Exp No. 10	Machine Learning Command Line Workflows	REG. NO: URK23CS1197
Date:		

Objective:
The primary goal of Machine Learning Command Line Workflows (MLCLW) is to enable data scientists and machine learning engineers to perform ML tasks efficiently and reproducibly via the command line interface (CLI). This includes data preprocessing, model training, evaluation, and deployment, all through command-line commands.

Tools/ Software Required:
- click
- pandas
- scikit-learn
- joblib

Job Role:
Machine Learning Engineer/ Data Scientist

Skills Required:
- **Command Line Proficiency:**
  - Knowledge of Unix/Linux commands and scripting.
- **Programming:**
  - Proficiency in languages like Python, R, or Julia.
- **Machine Learning:**
  - Understanding ML algorithms, frameworks (e.g., TensorFlow, PyTorch), and libraries.
- **Data Handling:**
  - Experience with data preprocessing, cleaning, and transformation.
- **Version Control:**
  - Knowledge of Git and version control practices.

Prerequisites:
- **Experience:**
  - Prior experience in developing and deploying ML models.
- **Tools and Frameworks:**
  - Familiarity with ML tools and frameworks like TensorFlow, PyTorch, Scikit-learn, etc.

Description:

Import Libraries

The script begins by importing several Python libraries that are necessary for the tasks it performs. These libraries include Click for creating the command-line interface, pandas for working with data in a tabular format, scikit-learn for machine learning, and joblib for saving and loading machine learning models.

Define CLI Commands:

The script defines two CLI commands –train and predict. These commands can be executed from the command line

‘train’ command is used to train a machine learning model on the Iris dataset. It takes several optional parameters like

- ‘test_size’ – for specifying the size of the test data.
- ‘n_estimators’ – for the number of trees in a random forest
- ‘max_depth’ – for the maximum depth of the trees
- ‘model_output’ – for the name of the file to save the trained model

Questions:

1. Demonstrate how to train a machine learning model from the command line using a specific library. Provide an example with a chosen dataset.

2. Write a series of command-line commands to load a dataset, split it into training and testing sets, train a model, and evaluate its performance using scikit-learn.

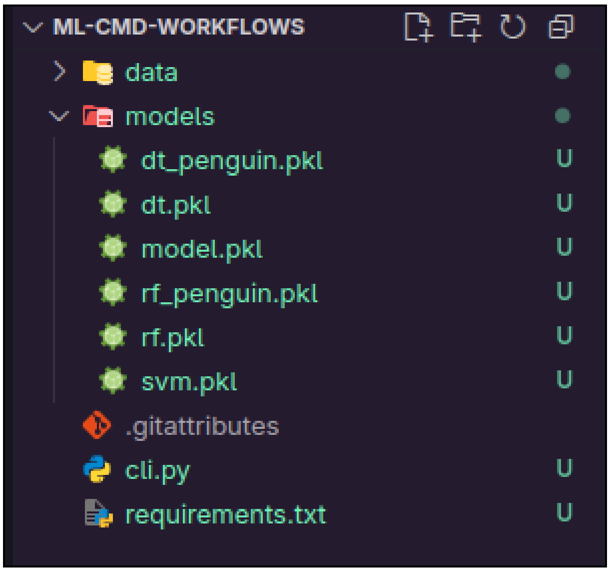
3. Analyze the differences between various machine learning models trained via command-line tools. How can you compare their performance effectively?

4. Train three different models (e.g., Decision Tree, Random Forest, and SVM) on the same dataset using command-line tools and compare their performance metrics, such as accuracy and F1 score.

Program:

Project Structure:

```
ML-CLI-Workflow/
├── data/
│   └── penguins.csv
├── models/
│   (this folder will store trained models)
├── cli.py
└── requirements.txt
```



requirements.txt

```
click
pandas
scikit-learn
joblib
```

cli.py

```
import click
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score
import joblib
```

