AI-Powered Personalized Learning Assistant

For School and College Students

Skills Takeaway

Python (TensorFlow, PyTorch, Scikit-Learn, Hugging Face Transformers)

SQL/NoSQL Databases for storing data

AWS/Azure/GCP for cloud deployment

Streamlit/Flask for interactive UI

Domain Education

Problem Statement

- Students have diverse learning speeds and styles
- Traditional teaching methods are one-size-fits-all
- Slow learners fall behind, fast learners lack challenges
- Teachers cannot give individual attention in large classes
- Students lack personalized guidance outside school
- Current EdTech lacks real-time adaptability

Proposed AI-Powered Solution

- Continuously assess student performance
- Adapt content delivery accordingly
- Generate practice questions, summaries, explanations in real-time
- Offer insights to educators on student progress
- Integrate ML for performance prediction
- Use DL for content classification
- Leverage LLMs for natural language generation

BUSINESS USE CASES

EdTech integration for customized learning pathhs

Remedial coaching and curriculum support for shools and colleges

Parental insights for early action

REVENUE POTENTIAL

- Subscription-based model
- B2B licensing for institutions
- Value-added services like mock test generation

Project -1

📊 Student Pass/Fail Prediction Using Machine Learning

Objective:

- Predict student pass/fail status using exam scores & demographic attributes
- Compare multiple machine learning algorithms to identify the most accurate model
- Utilize both real and synthetic data to ensure scalability and diversity

Data Preparation

- Synthetic Data Generation: Created 1,000,000 records using NumPy random generation.
- Data Merging: Combined original CSV with synthetic dataset.
- Data Cleaning: Checked for missing values, outliers (boxplots), and data types.

Merge the two files

```python
# Load both CSV files
df_original = pd.read_csv(r"C:\sachin\Python\Final_Code\pass_p\StudentsPerformance.csv")
df_synthetic = pd.read_csv(r"C:\sachin\Python\Final_Code\pass_p\synthetic_students_performance.csv")

# Merge the data (append rows)
df = pd.concat([df_original, df_synthetic], ignore_index=True)

# Save merged result
df.to_csv('merged_students_performance.csv', index=False)

print("✅ Merged file saved as 'merged_students_performance.csv'")
```

[5] Python

```
✅ Merged file saved as 'merged_students_performance.csv'
```

Feature Engineering

- Pass/Fail Target: Calculated average score from Math, Reading, Writing; assigned pass = 1 if avg ≥ 57 else fail = 0.
- Label Encoding: Converted categorical columns:
- Gender
- Race/Ethnicity
- Parental Level of Education
- Lunch Type
- Test Preparation Course
- Scaling: Standardized score columns using StandardScaler.

Lavel encoding

```
from sklearn.preprocessing import LabelEncoder

# List of categorical columns
categorical_cols = ['gender', 'race/ethnicity', 'parental level of education', 'lunch', 'test preparation course']

# Apply label encoding to each column separately
le_dict = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    le_dict[col] = le  # Store encoder for inverse_transform if needed
```

