ML and NLP
with Python

Python Libraries

NumPy	**Numerical computing, arrays**
Pandas	**Data manipulation**
Matplotlib	**Data visualization**
Seaborn	**Statistical data visualization**
Scikit-Learn	**Machine learning algorithms**
TensorFlow	Deep learning, neural networks
Keras	High-level API for deep learning
PyTorch	Deep learning (research-focused)
XGBoost	Gradient boosting for structured data
LightGBM	Fast boosting algorithm
OpenCV	Computer vision and image processing
NLTK	**Natural language processing**
SpaCy	Advanced NLP

scikit

- scikit-learn (sklearn) is a powerful machine learning library in Python that provides tools for:
  Data Preprocessing (handling missing data, scaling, encoding)
  Feature Extraction (Bag of Words, TF-IDF, PCA)
  Supervised Learning (Regression & Classification models)
  Unsupervised Learning (Clustering, Anomaly Detection)
  Model Selection & Evaluation (Cross-validation, Hyperparameter tuning)

- Task 1: Load & Explore a Dataset

```
import pandas as pd
df = pd.read_csv('data.csv')  # Load dataset
print(df.head())  # Show first 5 rows
print(df.info())  # Dataset summary
print(df.describe())  # Statistical summary
```

- Task 2: Train-Test Split

```
from sklearn.model_selection import train_test_split
X = df.drop('Target', axis=1)  # Features
y = df['Target']  # Labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

The random_state parameter ensures that the data split is reproducible. It controls the randomness of the train-test split, meaning:
Same random_state → Same Split Every Time
Different random_state → Different Split Every Time

- Task 3: Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)  # Train model
y_pred = model.predict(X_test)  # Make predictions
```

- from sklearn.model_selection import train_test_split
- import numpy as np

- # Create a simple dataset
- X = np.array(range(10)).reshape(-1, 1) # Features (0 to 9)
- y = np.array(range(10)) # Labels (0 to 9)

- # Split without random_state (results will change every time)
- X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

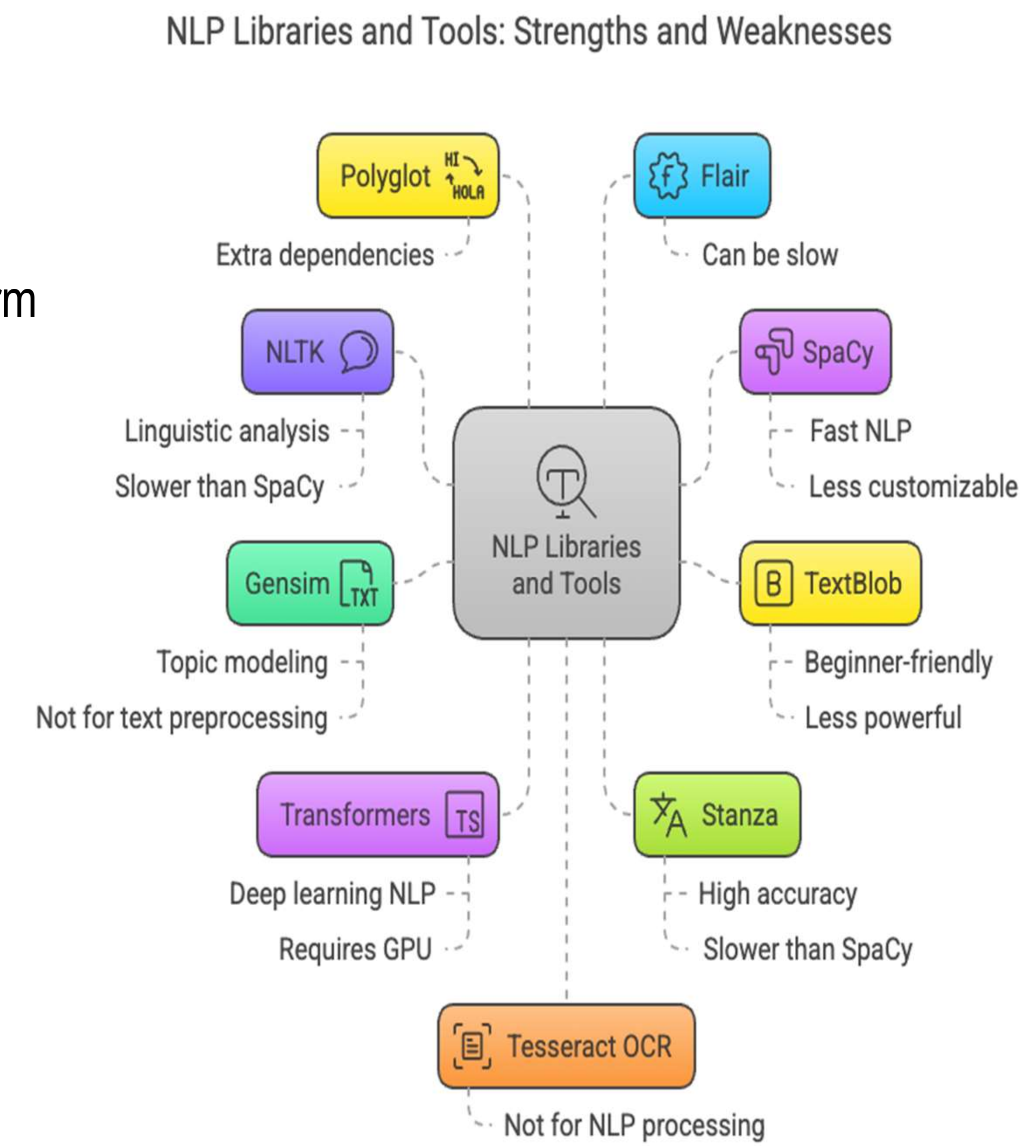
- print("X_test:", X_test.ravel()) # Different results each time