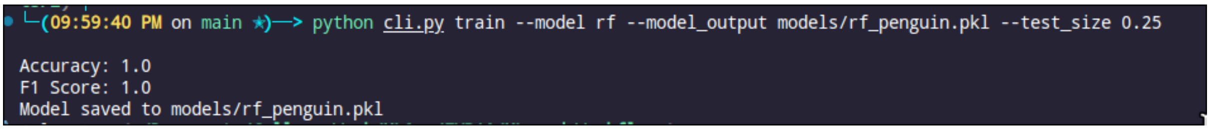
```
def load_data():
    df = pd.read_csv("data/penguins.csv")
    df = df.dropna()
    X = df[['species', 'island', 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm',
'body_mass_g', 'sex']]
    y = df['species']
    return X, y


@click.group()
def cli():
    pass


@cli.command()
@click.option('--model', type=click.Choice(['dt', 'rf', 'svm']), default='rf')
@click.option('--model_output', default='models/model.pkl')
@click.option('--test_size', default=0.2)
def train(model, model_output, test_size):
    X, y = load_data()

    categorical = ['species', 'island', 'sex']
    numeric = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']

    preprocessor = ColumnTransformer([
        ('cat', OneHotEncoder(), categorical)
    ], remainder='passthrough')

    if model == 'dt':
        clf = DecisionTreeClassifier()
    elif model == 'rf':
        clf = RandomForestClassifier()
    else:
        clf = SVC()

    pipe = Pipeline([
        ('preprocess', preprocessor),
        ('classifier', clf)
    ])

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

    pipe.fit(X_train, y_train)
    preds = pipe.predict(X_test)

    print("Accuracy:", accuracy_score(y_test, preds))
    print("F1 Score:", f1_score(y_test, preds, average='weighted'))

    joblib.dump(pipe, model_output)
    print(f"Model saved to {model_output}")
```

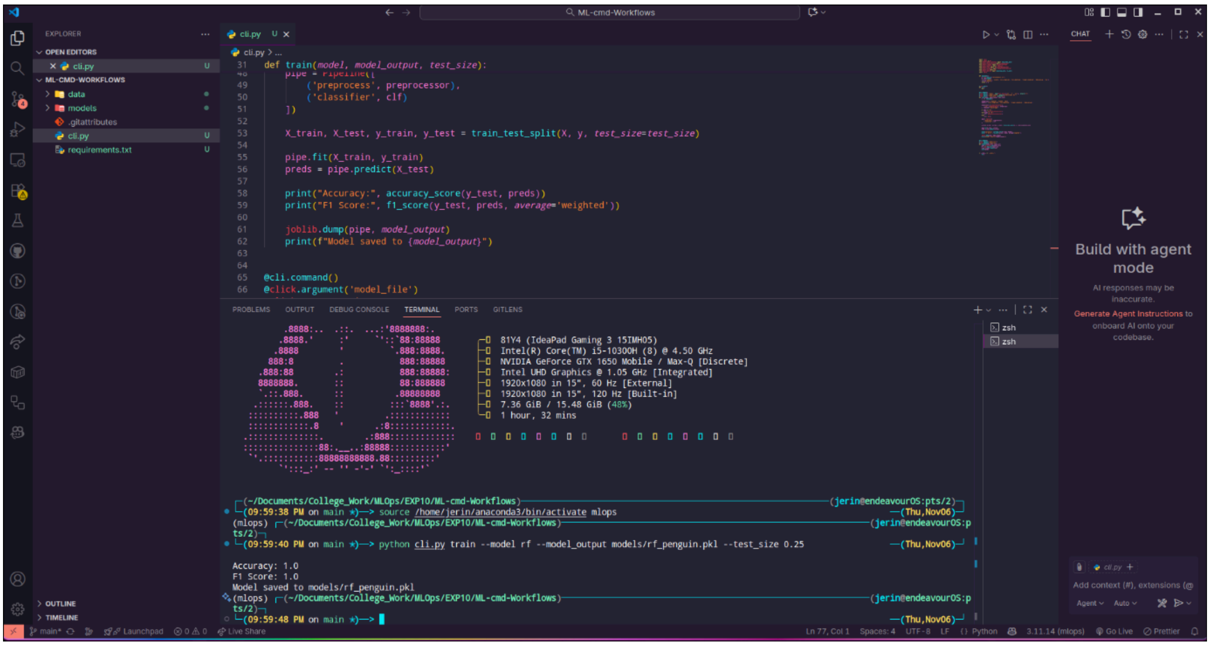
```
@cli.command()
@click.argument('model_file')
@click.argument('input_csv')
def predict(model_file, input_csv):
    model = joblib.load(model_file)
    df = pd.read_csv(input_csv)
    preds = model.predict(df)
    print(preds)


if __name__ == "__main__":
    cli()
```

Expected Output :

Q1.