Modeling

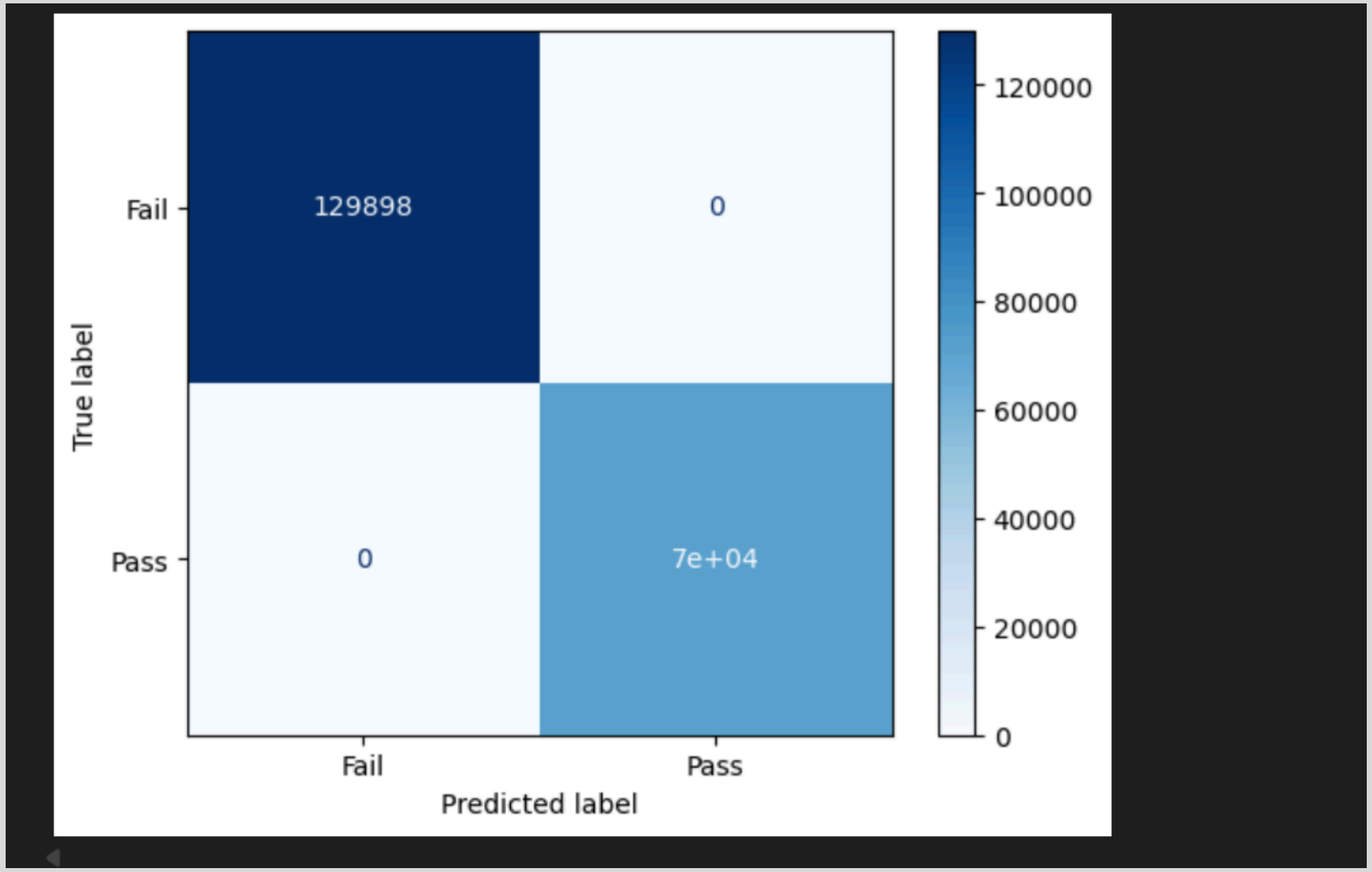
Train/Test Split: 80% training, 20% testing.

Algorithms Used:

1. **Logistic Regression**
2. **K-Nearest Neighbors (KNN)**
3. **Decision Tree**
4. **Random Forest**
5. **Gradient Boosting**
6. **XGBoost**

