criteria:

- Correctly handles invalid inputs (e.g., negative marks, non-numeric inputs).
- Produces a formatted report with accurate calculations.
- Saves the report to a file without errors.
- User-friendly interface with clear prompts.

Resources:

- Python Official Documentation: https://docs.python.org/3/
- Dataset: Create a simple list of dictionaries (e.g., `[{"name": "John", "roll": "R001", "marks": [85, 90, 88]}, ...]`).

Module 2: NumPy - Matrix Operations & Descriptive Statistics (Lab 03)

Duration: 3 Hours
Objective: Master NumPy for efficient array operations, matrix manipulations, and statistical analysis.
Content:

- **Arrays**: Create and manipulate 1D/2D arrays using `numpy.array()`.
- **Operations**: Perform element-wise operations, matrix multiplication (`np.dot`), and broadcasting.
- **Statistics**: Compute mean (`np.mean`), median (`np.median`), standard deviation (`np.std`).
- **Indexing/Slicing**: Access specific elements, rows, or columns.
- **Random Numbers**: Generate random arrays with `np.random`.

Detailed Explanation: NumPy is the foundation for numerical computing in Python, offering efficient array operations. Arrays are multi-dimensional data structures optimized for mathematical operations. Broadcasting allows operations on arrays of different shapes, while indexing/slicing enables precise data access. Descriptive statistics summarize data distributions, critical for ML preprocessing.

Code Snippet (Matrix Operations and Statistics):

```
import numpy as np

# Create arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([[1, 2], [3, 4]])
arr3 = np.random.randint(1, 10, (3, 3))

# Operations
print("Array Addition:", arr1 + 2)  # Broadcasting
print("Matrix Multiplication:", np.dot(arr2, arr2))
print("Mean:", np.mean(arr1))
print("Median:", np.median(arr1))
print("Standard Deviation:", np.std(arr1))

# Indexing/Slicing
print("First Element:", arr1[0])
print("First Row:", arr3[0, :])
```

Practice Questions:

1. Create a 1D array with numbers 1 to 10.
2. Create a 3x3 zero matrix.
3. Generate a 4x4 random integer matrix (1-10).
4. Sum all elements in a 2D array.
5. Compute mean of a 1D array of 10 random numbers.
6. Calculate standard deviation of an array.
7. Create a 5x5 identity matrix.
8. Multiply two 2x2 matrices.
9. Extract the first row of a 3x3 matrix.
10. Slice a 4x4 matrix to get the top-left 2x2 submatrix.
11. Add a scalar to all elements of an array.
12. Create an array of even numbers (2 to 20).
13. Reshape a 1D array (12 elements) into a 3x4 matrix.
14. Compute dot product of two vectors.
15. Find the maximum value in a 2D array.
16. Replace negative values with zero.
17. Generate 10 random numbers (0 to 1).
18. Compute element-wise square of an array.
19. Transpose a 3x3 matrix.
20. Sum diagonal elements of a matrix.
21. Create an array (1 to 100, step=5).
22. Find index of maximum value in a 1D array.
23. Concatenate two 1D arrays.
24. Compute mean of each row in a 3x3 matrix.
25. Create a 5x5 matrix with ones on the border.
26. Check if two arrays are equal.
27. Compute inverse of a 2x2 matrix.
28. Sort a 1D array in ascending order.
29. Find minimum value in each column of a 3x3 matrix.
30. Perform element-wise division of two arrays.

Real-World Project: *Sales Data Analysis*
Description: Analyze monthly sales data for multiple products, computing statistics and visualizing trends.
Dataset: Use a CSV file with columns: `Product`, `Month`, `Sales` (e.g., Kaggle's "Retail Sales Dataset").
Steps:

1. Load the CSV using `pandas.read_csv()` and convert to NumPy arrays.
2. Compute mean, median, and standard deviation for each product's sales.
3. Identify the top-performing product (highest average sales).
4. Use Matplotlib to plot sales trends over months.
5. Save results to a CSV and plot as a PNG.

Code Snippet:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("sales_data.csv")
sales = df.pivot_table(values="Sales", index="Month", columns="Product").to_numpy()

# Statistics
mean_sales = np.mean(sales, axis=0)
median_sales = np.median(sales, axis=0)
std_sales = np.std(sales, axis=0)
top_product = df["Product"].unique()[np.argmax(mean_sales)]

# Plot
plt.plot(sales)
plt.title("Sales Trends")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.legend(df["Product"].unique())
plt.savefig("sales_trends.png")
plt.show()

# Save results
```

```
results = pd.DataFrame({"Product": df["Product"].unique(), "Mean": mean_sales, "Median": median_sales, "Std": std_sales})
results.to_csv("sales_stats.csv")
print(f"Top Product: {top_product}")
```

Evaluation Criteria:

- Accurate loading and processing of sales data.
- Correct computation of statistics (mean, median, std).
- Clear visualization of trends with proper labels.
- Correct identification of the top product.
- Error-free CSV output.

Resources:

- NumPy Documentation: https://numpy.org/doc/stable/
- Dataset: Kaggle Retail Sales Dataset (or simulate with `np.random.randint`).

Module 3: Pandas - Data Manipulation & Analysis (Lab 04)

Duration: 3 Hours
Objective: Manipulate and analyze structured data using Pandas DataFrames and Series.
Content:

- **DataFrames/Series**: Create and manipulate tabular data structures.
- **Operations**: Handle missing values, sort, filter, and add columns.
- **File I/O**: Read/write CSV files.
- **Visualization**: Create plots with Pandas and Matplotlib.

Detailed Explanation: Pandas is a powerful library for data manipulation, built on NumPy. DataFrames handle tabular data (rows/columns), while Series represent single columns. Operations like filtering, grouping, and merging enable efficient data analysis. Visualization with Pandas simplifies exploratory data analysis (EDA).

Code Snippet (Data Manipulation):

```
import pandas as pd

# Create DataFrame
data = {"Name": ["Alice", "Bob"], "Marks": [85, 90]}
df = pd.DataFrame(data)

# Operations
df["Pass"] = df["Marks"] >= 50
df = df.sort_values("Marks")
df = df.fillna({"Marks": df["Marks"].mean()})

# Save to CSV
df.to_csv("students.csv", index=False)
print(df)
```

Practice Questions:

1. Create a DataFrame from a dictionary.
2. Read a CSV into a DataFrame.
3. Display first 5 rows.
4. Fill missing values with column mean.
5. Sort DataFrame by a column.
6. Filter rows (marks > 80).
7. Add pass/fail column.
8. Group by column and compute mean.
9. Merge two DataFrames.
10. Drop a column.
11. Replace values <50 with 0.
12. Sum a column.
13. Create a Series from a list.
14. Convert DataFrame column to Series.
15. Save DataFrame to CSV.
16. Find unique values in a column.
17. Count non-null values.
18. Create a pivot table.
19. Rename columns.
20. Check for missing values.
21. Compute column median.
22. Double values in a column.
23. Create a bar plot.
24. Filter with multiple conditions.
25. Compute column correlation.
26. Drop rows with missing values.
27. Create DataFrame with random integers.
28. Find row with maximum value.
29. Convert categorical column to codes.
30. Append two DataFrames.

Real-World Project: *Student Performance Dashboard*
Description: Build a dashboard to analyze student performance, handling missing data and visualizing results.
Dataset: CSV with columns: `Name`, `Roll`, `Math`, `Science`, `English`, `Subject` (e.g., Kaggle's "Student Performance Dataset").
Steps:

1. Load CSV using `pandas.read_csv()`.
2. Fill missing marks with column mean.
3. Add a pass/fail column (marks ≥50).
4. Create a bar chart of mark distribution.
5. Save processed data to a CSV.

Code Snippet:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("student_data.csv")

# Handle missing values
df = df.fillna(df.mean(numeric_only=True))

# Add pass/fail column
df["Pass"] = df[["Math", "Science", "English"]].mean(axis=1) >= 50

# Visualize
df[["Math", "Science", "English"]].mean().plot(kind="bar")
plt.title("Average Marks by Subject")
plt.savefig("marks_distribution.png")
plt.show()

# Save
df.to_csv("processed_student_data.csv", index=False)
```

