

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("rocki37/open-university-learning-analytics-dataset")

print("Path to dataset files:", path)

→ Downloading from https://www.kaggle.com/api/v1/datasets/download/rocki37/open-university-learning-analytics-dataset?dataset\_version=1
100%|██████████| 84.3M/84.3M [00:00<00:00, 97.5MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/rocki37/open-university-learning-analytics-dataset/versions/1

import pandas as pd
import os

# Set dataset path (update this after running your KaggleHub command)
dataset_path = "/root/.cache/kagglehub/datasets/rocki37/open-university-learning-analytics-dataset/versions/1" # Replace with your dataset path

# Load datasets
student_info = pd.read_csv(f"{dataset_path}/studentInfo.csv")
student_vle = pd.read_csv(f"{dataset_path}/studentVle.csv")
assessments = pd.read_csv(f"{dataset_path}/assessments.csv")
student_assessments = pd.read_csv(f"{dataset_path}/studentAssessment.csv")

# ♦ Print column names for debugging
print("✅ Loaded Datasets:")
print("\nStudent Info Columns:\n", student_info.columns.tolist())
print("\nStudent VLE Columns:\n", student_vle.columns.tolist())
print("\nAssessments Columns:\n", assessments.columns.tolist())
print("\nStudent Assessments Columns:\n", student_assessments.columns.tolist())

→ ✅ Loaded Datasets:

Student Info Columns:
['code_module', 'code_presentation', 'id_student', 'gender', 'region', 'highest_education', 'imd_band', 'age_band', 'num_situation']

Student VLE Columns:
['code_module', 'code_presentation', 'id_student', 'id_site', 'date', 'sum_click']
```

```
Assessments Columns:
```

```
['code_module', 'code_presentation', 'id_assessment', 'assessment_type', 'date', 'weight']
```

```
Student Assessments Columns:
```

```
['id_assessment', 'id_student', 'date_submitted', 'is_banked', 'score']
```

```
from google.colab import drive
import pandas as pd

drive.mount('/content/drive')

# Load the dataset from Google Drive
df_cleaned = pd.read_csv("/content/drive/MyDrive/cleaned_dataset.csv")

print("✅ Cleaned dataset loaded successfully!")
print("📊 Dataset Shape:", df_cleaned.shape)
```

→ Mounted at /content/drive

✅ Cleaned dataset loaded successfully!

📊 Dataset Shape: (207319, 16)

```
print(df.columns)
print(df.head()) # To preview the first few rows
```

→ Index(['gender', 'region', 'highest_education', 'imd_band', 'age_band',
 'num_of_prev_attempts', 'studied_credits', 'disability', 'final_result',
 'date_submitted', 'is_banked', 'score', 'assessment_type', 'date',
 'weight', 'sum_click'],
 dtype='object')

	gender	region	highest_education	imd_band	age_band	\
0	1	0		1	9	2
1	1	0		1	9	2
2	1	0		1	9	2
3	1	0		1	9	2
4	1	0		1	9	2

```

      num_of_prev_attempts  studied_credits  disability  final_result  \
0                  0.0          0.35          0            2
1                  0.0          0.35          0            2
2                  0.0          0.35          0            2
3                  0.0          0.35          0            2
4                  0.0          0.35          0            2

      date_submitted  is_banked  score  assessment_type  date  weight  \
0     0.046850      0.0    0.78           2  0.076336   0.1
1     0.103393      0.0    0.85           2  0.209924   0.2
2     0.203554      0.0    0.80           2  0.450382   0.2
3     0.282714      0.0    0.85           2  0.637405   0.2
4     0.360258      0.0    0.82           2  0.824427   0.3

sum_click
0  0.03264
1  0.03264
2  0.03264
3  0.03264
4  0.03264

```

```
print("🔍 Student VLE Columns:", student_vle.columns.tolist())
```

→ 🔎 Student VLE Columns: ['code_module', 'code_presentation', 'id_student', 'id_site', 'date', 'sum_click']