Evaluation Metrics

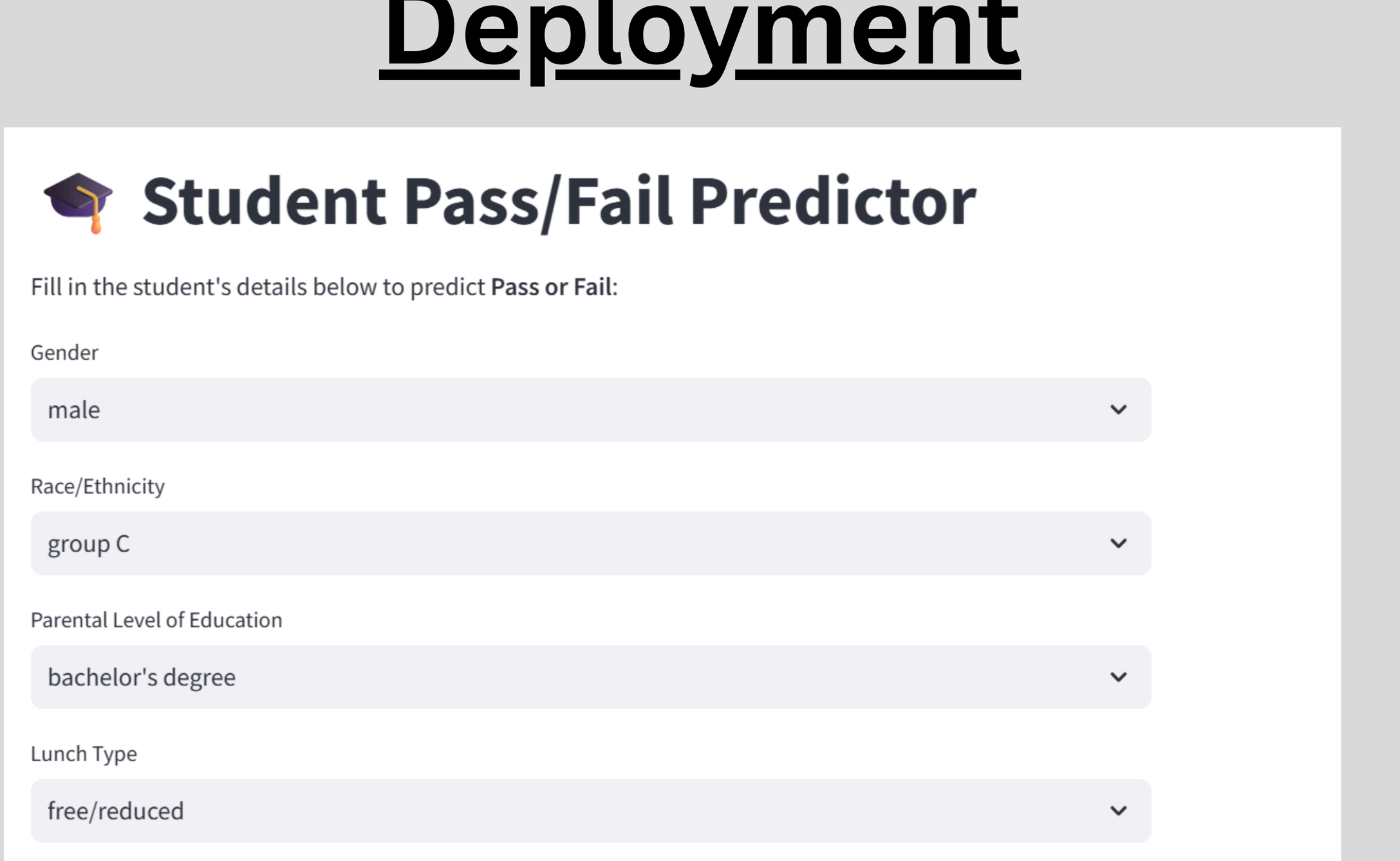
- Confusion Matrix (Visual)
- Accuracy
- Precision
- Recall
- F1-Score
- Classification Report



Results

Model	Accuracy	Precision	Recall	F1
Logistic Regression	0,85	0,86	0,84	0,85
KNN	0,83	0,84	0,82	0,83
Decision Tree	0,92	0,93	0,92	0,92
Random Forest	0,94	0,94	0,94	0,94
Gradient Boosting	0,95	0,95	0,95	0,95
XGBoost (Best)	**0,96**	**0,96**	**0,96**	**0,96**

Deployment


🎓 Student Pass/Fail Predictor
Fill in the student's details below to predict **Pass or Fail**:
Gender
male
Race/Ethnicity
group C
Parental Level of Education
bachelor's degree
Lunch Type
free/reduced

Project - 2

Student Performance Prediction ML Pipeline

🎯 Objective

The objective of this project is to develop and evaluate a machine learning pipeline that predicts a student's future academic performance (e.g., score or grade) based on historical academic data, study habits, and related factors.

