Challenge Lab 1: Create an End-to-End MLOps Pipeline – From Preprocessing to GitHub Automation with the Iris Dataset

Lab overview

In this lab, you will:

- Set up an ML project with GitHub
- Perform data exploration, preprocessing, and feature engineering
- Automate the ML pipeline using GitHub Actions
- Train and evaluate an ML model within the Pipeline

Estimated completion time

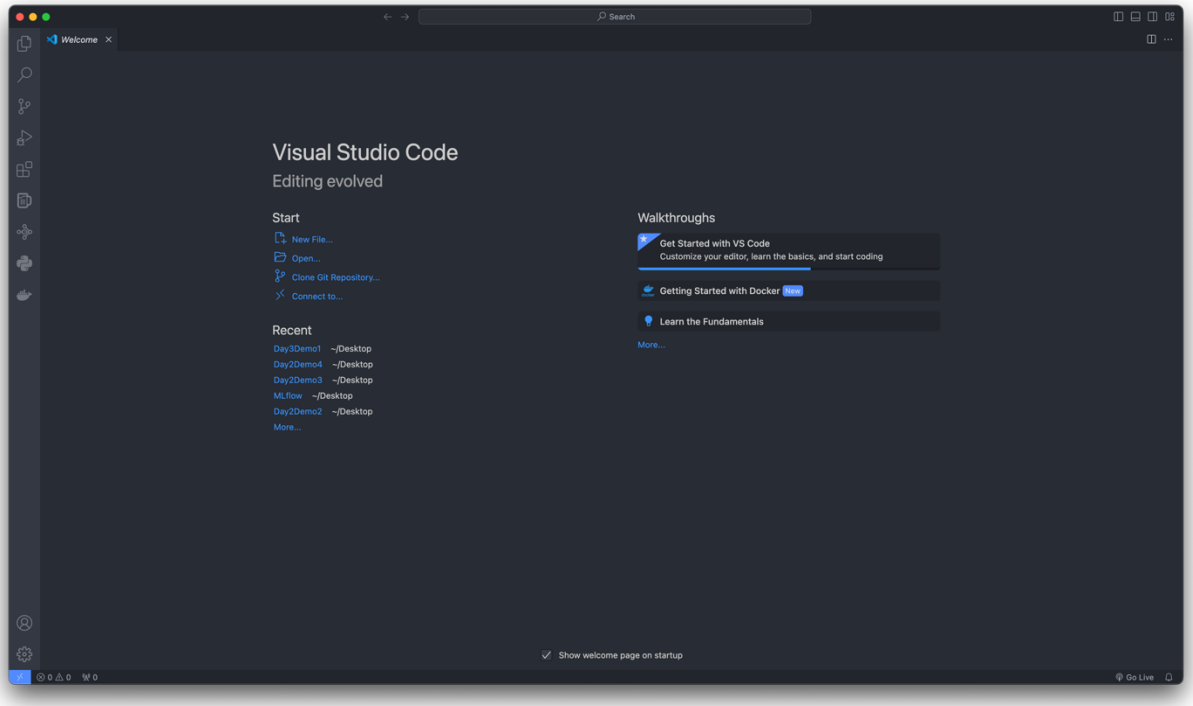
45 minutes

> **Note**
>
> You need a GitHub account to perform this lab.