```
# Identify common columns between df and student_vle
common_cols = list(set(df.columns) & set(student_vle.columns))
print("✅ Common Columns for Merging:", common_cols)

# Merge using only these columns
df = df.merge(student_vle, on=common_cols, how='left')
```

→ Common Columns for Merging: ['date', 'id_student']

```
print("✅ Merged Dataset Shape:", df.shape)
print(df.head()) # Display first few rows
```

→ Merged Dataset Shape: (967569, 25)

```
code_module_x code_presentation_x id_student gender      region \
0          AAA        2013J    11391      M  East Anglian Region
1          AAA        2013J    11391      M  East Anglian Region
2          AAA        2013J    11391      M  East Anglian Region
3          AAA        2013J    11391      M  East Anglian Region
4          AAA        2013J    11391      M  East Anglian Region

highest_education imd_band age_band num_of_prev_attempts studied_credits \
0  HE Qualification  90-100%   55<=           0            240
1  HE Qualification  90-100%   55<=           0            240
2  HE Qualification  90-100%   55<=           0            240
3  HE Qualification  90-100%   55<=           0            240
4  HE Qualification  90-100%   55<=           0            240

... score code_module_y code_presentation_y assessment_type date \
0 ... 78.0        AAA        2013J       TMA  19.0
1 ... 85.0        AAA        2013J       TMA  54.0
2 ... 80.0        AAA        2013J       TMA 117.0
3 ... 85.0        AAA        2013J       TMA 166.0
4 ... 82.0        AAA        2013J       TMA 215.0

weight code_module code_presentation id_site sum_click
0    10.0        NaN        NaN        NaN        NaN
1    20.0        AAA        2013J  546614.0      1.0
2    20.0        NaN        NaN        NaN        NaN
3    20.0        NaN        NaN        NaN        NaN
4    30.0        NaN        NaN        NaN        NaN
```

[5 rows x 25 columns]

```
# Drop duplicate columns
df = df.drop(columns=['code_module_y', 'code_presentation_y'])

# Rename _x columns to original names
df = df.rename(columns={'code_module_x': 'code_module', 'code_presentation_x': 'code_presentation'})

print("✅ Cleaned column names:", df.columns.tolist())

→ d', 'is_banked', 'score', 'assessment_type', 'date', 'weight', 'code_module', 'code_presentation', 'id_site', 'sum_click']

# Drop duplicate columns
df = df.loc[:, ~df.columns.duplicated()]

print("✅ Cleaned dataset columns:", df.columns.tolist())

→ ✅ Cleaned dataset columns: ['code_module', 'code_presentation', 'id_student', 'gender', 'region', 'highest_education', 'final_result']

# Map final_result categories to numerical values
df['final_result'] = df['final_result'].map({'Withdrawn': 0, 'Fail': 1, 'Pass': 2, 'Distinction': 3})

# Verify mapping
print("✅ Encoded final_result:\n", df['final_result'].value_counts()) # Check distribution

→ ✅ Encoded final_result:
final_result
2    530598
1    164193
3    138798
0    133980
Name: count, dtype: int64

# Check for missing values
missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0] # Only display columns with missing values
```

```
print("🔍 Missing Values:\n", missing_values)
```

➡️ 🔎 Missing Values:

imd_band	55931
id_assessment	5847
date_submitted	5847
is_banked	5847
score	6598
assessment_type	5847
date	9865
weight	5847
id_site	100373
sum_click	100373

dtype: int64

```
df.drop(columns=['id_site', 'sum_click'], inplace=True)
print("✅ Dropped highly missing columns: ['id_site', 'sum_click']")
```

➡️ ✅ Dropped highly missing columns: ['id_site', 'sum_click']