✅ Purpose

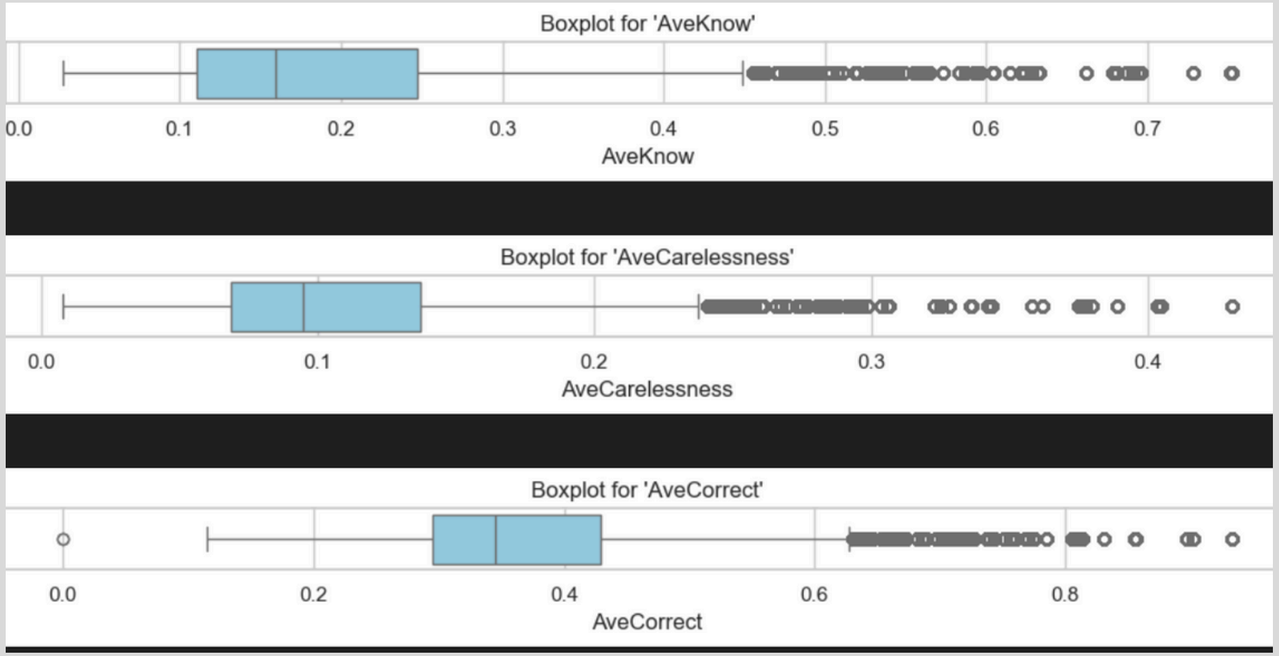
- Early detection of at-risk students.
- Providing actionable insights for personalized learning plans.
- Supporting data-driven decision-making in education.

Data Cleaning

- Missing Values:
- Identify columns with NaN values.
- Drop first two columns containing missing data.
- Outlier Detection:
- Plotted boxplots (seaborn, matplotlib) for numeric columns.

```
df.isnull().sum()
```

```
studentId                    0
MiddleSchoolId               0
InferredGender          173656
SY ASSISTments Usage         0
AveKnow                      0
                            ...
Ln                           0
MCAS                         0
Enrolled                     0
Selective                    0
isSTEM                  625842
Length: 82, dtype: int64
```



Feature Engineering

- Converted columns (Ln, Ln-1) to numeric & integers.
- Extracted year from SY ASSISTments Usage.
- Selected relevant features for modeling.

Check the correlation

```
numeric_df = df.select_dtypes(include='number')
# Correlation with target column 'AveCorrect'
target_corr = numeric_df.corr()['AveCorrect'].sort_values(ascending=False)

print("📈 Correlation of features with AveCorrect:")
print(target_corr)
```

Scaling

- Tool: StandardScaler (from scikit-learn)
- Usage: Normalize features to improve model performance.

