# ACTIVITY: 10

# DEEP LEARNING MODEL IMPLEMENTATION AND PERFORMANCE ANALYSIS

**Problem Statement:** In this project, we aim to improve the performance of existing regression and classification models by rebuilding them using deep learning techniques. The goal is to compare the performance of traditional machine learning (ML) approaches with deep learning (DL) methods, assessing metrics such as accuracy, precision, recall, and MSE, MAE, r2_score.

## Regression - Rebuild with deep learning model

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error, r2_score

from tensorflow import keras

# 1. Load and prepare data

iris_data = load_iris()

data = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)

data['target'] = iris_data.data[:, 2]   # Using petal width as the target

# 2. Display the first few rows of the dataset

print(data.head())

# 3. Visualize correlation with a heatmap

plt.figure(figsize=(10, 8))
```

```
sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm', center=0)
plt.title('Feature Correlation Heatmap')
plt.show()
# 4. Prepare features and target variable
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# 5. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# 6. Build the deep learning model
model = keras.Sequential([
keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
keras.layers.Dense(32, activation='relu'),
keras.layers.Dense(1)   # Output layer for regression
])

# 7. Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# 8. Train the model
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_val, y_val))

# 9. Evaluate the model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Custom accuracy-like metric: Percentage of predictions within a threshold
threshold = 0.1  # Set your acceptable range
accuracy_like = np.mean(np.abs(y_pred.flatten() - y_test) <= threshold)

# Display results
print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared Score: {r2:.2f}')

# Optional: Display a few predictions vs actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred.flatten()})
print(results.head())
```

## Output:

```
4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - loss: 14.1944 - val_loss: 15.0136
Epoch 4/100
4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 13.2979 - val_loss: 13.6908
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 12.5030 - val_loss: 12.3585
Epoch 6/100
```

4/4 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 10.8817 - val_loss: 11.0777

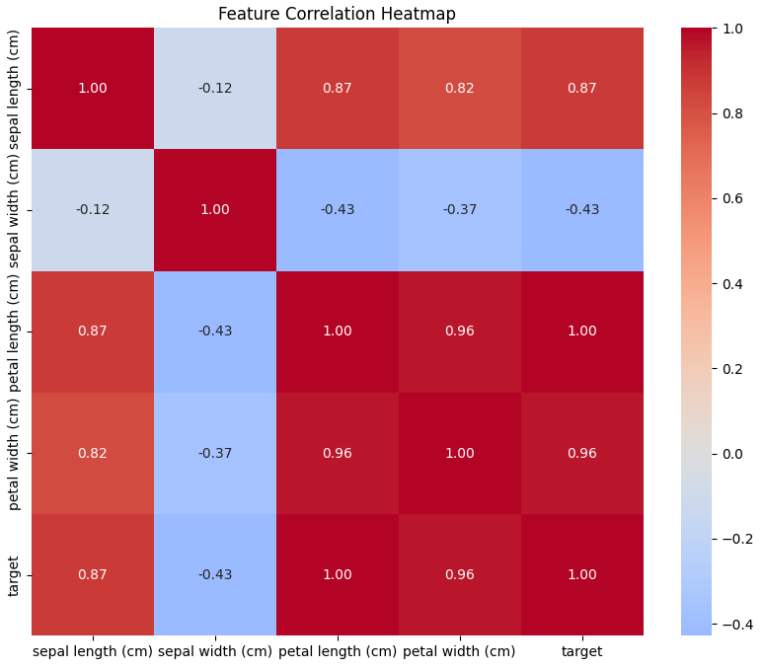
Epoch 7/100

Mean Squared Error: 0.03

R-squared Score: 0.99

| | Actual | Predicted |
|---|---|---|
| 143 | 5.9 | 5.900316 |
| 56 | 4.7 | 4.319361 |
| 128 | 5.6 | 5.713876 |
| 69 | 3.9 | 3.834024 |
| 68 | 4.5 | 4.914990 |

## Graph:



## Regression – Rebuild with machine learning model

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# 1. Load and prepare data
iris_data = load_iris()
data = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
data['target'] = iris_data.data[:, 2]   # Using petal width as the target

# 2. Display the first few rows of the dataset
print(data.head())

# 3. Visualize correlation with a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='coolwarm', center=0)
plt.title('Feature Correlation Heatmap')
plt.show()

# 4. Prepare features and target variable
X = data.drop('target', axis=1)
y = data['target']
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 6. Build the regression model
model = LinearRegression()

# 7. Train the model
model.fit(X_train, y_train)

# 8. Make predictions
y_pred = model.predict(X_test)

# 9. Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Custom accuracy-like metric: Percentage of predictions within a threshold
threshold = 0.1    # Set your acceptable range
accuracy_like = np.mean(np.abs(y_pred - y_test) <= threshold) * 100
print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared Score: {r2:.2f}')
print(f'Accuracy(within ±{threshold}): {accuracy_like:.2f}%')
# Optional: Display a few predictions vs actual values
```

```
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

## Output:

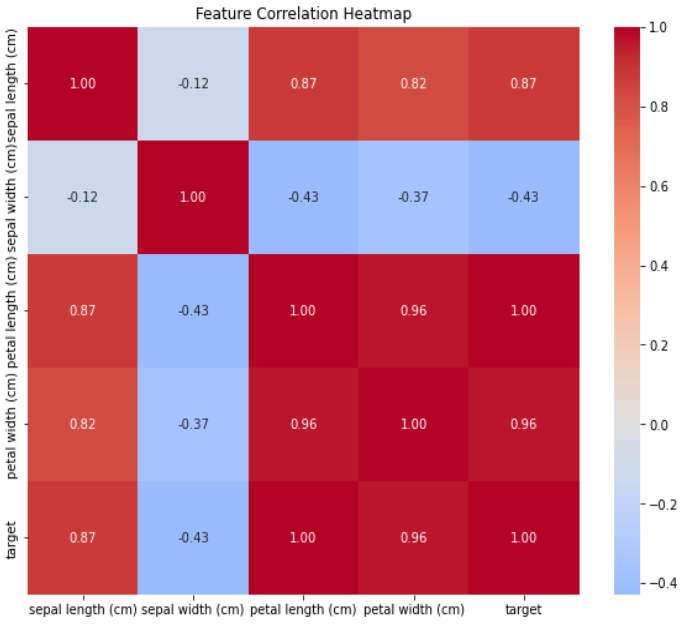
Mean Squared Error: 0.00

R-squared Score: 1.00

Accuracy(within ±0.1): 100.00%

| | Actual | Predicted |
|---|---|---|
| 73 | 4.7 | 4.7 |
| 18 | 1.7 | 1.7 |
| 118 | 6.9 | 6.9 |
| 78 | 4.5 | 4.5 |
| 76 | 4.8 | 4.8 |

## Graph:



## Classification – Rebuild with deep learning model

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tensorflow import keras

# 1. Load and prepare data
iris_data = load_iris()
data = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
data['target'] = iris_data.target

# 2. Display the first few rows of the dataset
print(data.head())

# 3. Visualize the data distribution
sns.pairplot(data, hue='target', palette='Set2')
plt.title('Iris Dataset Pairplot')
plt.show()

# 4. Prepare features and target variable
X = data.drop('target', axis=1)
y = data['target']
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 6. Build the deep learning model
model = keras.Sequential([
keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
keras.layers.Dense(32, activation='relu'),
keras.layers.Dense(3, activation='softmax')  # 3 classes for iris species
])

# 7. Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# 8. Train the model
history = model.fit(X_train, y_train, epochs=100, validation_split=0.2)

# 9. Evaluate the model
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Convert probabilities to class labels

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_classes)
```

```
print(f'Accuracy: {accuracy * 100:.2f}%')

# Optional: Plot training history

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()


# Optional: Display a few predictions vs actual values

results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_classes})

print(results.head())
```

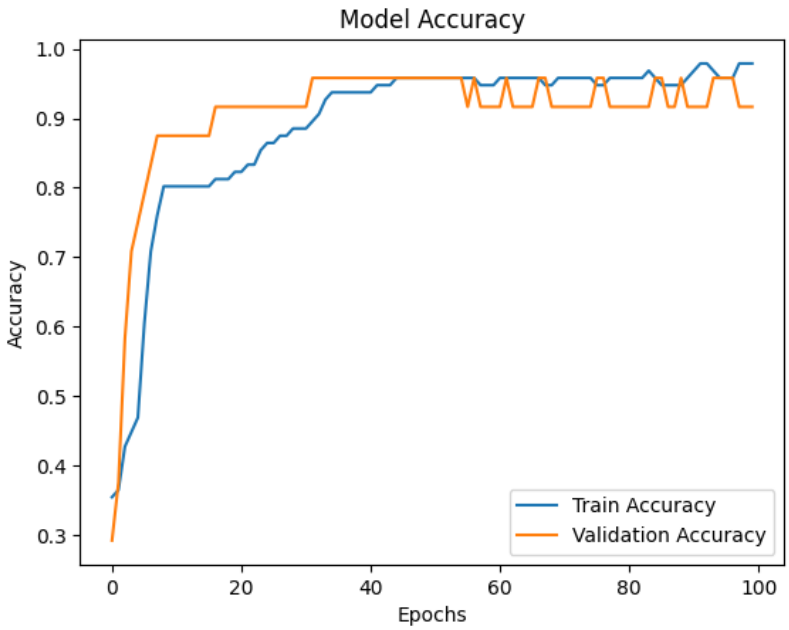
## Output:

```
3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step - accuracy: 0.3216 - loss: 1.2119 - val_accuracy: 0.2917 - val_loss: 1.0809

Epoch 2/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.3815 - loss: 1.1034 - val_accuracy: 0.3750 - val_loss: 1.0115

Epoch 3/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.4440 - loss: 1.0391 - val_accuracy: 0.5833 - val_loss: 0.9489

Epoch 4/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4544 - loss: 0.9871 - val_accuracy: 0.7083 - val_loss: 0.8893

Epoch 5/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4492 - loss: 0.9308 - val_accuracy: 0.7500 - val_loss: 0.8363
```