```
# Fill missing categorical values with mode
for col in ['imd_band', 'id_assessment', 'date_submitted', 'is_banked', 'assessment_type', 'date']:
    df[col].fillna(df[col].mode()[0], inplace=True)

# Fill missing numerical values with median
df['score'].fillna(df['score'].median(), inplace=True)
df['weight'].fillna(df['weight'].median(), inplace=True)

print("✅ Missing values filled successfully!")
```

➡️ ✅ Missing values filled successfully!

<ipython-input-25-b1c1c9ff7ae2>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through the behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we a

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[co

```
df[col].fillna(df[col].mode()[0], inplace=True)
<ipython-input-25-b1c1c9ff7ae2>:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we a
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[co

```
df['score'].fillna(df['score'].median(), inplace=True)
<ipython-input-25-b1c1c9ff7ae2>:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we a
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[co

```
df['weight'].fillna(df['weight'].median(), inplace=True)
```

```
print("🔍 Remaining Missing Values:\n", df.isnull().sum().sum()) # Should be 0
```

➡️ 🔎 Remaining Missing Values:
0

```
from sklearn.preprocessing import LabelEncoder
```

```
# Binary Encoding
```

```
df['gender'] = df['gender'].map({'M': 0, 'F': 1})
df['disability'] = df['disability'].map({'N': 0, 'Y': 1})
df['is_banked'] = df['is_banked'].astype(int) # Already binary
```

```
# Ordinal Encoding
```

```
education_order = ['No Formal Quals', 'Lower Than A Level', 'A Level or Equivalent', 'HE Qualification', 'Post Graduate Quali
df['highest_education'] = df['highest_education'].apply(lambda x: education_order.index(x) if x in education_order else -1)
```

```
age_order = ['0-35', '35-55', '55<=']
```

```
df['age_band'] = df['age_band'].apply(lambda x: age_order.index(x) if x in age_order else -1)
```

```
imd_order = ['0-10%', '10-20%', '20-30%', '30-40%', '40-50%', '50-60%', '60-70%', '70-80%', '80-90%', '90-100%']
```

```
df['imd_band'] = df['imd_band'].apply(lambda x: imd_order.index(x) if x in imd_order else -1)

# One-Hot Encoding
df = pd.get_dummies(df, columns=['region', 'assessment_type', 'code_module', 'code_presentation'])

print("✅ Categorical Encoding Complete!")
```

→ ✅ Categorical Encoding Complete!

```
print(df.head()) # Check if categorical values are now numerical
print(df.info()) # Confirm data types
```

→

```

14    weight          967569 non-null  float64
15  region_East Anglian Region   967569 non-null  bool
16  region_East Midlands Region  967569 non-null  bool
17  region_Ireland              967569 non-null  bool
18  region_London Region       967569 non-null  bool
19  region_North Region        967569 non-null  bool
20  region_North Western Region 967569 non-null  bool
21  region_Scotland            967569 non-null  bool
22  region_South East Region   967569 non-null  bool
23  region_South Region        967569 non-null  bool
24  region_South West Region   967569 non-null  bool
25  region_Wales               967569 non-null  bool
26  region_West Midlands Region 967569 non-null  bool
27  region_Yorkshire Region    967569 non-null  bool
28  assessment_type_CMA        967569 non-null  bool
29  assessment_type_Exam      967569 non-null  bool
30  assessment_type_TMA        967569 non-null  bool
31  code_module_AAA            967569 non-null  bool
32  code_module_BBB            967569 non-null  bool
33  code_module_CCC            967569 non-null  bool
34  code_module_DDD            967569 non-null  bool
35  code_module_EEE            967569 non-null  bool
36  code_module_FFF            967569 non-null  bool
37  code_module_GGG            967569 non-null  bool
38  code_presentation_2013B    967569 non-null  bool
39  code_presentation_2013J    967569 non-null  bool
40  code_presentation_2014B    967569 non-null  bool
41  code_presentation_2014J    967569 non-null  bool
dtypes: bool(27), float64(5), int64(10)
memory usage: 135.6 MB
None

```