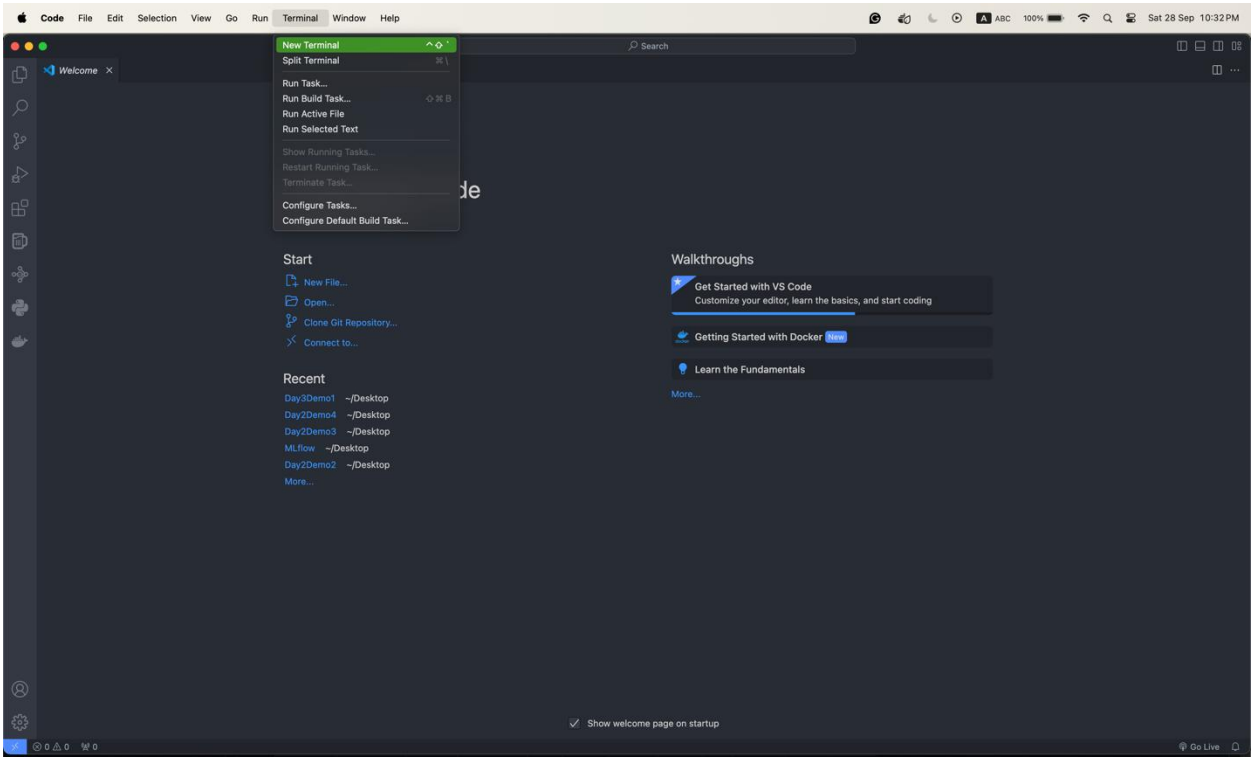
Task 1: Initializing an ML project in GitHub

In this task, you will learn to initialize a machine learning project, set up the necessary environment and dependencies, and ensure code quality by configuring pre-commit hooks for linting.

1. Open the browser, go to **https://github.com/,** and click the **Sign-in** button on the top right corner of the website.
2. Enter your username, email address, and password, then click the **Sign in** button.
3. Click on the green **New** button to create a new repository.
4. Name your repository (e.g., **mlops_Lab**). Enter this description: **This repository demonstrates the setup of an ML project, including GitHub repository initialization, Python virtual environment creation, pre-commit hooks configuration for linting and testing, and Git tracking for version control.** Choose **Public** and initialize it with a **README**.
5. Click on the green **Create Repository** button.
6. Open **VS Code**.



7. Click on the menu item **Terminal** and then select the sub-menu item **New Terminal.**



8. Enter the following command in the terminal using your GitHub username.

```
git clone https://github.com/your-username/mlops_Lab.git
```

9. Navigate into the folder by using the following command.

```
cd mlops_Lab
```

10. Install **virtualenv** if not already installed using the following command.

```
pip install virtualenv
```

11. In your project folder, create a virtual environment, activate it and verify the activation by confirming that the terminal displays the Python version.
12. Using **pip**, install the libraries needed for the project and save them to a **requirements.txt** file in the project folder. The required libraries are:

```
pandas scikit-learn matplotlib seaborn ipykernel
```

13. Commit and push changes to Github with the following commands.

```
git add .

git config --global user.email "you@example.com"

git commit -m "Initial setup with the virtual environment and pre-commit hooks"

git push origin main
```

Task 2: Loading the Iris dataset and performing exploratory data analysis (EDA)

Explore and visualize the dataset using various types of plots to understand the underlying patterns in the data. Perform a detailed analysis of the features and identify potential relationships between variables.

