ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

◇ TEXT PREPROCESSING: TOKENIZATION, STEMMING, LEMMATIZATION

📌 CHAPTER 1: INTRODUCTION TO TEXT PREPROCESSING

🔸 1.1 What is Text Preprocessing?

Text preprocessing is the process of preparing and cleaning text data before applying **Natural Language Processing (NLP)** techniques. Raw text contains **noise, variations, and redundancies**, making it difficult for machine learning models to interpret. Preprocessing converts text into a **structured, meaningful format**.

🔸 1.2 Why is Text Preprocessing Important?

✓ **Improves Model Accuracy** – Cleaner data leads to better predictions in NLP models.
✓ **Reduces Dimensionality** – Removes unnecessary words, reducing computational cost.
✓ **Enhances Text Normalization** – Unifies words with similar meanings.
✓ **Prepares Text for Tokenization, Sentiment Analysis, and Topic Modeling.**

🔹 1.3 Common Steps in Text Preprocessing

1️⃣ **Tokenization** – Splitting text into words or sentences.

2️⃣ **Lowercasing** – Standardizing text by converting it to lowercase.

3️⃣ **Removing Punctuation & Special Characters** – Eliminating unnecessary symbols.

4️⃣ **Stopword Removal** – Filtering out common words like "the", "is", "and".

5️⃣ **Stemming & Lemmatization** – Converting words to their root form.

📌 Example:

Raw Text:

➡ "The runners were running in the park happily."

Processed Text:

➡ "runner run park happy"

💡 Conclusion:

Text preprocessing helps transform raw text into **clean and structured data** for NLP applications.

📌 CHAPTER 2: TOKENIZATION

🔹 2.1 What is Tokenization?

Tokenization is the process of splitting text into **smaller units** (tokens), such as words or sentences. These tokens are then analyzed for patterns and relationships.

✓ **Word Tokenization** – Splitting text into individual words.

✓ **Sentence Tokenization** – Splitting text into sentences.

🔸 2.2 Implementing Tokenization in Python

Install Required Libraries

pip install nltk spacy

Word Tokenization using NLTK

```
import nltk

nltk.download('punkt')

from nltk.tokenize import word_tokenize


text = "Natural Language Processing (NLP) is exciting!"

tokens = word_tokenize(text)


print("Word Tokens:", tokens)
```

📌 **Output:**

['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'is', 'exciting', '!']

Sentence Tokenization using NLTK

```
from nltk.tokenize import sent_tokenize


text = "Hello! How are you? This is an NLP tutorial."

sentences = sent_tokenize(text)


print("Sentence Tokens:", sentences)
```

📌 **Output:**

['Hello!', 'How are you?', 'This is an NLP tutorial.']

🔹 **2.3 Tokenization using SpaCy**

```
import spacy

nlp = spacy.load("en_core_web_sm")


text = "Text preprocessing is a crucial step in NLP."

doc = nlp(text)


tokens = [token.text for token in doc]

print("Word Tokens:", tokens)
```

📌 **Output:**

['Text', 'preprocessing', 'is', 'a', 'crucial', 'step', 'in', 'NLP', '.']

💡 **Conclusion:**
Tokenization **splits text into words or sentences**, making it **easier to analyze**.

📌 CHAPTER 3: STEMMING

🔹 **3.1 What is Stemming?**

Stemming reduces words to their **root form** by **removing suffixes**. However, it may not always produce real words.

✓ Reduces **inflected words** to their base form.
✓ Useful for **search engines and text mining**.
✓ Doesn't always create meaningful words.

🔸 3.2 Implementing Stemming in Python

Using Porter Stemmer (NLTK)

```
from nltk.stem import PorterStemmer


stemmer = PorterStemmer()


words = ["running", "flies", "easily", "happiness"]
stemmed_words = [stemmer.stem(word) for word in words]


print("Stemmed Words:", stemmed_words)
```

📌 **Output:**

```
['run', 'fli', 'easili', 'happi']
```

✓ **"running" → "run"**
✓ **"flies" → "fli" (not a real word)**

Using Lancaster Stemmer (More Aggressive)

```
from nltk.stem import LancasterStemmer


lancaster = LancasterStemmer()
```

```
words = ["running", "flies", "easily", "happiness"]

stemmed_words = [lancaster.stem(word) for word in words]


print("Stemmed Words:", stemmed_words)
```

📌 **Output:**

```
['run', 'fli', 'easy', 'happy']
```

💡 **Conclusion:**
Stemming **reduces words to their root but may not retain meaning**. It is useful for **text classification and search engines**.

📌 Chapter 4: Lemmatization

🔹 4.1 What is Lemmatization?

Lemmatization converts words to their **dictionary base form (lemma)**. Unlike stemming, lemmatization produces **real words**.

✓ Uses a **vocabulary-based approach**.
✓ More **accurate than stemming**.
✓ Requires **Part-of-Speech (POS) tagging**.

🔹 4.2 Implementing Lemmatization in Python

Using WordNetLemmatizer (NLTK)

```
from nltk.stem import WordNetLemmatizer

from nltk.corpus import wordnet
```

```
nltk.download('wordnet')


lemmatizer = WordNetLemmatizer()


words = ["running", "flies", "easily", "happiness"]

lemmas = [lemmatizer.lemmatize(word, pos="v") for word in words]
# 'v' = verb


print("Lemmatized Words:", lemmas)
```

📌 **Output:**

```
['run', 'fly', 'ease', 'happiness']
```

✓ **"running" → "run"**
✓ **"flies" → "fly"**
✓ **"easily" → "ease"**

Using SpaCy for Lemmatization

```
import spacy

nlp = spacy.load("en_core_web_sm")


text = "The runners are running faster than usual."

doc = nlp(text)


lemmas = [token.lemma_ for token in doc]
```

```
print("Lemmatized Text:", lemmas)
```

📌 **Output:**

```
['the', 'runner', 'be', 'run', 'fast', 'than', 'usual', '.']
```

💡 **Conclusion:**
Lemmatization **preserves the correct meaning** of words and is **better for NLP tasks** than stemming.