```

# Drop unnecessary ID columns
drop_columns = ['id_student', 'id_assessment', 'id_site', 'code_module', 'code_presentation']
df = df.drop(columns=drop_columns, errors='ignore')

print("✅ Dropped unnecessary columns.")
print("📊 New Shape:", df.shape)

```

→ ✅ Dropped unnecessary columns.
📊 New Shape: (967569, 40)

```
print(df.columns.tolist()) # Check current columns

# Adjust numerical columns based on available data
num_cols = [col for col in ['date_submitted', 'score', 'date', 'weight', 'sum_click'] if col in df.columns]

# Fill missing numerical values with median
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

# Fill categorical missing values with mode (most frequent value)
df = df.fillna(df.mode().iloc[0])

print("✅ Missing values handled.")
print("🔍 Remaining missing values:\n", df.isnull().sum().sum()) # Should be 0

→ ['gender', 'highest_education', 'imd_band', 'age_band', 'num_of_prev_attempts', 'studied_credits', 'disability', 'final_r
  ✅ Missing values handled.
  🔎 Remaining missing values:
  0

from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = df.drop(columns=['final_result']) # Features
y = df['final_result'] # Target

# Split dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("✅ Data split complete!")
print("📊 Training Set Shape:", X_train.shape, y_train.shape)
print("📊 Test Set Shape:", X_test.shape, y_test.shape)

→ ✅ Data split complete!
  📊 Training Set Shape: (774055, 39) (774055,)
  📊 Test Set Shape: (193514, 39) (193514,)
```

```
time_steps = 3 # Change this to a factor of 39 (e.g., 3, 13)
features_per_timestep = num_features // time_steps # Ensure divisibility

# Reshape
X_train_lstm = np.reshape(X_train.values, (X_train.shape[0], time_steps, features_per_timestep))
X_test_lstm = np.reshape(X_test.values, (X_test.shape[0], time_steps, features_per_timestep))

print(f"✓ Reshaped X_train for LSTM: {X_train_lstm.shape}")
print(f"✓ Reshaped X_test for LSTM: {X_test_lstm.shape}")
```

→ ✓ Reshaped X_train for LSTM: (774055, 3, 13)
✓ Reshaped X_test for LSTM: (193514, 3, 13)

◆ LSTM Model Code

```
print(X_train_lstm.dtype) # Check the dtype of the array
```

→ object

```
X_train = pd.DataFrame(X_train) # If X_train is a NumPy array, convert to DataFrame
X_train = X_train.apply(pd.to_numeric, errors='coerce')

X_test = pd.DataFrame(X_test)
X_test = X_test.apply(pd.to_numeric, errors='coerce')

X_train_lstm = np.reshape(X_train.values, (X_train.shape[0], time_steps, num_features // time_steps))
X_test_lstm = np.reshape(X_test.values, (X_test.shape[0], time_steps, num_features // time_steps))
```

```
print(X_train_lstm.dtype) # Should be float32 or int64
print(y_train.dtype) # Should be int
```

→ object
int64

```
non_numeric_cols = X_train.select_dtypes(exclude=['number']).columns
print("🔴 Non-Numeric Columns:", non_numeric_cols.tolist())
```

→ 🔴 Non-Numeric Columns: ['region_East Anglian Region', 'region_East Midlands Region', 'region_Ireland', 'region_London Re

```
X_train = X_train.astype(float)
X_test = X_test.astype(float)
```

```
X_train_lstm = np.reshape(X_train.values, (X_train.shape[0], time_steps, X_train.shape[1] // time_steps))
X_test_lstm = np.reshape(X_test.values, (X_test.shape[0], time_steps, X_test.shape[1] // time_steps))
```