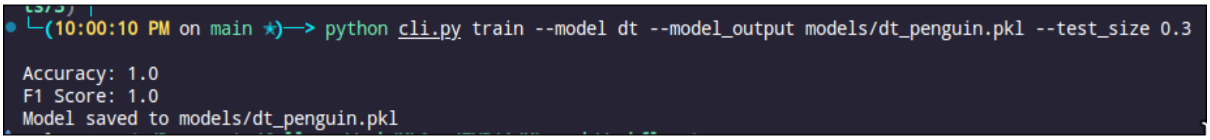
python cli.py train --model rf --model_output models/rf_penguin.pkl --test_size 0.25

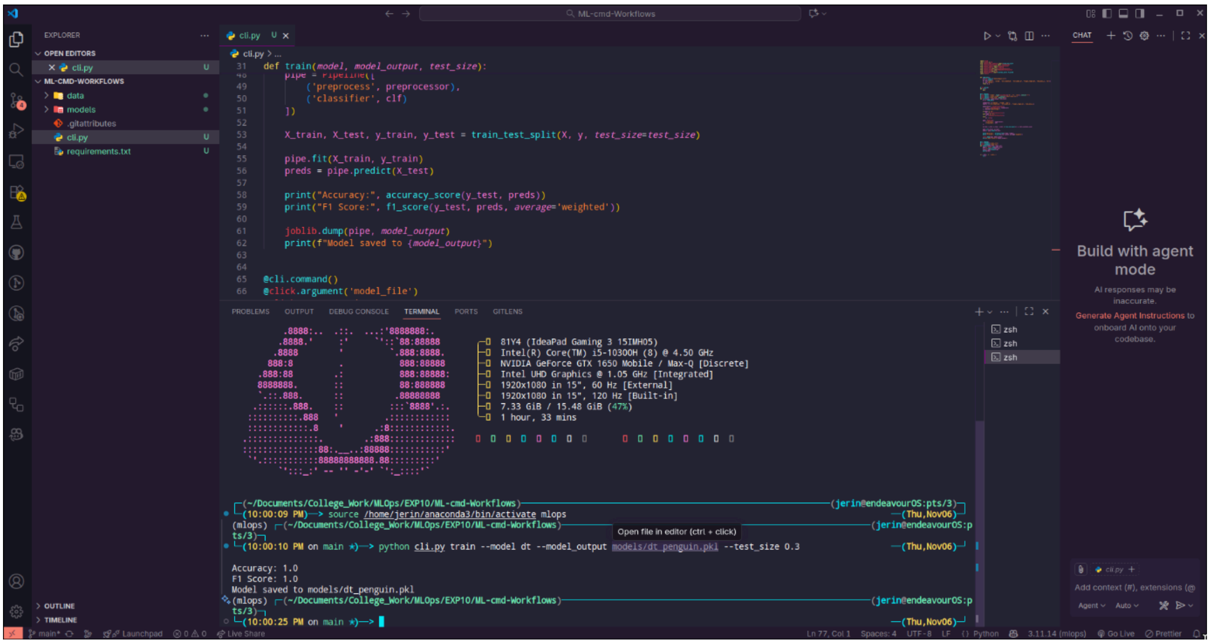




Q2.