📌 Chapter 5: Stemming vs. Lemmatization

Feature	Stemming	Lemmatization
Definition	Removes suffixes	Converts to dictionary form
Speed	Faster	Slower
Accuracy	Less accurate	More accurate
Use Case	Search engines, quick NLP tasks	Sentiment analysis, AI chatbots

📌 **Example:**
✓ **"Caring" → Stemmed: "car", Lemmatized: "care"**
✓ **"Better" → Stemmed: "better", Lemmatized: "good"**

💡 **Conclusion:**
✓ Use **stemming** for **quick NLP tasks**.
✓ Use **lemmatization** for **AI chatbots, sentiment analysis, and deep learning**.

📌 Summary & Next Steps

✅ **Key Takeaways:**

✓ **Tokenization** splits text into **words/sentences** for NLP.

✓ **Stemming** reduces words to **root form** (may not be real words).

✓ **Lemmatization** converts words to their **dictionary base form** (more accurate).

📌 **Next Steps:**

- **Use NLP libraries (NLTK, SpaCy) for text preprocessing.**
- **Apply stemming and lemmatization on real-world datasets (tweets, news articles).**
- **Build a text classification model using preprocessed text.** 🚀

◇ Sentiment Analysis & Named Entity Recognition (NER)

📌 Chapter 1: Introduction to Sentiment Analysis & Named Entity Recognition (NER)

🔹 1.1 What is Sentiment Analysis?

Sentiment Analysis (also called **opinion mining**) is a **Natural Language Processing (NLP) technique** that determines the **emotional tone of a piece of text**. It is widely used in social media monitoring, product reviews, customer feedback analysis, and more.

Types of Sentiment Analysis:

✓ **Binary Sentiment Analysis** – Classifies text as **Positive or Negative**.
✓ **Multiclass Sentiment Analysis** – Includes **Neutral**, along with Positive and Negative.
✓ **Fine-grained Sentiment Analysis** – Uses a scale (e.g., **very positive, positive, neutral, negative, very negative**).
✓ **Aspect-based Sentiment Analysis** – Analyzes **sentiment towards specific aspects** of a product/service.

📌 Example:

✓ "I love this phone! The camera is amazing!" → **Positive**
✓ "The battery life is terrible." → **Negative**
✓ "The movie was okay, but too long." → **Neutral**

💡 Conclusion:

Sentiment Analysis helps businesses **understand customer emotions, track brand reputation, and improve user experiences**.

🔹 1.2 What is Named Entity Recognition (NER)?

Named Entity Recognition (NER) is an NLP technique that **identifies and classifies entities in text** into predefined categories, such as:

✓ **Person Names** → "Elon Musk", "Barack Obama"
✓ **Organizations** → "Google", "NASA", "Apple Inc."
✓ **Locations** → "New York", "Mount Everest"
✓ **Dates & Time** → "January 1st, 2023", "5 PM"
✓ **Monetary Values** → "$100", "₹5000"
✓ **Product Names** → "iPhone 13", "Tesla Model S"

📌 **Example:**
✓ **Sentence:** "Amazon was founded by Jeff Bezos in 1994."
✓ **NER Output:**

- **Organization:** Amazon
- **Person:** Jeff Bezos
- **Date:** 1994

💡 **Conclusion:**
NER is crucial for **automated information extraction, chatbot development, and knowledge graph creation**.

📌 CHAPTER 2: SENTIMENT ANALYSIS USING NLP

🔹 2.1 Preprocessing Text for Sentiment Analysis

Before applying Sentiment Analysis, we need to **clean and preprocess** the text.

Key Preprocessing Steps:

✓ **Lowercasing** – Convert text to lowercase.
✓ **Removing Punctuation & Special Characters** – Remove , . ? ! etc.
✓ **Tokenization** – Splitting text into words.
✓ **Stopword Removal** – Removing common words (e.g., "the", "is", "and").
✓ **Lemmatization** – Convert words to base form ("running" → "run").

Example of Preprocessing:

```
import re

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')


# Define preprocessing function

def preprocess_text(text):

  text = text.lower()  # Convert to lowercase

  text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation

  tokens = word_tokenize(text)  # Tokenization
```

```
    tokens = [word for word in tokens if word not in
stopwords.words('english')]  # Remove stopwords
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]  #
Lemmatization
    return " ".join(tokens)

# Example usage
sample_text = "The movie was absolutely amazing, I loved it!"
print(preprocess_text(sample_text))
```

📌 **Output:** "movie absolutely amazing loved"

🔸 **2.2 Sentiment Analysis using TextBlob (Simple Approach)**

```
from textblob import TextBlob

# Function to predict sentiment
def get_sentiment(text):
    score = TextBlob(text).sentiment.polarity
    if score > 0:
        return "Positive"
    elif score < 0:
        return "Negative"
    else:
```

```
    return "Neutral"


# Test cases
print(get_sentiment("The product is fantastic!"))
print(get_sentiment("I hate the slow customer service."))
print(get_sentiment("It's an okay experience."))
```

📌 **Output:**

✓ "Positive"

✓ "Negative"

✓ "Neutral"

🔹 **2.3 Sentiment Analysis using Machine Learning (TF-IDF + Logistic Regression)**

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Sample dataset
data = {'text': ["I love this phone!", "This is the worst movie ever.",
"The food was okay.",
        "Amazing experience!", "Horrible customer service."],
    'label': [1, 0, 2, 1, 0]}  # 1 = Positive, 0 = Negative, 2 = Neutral
```

```
df = pd.DataFrame(data)

# Convert text into numerical features
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['text'])
y = df['label']

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

# Train model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

📌 **Output:**

✓ Accuracy of the model on test data (e.g., **80-90%**).