```
print("✅ Reshaped X_train_lstm:", X_train_lstm.shape)
print("✅ Reshaped X_test_lstm:", X_test_lstm.shape)
```

→ ✅ Reshaped X_train_lstm: (774055, 3, 13)
✅ Reshaped X_test_lstm: (193514, 3, 13)

```
history = model.fit(
    X_train_lstm, y_train,
    validation_data=(X_test_lstm, y_test),
    epochs=20, batch_size=32
)
```

→ Epoch 1/20
24190/24190 172s 7ms/step - accuracy: 0.5729 - loss: 1.0240 - val_accuracy: 0.6009 - val_loss: 0.943

```
Epoch 2/20  
24190/24190 159s 7ms/step - accuracy: 0.6036 - loss: 0.9380 - val_accuracy: 0.6252 - val_loss: 0.8851  
Epoch 3/20  
24190/24190 207s 7ms/step - accuracy: 0.6240 - loss: 0.8930 - val_accuracy: 0.6442 - val_loss: 0.8461  
Epoch 4/20  
24190/24190 157s 6ms/step - accuracy: 0.6373 - loss: 0.8626 - val_accuracy: 0.6590 - val_loss: 0.8121  
Epoch 5/20  
24190/24190 204s 7ms/step - accuracy: 0.6508 - loss: 0.8361 - val_accuracy: 0.6693 - val_loss: 0.7931  
Epoch 6/20  
24190/24190 198s 6ms/step - accuracy: 0.6598 - loss: 0.8182 - val_accuracy: 0.6814 - val_loss: 0.7691  
Epoch 7/20  
24190/24190 200s 6ms/step - accuracy: 0.6658 - loss: 0.8018 - val_accuracy: 0.6903 - val_loss: 0.7471  
Epoch 8/20  
24190/24190 198s 6ms/step - accuracy: 0.6743 - loss: 0.7861 - val_accuracy: 0.6896 - val_loss: 0.7501  
Epoch 9/20  
24190/24190 160s 7ms/step - accuracy: 0.6802 - loss: 0.7756 - val_accuracy: 0.6965 - val_loss: 0.7471  
Epoch 10/20  
24190/24190 200s 7ms/step - accuracy: 0.6838 - loss: 0.7670 - val_accuracy: 0.7038 - val_loss: 0.7221  
Epoch 11/20  
24190/24190 195s 6ms/step - accuracy: 0.6889 - loss: 0.7558 - val_accuracy: 0.7082 - val_loss: 0.7141  
Epoch 12/20  
24190/24190 153s 6ms/step - accuracy: 0.6940 - loss: 0.7468 - val_accuracy: 0.7166 - val_loss: 0.6991  
Epoch 13/20  
24190/24190 201s 6ms/step - accuracy: 0.6982 - loss: 0.7393 - val_accuracy: 0.7255 - val_loss: 0.6791  
Epoch 14/20  
24190/24190 161s 7ms/step - accuracy: 0.7007 - loss: 0.7334 - val_accuracy: 0.7248 - val_loss: 0.6781  
Epoch 15/20  
24190/24190 202s 7ms/step - accuracy: 0.7039 - loss: 0.7263 - val_accuracy: 0.7279 - val_loss: 0.6721  
Epoch 16/20  
24190/24190 194s 6ms/step - accuracy: 0.7064 - loss: 0.7216 - val_accuracy: 0.7243 - val_loss: 0.6851  
Epoch 17/20  
24190/24190 208s 7ms/step - accuracy: 0.7100 - loss: 0.7140 - val_accuracy: 0.7222 - val_loss: 0.6891  
Epoch 18/20  
24190/24190 197s 6ms/step - accuracy: 0.7113 - loss: 0.7116 - val_accuracy: 0.7326 - val_loss: 0.6611  
Epoch 19/20  
24190/24190 161s 7ms/step - accuracy: 0.7130 - loss: 0.7086 - val_accuracy: 0.7331 - val_loss: 0.6601  
Epoch 20/20  
24190/24190 199s 7ms/step - accuracy: 0.7160 - loss: 0.7031 - val_accuracy: 0.7410 - val_loss: 0.6481
```