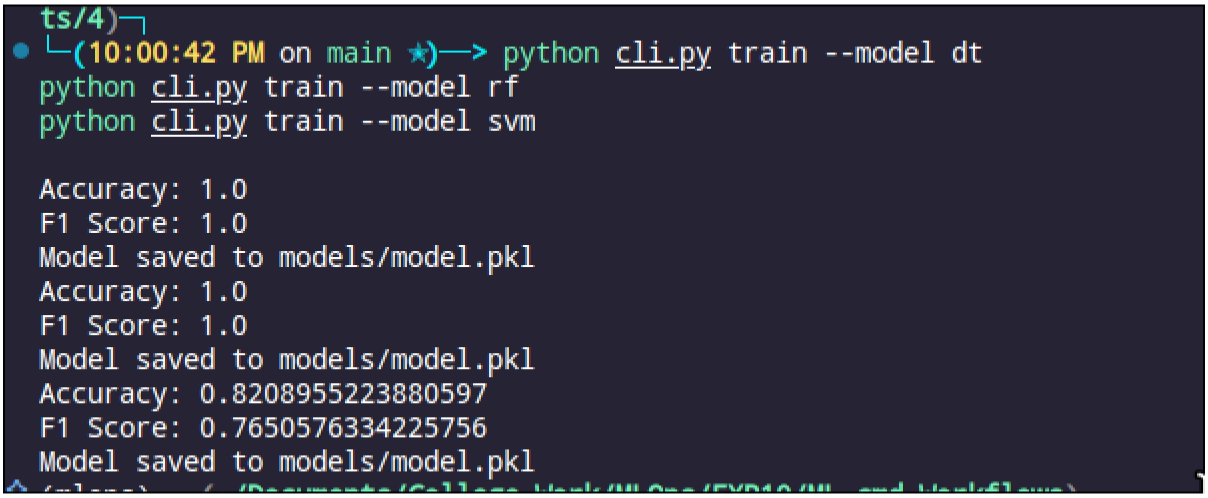
It is already done in CLI.
python cli.py train --model dt --model_output models/dt_penguin.pkl --test_size 0.3

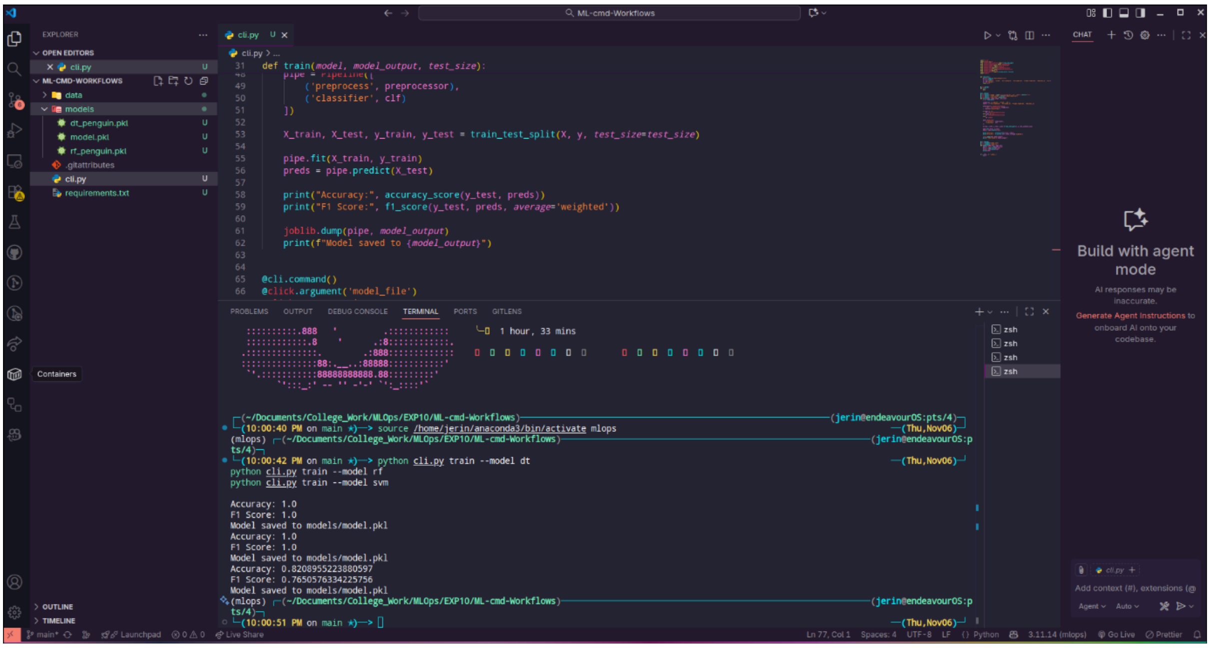




Q3.