Evaluation Criteria:

- Correctly handles missing data.
- Accurately computes pass/fail status.
- Produces clear bar chart with labels.
- Saves processed data without errors.

Resources:

- Pandas Documentation: https://pandas.pydata.org/docs/
- Dataset: Kaggle Student Performance Dataset.

Module 4: Data Visualization with Matplotlib and Seaborn (Lab 05 & 06)

Duration: 6 Hours
Objective: Create and interpret visualizations to explore data patterns.
Content:

- **Matplotlib**: Line, bar, scatter plots.
- **Seaborn**: Advanced visualizations (`histplot`, `pairplot`, `heatmap`, `boxplot`).
- **Correlation**: Visualize correlation matrices.
- **Subplots**: Multiple plots in one figure.

Detailed Explanation: Visualization is critical for EDA in ML. Matplotlib provides flexible plotting, while Seaborn simplifies statistical visualizations. Correlation matrices reveal relationships between variables, and subplots allow multiple visualizations in a single figure.

Code Snippet (Visualization Example):

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load data
df = pd.DataFrame({"x": [1, 2, 3, 4], "y": [10, 20, 25, 30]})

# Plots
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(df["x"], df["y"])
plt.title("Scatter Plot")

plt.subplot(1, 2, 2)
sns.heatmap(df.corr(), annot=True)
plt.title("Correlation Heatmap")
plt.savefig("plots.png")
plt.show()
```

Practice Questions:

1. Create a line plot (1 to 10).
2. Plot student counts by department (bar).
3. Create scatter plot of two variables.
4. Generate histogram with KDE (Seaborn).

5. Create pairplot for Iris dataset.
6. Plot correlation heatmap.
7. Create boxplot for a column.
8. Plot two subplots side by side.
9. Add title/labels to Matplotlib plot.
10. Change Seaborn plot color.
11. Create pie chart of data distribution.
12. Plot violin plot.
13. Save plot as PNG.
14. Scatter plot with category colors.
15. Line chart with legend.
16. Stacked bar chart.
17. Add grid lines to Matplotlib plot.
18. Histogram with 20 bins.
19. Boxen plot (Seaborn).
20. Regression line in pairplot.
21. Customize plot label font size.
22. 3D scatter plot (Matplotlib).
23. Time series plot with dates.
24. Subplot with bar and line charts.
25. Heatmap with annotations.
26. Violin plot comparing groups.
27. Bar chart with error bars.
28. Scatter plot with size proportional to variable.
29. Density plot (Seaborn).
30. Combine boxplot and stripplot.

Real-World Project: *Sales Trend Analysis Dashboard*
Description: Visualize sales trends for multiple products over time.
Dataset: CSV with `Product`, `Month`, `Sales` (e.g., Kaggle Retail Sales).
Steps:

1. Load sales data with Pandas.
2. Create line chart for sales trends.
3. Generate bar chart for total sales by product.
4. Plot correlation heatmap.
5. Save plots as PNGs.

Code Snippet:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
df = pd.read_csv("sales_data.csv")
pivot_df = df.pivot_table(values="Sales", index="Month", columns="Product")

# Plots
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
pivot_df.plot(kind="line")
plt.title("Sales Trends")

plt.subplot(2, 2, 2)
pivot_df.sum().plot(kind="bar")
plt.title("Total Sales by Product")

plt.subplot(2, 2, 3)
sns.heatmap(pivot_df.corr(), annot=True)
plt.title("Sales Correlation")

plt.tight_layout()
plt.savefig("sales_dashboard.png")
plt.show()
```

Evaluation Criteria:

- Accurate data loading and pivoting.
- Clear, labeled visualizations.
- Correct correlation computation.
- Error-free PNG output.

Resources:

- Matplotlib Documentation: https://matplotlib.org/stable/contents.html
- Seaborn Documentation: https://seaborn.pydata.org/
- Dataset: Kaggle Retail Sales Dataset.

Module 5: Descriptive Statistics & Probability (Lab 07)

Duration: 3 Hours
Objective: Understand statistical measures and probability simulations.
Content:

- **Central Tendency**: Mean, median, mode.
- **Dispersion**: Variance, standard deviation.
- **Probability**: Simulate coin tosses, dice rolls.
- **Distributions**: Analyze normal distributions, skewness, kurtosis.

Detailed Explanation: Descriptive statistics summarize data characteristics. Central tendency measures (mean, median, mode) describe typical values, while dispersion measures (variance, std) describe spread. Probability simulations model real-world randomness, and distribution analysis helps understand data shapes.

Code Snippet (Statistics and Simulation):

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Data
data = np.random.normal(50, 10, 1000)

# Statistics
mean = np.mean(data)
median = np.median(data)
mode = stats.mode(data)[0][0]
variance = np.var(data)
std = np.std(data)

# Simulation
coin_tosses = np.random.choice([0, 1], size=100)
heads_prob = np.mean(coin_tosses)

# Plot
plt.hist(data, bins=30, density=True)
plt.title(f"Mean: {mean:.2f}, Median: {median:.2f}, Std: {std:.2f}")
plt.savefig("distribution.png")
plt.show()
```

Practice Questions:

1. Compute mean of a list.
2. Calculate median.
3. Find mode using SciPy.
4. Compute variance.
5. Calculate standard deviation.
6. Simulate 100 coin tosses.
7. Simulate 1000 dice rolls.
8. Generate normal distribution.
9. Plot histogram with KDE.
10. Compute skewness.
11. Calculate kurtosis.
12. Find 25th/75th percentiles.
13. Simulate biased coin toss.
14. Compute range.
15. Generate uniform distribution.
16. Plot CDF.
17. Mean of 50-number sample.
18. Standard error of mean.
19. Simulate binomial distribution.
20. Compute covariance.
21. Generate Poisson sample.
22. Histogram with 30 bins.
23. 90th percentile.
24. Simulate random walk.
25. Calculate IQR.
26. Median absolute deviation.
27. Exponential distribution sample.
28. Plot boxplot.
29. Probability of rolling two 6s.
30. Simulate card draw probabilities.

Real-World Project: *Customer Behavior Analysis*
Description: Analyze simulated customer purchase data to understand spending patterns.
Dataset: Generate data with `np.random.normal` (e.g., amounts, frequency).
Steps:

1. Simulate purchase data (amounts, dates).
2. Compute mean, median, variance, IQR.
3. Simulate repeat purchase probabilities.
4. Plot histogram with KDE.
5. Save results to CSV.

Code Snippet:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Simulate data
```

```
data = pd.DataFrame({
    "CustomerID": range(1000),
    "Amount": np.random.normal(100, 20, 1000)
})

# Statistics
stats_df = pd.DataFrame({
    "Metric": ["Mean", "Median", "Variance", "IQR"],
    "Value": [
        np.mean(data["Amount"]),
        np.median(data["Amount"]),
        np.var(data["Amount"]),
        np.percentile(data["Amount"], 75) - np.percentile(data["Amount"], 25)
    ]
})

# Plot
plt.hist(data["Amount"], bins=30, density=True)
plt.title("Purchase Amount Distribution")
plt.savefig("purchase_distribution.png")
plt.show()

stats_df.to_csv("purchase_stats.csv")
```