```
# Evaluate on test set  
test_loss, test_accuracy = model.evaluate(X_test_lstm, y_test)
```

```
print(f"✓ Test Accuracy: {test_accuracy:.4f}")
```

```
→ 6048/6048 ━━━━━━━━━━ 17s 3ms/step - accuracy: 0.7402 - loss: 0.6502
✓ Test Accuracy: 0.7410
```

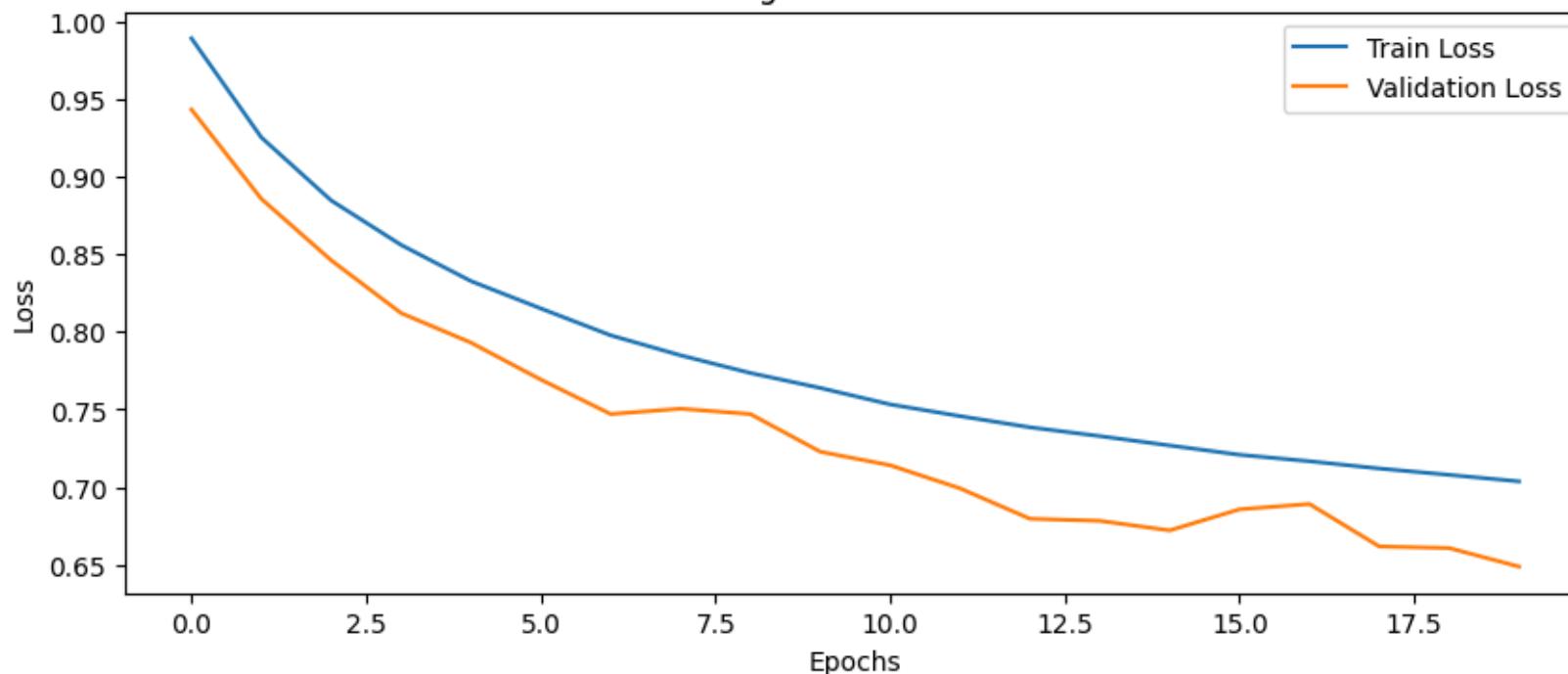
```
import matplotlib.pyplot as plt