Epoch 6/100

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5404 - loss: 0.8903 - val_accuracy: 0.7917 - val_loss: 0.7892

Epoch 7/100




Accuracy: 96.67%

| | Actual | Predicted |
|---|---|---|
| 73 | 1 | 1 |
| 18 | 0 | 0 |
| 118 | 2 | 2 |
| 78 | 1 | 1 |
| 76 | 1 | 1 |

## Classification - Rebuild with machine learning model

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
impor tseaborn as sns

# 1. Load and prepare data
iris_data = load_iris()
data = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
data['target'] = iris_data.target

# 2. Display the first few rows of the dataset
print(data.head())

# 3. Visualize the data distribution using pairplot
plt.figure(figsize=(10, 6))
sns.pairplot(data, hue='target', palette='Set2')
plt.title('Iris Dataset Pairplot')
plt.show()

# 4. Prepare features and target variable
```

```
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 6. Build the Logistic Regression model
model = LogisticRegression()

# 7. Train the model
model.fit(X_train, y_train)

# 8. Make predictions
y_pred = model.predict(X_test)

# 9. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%')
print('Classification Report:\n', class_report)
```

```
# 10. Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris_data.target_names, yticklabels=iris_data.target_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Optional: Display a few predictions vs actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

## Output:

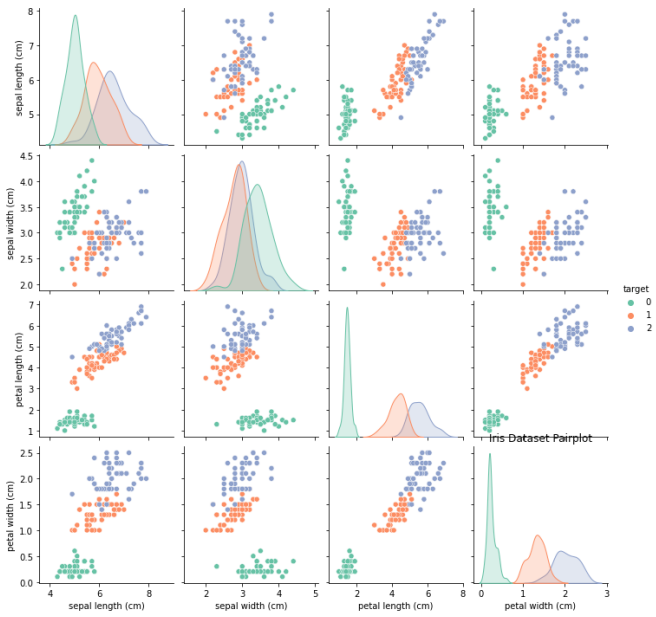
Accuracy: 100.00%

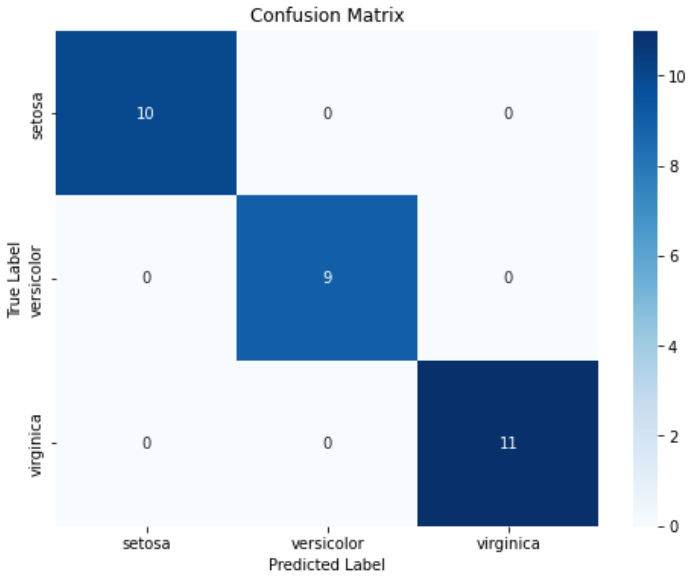
Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 10 |
| 1 | 1.00 | 1.00 | 1.00 | 9 |
| 2 | 1.00 | 1.00 | 1.00 | 11 |
| accuracy | | | 1.00 | 30 |
| macroavg | 1.00 | 1.00 | 1.00 | 30 |
| weightedavg | 1.00 | 1.00 | 1.00 | 30 |