Evaluation Criteria:

- Accurate simulation and statistics.
- Clear visualization with KDE.
- Correct IQR calculation.
- Error-free CSV output.

Resources:

- SciPy Documentation: https://docs.scipy.org/doc/scipy/
- Dataset: Simulated with `np.random.normal`.

Module 6: Hypothesis Testing & Outlier Detection (Lab 08)

Duration: 3 Hours
Objective: Perform hypothesis testing and detect outliers.
Content:

- **Hypothesis Testing**: Independent/paired t-tests.
- **Outlier Detection**: Z-scores, IQR method.
- **Visualization**: Highlight outliers in plots.

Detailed Explanation: Hypothesis testing evaluates if observed differences are statistically significant. T-tests compare means between groups. Outliers, extreme values, can skew analysis and are detected using Z-scores or IQR.

Code Snippet (T-Test and Outlier Detection):

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Data
group1 = np.random.normal(50, 10, 100)
group2 = np.random.normal(55, 10, 100)

# T-test
t_stat, p_val = stats.ttest_ind(group1, group2)

# Outliers
z_scores = np.abs((group1 - np.mean(group1)) / np.std(group1))
outliers = group1[z_scores > 3]

# Plot
plt.scatter(range(len(group1)), group1, c=["red" if z > 3 else "blue" for z in z_scores])
plt.title(f"T-test p-value: {p_val:.4f}")
plt.savefig("outliers.png")
plt.show()
```

Practice Questions:

1. Independent t-test.
2. Compute p-value.
3. Paired t-test.
4. Calculate Z-scores.
5. Identify outliers (Z-score > 3).
6. Compute IQR.
7. Find outliers (IQR method).
8. Plot with outliers highlighted.
9. T-test (α=0.05).
10. Degrees of freedom.
11. Compare means (t-test).
12. Boxplot for outliers.
13. Manual t-statistic.

14. Interpret p-value.
15. One-sample t-test.
16. Standard error.
17. Outliers in time series.
18. T-test on skewed data.
19. Confidence interval.
20. Histogram with outliers.
21. Filter outliers (IQR).
22. T-test on normal data.
23. Compare Z-score/IQR.
24. Test equal variances.
25. Effect size of t-test.
26. Simulate data with outliers.
27. T-test on small sample.
28. Visualize t-statistics.
29. Compute t-test power.
30. Compare outlier methods.

Real-World Project: *Quality Control Analysis*
Description: Analyze defect rates in manufacturing to compare factories and detect outliers.
Dataset: Simulate defect counts with `np.random.poisson`.
Steps:

1. Simulate defect data for two factories.
2. Perform t-test to compare defect rates.
3. Detect outliers using Z-scores and IQR.
4. Plot boxplot with outliers highlighted.
5. Save results to CSV.

Code Snippet:

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt

# Simulate data
factory1 = np.random.poisson(5, 100)
factory2 = np.random.poisson(6, 100)

# T-test
t_stat, p_val = stats.ttest_ind(factory1, factory2)

# Outliers (IQR)
q1, q3 = np.percentile(factory1, [25, 75])
iqr = q3 - q1
outliers = factory1[(factory1 < q1 - 1.5 * iqr) | (factory1 > q3 + 1.5 * iqr)]

# Plot
plt.boxplot([factory1, factory2], labels=["Factory 1", "Factory 2"])
plt.title(f"T-test p-value: {p_val:.4f}")
plt.savefig("defect_analysis.png")
plt.show()

pd.DataFrame({"Factory 1 Outliers": outliers}).to_csv("defect_outliers.csv")
```

Evaluation Criteria:

- Accurate t-test results.
- Correct outlier detection.
- Clear boxplot visualization.
- Error-free CSV output.

Resources:

- SciPy Stats: https://docs.scipy.org/doc/scipy/reference/stats.html
- Dataset: Simulated with `np.random.poisson`.

Module 7: Data Scaling, Encoding & Outlier Handling (Lab 09)

Duration: 3 Hours
Objective: Preprocess data for ML using scaling, encoding, and outlier handling.
Content:

- **Scaling**: MinMaxScaler, StandardScaler.
- **Encoding**: OneHotEncoder, LabelEncoder.
- **Outlier Handling**: Z-scores, IQR.

Detailed Explanation: Preprocessing ensures data is suitable for ML models. Scaling normalizes features to a common range, encoding converts categorical data to numerical, and outlier handling prevents skewed results.

Code Snippet (Preprocessing Pipeline):

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd
```

```
import numpy as np

# Data
df = pd.DataFrame({"Age": [25, 30, 100], "Category": ["A", "B", "A"]})

# Scaling
scaler = StandardScaler()
df["Age_Scaled"] = scaler.fit_transform(df[["Age"]])

# Encoding
encoder = OneHotEncoder(sparse_output=False)
encoded = encoder.fit_transform(df[["Category"]])
df_encoded = pd.concat([df, pd.DataFrame(encoded, columns=encoder.get_feature_names_out())], axis=1)