- Task 4: Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
clf = LogisticRegression()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

NLP Libraries in Python

- Python has number of libraries for NLP to perform tokenization, sentiment analysis, machine translation, text summarization, and more.
    - **NLTK (Natural Language Toolkit)**
    - **spaCy**
    - **TextBlob**
    - **Transformers (by Hugging Face)**
    - **Gensim**
    - **Tesseract OCR (for Text Extraction from Images)**
    - **Polyglot**
    - **Keras (for deep learning NLTK)**



NLP-II

BoW in Python

```
from sklearn.feature_extraction.text import CountVectorizer
texts = ["I love machine learning", "Machine learning is amazing", "I love coding"]
vectorizer = CountVectorizer()
bow = vectorizer.fit_transform(texts) //Learn the vocabulary dictionary and return document-term matrix.
print(vectorizer.get_feature_names_out())
print(bow.toarray())
```

```
['amazing' 'coding' 'is' 'learning' 'love' 'machine']
[[0 0 0 1 1 1]
 [1 0 1 1 0 1]
 [0 1 0 0 1 0]]
```

```
from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [ ... 'This is the first document.', ... 'This document is the second document.',
... 'And this is the third one.', ... 'Is this the first document?', ... ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> vectorizer.get_feature_names_out()
array(['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this'], ...)
>>> print(X.toarray())
 [[0 1 1 1 0 0 1 0 1]
[0 2 0 1 0 1 1 0 1]
[1 0 0 1 1 0 1 1 1]
[0 1 1 1 0 0 1 0 1]]
```

```
vectorizer2 = CountVectorizer(analyzer='word', ngram_range=(2, 2))
>>> X2 = vectorizer2.fit_transform(corpus)
>>> vectorizer2.get_feature_names_out()
array(['and this', 'document is', 'first document', 'is the', 'is this', 'second
document', 'the first', 'the second', 'the third', 'third one', 'this document',
'this is', 'this the'], ...)
>>> print(X2.toarray())
[[0 0 1 1 0 0 1 0 0 0 0 1 0]
[0 1 0 1 0 1 0 1 0 0 1 0 0]
[1 0 0 1 0 0 0 0 1 1 0 1 0]
[0 0 1 0 1 0 1 0 0 0 0 0 1]]
```

TF-IDF in Python

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(texts)
print(tfidf.get_feature_names_out())
print(X.toarray())
```

```
['amazing' 'coding' 'is' 'learning' 'love' 'machine’]
[[0. 0. 0. 0.57735027 0.57735027 0.57735027]
[0.5628291 0. 0.5628291 0.42804604 0. 0.42804604]
[0. 0.79596054 0. 0. 0.60534851 0. ]]
```

Similarity in Texts

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

```
text1 = set("machine learning is fun".split())
text2 = set("learning about machine intelligence".split())
jaccard = len(text1 & text2) / len(text1 | text2)
print("Jaccard Similarity:", jaccard)
```

Jaccard Similarity: 0.3333333333333333

Cosine Similarity

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

```
from sklearn.metrics.pairwise import cosine_similarity
tfidf_vec = TfidfVectorizer()
vecs = tfidf_vec.fit_transform(["machine learning is fun", "learning about machine intelligence"])
cos_sim = cosine_similarity(vecs[0:1], vecs[1:2])
print("Cosine Similarity:", cos_sim[0][0])
```

Cosine Similarity: 0.3360969272762575

Jaccard compares token sets;
Cosine compares vector angles (good for longer texts).

Sentiment Analysis

```
from textblob import TextBlob
review = "The service was excellent and the staff was friendly."
blob = TextBlob(review)
print("Polarity:", blob.sentiment.polarity)
print("Subjectivity:", blob.sentiment.subjectivity)
```

Word Cloud

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
text = "Python is simple and powerful. I love Python programming!"
wordcloud = WordCloud().generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Text Generation using Keras

```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
text = "Machine learning is fun and exciting to learn"
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
sequences = []
words = text.split()
for i in range(1, len(words)):
   seq = words[:i+1]
   tokenized_seq = tokenizer.texts_to_sequences([' '.join(seq)])[0]
   sequences.append(tokenized_seq)
# Pad the sequences
padded = pad_sequences(sequences)
print(padded)
```

Build a Model (LSTM Example)

```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
model = Sequential()
model.add(Embedding(input_dim=50, output_dim=10,
input_length=padded.shape[1]))
model.add(LSTM(50))
model.add(Dense(50, activation='relu'))
model.add(Dense(len(tokenizer.word_index) + 1, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
# Normally you'd train the model with model.fit(), then use it to predict.
```

- **Long Short-Term Memory**.
  It is a **type of Recurrent Neural Network (RNN)** that is specially designed to **remember long sequences** and patterns in data — especially useful in **Natural Language Processing (NLP)**, time series, and speech.