💡 **Conclusion:**

Machine learning improves **sentiment classification** by learning from labeled data.

📌 Chapter 3: Named Entity Recognition (NER) with SpaCy

🔹 3.1 Implementing NER using SpaCy

```
import spacy

# Load pre-trained SpaCy model
nlp = spacy.load("en_core_web_sm")

# Sample text
text = "Elon Musk is the CEO of Tesla and SpaceX, and he lives in California."

# Process text
doc = nlp(text)

# Print Named Entities
for ent in doc.ents:
    print(f"{ent.text} → {ent.label_}")
```

📌 **Output:**

Elon Musk ⟶ PERSON

Tesla ⟶ ORG

SpaceX ⟶ ORG

California ⟶ GPE

✓ **PERSON** – Names of people.
✓ **ORG** – Organizations.
✓ **GPE** – Geographic locations.

🔸 **3.2 Custom Named Entity Recognition Model (Training on New Data)**

We can **train a custom NER model** to recognize new entity types (e.g., product names, chemical compounds).

```
import spacy

from spacy.training import Example


# Load blank model

nlp = spacy.blank("en")


# Create training data

train_data = [

  ("Apple released a new iPhone.", {"entities": [(0, 5, "ORG"), (20, 26, "PRODUCT")]}),
```

```
  ("Google acquired Fitbit.", {"entities": [(0, 6, "ORG"), (15, 21, "PRODUCT")]}),
]

# Add NER component
ner = nlp.add_pipe("ner", last=True)
ner.add_label("PRODUCT")  # Add custom label

# Train model
optimizer = nlp.begin_training()
for _ in range(10):
  for text, annotations in train_data:
    example = Example.from_dict(nlp.make_doc(text), annotations)
    nlp.update([example], drop=0.5)

# Test new model
doc = nlp("Samsung launched Galaxy S21.")
for ent in doc.ents:
  print(f"{ent.text} → {ent.label_}")
```

📌 **Output:**

Samsung → ORG

Galaxy S21 → PRODUCT

💡 **Conclusion:**

NER is **customizable** and can be trained for **specific entity recognition**.

📌 Summary & Next Steps

✅ **Key Takeaways:**

✓ **Sentiment Analysis** predicts emotions from text.

✓ **NER identifies and categorizes key entities.**

✓ **Machine learning models improve sentiment detection.**

✓ **SpaCy enables custom Named Entity Recognition.**

📌 **Next Steps:**

🔹 **Train a deep learning model (LSTM, BERT) for sentiment analysis.**

🔹 **Expand NER to include domain-specific entities (medicine, finance, law).**

🔹 **Deploy sentiment analysis as a chatbot feature.** 🚀

◇ Building AI Chatbots with NLP

📌 Chapter 1: Introduction to AI Chatbots

🔹 1.1 What is an AI Chatbot?

An **AI chatbot** is a software application that uses **Natural Language Processing (NLP)** and **Machine Learning (ML)** to understand and generate human-like conversations. AI chatbots are used for **customer support, virtual assistants, e-commerce, healthcare, and education**.

Types of Chatbots:

1. **Rule-Based Chatbots** – Follow predefined rules and responses.
2. **AI-Powered Chatbots** – Use **NLP & ML** to generate responses dynamically.
3. **Retrieval-Based Chatbots** – Select the best response from a set of predefined responses.
4. **Generative Chatbots** – Use **Deep Learning models** like GPT to generate human-like text.

Key Features of AI Chatbots:

✓ **Understand user input** – NLP enables text analysis.
✓ **Context retention** – Memory-based AI for smarter responses.
✓ **Multilingual support** – Translate and process multiple languages.
✓ **Voice integration** – Supports speech recognition & text-to-speech.

📌 **Example:**
✓ **Siri & Alexa** – AI-powered voice chatbots.
✓ **Customer Support Bots** – Used by banks, e-commerce sites.

💡 **Conclusion:**
AI chatbots **improve user experience, automate tasks, and enhance customer engagement**.

📌 Chapter 2: Fundamentals of Natural Language Processing (NLP)

🔹 2.1 What is NLP?

Natural Language Processing (NLP) is a branch of AI that enables machines to understand and process human language. It helps chatbots extract meaning from user input and respond intelligently.

Key Components of NLP in Chatbots:

✓ **Tokenization** – Splitting text into words or sentences.
✓ **Stemming & Lemmatization** – Reducing words to their base form (e.g., "running" → "run").
✓ **Part-of-Speech (POS) Tagging** – Identifying nouns, verbs, adjectives.
✓ **Named Entity Recognition (NER)** – Detecting entities (e.g., locations, dates, names).
✓ **Sentiment Analysis** – Understanding user emotions (positive, negative, neutral).
✓ **Intent Recognition** – Identifying the intent behind user queries.