# Outliers
z_scores = np.abs((df["Age"] - df["Age"].mean()) / df["Age"].std())
df["Outlier"] = z_scores > 3

print(df_encoded)
```

Practice Questions:

1. Apply MinMaxScaler.
2. Use StandardScaler.
3. One-hot encode a column.
4. Apply LabelEncoder.
5. Detect outliers (Z-scores).
6. Remove outliers (IQR).
7. Create scaling/encoding pipeline.
8. Transform mixed data types.
9. Compare MinMaxScaler/StandardScaler.
10. Encode ordinal categories.
11. Handle missing values.
12. Apply VarianceThreshold.
13. Preprocess mixed DataFrame.
14. Visualize scaled vs. original data.
15. Compute variance pre/post-scaling.
16. Encode multiple categorical columns.
17. Detect outliers in scaled data.
18. Pipeline for Iris dataset.
19. Compare scaling methods.
20. Remove low-variance features.
21. Encode missing categorical values.
22. Scale time series data.
23. Visualize scaled feature.
24. Combine scaling/encoding pipeline.
25. Detect multi-feature outliers.
26. One-hot encode high-cardinality column.
27. Scale skewed data.
28. Compare Z-score/IQR on scaled data.
29. Preprocess numeric/categorical outliers.
30. Pipeline with scaling, encoding, feature selection.

Real-World Project: *Customer Segmentation Preprocessing*
Description: Preprocess customer data for clustering.
Dataset: CSV with `Age`, `Income`, `Purchase_Category` (e.g., Kaggle Customer Dataset).
Steps:

1. Load customer data.
2. Scale `Age`, `Income` with StandardScaler.
3. Encode `Purchase_Category` with OneHotEncoder.
4. Remove outliers using IQR.
5. Save preprocessed data.

Code Snippet:

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import pandas as pd
import numpy as np

# Load data
df = pd.read_csv("customer_data.csv")

# Preprocessing pipeline
numeric_features = ["Age", "Income"]
categorical_features = ["Purchase_Category"]
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(), categorical_features)
    ])

# Apply preprocessing
preprocessed = preprocessor.fit_transform(df)
df_preprocessed = pd.DataFrame(preprocessed, columns=numeric_features + list(preprocessor.named_transformers_["cat"].get_fe

# Remove outliers (IQR)
```

```
q1, q3 = df_preprocessed["Age"].quantile([0.25, 0.75])
iqr = q3 - q1
df_preprocessed = df_preprocessed[(df_preprocessed["Age"] >= q1 - 1.5 * iqr) & (df_preprocessed["Age"] <= q3 + 1.5 * iqr)]

df_preprocessed.to_csv("preprocessed_customers.csv")
```

Evaluation Criteria:

- Correct scaling and encoding.
- Accurate outlier removal.
- Error-free CSV output.

Resources:

- Scikit-learn Documentation: https://scikit-learn.org/stable/
- Dataset: Kaggle Customer Segmentation Dataset.

Module 8: Linear Regression & KNN (Lab 10 & 11)

Duration: 6 Hours
Objective: Implement supervised learning with linear regression and KNN.
Content:

- **Linear Regression**: Predict continuous outcomes.
- **KNN**: Classify based on nearest neighbors.
- **Evaluation**: MSE, R^2, accuracy, confusion matrix.

Detailed Explanation: Linear regression models linear relationships between features and outcomes, minimizing squared errors. KNN classifies based on the majority class of k-nearest neighbors. Evaluation metrics quantify model performance.

Code Snippet (Linear Regression and KNN):

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
import pandas as pd

# Load data
df = pd.read_csv("housing.csv")
X = df[["Size", "Bedrooms"]]
y = df["Price"]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred_lr))
print("R²:", r2_score(y_test, y_pred_lr))

# KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train > y_train.median())
y_pred_knn = knn.predict(X_test)
print("Accuracy:", accuracy_score(y_test > y_test.median(), y_pred_knn))
print("Confusion Matrix:\n", confusion_matrix(y_test > y_test.median(), y_pred_knn))
```

Practice Questions:

1. Train linear regression model.
2. Compute MSE.
3. Calculate R^2.
4. Split dataset.
5. Train KNN (k=5).
6. Compute KNN accuracy.
7. Confusion matrix for KNN.
8. Predict with linear regression.
9. Visualize regression line.
10. Train KNN (k=3, k=9).
11. Compare k values.
12. Scale before KNN.
13. Plot KNN decision boundary.
14. Linear regression on diabetes dataset.
15. Compute regression coefficients.
16. KNN on Iris dataset.
17. Cross-validation for regression.
18. KNN on binary dataset.
19. Precision/recall for KNN.
20. Plot regression residuals.
21. Linear regression with one feature.
22. KNN with/without scaling.
23. Cross-validate k in KNN.

24. Visualize regression predictions.
25. KNN on wine dataset.
26. F1 score for KNN.
27. Linear regression on time series.
28. Scatter true vs. predicted.
29. KNN with weighted voting.
30. Compare regression/KNN.

Real-World Project: *House Price Prediction*
Description: Predict house prices and classify affordability.
Dataset: CSV with `Size`, `Bedrooms`, `Price` (e.g., Kaggle House Prices).
Steps:

1. Load housing data.
2. Train linear regression for price prediction.
3. Evaluate with MSE, R^2.
4. Train KNN to classify affordable/expensive.
5. Visualize predictions and boundaries.

Code Snippet:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

# Load data
df = pd.read_csv("housing.csv")
X = df[["Size", "Bedrooms"]]
y = df["Price"]

# Linear Regression
lr = LinearRegression()
lr.fit(X, y)
y_pred = lr.predict(X)

# KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y > y.median())

# Plot
plt.scatter(X["Size"], y, label="Actual")
plt.plot(X["Size"], y_pred, color="red", label="Predicted")
plt.title(f"MSE: {mean_squared_error(y, y_pred):.2f}, R²: {r2_score(y, y_pred):.2f}")
plt.legend()
plt.savefig("house_prices.png")
plt.show()
```

Evaluation Criteria:

- Accurate model training.
- Correct evaluation metrics.
- Clear visualization.

Resources:

- Scikit-learn: https://scikit-learn.org/stable/
- Dataset: Kaggle House Prices Dataset.

Module 9: Logistic Regression & Decision Trees (Lab 13)

Duration: 3 Hours
Objective: Build classifiers using logistic regression and decision trees.
Content:

- **Logistic Regression**: Model binary outcomes with sigmoid.
- **Decision Trees**: Use CART algorithm for classification.
- **Visualization**: Plot trees, evaluate with metrics.

Detailed Explanation: Logistic regression predicts probabilities using the sigmoid function, suitable for binary classification. Decision trees split data based on feature thresholds, creating interpretable models.

Code Snippet:

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

# Load data
data = load_breast_cancer()
X, y = data.data, data.target

# Logistic Regression
lr = LogisticRegression(max_iter=1000)
```

```
lr.fit(X, y)
print("Logistic Accuracy:", accuracy_score(y, lr.predict(X)))

# Decision Tree
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X, y)
print("Tree Accuracy:", accuracy_score(y, dt.predict(X)))

# Plot tree
from sklearn.tree import plot_tree
plt.figure(figsize=(10, 5))
plot_tree(dt, feature_names=data.feature_names, class_names=data.target_names)
plt.savefig("decision_tree.png")
plt.show()
```

Practice Questions:

1. Train logistic regression.
2. Compute accuracy.
3. Confusion matrix.
4. Train decision tree.
5. Visualize tree.
6. Classification report.
7. Predict probabilities.
8. Decision tree (depth=3).
9. Compare accuracy.
10. Logistic regression on breast cancer dataset.
11. Plot decision boundary.
12. Feature importance (tree).
13. Tree without depth limit.
14. Cross-validate logistic regression.
15. ROC curve.
16. Tree on Iris dataset.
17. Precision/recall (tree).
18. Multiclass logistic regression.
19. Prune tree.
20. Plot logistic coefficients.
21. Tree with entropy criterion.
22. Compare tree depths.
23. Logistic regression with L1.
24. Visualize tree splits.
25. Logistic regression with balanced weights.
26. F1 score (tree).
27. Tree on binary dataset.
28. Confusion matrix (logistic).
29. Logistic regression (high max_iter).
30. Compare tree/logistic.

Real-World Project: *Medical Diagnosis Classifier*
Description: Classify tumors as malignant/benign.
Dataset: Breast cancer dataset (Scikit-learn).
Steps:

1. Load breast cancer data.
2. Train logistic regression and decision tree.
3. Evaluate with accuracy, precision, recall, F1.
4. Visualize tree and coefficients.

Code Snippet:

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Load data
data = load_breast_cancer()
X, y = data.data, data.target

# Models
lr = LogisticRegression(max_iter=1000)
lr.fit(X, y)
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X, y)

# Evaluate
print("Logistic Report:\n", classification_report(y, lr.predict(X)))
print("Tree Report:\n", classification_report(y, dt.predict(X)))

# Plot
plt.figure(figsize=(10, 5))
plt.bar(range(len(lr.coef_[0])), lr.coef_[0])
plt.xticks(range(len(data.feature_names)), data.feature_names, rotation=90)
plt.title("Logistic Regression Coefficients")
plt.savefig("logistic_coeffs.png")
plt.show()
```

Evaluation Criteria:

- Accurate model training.
- Comprehensive evaluation.
- Clear visualizations.