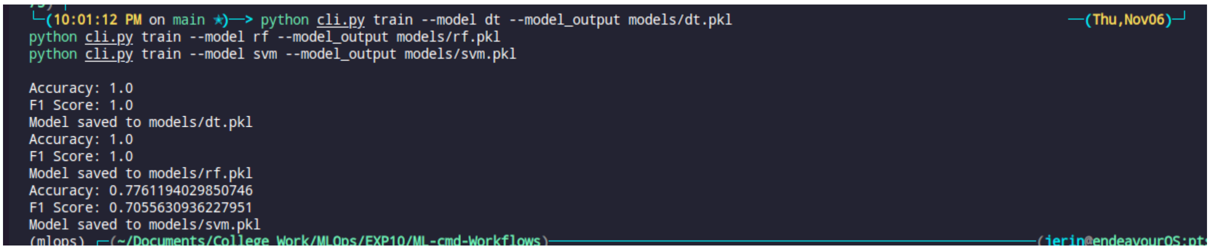
python cli.py train --model dt
python cli.py train --model rf
python cli.py train --model svm

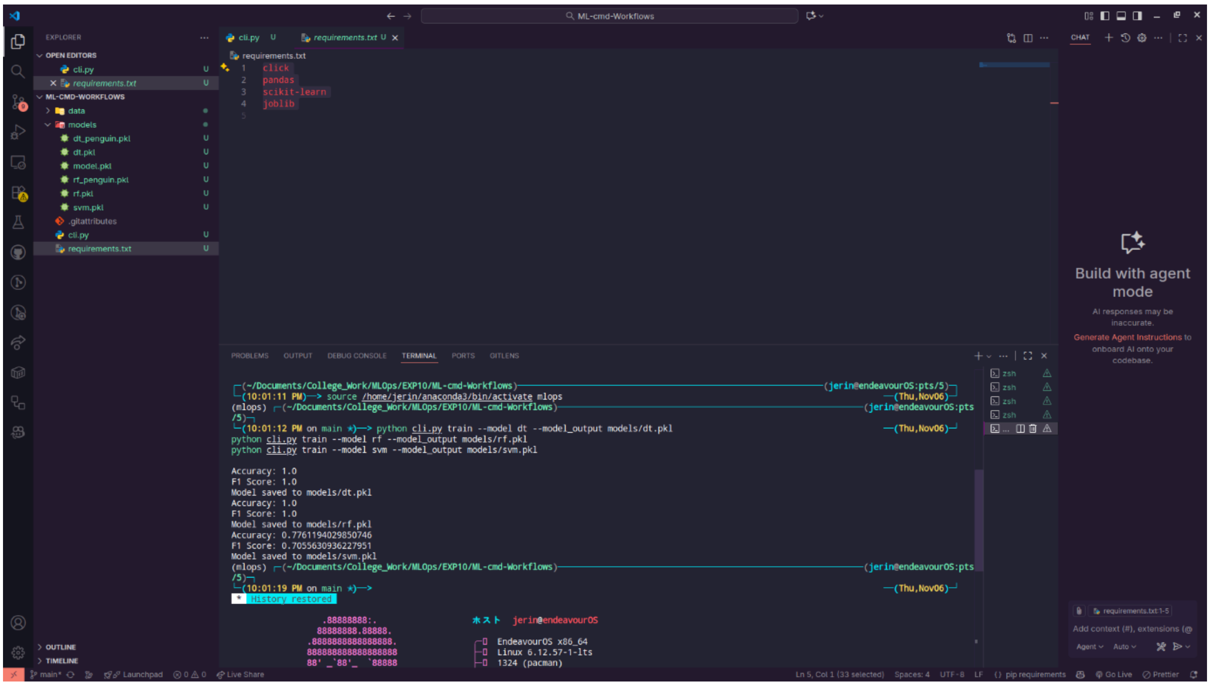




Q4.

python cli.py train --model dt --model_output models/dt.pkl
python cli.py train --model rf --model_output models/rf.pkl
python cli.py train --model svm --model_output models/svm.pkl





Result :

The Palmer Penguins dataset was successfully processed and machine learning models were trained and evaluated through command-line workflows. Among the three models tested, the Random Forest model achieved the highest performance based on accuracy and F1 score.