| | Actual | Predicted |
|---|---|---|
| 73 | 1 | 1 |
| 18 | 0 | 0 |
| 118 | 2 | 2 |
| 78 | 1 | 1 |
| 76 | 1 | 1 |

**Graph:**



# Analyse the performance of ML and DL

**1. Handling Complex Patterns:** Deep learning models, particularly neural networks, can capture complex patterns and relationships in large datasets that traditional algorithms may struggle with.

**2. Feature Learning:** Deep learning can automatically extract and learn features from raw data (e.g., images, text), reducing the need for manual feature engineering.

**3. Scalability:** Deep learning models often scale better with large datasets. As the amount of data increases, their performance typically improves.

**4. Performance on Unstructured Data:** Deep learning excels at processing unstructured data (e.g., images, audio, text) due to its architecture, which is designed to work with high-dimensional input.

**5. State-of-the-Art Results:** For many tasks, such as image recognition, natural language processing, and speech recognition, deep learning models have achieved state-of-the-art performance.

**6. Transfer Learning:** Pre-trained deep learning models can be fine-tuned for specific tasks, making them more efficient in terms of training time and data requirements.

**7. End-to-End Learning:** Deep learning allows for end-to-end learning, meaning that the entire pipeline (feature extraction to prediction) can be trained simultaneously, which can improve overall performance.

**8. Robustness:** Deep learning models can be more robust to noise and variations in data, as they can learn high-level abstractions.

**9. Advanced Architectures:** Deep learning allows for the use of advanced architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), which are tailored for specific types of data.

**10. Continuous Improvement:** With ongoing advancements in deep learning techniques and architectures, models are continually improving, leading to better performance in various applications.

# Uploading Files to GitHub

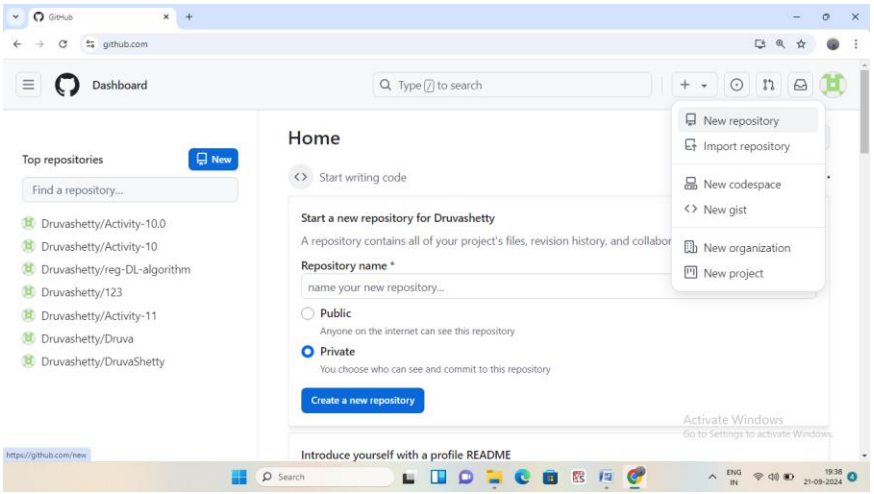
## Creating git repository for the project:

### Step 1: Sign In to GitHub

- Go to GitHub.com and log in to your account.