Resources:

- Scikit-learn: https://scikit-learn.org/stable/
- Dataset: Scikit-learn Breast Cancer Dataset.

Module 10: Random Forest & SVM (Lab 14)

Duration: 3 Hours
Objective: Implement ensemble and kernel-based classifiers.
Content:

- **Random Forest**: Ensemble of decision trees.
- **SVM**: Kernel-based classification (RBF, linear).
- **Tuning**: GridSearchCV for hyperparameters.

Detailed Explanation: Random Forest combines multiple trees to reduce overfitting, improving robustness. SVM finds optimal hyperplanes using kernels for non-linear data. Hyperparameter tuning optimizes performance.

Code Snippet:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris

# Load data
data = load_iris()
X, y = data.data, data.target

# Random Forest
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X, y)
print("RF Accuracy:", accuracy_score(y, rf.predict(X)))

# SVM
svm = SVC(kernel="rbf")
svm.fit(X, y)
print("SVM Accuracy:", accuracy_score(y, svm.predict(X)))

# GridSearch
param_grid = {"C": [0.1, 1, 10], "gamma": ["scale", "auto"]}
grid = GridSearchCV(SVC(), param_grid)
grid.fit(X, y)
print("Best SVM Params:", grid.best_params_)
```

Practice Questions:

1. Train Random Forest (100 trees).
2. Compute RF accuracy.
3. RF confusion matrix.
4. Train SVM (RBF).
5. GridSearch for SVM.
6. RF feature importance.
7. RF with 50/200 trees.
8. SVM with linear kernel.
9. Compare RF/SVM accuracy.
10. SVM on breast cancer dataset.
11. SVM decision boundary.
12. RF on Iris dataset.
13. SVM classification report.
14. GridSearch for RF.
15. SVM with polynomial kernel.
16. Plot RF feature importance.
17. SVM with balanced weights.
18. Compare SVM kernels.
19. RF with entropy.
20. Cross-validate RF.
21. PCA before SVM.
22. Precision/recall for RF.
23. SVM on imbalanced data.
24. SVM confusion matrix.
25. RF with max_depth=5.
26. Compare RF/SVM.
27. GridSearch for RF trees.
28. SVM with different C.
29. RF ROC curve.
30. RF multiclass classification.

Real-World Project: *Credit Risk Assessment*
Description: Predict loan default risk.
Dataset: CSV with `Credit_Score`, `Income`, `Loan_Status` (e.g., Kaggle Credit Risk).
Steps:

1. Load credit data.
2. Train RF and SVM models.
3. Tune hyperparameters with GridSearchCV.
4. Evaluate with accuracy, precision, recall, ROC.
5. Visualize feature importance, boundaries.

Code Snippet:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, roc_curve
import matplotlib.pyplot as plt
import pandas as pd

# Load data
df = pd.read_csv("credit_data.csv")
X = df[["Credit_Score", "Income"]]
y = df["Loan_Status"]

# Models
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X, y)
svm = SVC(probability=True)
svm.fit(X, y)

# Evaluate
print("RF Report:\n", classification_report(y, rf.predict(X)))
print("SVM Report:\n", classification_report(y, svm.predict(X)))

# Plot ROC
fpr, tpr, _ = roc_curve(y, rf.predict_proba(X)[:, 1])
plt.plot(fpr, tpr)
plt.title("RF ROC Curve")
plt.savefig("roc_curve.png")
plt.show()
```

Evaluation Criteria:

- Accurate model training and tuning.
- Comprehensive evaluation.
- Clear visualizations.

Resources:

- Scikit-learn: https://scikit-learn.org/stable/
- Dataset: Kaggle Credit Risk Dataset.

Module 11: Clustering & Silhouette Score (Lab 15)

Duration: 3 Hours
Objective: Implement unsupervised learning with K-Means and Hierarchical Clustering.
Content:

- **K-Means**: Cluster based on centroids.
- **Hierarchical Clustering**: Build dendrograms (Ward's method).
- **Silhouette Score**: Evaluate clustering quality.

Detailed Explanation: Clustering groups similar data points without labels. K-Means minimizes variance within clusters, while hierarchical clustering builds a tree of clusters. Silhouette score measures cluster cohesion and separation.

Code Snippet:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load data
data = load_iris()
X = data.data

# K-Means
kmeans = KMeans(n_clusters=3)
labels = kmeans.fit_predict(X)
print("Silhouette Score:", silhouette_score(X, labels))

# Hierarchical
Z = linkage(X, method="ward")
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title("Hierarchical Clustering Dendrogram")
```

```
plt.savefig("dendrogram.png")
plt.show()
```

Practice Questions:

1. K-Means (k=3).
2. Compute silhouette score.
3. Create dendrogram.
4. Hierarchical (Ward's).
5. Visualize K-Means clusters.
6. Compute K-Means inertia.
7. K-Means (k=2, k=5).
8. Compare silhouette scores.
9. Hierarchical (single linkage).
10. Visualize K-Means centers.
11. K-Means on Iris.
12. Scatter with cluster labels.
13. Distance matrix for hierarchical.
14. Hierarchical (complete linkage).
15. Adjusted Rand index.
16. Dendrogram with leaf counts.
17. K-Means (4 features).
18. Compare K-Means/hierarchical.
19. Silhouette for hierarchical.
20. Visualize clusters.
21. K-Means random initialization.
22. Clustering comparison table.
23. Cluster scaled data.
24. Elbow curve for K-Means.
25. Hierarchical on large dataset.
26. Silhouette with outliers.
27. Truncated dendrogram.
28. K-Means on synthetic data.
29. Compare linkage methods.
30. Save clustering plot.

Real-World Project: *Customer Segmentation*
Description: Segment customers based on behavior.
Dataset: CSV with `Age`, `Income`, `Purchase_History` (e.g., Kaggle Customer Data).
Steps:

1. Load customer data.
2. Apply K-Means and hierarchical clustering.
3. Evaluate with silhouette score.
4. Visualize clusters and dendrogram.
5. Save results.

Code Snippet:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("customer_data.csv")
X = df[["Age", "Income", "Purchase_History"]]

# K-Means
kmeans = KMeans(n_clusters=3)
labels = kmeans.fit_predict(X)
print("Silhouette Score:", silhouette_score(X, labels))

# Hierarchical
Z = linkage(X, "ward")
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title("Customer Segmentation Dendrogram")
plt.savefig("customer_dendrogram.png")
plt.show()

# Save clusters
df["Cluster"] = labels
df.to_csv("segmented_customers.csv")
```

Evaluation Criteria:

- Accurate clustering.
- Correct silhouette score.
- Clear visualizations.
- Error-free CSV output.