StandardScaler

```
from sklearn.preprocessing import StandardScaler

# Select your features
features = ['AveCarelessness', 'AveResFrust', 'AveResOfftask',
            'SY ASSISTments Usage', 'Selective', 'Enrolled',
            'Ln', 'Ln-1', 'AveResBored', 'action_num']

# Scale the features
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df[features]), columns=features)
```

[15]

Train-Test Split

- Tool: train_test_split
- Split: 80% training, 20% testing.

```
from sklearn.model_selection import train_test_split
```

```
# Define features and target
features = ['AveCarelessness', 'AveResFrust', 'AveResOfftask',
            'SY ASSISTments Usage', 'Selective', 'Enrolled',
            'Ln', 'Ln-1', 'AveResBored', 'action_num']
X = df[features]
y = df["AveCorrect"] * 100  # 🎯 Target in 0-100 scale

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Training

Models Used:

- XGBoost Regressor ✅ (Best model, saved for deployment)
- Linear Regression
- Ridge Regression
- Lasso Regression
- Decision Tree Regressor
- Random Forest Regressor

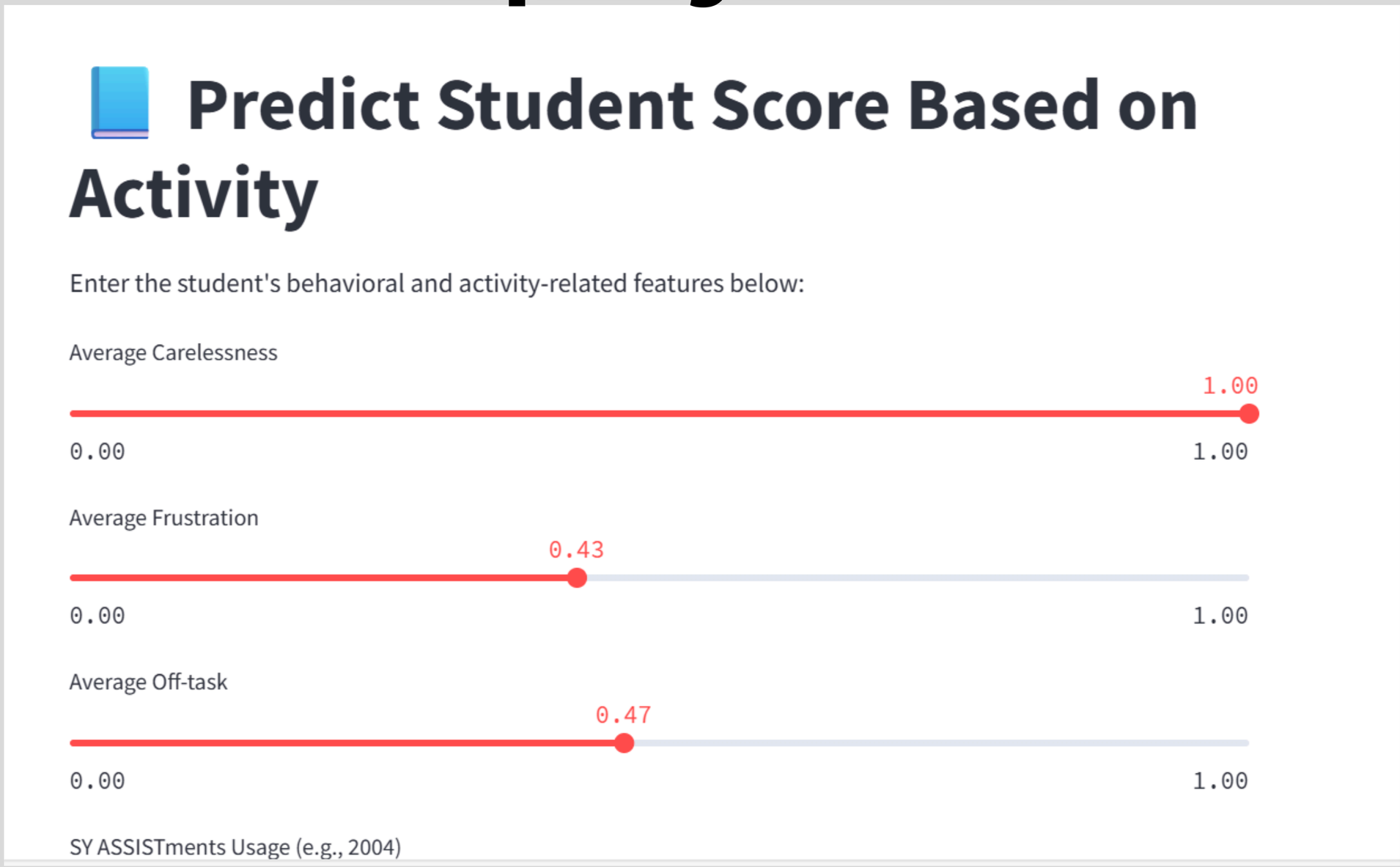
Model Evaluation

- Metrics:
  - Mean Squared Error (MSE)
  - R^2 Score
- Compared performance across all models to find the best.