📌 **Example:**
✓ NLP allows a chatbot to understand:
User: "What's the weather like today?"
Chatbot: "The temperature today is 25°C with clear skies."

💡 **Conclusion:**
NLP enables chatbots to **process, analyze, and respond to human language intelligently**.

📌 Chapter 3: Steps to Build an AI Chatbot

🔹 3.1 Chatbot Development Pipeline

✓ **Step 1**: **Define Chatbot Objectives** – Set chatbot goals (e.g., answering FAQs, customer support).

✓ **Step 2**: **Preprocess Input Data** – Clean, tokenize, and vectorize text.

✓ **Step 3**: **Train an NLP Model** – Use machine learning (NLTK, spaCy, Transformers).

✓ **Step 4**: **Develop Response Generation** – Retrieval-based or generative chatbot.

✓ **Step 5**: **Integrate API & Deployment** – Deploy chatbot on cloud or messaging platforms.

📌 Example:

✓ A chatbot for an **e-commerce store** can assist users with **product recommendations, order tracking, and FAQs**.

💡 Conclusion:

A structured pipeline helps in **building, training, and deploying efficient AI chatbots**.

📌 Chapter 4: Implementing a Basic AI Chatbot

🔹 4.1 Step 1: Install Required Libraries

```
import numpy as np

import nltk

import random

import string
```

```
import re


from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords
```

4.2 Step 2: Load and Preprocess Data

```
nltk.download('punkt')

nltk.download('stopwords')


# Sample chatbot responses

responses = [

  "Hello! How can I help you?",

  "I'm an AI chatbot. Ask me anything!",

  "I'm here to assist you with your queries.",

  "Thank you for chatting with me!"

]


# Preprocessing function

def preprocess_text(text):

  text = text.lower()  # Convert to lowercase
```

```
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = text.translate(str.maketrans("", "", string.punctuation))  # Remove punctuation
    tokens = word_tokenize(text)  # Tokenize words
    text = [word for word in tokens if word not in stopwords.words('english')]  # Remove stopwords
    return " ".join(text)

# Example
print(preprocess_text("Hello! How are you today?"))
```

4.3 Step 3: Implement a Simple Rule-Based Chatbot

```
def chatbot_response(user_input):
    user_input = preprocess_text(user_input)
    responses_lower = [preprocess_text(resp) for resp in responses]

    # Convert text to vectors using TF-IDF
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform([user_input] + responses_lower)

    # Compute similarity scores
    similarity_scores = cosine_similarity(vectors[0], vectors[1:])
```

```
    # Get best response
    best_response_idx = similarity_scores.argmax()

    return responses[best_response_idx]

# Chatbot interaction loop
while True:
    user_input = input("You: ")
    if user_input.lower() in ["exit", "quit", "bye"]:
        print("Chatbot: Goodbye!")
        break
    response = chatbot_response(user_input)
    print("Chatbot:", response)
```

📌 **Example:**
✓ **User:** "Hi"
✓ **Chatbot:** "Hello! How can I help you?"

💡 **Conclusion:**
Rule-based chatbots provide **quick and simple responses but lack deep understanding**.

📌 Chapter 5: Implementing an AI Chatbot with RNN/LSTM

🔹 5.1 Step 1: Train an LSTM Model for Chatbot Responses

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

from tensorflow.keras.preprocessing.sequence import
pad_sequences

from tensorflow.keras.preprocessing.text import Tokenizer


# Sample chatbot training data

training_sentences = ["Hello!", "How are you?", "What can you do?",
"Goodbye!"]

training_labels = [0, 1, 2, 3]  # Assign numeric labels


# Tokenize the sentences

tokenizer = Tokenizer()

tokenizer.fit_on_texts(training_sentences)

total_words = len(tokenizer.word_index) + 1


# Convert text to sequences

input_sequences =
tokenizer.texts_to_sequences(training_sentences)

input_sequences = pad_sequences(input_sequences,
padding="post")


# Define LSTM model

model = Sequential([
```

```
    Embedding(total_words, 16, input_length=input_sequences.shape[1]),
    LSTM(16, return_sequences=True),
    LSTM(16),
    Dense(16, activation="relu"),
    Dense(4, activation="softmax")  # 4 possible responses
])

# Compile Model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train Model
model.fit(input_sequences, np.array(training_labels), epochs=100, verbose=0)