Resources:

- Scikit-learn Clustering: https://scikit-learn.org/stable/modules/clustering.html

- Dataset: Kaggle Customer Segmentation Dataset.

Module 12: PCA & Anomaly Detection (Lab 16)

Duration: 3 Hours
Objective: Reduce dimensionality with PCA and detect anomalies.
Content:

- **PCA**: Transform to principal components.
- **Anomaly Detection**: Use Z-scores.
- **Visualization**: Plot components and anomalies.

Detailed Explanation: PCA reduces dimensionality by projecting data onto principal components, preserving variance. Anomalies are detected as extreme values using Z-scores.

Code Snippet:

```
from sklearn.decomposition import PCA
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("transaction_data.csv")
X = df[["Amount", "Time"]]

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
print("Explained Variance:", pca.explained_variance_ratio_)

# Anomalies
z_scores = np.abs((X["Amount"] - X["Amount"].mean()) / X["Amount"].std())
anomalies = X[z_scores > 3]

# Plot
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=["red" if z > 3 else "blue" for z in z_scores])
plt.title("PCA with Anomalies")
plt.savefig("pca_anomalies.png")
plt.show()
```

Practice Questions:

1. PCA (2 components).
2. Explained variance ratio.
3. Visualize PCA components.
4. PCA on Iris dataset.
5. Z-scores for anomalies.
6. Anomalies (Z-score > 3).
7. Plot PCA with anomalies.
8. PCA (3 components).
9. Covariance matrix.
10. Scree plot.
11. PCA on high-dimensional data.
12. Cumulative explained variance.
13. Anomalies in PCA data.
14. 3D PCA plot.
15. PCA on scaled data.
16. PCA eigenvalues.
17. Reconstruct PCA data.
18. Compare PCA with/without scaling.
19. Histogram of Z-scores.
20. Anomaly detection in time series.
21. Manual PCA components.
22. Explained variance bar plot.
23. PCA with missing values.
24. Different anomaly threshold.
25. Compare PCA/t-SNE.
26. PCA with categorical features.
27. Visualize PCA anomalies.
28. PCA reconstruction error.
29. PCA on large dataset.
30. Compare anomaly detection methods.

Real-World Project: *Fraud Detection System*
Description: Detect fraudulent transactions using PCA and Z-scores.
Dataset: CSV with `Amount`, `Time`, `Location` (e.g., Kaggle Credit Card Fraud).
Steps:

1. Load transaction data.
2. Apply PCA (2 components).
3. Detect anomalies with Z-scores.
4. Visualize PCA with anomalies.
5. Save results.

Code Snippet:

```
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("transaction_data.csv")
X = df[["Amount", "Time", "Location"]]

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Anomalies
z_scores = np.abs((X["Amount"] - X["Amount"].mean()) / X["Amount"].std())
df["Anomaly"] = z_scores > 3

# Plot
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df["Anomaly"].map({True: "red", False: "blue"}))
plt.title("Fraud Detection with PCA")
plt.savefig("fraud_pca.png")
plt.show()

df.to_csv("fraud_results.csv")
```

Evaluation Criteria:

- Accurate PCA transformation.
- Correct anomaly detection.
- Clear visualization.
- Error-free CSV output.

Resources:

- Scikit-learn PCA: https://scikit-learn.org/stable/modules/decomposition.html
- Dataset: Kaggle Credit Card Fraud Dataset.

Module 13: Artificial Neural Networks with Keras (Lab 17)

Duration: 3 Hours
Objective: Build and train ANNs using Keras on MNIST.
Content:

- **ANN Architecture**: Sequential, Dense, Flatten layers.
- **Training**: Adam optimizer, categorical crossentropy.
- **Evaluation**: Accuracy, loss.

Detailed Explanation: ANNs model complex relationships using layers of neurons. Keras simplifies ANN construction with high-level APIs. MNIST digit classification is a standard ANN task.

Code Snippet:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

# Model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=5, validation_split=0.2)

# Plot
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("ANN Loss Curves")
plt.legend()
plt.savefig("ann_loss.png")
plt.show()
```

Practice Questions:

1. ANN with 128 neurons.
2. Train on MNIST.
3. Test accuracy.
4. Plot loss curves.
5. 256 neurons.

6. Train for 20 epochs.
7. Use SGD optimizer.
8. Visualize 10 predictions.
9. Confusion matrix.
10. Add second hidden layer.
11. Batch size 64.
12. Plot accuracy curves.
13. ReLU activation.
14. Sparse categorical loss.
15. Evaluate subset.
16. Change learning rate.
17. Visualize weights.
18. Different neuron counts.
19. Precision/recall.
20. Add dropout.
21. Early stopping.
22. Plot learning curves.
23. Sigmoid activation.
24. Train on CIFAR-10.
25. F1 score.
26. Visualize feature maps.
27. Larger batch size.
28. Compare activation functions.
29. Save model.
30. Load model for predictions.

Real-World Project: *Digit Recognizer*
Description: Classify handwritten digits.
Dataset: MNIST (Keras).
Steps:

1. Load MNIST data.
2. Train ANN for digit classification.
3. Evaluate with accuracy, confusion matrix.
4. Visualize predictions.
5. Save model.

Code Snippet:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

# Load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

# Model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=5)

# Evaluate
y_pred = model.predict(X_test).argmax(axis=1)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Visualize predictions
plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(X_test[i], cmap="gray")
    plt.title(f"Pred: {y_pred[i]}")
plt.savefig("digit_predictions.png")
plt.show()

model.save("digit_recognizer.h5")
```

Evaluation Criteria:

- Accurate ANN training.
- Correct evaluation metrics.
- Clear prediction visualizations.
- Error-free model saving.

Resources:

- Keras Documentation: https://keras.io/
- Dataset: Keras MNIST.

Module 14: Advanced ANN Techniques (Lab 18)

Duration: 3 Hours
Objective: Enhance ANN performance with advanced techniques.
Content:

- **Activation Functions**: ReLU, sigmoid, tanh.
- **Early Stopping**: Prevent overfitting.
- **Dropout**: Randomly drop neurons.

Detailed Explanation: Advanced techniques improve ANN generalization. ReLU accelerates training, sigmoid/tanh handle specific cases, early stopping halts training when validation loss plateaus, and dropout reduces overfitting.