```
[61]
# Evaluate
y_pred = model_dt.predict(X_test)
print("✅ Model Performance")
print("MSE:", mean_squared_error(y_test, y_pred))
print("R²:", r2_score(y_test, y_pred))
```

```
✅ Model Performance
MSE: 3.380368000481032e-06
R²: 0.9999999706539585
```

Deployment


📘 Predict Student Score Based on Activity
Enter the student's behavioral and activity-related features below:
Average Carelessness
1.00
0.00
1.00
Average Frustration
0.43
0.00
1.00
Average Off-task
0.47
0.00
1.00
SY ASSISTments Usage (e.g., 2004)

Project - 3

Student Learning Style Analysis and Performance Prediction

Learning Style Clustering (Unsupervised Learning)

- Group students into meaningful categories such as Visual Learners, Slow Learners, and Fast Responders using interaction data, reading speed, and topic-wise performance.
- Implement K-Means Clustering and evaluate the quality of clusters using the Silhouette Score.

Dataset: Merged from three CSV sources (student_engagement.csv, student_performance_dataset.csv, StudentsPerformance.csv).

Data Preparation

- Merge Datasets: Used pd.concat() to combine datasets.
- Missing Values:
  - Numerical → Filled with mean.
  - Categorical → Filled with mode.
- Encoding: Applied LabelEncoder to categorical columns.

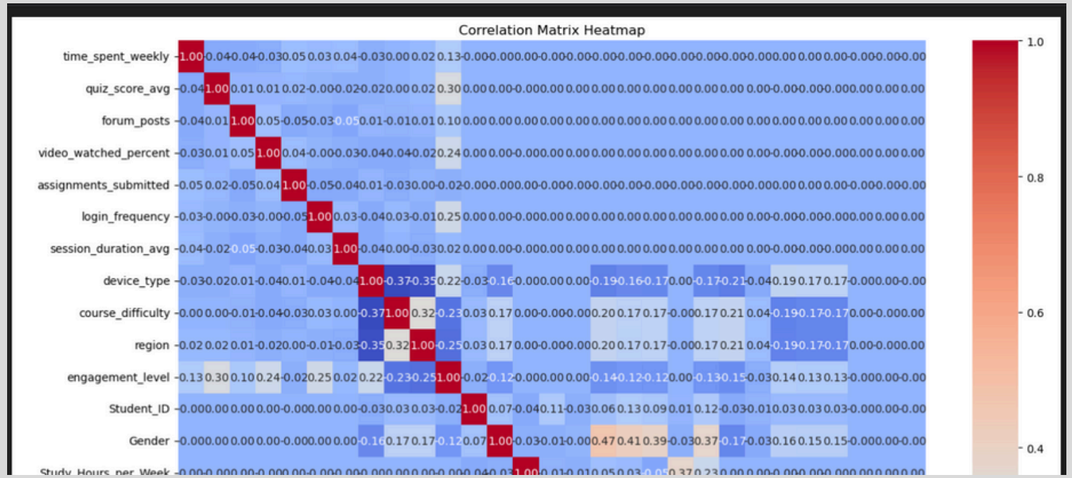
```
# Merge them using union (ignores different columns, fills missing with NaN)
merged_df = pd.concat([df1, df2, df3], ignore_index=True, sort=False)