print("Chatbot model trained successfully!")
```

📌 **Key Features:**

✓ Uses **LSTM (Long Short-Term Memory)** networks to **understand context**.

✓ Provides **more accurate responses** compared to rule-based chatbots.

📌 Summary & Next Steps

✅ **Key Takeaways:**

✓ **AI Chatbots use NLP to understand and generate human-like conversations.**

✓ **Rule-Based Chatbots** are simple but lack learning capability.

✓ **AI Chatbots with Deep Learning** provide more advanced interactions.

✓ **Deploying Chatbots** on cloud services enables real-time AI assistance.

📌 **Next Steps:**

🔹 **Train an AI chatbot with a large dataset (e.g., Cornell Movie Dialogs Corpus).**

🔹 **Implement advanced NLP techniques like Transformers (BERT, GPT).**

🔹 **Deploy chatbot using Flask & integrate it with WhatsApp, Telegram, or Messenger.** 🚀

◇ Recommendation Systems: Personalized User Experience

📌 Chapter 1: Introduction to Recommendation Systems

🔹 1.1 What is a Recommendation System?

A **Recommendation System** is an AI-driven technique that suggests relevant content, products, or services to users based on their preferences, behaviors, or past interactions. It helps businesses **personalize user experiences, increase engagement, and boost sales**.

Why are Recommendation Systems Important?

✓ **Enhances User Experience** – Provides personalized recommendations, making it easier for users to find relevant content.
✓ **Increases Engagement & Retention** – Personalized suggestions improve customer satisfaction and retention rates.
✓ **Boosts Revenue** – Drives sales by recommending the right products at the right time.
✓ **Optimizes Content Discovery** – Helps users discover content they might have otherwise missed.

Where are Recommendation Systems Used?

✓ **E-commerce** – Amazon, eBay (product recommendations).
✓ **Streaming Services** – Netflix, YouTube, Spotify (movie & music suggestions).
✓ **Social Media** – Facebook, Instagram, LinkedIn (friend and content recommendations).
✓ **Online Education** – Udemy, Coursera (course recommendations).

📌 **Example:**
Netflix uses **collaborative filtering and deep learning models** to recommend movies based on user viewing history.

💡 **Conclusion:**
Recommendation systems are **essential for modern digital businesses** to enhance user engagement and improve sales.

📌 Chapter 2: Types of Recommendation Systems

🔹 2.1 Content-Based Filtering

Content-Based Filtering recommends items based on the **features of the items and the user's preferences**. It assumes that if a user likes an item, they will like similar items.

How It Works:

✓ Extracts **features** (e.g., genre, keywords, price).
✓ Compares user preferences with item features.
✓ Uses similarity metrics (Cosine Similarity, TF-IDF) to rank items.

📌 **Example:**
✓ A user watches **science-fiction movies** on Netflix ⟶ The system recommends **other sci-fi movies**.

🔹 2.2 Collaborative Filtering

Collaborative Filtering (CF) recommends items based on **user interactions and similarities** between users or items.

Types of Collaborative Filtering:

1️⃣User-Based Collaborative Filtering

- Finds users with **similar tastes** and recommends items liked by similar users.
- Example: **"People who liked this also liked that."**

2. Item-Based Collaborative Filtering

- Finds items that are **similar to what a user has interacted with**.
- Example: **Amazon's "Customers who bought this also bought..."**

📌 **Example:**
✔ A user buys a **gaming laptop** → The system recommends **gaming accessories**.

🔹 2.3 Hybrid Recommendation Systems

Hybrid models combine **content-based and collaborative filtering** to provide better recommendations.

📌 **Example:**
✔ Netflix **blends collaborative filtering and deep learning models** to personalize recommendations.

💡 **Conclusion:**
Choosing the right **recommendation technique depends on data availability and business goals**.

📌 Chapter 3: Building a Recommendation System in Python

🔹 3.1 Dataset Selection

We will use the **MovieLens dataset,** which contains **user ratings for movies**.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics.pairwise import cosine_similarity
```

Load the Data

```
# Load Movie Ratings dataset

movies = pd.read_csv("movies.csv")  # Movie information

ratings = pd.read_csv("ratings.csv")  # User ratings


# Merge datasets

df = pd.merge(ratings, movies, on="movieId")


# Display first few rows

df.head()
```

3.2 Building a Content-Based Recommendation System

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import linear_kernel


# Convert genres to lowercase
```

```
df['genres'] = df['genres'].str.lower()


# Convert movie genres into numerical features
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(df['genres'])


# Compute similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)


# Function to recommend movies
def recommend_movies(movie_title, num_recommendations=5):
    movie_idx = df[df['title'] == movie_title].index[0]
    similarity_scores = list(enumerate(cosine_sim[movie_idx]))
    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)
    recommended_movies = [df['title'][i[0]] for i in similarity_scores[1:num_recommendations+1]]
    return recommended_movies


# Test the Recommendation System
print(recommend_movies("Toy Story (1995)"))
```

📌 **Expected Output:**

['A Bug's Life (1998)', 'Toy Story 2 (1999)', 'Monsters, Inc. (2001)', 'Finding Nemo (2003)', 'The Incredibles (2004)']

✓ This model **recommends movies based on genre similarity**.

🔸 3.3 Building a Collaborative Filtering Recommendation System

```
from surprise import SVD

from surprise import Dataset, Reader

from surprise.model_selection import train_test_split

from surprise import accuracy


# Load data into Surprise format

reader = Reader(rating_scale=(0.5, 5.0))

data = Dataset.load_from_df(df[['userId', 'movieId', 'rating']], reader)


# Train-Test Split

trainset, testset = train_test_split(data, test_size=0.2, random_state=42)


# Train a Collaborative Filtering Model

model = SVD()

model.fit(trainset)
```

```
# Make Predictions
predictions = model.test(testset)


# Evaluate Model
rmse = accuracy.rmse(predictions)
print(f"RMSE: {rmse}")
```

📌 **Example Output:**

RMSE: 0.84

✓ This means the model's predictions are **close to real user ratings**.

💡 **Conclusion:**
Collaborative filtering **predicts user preferences based on previous interactions**.

📌 Chapter 4: Deploying a Recommendation API

🔹 4.1 Using Flask to Deploy the Model

```
from flask import Flask, request, jsonify


app = Flask(__name__)


@app.route('/recommend', methods=['GET'])
def recommend():
    movie_title = request.args.get('title')
    recommendations = recommend_movies(movie_title)
```