1. Load the dataset and display the first few rows to confirm it's loaded correctly. Open the mlops_Lab folder in VS Code. Create a new IPYNB file (mlops_Lab.ipynb), create a new code cell, enter the following code and execute it.

```
import pandas as pd

import matplotlib

import matplotlib.pyplot as plt

# Load Iris dataset

data = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')

# Display the first few rows

print(data.head())
```

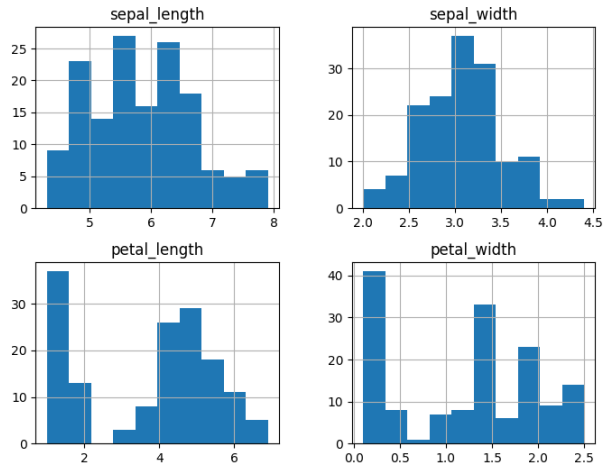
2. Analyze the dataset structure and look for missing values, unique values, and basic statistical summaries. Create a new code cell, enter the following code, and execute it.

```
# Summary of the dataset

print(data.info())

print(data.describe())

print("Missing values in the dataset:\n", data.isnull().sum())
```

3. Use histograms, pair plots, scatter plots, and box plots to understand the data distribution and identify any potential outliers. Create a new code cell, enter the following code, and execute it to visualize the distribution of each numerical feature.

```
data.hist(figsize=(8, 6))

plt.show()
```

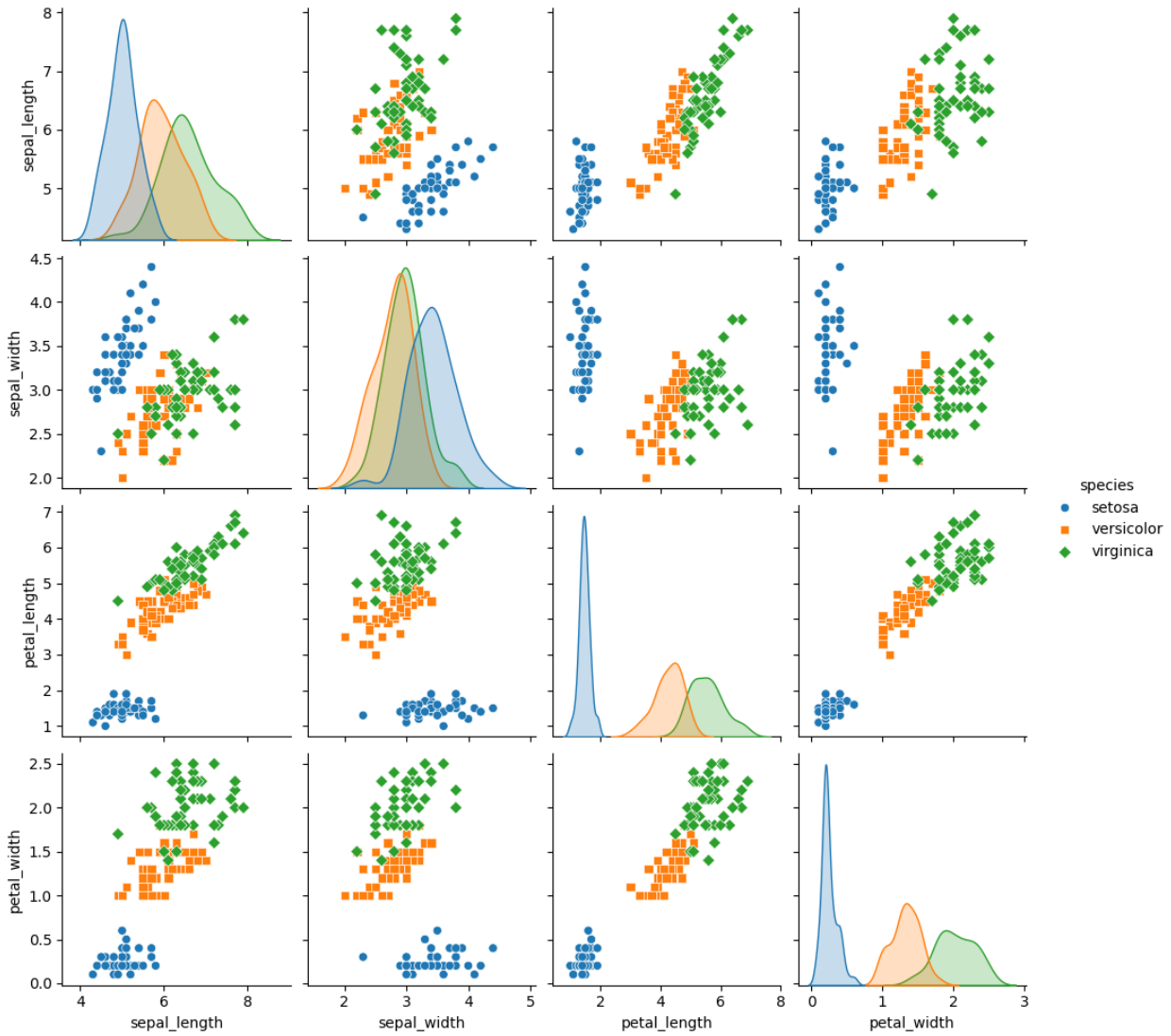
The following plots will be shown as output when you execute the above code.