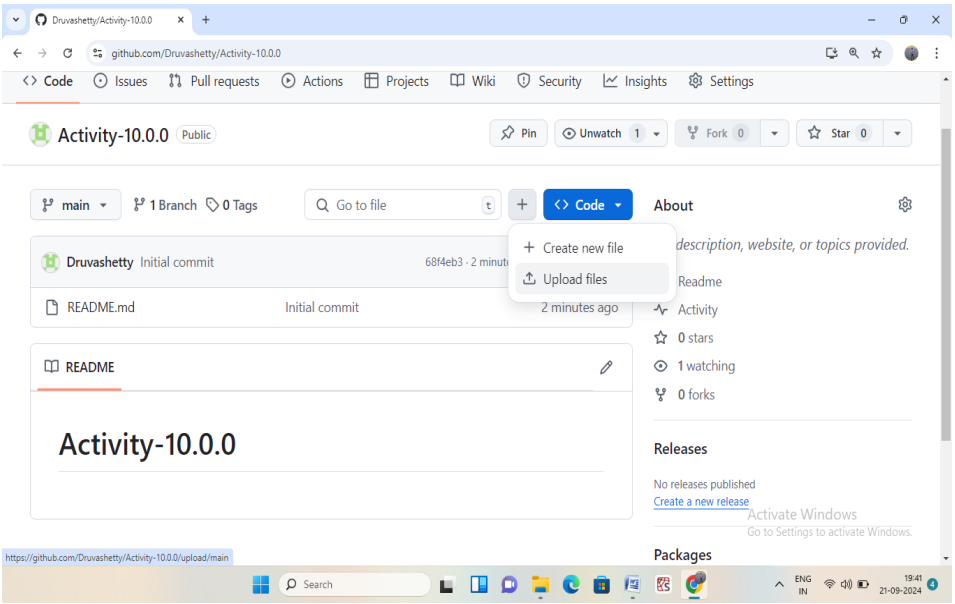
### Step 2: Navigate to the Repository

- Click on the repository where you want to upload the file. If you don't have a repository yet, you can create one by clicking the "+" icon in the top right and selecting "New repository."



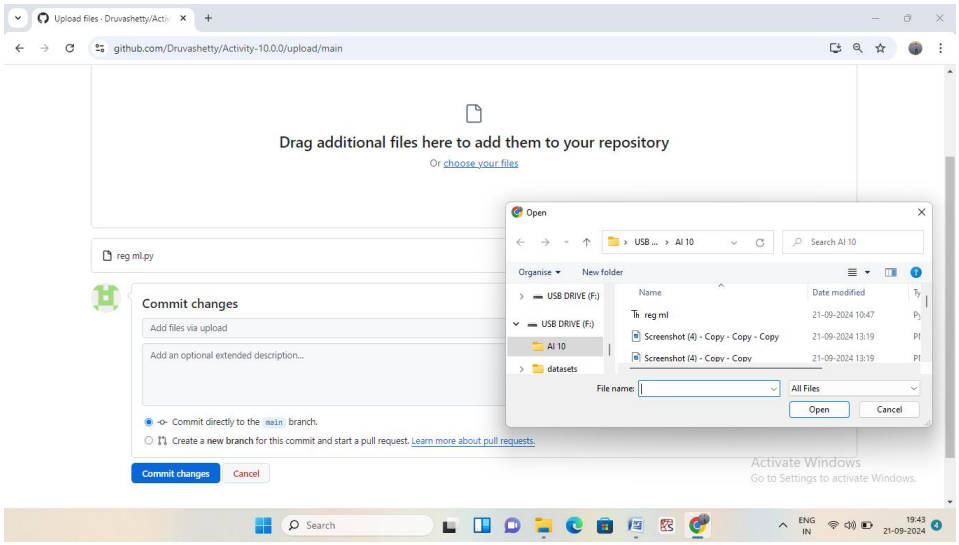

### Step 3: Open the Upload File Option

- In your repository, click on the "Add file" button, then select "Upload files" from the dropdown menu.



**Step 4: Drag and Drop or Select Files**

- You can drag and drop your file(s) into the upload area, or click “choose your files” to browse your computer for the files you want to upload.

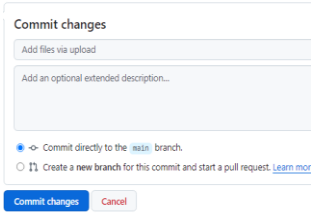



**Step 5: Commit Your Changes**

- After selecting your files, scroll down to the “Commit changes” section. Here, you can add a commit message that describes your changes.

**Step 6: Choose Commit Options**

- Select whether to commit directly to the main branch or to create a new branch for your changes.




**Step 7: Click on Commit Changes**

- Once you’ve filled out the commit message and selected your options, click the “Commit changes” button to finalize the upload.

**Step 8: Verify Your Upload**

- After committing, you’ll be redirected to your repository. You should see the uploaded file listed there. Click on it to confirm that it has been uploaded correctly.

**Click [https://github.com/Druvashetty/Activity-10](https://github.com/Druvashetty/Activity-10) to view the uploaded file**