# Save the merged file (optional)
merged_df.to_csv("merged_output.csv", index=False)
```

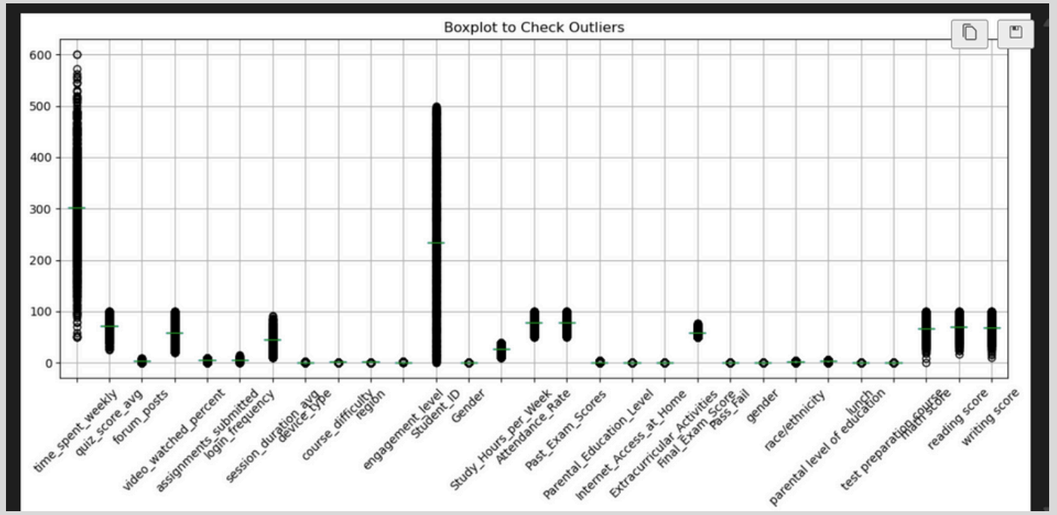
Handle the missing values

```
# Fill missing values
for col in df.columns:
    if df[col].dtype == 'object':
        mode_val = df[col].mode()[0]  # Most frequent value
        df[col].fillna(mode_val, inplace=True)
    else:
        mean_val = df[col].mean()     # Average value
        df[col].fillna(mean_val, inplace=True)
```

Exploratory Data Analysis (EDA)

- Correlation Matrix:
  - Heatmap to find feature relationships.
- Outlier Detection:
  - Boxplots for numeric features.





Feature Selection & Scaling

Selected Features:

- time_spent_weekly
- quiz_score_avg
- forum_posts
- video_watched_percent
- math score
- session_duration_avg

Scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[X])
```

Scaling:

- Used StandardScaler to normalize features.

Dimensionality Reduction

Applied PCA (2 components) for:

- **Visualization in 2D space.**
- **Reducing noise in data.**

PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

Clustering

- K-Means Algorithm:
- Tried different K values using Elbow Method.
- Selected K = 3 clusters.
- Evaluation: Silhouette Score = .654