4. To visualize the relationship between all pairs of features, use the pair plot. Create a new code cell, enter the following code, and execute it.

```
import seaborn as sns

sns.pairplot(data, hue='species', markers=["o", "s", "D"])

plt.show()
```

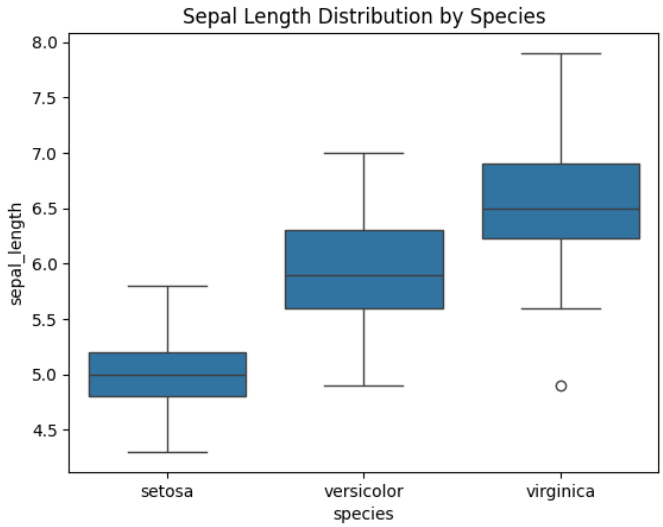
When you execute the above code, the following plot will be displayed as the output.



5. Boxplots are useful for identifying outliers within each species. Create a new code cell, and enter the following code to display the boxplot for selected features.

```
sns.boxplot(x='species', y='sepal_length', data=data)

plt.title('Sepal Length Distribution by Species')

plt.show()
```

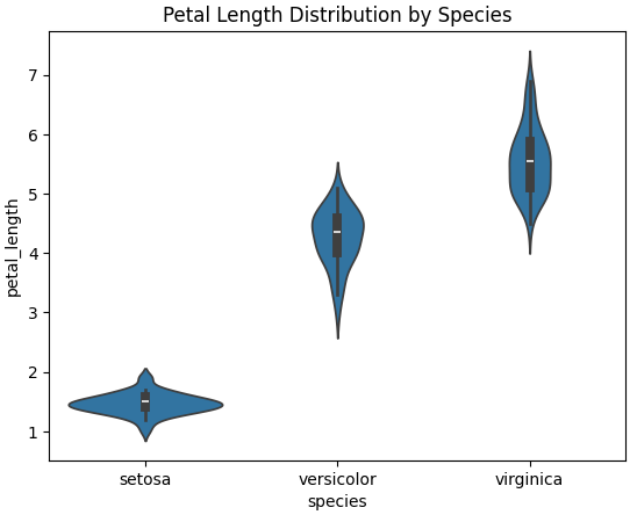
Executing the above code will display the following plot as the output.



