**Key Terms**

**Azure CLI (Azure Command-Line Interface)**: A command-line tool that allows management and interaction with Azure services and resources.

import azure.cli.core

cli = azure.cli.core.AzureCli()

cli.invoke(["storage", "account", "list"])

**Azure ML (Azure Machine Learning) Studio**: A cloud-based environment for machine learning development and deployment.

from azureml.core import Workspace

ws = Workspace.create(name='myworkspace',

                     subscription\_id='...',

                     resource\_group='myresourcegroup',

                     location='eastus')

**Transformers**: A Hugging Face library for natural language processing tasks.

from transformers import pipeline

classifier = pipeline("sentiment-analysis")

result = classifier("I love this course!")

print(result)

**Datasets**: A Hugging Face library for loading datasets.

from datasets import load\_dataset

dataset = load\_dataset("glue", "mrpc")

train\_dataset = dataset["train"]

print(len(train\_dataset))

**Open Datasets**: An Azure resource for loading curated public datasets.

from azureml.opendatasets import **PublicHolidays**

holidays = PublicHolidays()

holidays\_df = holidays.to\_pandas\_dataframe()

print(holidays\_df.head())

**Azure CLI**

which az

az –version

az extension list

az extension add -n ml -y

az extension list

**Authenticate**:

Az login

Az account list -o table

Pip install azure-ml

# Key Terms

**Click**: A Python package for building command line interfaces.

import click

@click.command()

@click.option('--count', default=1)

def hello(count):

    for x in range(count):

        click.echo('Hello World!')

if \_\_name\_\_ == '\_\_main\_\_':

    hello()

**ArgParse**: A Python module for parsing command-line arguments.

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument('--verbosity', action='store\_true')

args = parser.parse\_args()

if args.verbosity:

    print("Verbose mode enabled")

**sys.argv**: A Python module containing command line arguments.

import sys

print(f"Script name: {sys.argv[0]}")

print(f"First argument: {sys.argv[1]}")

print(f"Second argument: {sys.argv[2]}")

**Setuptools**: A package for building and distributing Python projects.

from setuptools import setup

setup(

    name='mypackage',

    version='1.0',

    install\_requires=['requests', 'click'])

**Entry points**: Definitions linking scripts to functions in Setuptools.

from setuptools import setup

setup(

  #...,

  entry\_points = {

    'console\_scripts': [

      'myscript = mypackage.mymodule:main\_func',

    ]

  }

)

**Summary of Lesson**

This lesson covered building automation by creating reusable command-line interface (CLI) tools in Python. It showed parsing command arguments, handling flags, declaring dependencies, packaging projects, and using frameworks like Click and ArgParse.

**Top 3 Key Points**

* CLI tools allow automating tasks instead of manual work
* Frameworks like Click simplify building robust CLIs
* Packaging projects enables sharing and reusing CLIs

**APIs**

gunicorn

apache

uvicorn

# Key Terms

**FastAPI**: A Python web framework for building APIs.

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

def read\_root():

    return {"message": "Hello World"}

print(read\_root())

**Flask**: A lightweight Python web framework.

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def home():

    return "Home Page"

if \_\_name\_\_ == "\_\_main\_\_":

    app.run()

**Transforms**: Hugging Face library for NLP tasks.

from transformers import pipeline

classifier = pipeline("sentiment-analysis")

result = classifier("I love this course")

print(result)

**Onyx**: ML model server for high-performance inferencing.

import onnxruntime as rt

sess = rt.InferenceSession("model.onnx")

input\_name = sess.get\_inputs()[0].name

res = sess.run(None, {input\_name: x})

**OpenAPI**: Specification for API documentation.

from fastapi import FastAPI

from fastapi.openapi.utils import get\_openapi

app = FastAPI()

def custom\_openapi():

    return get\_openapi(

        title="Custom title",

        version="2.5.0",

        description="Custom description",)

app.openapi = custom\_openapi

**Summary of Lesson**

This lesson covered building machine learning APIs with Python frameworks like Flask and FastAPI. It showed serving models with Onyx, leveraging transformers for text generation, documenting and testing endpoints, and following best practices around errors, requests, and responses.

**Top 3 Key Points**

* APIs enable accessing ML models via HTTP requests
* FastAPI and Flask simplify building and running APIs
* Best practices improve reliability and usability