Code Snippet:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

# Model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation="relu"),
    Dropout(0.2),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
early_stop = EarlyStopping(patience=3)
history = model.fit(X_train, y_train, epochs=20, validation_split=0.2, callbacks=[early_stop])

# Plot
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("ANN with Dropout and Early Stopping")
plt.legend()
plt.savefig("ann_advanced.png")
plt.show()
```

Practice Questions:

1. ANN with ReLU.
2. Early stopping (patience=3).
3. Dropout (rate=0.2).
4. Compare sigmoid/ReLU.
5. ANN with tanh.
6. Dropout rate 0.5.
7. Early stopping (patience=5).
8. Plot dropout curves.
9. Multiple dropout layers.
10. Compare activation functions.
11. Early stopping on MNIST.
12. ReLU + sigmoid.
13. Visualize dropout effect.
14. High dropout rate.
15. Different monitor metric.
16. Compare loss curves.
17. Dropout + early stopping.
18. Plot dropout accuracy.
19. Custom learning rate.
20. Tanh vs. ReLU.
21. Dropout specific layer.
22. Large patience value.
23. Visualize early stopping.
24. Multiple activation functions.
25. Accuracy with/without dropout.
26. Plot dropout loss.
27. Combined dropout/early stopping.
28. Compare patience values.
29. Visualize validation loss.
30. Custom optimizer with dropout.

Real-World Project: *Customer Churn Prediction*
Description: Predict customer churn with an ANN.
Dataset: CSV with `Tenure`, `Monthly_Charges`, `Churn` (e.g., Kaggle Telco Churn).
Steps:

1. Load churn data.
2. Train ANN with ReLU, dropout, early stopping.
3. Compare activation functions.
4. Visualize training/validation curves.

5. Evaluate with accuracy, ROC.

Code Snippet:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import roc_curve
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("churn_data.csv")
X = df[["Tenure", "Monthly_Charges"]]
y = df["Churn"]

# Model
model = Sequential([
    Dense(64, activation="relu", input_shape=(X.shape[1],)),
    Dropout(0.3),
    Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
early_stop = EarlyStopping(patience=3)
history = model.fit(X, y, epochs=20, validation_split=0.2, callbacks=[early_stop])

# Plot ROC
fpr, tpr, _ = roc_curve(y, model.predict(X))
plt.plot(fpr, tpr)
plt.title("Churn Prediction ROC")
plt.savefig("churn_roc.png")
plt.show()
```

Evaluation Criteria:

- Accurate ANN training.
- Effective use of dropout/early stopping.
- Clear ROC visualization.
- Correct evaluation metrics.

Resources:

- Keras: https://keras.io/
- Dataset: Kaggle Telco Churn Dataset.

Module 15: Convolutional Neural Networks with Keras (Lab 19)

Duration: 3 Hours
Objective: Build CNNs for image classification on MNIST.
Content:

- **CNN Architecture**: Conv2D, MaxPooling2D, Flatten, Dense.
- **Training**: Adam optimizer.
- **Evaluation**: Accuracy, loss.

Detailed Explanation: CNNs excel at image tasks by learning hierarchical features. Convolutional layers extract features, pooling reduces dimensionality, and dense layers classify.

Code Snippet:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1) / 255.0
X_test = X_test.reshape(-1, 28, 28, 1) / 255.0

# Model
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=5)

# Plot
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.title("CNN Accuracy")
plt.legend()
plt.savefig("cnn_accuracy.png")
plt.show()
```

Practice Questions:

1. CNN with one conv layer.
2. Train on MNIST.
3. Add max pooling.
4. Test accuracy.
5. Visualize feature maps.
6. Train for 10 epochs.
7. Add second conv layer.
8. Plot loss curves.
9. Add dropout.
10. Batch size 64.
11. Visualize 10 predictions.
12. Confusion matrix.
13. Second max pooling.
14. ReLU activation.
15. SGD optimizer.
16. Accuracy curves.
17. Train on CIFAR-10.
18. Precision/recall.
19. Dropout after conv layers.
20. Early stopping.
21. Visualize filters.
22. Larger batch size.
23. Compare filter sizes.
24. Batch normalization.
25. Custom learning rate.
26. F1 score.
27. Visualize multiple feature maps.
28. Deeper CNN.
29. Save model.
30. Load model for predictions.

Real-World Project: *Image-Based Product Classifier*
Description: Classify product images into categories.
Dataset: MNIST or similar (e.g., Kaggle Product Images).
Steps:

1. Load product images.
2. Train CNN for classification.
3. Evaluate with accuracy, confusion matrix.
4. Visualize feature maps, predictions.
5. Save model.

Code Snippet:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

# Load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1) / 255.0
X_test = X_test.reshape(-1, 28, 28, 1) / 255.0

# Model
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation="relu"),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=5)

# Evaluate and visualize
y_pred = model.predict(X_test).argmax(axis=1)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(X_test[i].reshape(28, 28), cmap="gray")
    plt.title(f"Pred: {y_pred[i]}")
plt.savefig("product_predictions.png")
plt.show()

model.save("product_classifier.h5")
```

Evaluation Criteria:

- Accurate CNN training.
- Correct evaluation metrics.

- Clear visualizations.
- Error-free model saving.

Resources:

- Keras: https://keras.io/
- Dataset: Keras MNIST or Kaggle Product Images.

Module 16: Transfer Learning with VGG16 & ResNet50 (Lab 21)

Duration: 3 Hours
Objective: Apply transfer learning for image classification.
Content:

- **VGG16**: Pretrained model for classification.
- **ResNet50**: Fine-tune for custom datasets.
- **Data Augmentation**: ImageDataGenerator.

Detailed Explanation: Transfer learning leverages pretrained models (VGG16, ResNet50) to improve performance on small datasets. Data augmentation enhances robustness by generating varied training images.

Code Snippet:

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load VGG16
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False

# Model
model = Sequential([
    base_model,
    Flatten(),
    Dense(128, activation="relu"),
    Dense(2, activation="softmax")
])

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

# Data augmentation
datagen = ImageDataGenerator(rotation_range=20, horizontal_flip=True)
# Example: datagen.flow(X_train, y_train, batch_size=32)
```

Practice Questions:

1. Load VGG16 without top layers.
2. Fine-tune VGG16.
3. Apply data augmentation.
4. Train ResNet50.
5. Freeze VGG16 conv layers.
6. VGG16 accuracy.
7. Visualize VGG16 feature maps.
8. ResNet50 with custom output.
9. Augmentation with rotation/flipping.
10. Fine-tune ResNet50 layers.
11. VGG16 confusion matrix.
12. VGG16 batch size 32.
13. Plot loss curves.
14. Pretrained multiclass.
15. Visualize ResNet50 predictions.
16. Transfer learning on small dataset.
17. Precision/recall VGG16.
18. ResNet50 early stopping.
19. Dropout in transfer model.
20. ResNet50 feature maps.
21. VGG16 custom learning rate.
22. Compare VGG16/ResNet50.
23. Augmentation with shear/zoom.
24. Fine-tune VGG16 large dataset.
25. F1 score ResNet50.
26. Visualize training curves.
27. Pretrained binary classification.
28. Freeze ResNet50 except last block.
29. Save pretrained model.
30. Load pretrained model.

Real-World Project: *Image-Based Disease Classifier*
Description: Classify medical images for disease detection.
Dataset: CSV with image paths and labels (e.g., Kaggle Chest X-Ray).
Steps:

1. Load medical images.
2. Use VGG16/ResNet50 for classification.
3. Apply data augmentation.
4. Evaluate with accuracy, precision, recall.
5. Visualize feature maps, predictions.

Code Snippet:

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Load VGG16
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False

# Model
model = Sequential([
    base_model,
    Flatten(),
    Dense(128, activation="relu"),
    Dense(2, activation="softmax")
])

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

# Data augmentation
datagen = ImageDataGenerator(rotation_range=20, horizontal_flip=True)
# Example: model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=5)

# Evaluate
# y_pred = model.predict(X_test)
# print(classification_report(y_test, y_pred.argmax(axis=1)))

plt.title("VGG16 Training")
plt.savefig("vgg16_training.png")
plt.show()

model.save("disease_classifier.h5")
```

Evaluation Criteria:

- Accurate transfer learning.
- Effective augmentation.
- Clear visualizations.
- Correct evaluation metrics.