6. A violin plot allows comparison of the distribution of features across species. To generate the violin plot for some features in the dataset, create a new code cell and enter the following code in it.

```
sns.violinplot(x='species', y='petal_length', data=data)

plt.title('Petal Length Distribution by Species')

plt.show()
```

Executing the above code will result in displaying the following plot as output.



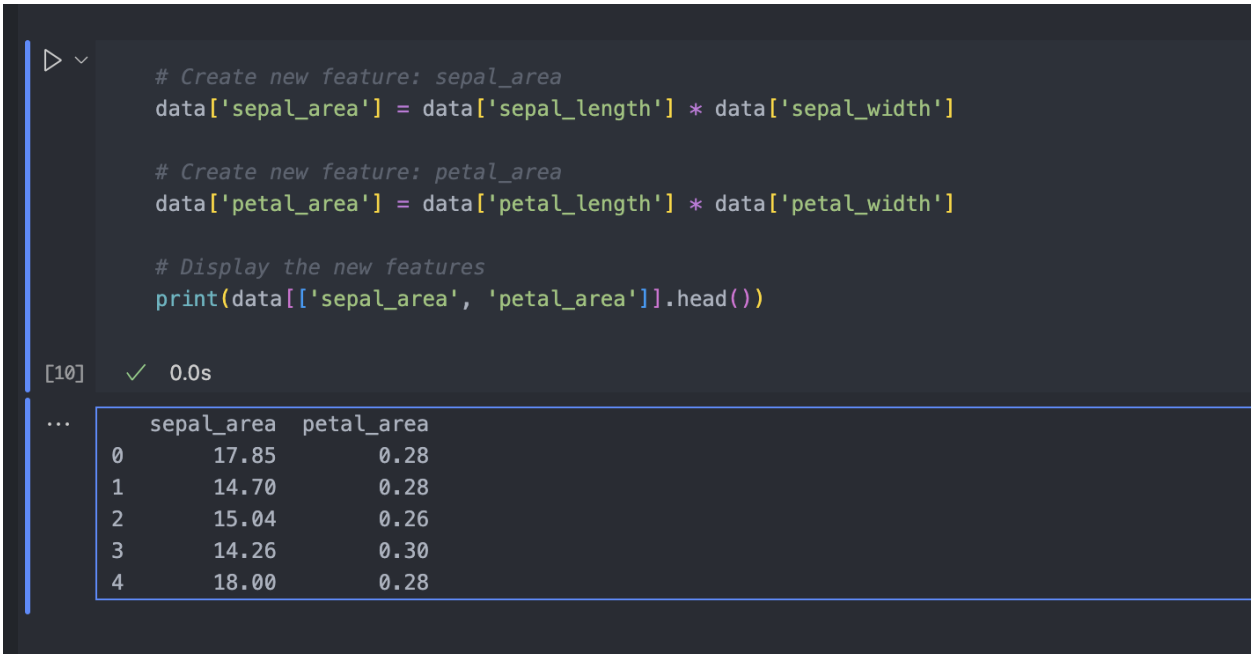
Task 3: Applying data preprocessing techniques such as scaling and feature engineering

Apply appropriate preprocessing techniques, such as scaling numerical features and encoding categorical variables. Additionally, enhance the dataset with new engineered features. Since the Iris dataset doesn't have missing values, you can skip this step.

