■ Transformers and Tokenization in Python Transformers are a deep learning model architecture widely used for NLP tasks like text classification, translation, and sentiment analysis. They utilize the **self-attention mechanism for faster and more accurate sequence processing.**

■ Tokenization Tokenization is the process of splitting text into smaller units called tokens. Types of tokenization: 1. **Word Tokenization: Splits text into words. 2. **Subword Tokenization**: Splits words into meaningful parts. 3. **Character Tokenization**: Splits text into individual characters.**

■ Python Code Examples

■ Tokenization with Transformers ```python # Install libraries pip install transformers datasets

Importing libraries from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline import torch

Load pre-trained BERT model tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased") model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased")

Tokenize sentence sentence = "I love Python programming!" tokens = tokenizer(sentence) print("Tokens:", tokens)

Decode tokens decoded = tokenizer.decode(tokens['input_ids']) print("Decoded Text:", decoded) ```

■ Sentiment Analysis with Transformers ```python # Sentiment analysis nlp = pipeline("sentiment-analysis") sentences = ["I love Python!", "This is the worst day ever."] results = nlp(sentences)

for sentence, result in zip(sentences, results): print(f"Sentence: {sentence}") print(f"Sentiment: {result['label']}, Score: {result['score']:.4f}") ```

■ Text Generation with GPT-2 ```python # GPT-2 text generation generator = pipeline("text-generation", model="gpt2") prompt = "Artificial Intelligence will" result = generator(prompt, max_length=50, num_return_sequences=1) print("Generated Text:", result[0]['generated_text']) ```

■ XGBoost with Optuna Tuning ```python import xgboost as xgb from sklearn.datasets import load_boston from sklearn.model_selection import train_test_split from sklearn.metrics import mean_squared_error import optuna

Load dataset data = load_boston() X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.2, random_state=42)

Optuna optimization function def objective(trial): params = { 'objective': 'reg:squarederror', 'eval_metric': 'rmse', 'booster': 'gbtree', 'max_depth': trial.suggest_int('max_depth', 3, 10), 'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3), 'n_estimators': trial.suggest_int('n_estimators', 100, 1000), 'subsample': trial.suggest_float('subsample', 0.5, 1.0), 'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0) }

dtrain = xgb.DMatrix(X_train, label=y_train) dtest = xgb.DMatrix(X_test, label=y_test)

model = xgb.train(params, dtrain, num_boost_round=trial.suggest_int('num_boost_round', 100, 1000)) preds = model.predict(dtest) rmse = mean_squared_error(y_test, preds, squared=False) return rmse

Run Optuna optimization study = optuna.create_study(direction='minimize') study.optimize(objective, n_trials=10)

Best parameters print("Best parameters:", study.best_params) ```

■ This document contains a complete overview of **transformers, tokenization, and XGBoost with Optuna tuning**.