```
    return jsonify(recommendations)


if __name__ == '__main__':
    app.run(debug=True)
```

📌 **Example API Call:**

http://127.0.0.1:5000/recommend?title=Toy Story (1995)

✓ This API returns **recommended movies for a given movie title**.

📌 Chapter 5: Future Trends in Recommendation Systems

🔹 5.1 Advanced Techniques

✓ **Deep Learning (Neural Networks)** – Netflix and YouTube use deep learning-based recommendation models.
✓ **Reinforcement Learning** – Helps **optimize personalized recommendations** in real-time.
✓ **Hybrid Filtering with Graph Neural Networks (GNNs)** – Used by **LinkedIn for connection recommendations**.
✓ **Real-time Personalization** – AI-driven systems adapt recommendations dynamically based on user behavior.

📌 **Example:**
✓ Spotify uses **Reinforcement Learning and Neural Networks** to provide **personalized song recommendations**.

💡 **Conclusion:**
The future of recommendation systems lies in **AI-powered real-time personalization**.

📌 Summary & Next Steps

✅ **Key Takeaways:**

✓ **Content-Based Filtering** recommends items based on item similarity.

✓ **Collaborative Filtering** recommends items based on user behavior.

✓ **Hybrid Systems** combine multiple techniques for better accuracy.

✓ **Advanced AI techniques** like deep learning enhance recommendations.

📌 **Next Steps:**

🔹 **Apply recommendation systems to real-world datasets (e.g., Netflix, Amazon, Yelp).**

🔹 **Train deep learning-based recommenders using TensorFlow & PyTorch.**

🔹 **Deploy recommendation models as APIs for real-world applications.** 🚀

◇ Real-World AI Applications in Business & Healthcare

📌 Chapter 1: Introduction to AI in Business & Healthcare

🔹 1.1 What is Artificial Intelligence (AI)?

Artificial Intelligence (AI) is a branch of computer science that enables machines to **mimic human intelligence** by learning from data, recognizing patterns, and making decisions. AI is revolutionizing industries by **automating processes, improving efficiency, and enhancing decision-making**.

Why is AI Important?

✓ **Automation** – Reduces manual effort and human errors.
✓ **Personalization** – Tailors user experiences in marketing, healthcare, and finance.
✓ **Predictive Analytics** – Anticipates future trends and behaviors.
✓ **Cost Efficiency** – Reduces operational expenses while increasing productivity.

Common AI Techniques Used in Business & Healthcare

✓ **Machine Learning (ML)** – Algorithms learn from data to make predictions.
✓ **Deep Learning** – Uses neural networks for complex problem-solving.
✓ **Natural Language Processing (NLP)** – Enables AI to understand human language.
✓ **Computer Vision** – AI interprets images and videos.

📌 **Example:**
Netflix uses **AI-driven recommendation algorithms** to suggest personalized content to users, increasing engagement and customer retention.

💡 **Conclusion:**
AI is **transforming industries** by enhancing decision-making, optimizing operations, and providing personalized services.

📌 Chapter 2: AI Applications in Business

🔹 2.1 AI in Customer Service

AI-powered chatbots and virtual assistants **enhance customer support by providing instant responses, reducing wait times, and improving user experience**.

How AI is Used in Customer Service?

✓ **Chatbots** – AI-powered bots answer customer inquiries.
✓ **Voice Assistants** – AI-driven virtual assistants like **Alexa & Siri** improve customer interactions.
✓ **Sentiment Analysis** – AI detects customer emotions and improves responses.

📌 **Example:**
✓ **Banking Sector:** AI chatbots like **Bank of America's "Erica"** assist customers with transactions.
✓ **E-commerce: Amazon's Alexa** helps users track orders and shop hands-free.

💡 **Conclusion:**
AI-driven customer service improves **response time, reduces costs, and enhances customer experience**.

🔹 2.2 AI in Marketing & Sales

AI helps businesses optimize **advertising, customer segmentation, and lead generation**.

How AI Improves Marketing?

✓ **Recommendation Engines** – Suggests products based on user behavior (e.g., Amazon, Netflix).
✓ **Personalized Ads** – AI targets the right audience for marketing campaigns.
✓ **AI-Driven Content Creation** – Generates personalized marketing copy.
✓ **Customer Segmentation** – Analyzes customer behavior to create targeted campaigns.

📌 Example:

✓ **Spotify** uses AI to analyze users' listening history and suggest new songs.
✓ **Google Ads** uses AI-powered **predictive analytics** to optimize ad targeting.

💡 Conclusion:

AI enables businesses to **target the right audience, increase engagement, and boost sales** through personalization.

🔹 2.3 AI in Finance & Banking

AI enhances **fraud detection, risk assessment, and financial planning**.

AI in Finance Applications

✓ **Fraud Detection** – AI identifies fraudulent transactions in real-time.

✓ **Algorithmic Trading** – AI-powered bots execute trades based on market patterns.

✓ **Credit Scoring & Loan Approval** – AI evaluates credit risk using alternative data sources.

✓ **AI-Powered Robo-Advisors** – Provides investment advice based on market trends.