# Plot training & validation loss
plt.figure(figsize=(10,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training & Validation Loss')
plt.show()

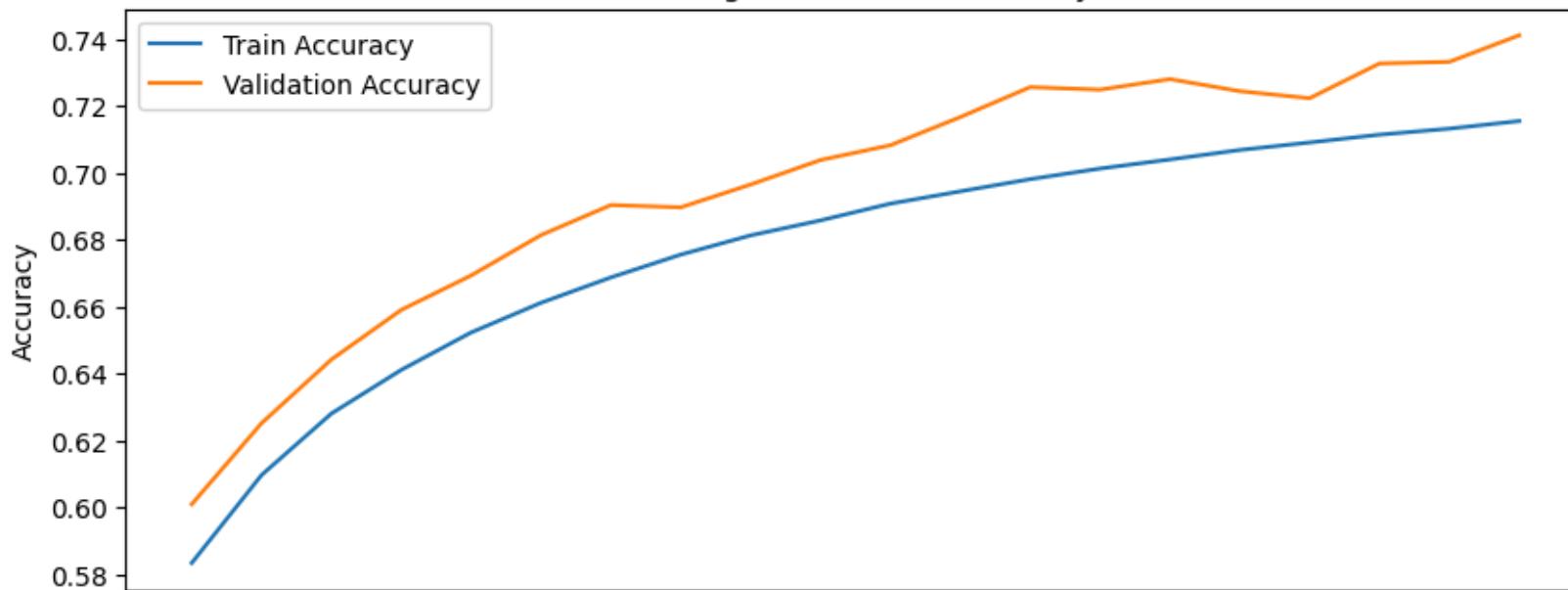
# Plot training & validation accuracy
plt.figure(figsize=(10,4))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training & Validation Accuracy')
plt.show()
```



Training & Validation Loss



Training & Validation Accuracy





```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import numpy as np

# Predict classes
y_pred = model.predict(X_test_lstm)
y_pred_classes = np.argmax(y_pred, axis=1)

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Print classification report
print(classification_report(y_test, y_pred_classes))
```

6048/6048

18s 3ms/step