1. Create new features based on existing ones. For example, create features that represent areas (like sepal area and petal area). Create a new code cell and enter the following code in it.

```
# Create a new feature: sepal_area

data['sepal_area'] = data['sepal_length'] * data['sepal_width']

# Create a new feature: petal_area

data['petal_area'] = data['petal_length'] * data['petal_width']

# Display the new features

print(data[['sepal_area', 'petal_area']].head())
```

Executing the above code cell will result in creating the features sepal_area and petal_area.



2. Scale the numerical features to ensure consistent ranges for model training. To do this, create a new code cell and enter the following code in it.

```
from sklearn.preprocessing import StandardScaler

# Select numerical columns

numerical_cols = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width', 'sepal_area', 'petal_area']

# Apply Standard Scaling

scaler = StandardScaler()

data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# Display scaled features

print(data[numerical_cols].head())
```

Executing the above code cell will result in the following output.

```
from sklearn.preprocessing import StandardScaler

# Select numerical columns
numerical_cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'sepal_area', 'petal_area']

# Apply Standard Scaling
scaler = StandardScaler()
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# Display scaled features
print(data[numerical_cols].head())

[11]  ✓ 3.6s

   sepal_length  sepal_width  petal_length  petal_width  sepal_area  \
0     -0.900681     1.019004     -1.340227    -1.315444    0.008098
1     -1.143017    -0.131979     -1.340227    -1.315444   -0.932024
2     -1.385353     0.328414     -1.397064    -1.315444   -0.830551
3     -1.506521     0.098217     -1.283389    -1.315444   -1.063343
4     -1.021849     1.249201     -1.340227    -1.315444    0.052866

   petal_area
0   -1.174041
1   -1.174041
2   -1.178299
3   -1.169783
4   -1.174041
```

Task 4: Building an automated ML pipeline using scikit-learn and setting up GitHub Actions to automate the model training and evaluation pipeline

This task aims to set up a machine learning pipeline that automates data preprocessing and model training. Integrate this pipeline with GitHub Actions to automate the workflow.

1. For defining the ML pipeline, use scikit-learn's Pipeline to chain together the preprocessing and model training steps. To do this, create a new code cell, enter the following code, and execute it.

```
from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split


# Split data into features and target

X = data.drop('species', axis=1)

y = data['species']


# Split into training and testing sets

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


# Build an ML pipeline with preprocessing and model training steps

pipeline = Pipeline(steps=[

    ('classifier', RandomForestClassifier(n_estimators=100,
random_state=42))

])
```

```
# Train the pipeline

pipeline.fit(X_train, y_train)
```

2. Test the trained pipeline on the test data and evaluate its performance using accuracy and other metrics. Create a new code cell, enter the following code, and execute it.

```
from sklearn.metrics import accuracy_score, classification_report


# Predict on test set

y_pred = pipeline.predict(X_test)


# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")


# Classification report

print(classification_report(y_test, y_pred))
```

Executing the above code will result in providing the results of precision, recall, f1-score, and support.

```
Model Accuracy: 1.00
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

3. Create a file **ml_pipeline.py** and paste all the Python code from the **ipynb** file to it. Set up GitHub Actions to automate the ML pipeline. Create a **.github/workflows/main.yml** file to automate the pipeline on push events and add the following code to it.

```
name: ML Pipeline Automation

on: [push]

jobs:

  build:

    runs-on: ubuntu-latest


    steps:

    - name: Checkout code

      uses: actions/checkout@v2


    - name: Set up Python

      uses: actions/setup-python@v2

      with:

        python-version: 3.9


    - name: Install dependencies

      run: |

        python -m venv venv

        source venv/bin/activate

        pip install -r requirements.txt


    - name: Run pipeline

      run: |
```

```
source venv/bin/activate

python ml_pipeline.py4.
```

4. Commit and push to Github, run the following commands in the terminal.

```
git add .

git commit -m "Added automated ML pipeline and GitHub Actions"

git push origin main
```

5. Verify that your GitHub repository now has the files upload.

> **Note**
>
> Ensure that you logout of your GitHub account on completion of this exercise.

Lab review

1. What is the purpose of the requirements.txt file?
   A. To store project data
   B. To list and manage project dependencies
   C. To configure GitHub Actions workflows
   D. To automate the ML pipeline

> **STOP**
>
> You have successfully completed this lab.