📌 **Example:**

✓ **JP Morgan Chase** uses AI for fraud detection and risk assessment.

✓ **PayPal** utilizes AI to detect **suspicious financial activities** and prevent cyber fraud.

💡 **Conclusion:**

AI helps **banks, fintech startups, and investment firms** improve **security, efficiency, and profitability**.

🔹 **2.4 AI in Supply Chain & Logistics**

AI **optimizes inventory management, route planning, and demand forecasting**.

How AI Helps in Logistics?

✓ **Predictive Maintenance** – AI detects potential failures in machinery.

✓ **Automated Warehouses** – AI-powered robots streamline warehouse operations.

✓ **Route Optimization** – AI reduces delivery costs and time.

✓ **Inventory Forecasting** – AI predicts demand trends.

📌 **Example:**

✓ **Amazon's AI-driven warehouse robots** optimize logistics and order fulfillment.

✓ **FedEx uses AI** for **real-time package tracking and delivery optimization**.

💡 **Conclusion:**

AI enhances **efficiency, reduces operational costs, and improves accuracy** in logistics and supply chain management.

📌 Chapter 3: AI Applications in Healthcare

🔹 3.1 AI in Medical Diagnosis

AI improves **medical imaging analysis, early disease detection, and clinical decision-making**.

AI in Medical Diagnosis Applications

✓ **AI-Powered Radiology** – Detects diseases in X-rays, MRIs, and CT scans.

✓ **Early Cancer Detection** – AI identifies cancerous cells faster than human doctors.

✓ **AI-Assisted Pathology** – Helps in diagnosing rare diseases.

📌 **Example:**

✓ **Google's DeepMind AI** detects **eye diseases from retinal scans** with higher accuracy than human doctors.

✓ **IBM Watson Health** assists oncologists in **predicting cancer treatment outcomes**.

💡 **Conclusion:**

AI enhances **early disease detection, speeds up diagnosis, and improves patient outcomes**.

🔹 3.2 AI in Drug Discovery & Development

AI speeds up **drug discovery** by analyzing chemical compounds and predicting drug effectiveness.

AI in Drug Development Applications

✓ **Predicts Drug Efficacy** – AI analyzes drug interactions.
✓ **Identifies Potential Drug Candidates** – AI screens thousands of compounds.
✓ **Accelerates Clinical Trials** – AI predicts patient responses to drugs.

📌 **Example:**
✓ **Pfizer** uses AI to develop new **COVID-19 vaccines and antiviral treatments**.
✓ **BenevolentAI** identifies drug repurposing opportunities using AI.

💡 **Conclusion:**
AI **reduces the time and cost of drug development,** leading to faster medical advancements.

🔹 3.3 AI in Personalized Medicine

AI enables **customized treatments based on genetic and lifestyle factors**.

AI in Personalized Medicine Applications

✓ **Genomic AI Analysis** – Analyzes patient DNA to personalize treatments.
✓ **AI-Based Treatment Plans** – Recommends therapies tailored to

individual patients.

✓ **AI-Powered Wearables** – Tracks patient health data in real-time.

📌 **Example:**

✓ **23andMe** uses AI for **genetic analysis and health predictions**.

✓ **Apple's HealthKit AI** monitors heart rate, sleep, and activity.

💡 **Conclusion:**

AI enhances **precision medicine, improving treatment effectiveness and patient care**.

🔹 **3.4 AI in Hospital & Patient Management**

AI **automates hospital workflows, improves efficiency, and enhances patient care**.

AI in Hospital Management Applications

✓ **AI-Powered Chatbots** – Assist patients with appointment scheduling.

✓ **AI in Electronic Health Records (EHRs)** – Automates data entry and analysis.

✓ **Predictive Healthcare Analytics** – Prevents hospital readmissions.

📌 **Example:**

✓ **Mayo Clinic** uses AI to optimize **patient scheduling and treatment planning**.

✓ **Google's Med-PaLM AI** processes **patient queries and medical documentation**.

💡 **Conclusion:**

AI improves **hospital efficiency, patient experience, and healthcare administration**.

📌 Summary & Next Steps

✅ Key Takeaways:

✓ AI **optimizes business operations, marketing, finance, and logistics**.

✓ AI **enhances healthcare through early diagnosis, personalized treatments, and hospital management**.

✓ AI is driving **automation, cost reduction, and real-time decision-making**.

📌 Next Steps:

🔹 **Explore AI-powered business solutions (e.g., ChatGPT, Google AI, IBM Watson).**

🔹 **Learn about AI frameworks like TensorFlow & PyTorch for real-world applications.**

🔹 **Apply AI techniques to real datasets in business and healthcare.** 🚀

📌 ASSIGNMENT:

☑ DEVELOP A RECOMMENDATION SYSTEM FOR E-COMMERCE PLATFORMS.

📌 Solution: Develop a Recommendation System for E-Commerce Platforms

- **Objective**

The goal of this assignment is to develop a **Recommendation System** for an **E-Commerce platform** using **collaborative filtering and content-based filtering**. The system will suggest products based on user preferences, purchase history, or product similarities.

- **Step 1: Import Required Libraries**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import csr_matrix
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
```