Resources:

- Keras Applications: https://keras.io/api/applications/
- Dataset: Kaggle Chest X-Ray Dataset.

Module 17: Text Preprocessing & Sentiment Classification (Lab 23)

Duration: 3 Hours
Objective: Preprocess text and classify sentiment.
Content:

- **Text Preprocessing**: Regex, tokenization.
- **TF-IDF**: Transform text to vectors.
- **Sentiment Classification**: Logistic regression.
- **Visualization**: Word clouds, frequency plots.

Detailed Explanation: Text preprocessing cleans and structures text data. TF-IDF weights words by importance. Logistic regression classifies sentiment based on text features.

Code Snippet:

```
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Data
texts = ["I love this movie!", "Terrible experience."]
labels = [1, 0]

# Preprocessing
texts = [re.sub(r"[^a-zA-Z\s]", "", text.lower()) for text in texts]

# TF-IDF
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(texts)

# Model
```

```
model = LogisticRegression()
model.fit(X, labels)

# Word cloud
wordcloud = WordCloud().generate(" ".join(texts))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.savefig("wordcloud.png")
plt.show()
```

Practice Questions:

1. Clean text with regex.
2. Tokenize into words.
3. Create TF-IDF matrix.
4. Train logistic regression for sentiment.
5. Create word cloud.
6. Compute word frequencies.
7. TF-IDF on reviews.
8. Sentiment classifier.
9. Plot word frequencies.
10. Clean special characters.
11. Tokenize into sentences.
12. Vocabulary size.
13. Sentiment with SVM.
14. Custom word cloud shape.
15. Apply stemming.
16. TF-IDF sentiment classifier.
17. Top 10 words.
18. Clean emojis.
19. Apply lemmatization.
20. Sentiment accuracy.
21. Sentiment confusion matrix.
22. Multiclass sentiment.
23. Visualize TF-IDF scores.
24. Preprocess large text dataset.
25. Cross-validate sentiment.
26. Sentiment distribution bar chart.
27. Precision/recall for sentiment.
28. TF-IDF with missing values.
29. Balanced class weights.
30. Positive/negative word clouds.

Real-World Project: *Movie Review Sentiment Analyzer*
Description: Classify movie reviews as positive/negative.
Dataset: CSV with `Review`, `Sentiment` (e.g., Kaggle IMDB Dataset).
Steps:

1. Load review data.
2. Preprocess (clean, tokenize, TF-IDF).
3. Train logistic regression.
4. Visualize word clouds.
5. Evaluate with accuracy, precision, recall.

Code Snippet:

```
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("movie_reviews.csv")
texts = df["Review"].apply(lambda x: re.sub(r"[^a-zA-Z\s]", "", x.lower()))
y = df["Sentiment"]

# TF-IDF and model
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(texts)
model = LogisticRegression()
model.fit(X, y)

# Word clouds
positive = " ".join(texts[y == 1])
negative = " ".join(texts[y == 0])
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(WordCloud().generate(positive), interpolation="bilinear")
plt.title("Positive Reviews")
plt.subplot(1, 2, 2)
plt.imshow(WordCloud().generate(negative), interpolation="bilinear")
plt.title("Negative Reviews")
plt.savefig("sentiment_wordclouds.png")
plt.show()

print(classification_report(y, model.predict(X)))
```

Evaluation Criteria:

- Accurate text preprocessing.
- Correct sentiment classification.
- Clear word clouds.
- Comprehensive evaluation.

Resources:

- Scikit-learn Text: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
- Dataset: Kaggle IMDB Dataset.

Module 18: Recommendation Systems & Face Detection (Lab 25)

Duration: 3 Hours
Objective: Build recommendation systems and detect faces with OpenCV.
Content:

- **Recommendation System**: Cosine similarity for user-based recommendations.
- **Face Detection**: Haar cascade classifiers.
- **Visualization**: Display detected faces.

Detailed Explanation: Recommendation systems suggest items based on user similarity (cosine similarity). Face detection uses pretrained Haar cascades to identify faces in images.

Code Snippet:

```
from sklearn.metrics.pairwise import cosine_similarity
import cv2
import matplotlib.pyplot as plt

# Recommendation
ratings = [[5, 3, 0], [4, 0, 2]]
similarity = cosine_similarity(ratings)
print("Cosine Similarity:\n", similarity)

# Face detection
img = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
faces = face_cascade.detectMultiScale(img, scaleFactor=1.1, minNeighbors=5)

# Draw rectangles
for
```

```
<style type="text/css">@media print {
    *, :after, :before {background: 0 0 !important;color: #000 !important;box-shadow: none !important;text-shadow: none !im
    a, a:visited {text-decoration: underline}
    a[href]:after {content: " (" attr(href) ")"}
    abbr[title]:after {content: " (" attr(title) ")"}
    a[href^="#"]:after, a[href^="javascript:"]:after {content: ""}
    blockquote, pre {border: 1px solid #999;page-break-inside: avoid}
    thead {display: table-header-group}
    img, tr {page-break-inside: avoid}
    img {max-width: 100% !important}
    h2, h3, p {orphans: 3;widows: 3}
    h2, h3 {page-break-after: avoid}
}
html {font-size: 12px}
@media screen and (min-width: 32rem) and (max-width: 48rem) {
    html {font-size: 15px}
}
@media screen and (min-width: 48rem) {
    html {font-size: 16px}
}
body {line-height: 1.85}
.air-p, p {font-size: 1rem;margin-bottom: 1.3rem}
.air-h1, .air-h2, .air-h3, .air-h4, h1, h2, h3, h4 {margin: 1.414rem 0 .5rem;font-weight: inherit;line-height: 1.42}
.air-h1, h1 {margin-top: 0;font-size: 3.998rem}
.air-h2, h2 {font-size: 2.827rem}
.air-h3, h3 {font-size: 1.999rem}
.air-h4, h4 {font-size: 1.414rem}
.air-h5, h5 {font-size: 1.121rem}
.air-h6, h6 {font-size: .88rem}
.air-small, small {font-size: .707em}
canvas, iframe, img, select, svg, textarea, video {max-width: 100%}
body {color: #444;font-family: 'Open Sans', Helvetica, sans-serif;font-weight: 300;margin: 0;text-align: center}
img {border-radius: 50%;height: 200px;margin: 0 auto;width: 200px}
a, a:visited {color: #3498db}
a:active, a:focus, a:hover {color: #2980b9}
pre {background-color: #fafafa;padding: 1rem;text-align: left}
blockquote {margin: 0;border-left: 5px solid #7a7a7a;font-style: italic;padding: 1.33em;text-align: left}
li, ol, ul {text-align: left}
p {color: #777}</style>
```