- **Step 2: Load and Explore the Dataset**

For this solution, we will use a **sample dataset** that contains **users, products, ratings, and reviews**. If you don't have a dataset, you can use public datasets from **Kaggle** or **Amazon Product Reviews**.

```
# Load dataset
df = pd.read_csv("ecommerce_product_ratings.csv")  # Example file


# Display first few rows
df.head()
```

Dataset Structure

UserID	ProductID	Rating	Review Text	Category
1001	P001	4.0	"Great phone!"	Electronics
1002	P003	5.0	"Loved it!"	Clothing
1003	P005	3.5	"Average quality"	Home Decor

📌 **Understanding the Data:**

✓ UserID – Identifies the customer.

✓ ProductID – Identifies the product.

✓ Rating – User's rating (1 to 5 stars).

✓ Review Text – User's review.

✓ Category – Product category.

🔹 **Step 3: Implement Content-Based Filtering**

Content-based filtering recommends products **similar to what a user has liked** based on product descriptions, categories, or features.

3.1 Convert Product Descriptions into Numerical Vectors

```
# Convert text data into numerical format using TF-IDF
tfidf_vectorizer = TfidfVectorizer(stop_words='english')

# Transform product descriptions into TF-IDF matrix
product_tfidf_matrix = tfidf_vectorizer.fit_transform(df["Review Text"].fillna(""))

# Compute Cosine Similarity between products
cosine_sim = cosine_similarity(product_tfidf_matrix)
```

3.2 Build a Function to Recommend Similar Products

```
# Create a product recommendation function
def recommend_similar_products(product_id, num_recommendations=5):
    # Get index of the given product
    product_index = df[df["ProductID"] == product_id].index[0]

    # Get similarity scores for all products
    similarity_scores = list(enumerate(cosine_sim[product_index]))

    # Sort products based on similarity scores
    sorted_products = sorted(similarity_scores, key=lambda x: x[1], reverse=True)[1:num_recommendations+1]
```

```
    # Get product IDs
    recommended_products = [df.iloc[i[0]]["ProductID"] for i in
sorted_products]

    return recommended_products

# Test the Recommendation System
print("Recommended Products:",
recommend_similar_products("P001"))
```

📌 **Example Output:**

Recommended Products: ['P003', 'P007', 'P005']

💡 **Conclusion:**

✓ Content-based filtering suggests products **similar to past purchases or reviewed items**.

🔹 **Step 4: Implement Collaborative Filtering**

Collaborative filtering recommends products based on **user interactions** rather than product descriptions.

4.1 Prepare Data for Collaborative Filtering

```
# Prepare dataset for collaborative filtering
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df[['UserID', 'ProductID', 'Rating']],
reader)
```

```
# Split dataset into training and testing sets
trainset, testset = train_test_split(data, test_size=0.2)
```

4.2 Train a Collaborative Filtering Model (SVD Algorithm)

```
# Train Singular Value Decomposition (SVD) model
model = SVD()
model.fit(trainset)
```

4.3 Build a Function to Recommend Products for Users

```
# Function to get top N recommendations for a user
def recommend_products_for_user(user_id, num_recommendations=5):
    # Get all unique product IDs
    product_ids = df["ProductID"].unique()

    # Predict ratings for all products
    predictions = [model.predict(user_id, product_id) for product_id in product_ids]

    # Sort predictions based on estimated ratings
    sorted_predictions = sorted(predictions, key=lambda x: x.est, reverse=True)[:num_recommendations]
```

```
    # Return top recommended product IDs
    recommended_products = [pred.iid for pred in sorted_predictions]
    return recommended_products

# Test the recommendation system for a user
print("Recommended Products for User 1001:", recommend_products_for_user(1001))
```

📌 **Example Output:**

Recommended Products for User 1001: ['P009', 'P002', 'P008']

💡 **Conclusion:**

✓ Collaborative filtering recommends products **based on what similar users have liked or rated highly**.

🔹 **Step 5: Hybrid Recommendation System**

To **combine content-based filtering and collaborative filtering**, we can take the **intersection of both recommendations**.

```
# Function to get hybrid recommendations
def hybrid_recommendation(user_id, product_id, num_recommendations=5):
    content_based = set(recommend_similar_products(product_id, num_recommendations))
    collaborative_based = set(recommend_products_for_user(user_id, num_recommendations))
```

```
    # Get intersection of both recommendations
    hybrid_recommendations = list(content_based.intersection(collaborative_based))

    # If not enough recommendations, combine both methods
    if len(hybrid_recommendations) < num_recommendations:
        hybrid_recommendations = list(content_based.union(collaborative_based))[:num_recommendations]

    return hybrid_recommendations

# Test hybrid recommendation system
print("Hybrid Recommendations for User 1001 on Product P001:", hybrid_recommendation(1001, "P001"))
```

📌 Example Output:

Hybrid Recommendations for User 1001 on Product P001: ['P009', 'P002', 'P007']

💡 Conclusion:

✓ Hybrid models **combine the strengths of content-based and collaborative filtering** to give better recommendations.

📌 Chapter 6: Deploying the Recommendation System using Flask API

6.1 Install Flask

pip install flask

6.2 Create a Flask App (app.py)

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route("/")
def home():
    return "E-Commerce Recommendation System is Running!"

@app.route("/recommend", methods=["POST"])
def recommend():
    data = request.get_json()
    user_id = data.get("user_id")
    product_id = data.get("product_id")

    recommendations = hybrid_recommendation(user_id, product_id)

    return jsonify({"recommended_products": recommendations})
```

```
if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=5000)
```

6.3 Run the Flask API Locally

python app.py

✓ API will start at **http://127.0.0.1:5000/**

6.4 Test the API using Postman or CURL

curl -X POST "http://127.0.0.1:5000/recommend" -H "Content-Type: application/json" -d '{"user_id": 1001, "product_id": "P001"}'

📌 **Expected Output:**

```
{
  "recommended_products": ["P009", "P002", "P007"]
}
```

📌 Summary & Next Steps

✅ **Key Takeaways:**

✓ **Content-Based Filtering** recommends **similar products based on descriptions**.

✓ **Collaborative Filtering** suggests **products based on user preferences**.

✓ **Hybrid Recommendation Systems** combine both approaches for better accuracy.

✓ **Flask API enables real-time product recommendations** for e-commerce platforms.

📌 **Next Steps:**

- **Deploy the API on AWS for real-world use.**
- **Use Deep Learning models like Neural Collaborative Filtering (NCF).**
- **Integrate the system with an e-commerce website.** 🚀