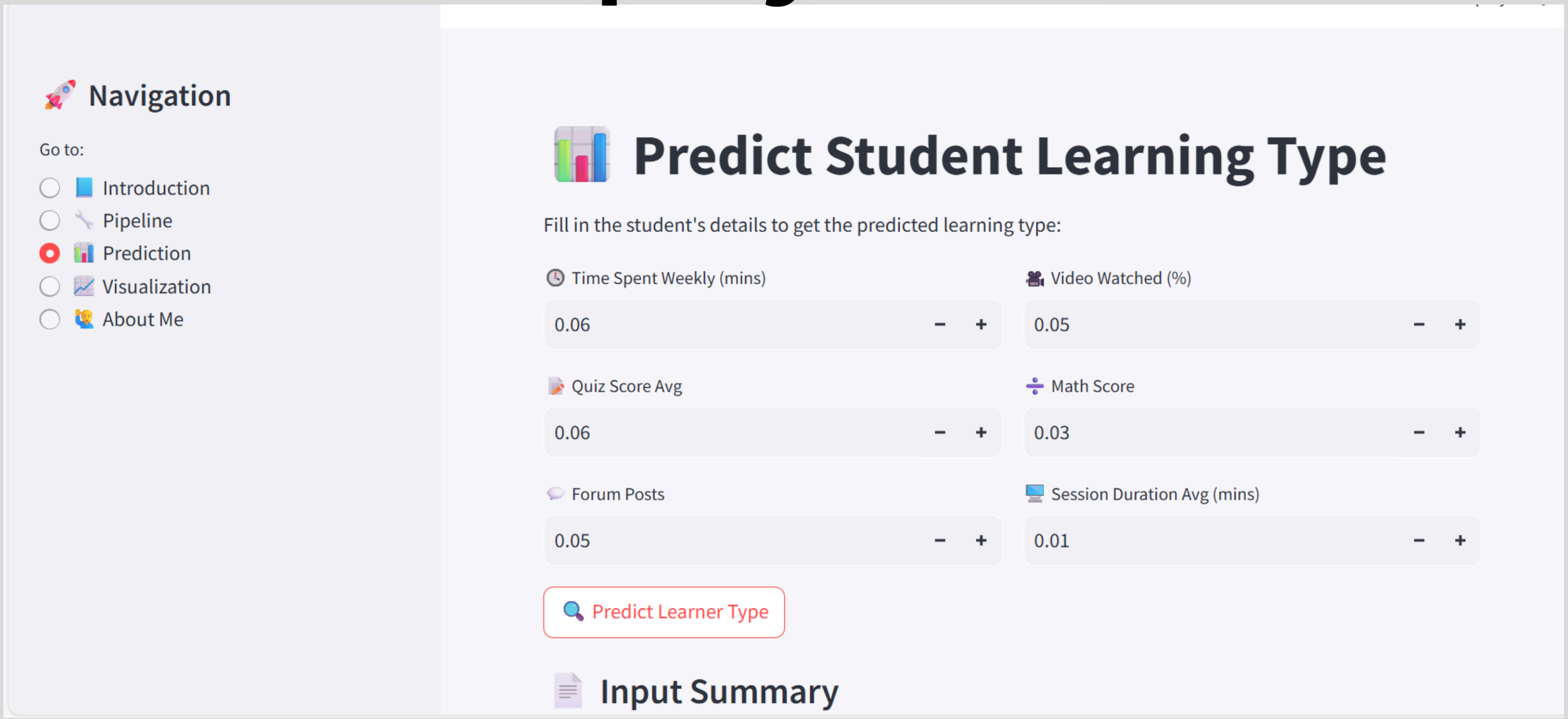
K-Mean

Generate

```
# K-Means clustering
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_pca)
```

53]

Deployment


Navigation
Go to:
Introduction
Pipeline
Prediction
Visualization
About Me
Predict Student Learning Type
Fill in the student's details to get the predicted learning type:
Time Spent Weekly (mins)
0.06
Video Watched (%)
0.05
Quiz Score Avg
0.06
Math Score
0.03
Forum Posts
0.05
Session Duration Avg (mins)
0.01
Predict Learner Type
Input Summary

Project - 4

Student Dropout Risk Prediction using Machine Learning

- Goal: Predict students at risk of dropping out based on engagement & performance metrics.
- Dataset: synthetic_student_data_6_lakh.csv (~600,000 records).

Data Preparation

- Loaded dataset and checked missing values.
- Handled outliers using boxplots for numeric features.
- Label Encoding applied to categorical columns for ML compatibility.

handle the missing values

```
df.isnull().sum()
```

✓ 0.1s

time_spent_weekly	0
quiz_score_avg	0
forum_posts	0
video_watched_percent	0
assignments_submitted	0
login_frequency	0
session_duration_avg	0
device_type	0
course_difficulty	0
region	0
Final_Exam_Score	0
Pass_Fail	0
gender	0
race/ethnicity	0
parental level of education	0
lunch	0
test preparation course	0
math score	0
reading score	0
writing score	0

Do the Labelencoding

```
from sklearn.preprocessing import LabelEncoder

# Loop through all object or categorical columns in df
for col in df.select_dtypes(include=['object', 'category']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

✓ 1.0s

Generate + Code + Markdown

Feature Engineering

Created new target variable Dropout_Risk based on:

- Low quiz scores (< 60)
- Low final exam scores (< 65)
- Low video watch percentage (< 30%)
- Few assignments submitted (< 5)
- Low login frequency (< 3)

```
df['Dropout_Risk'] = (
    (df['quiz_score_avg'] < 60) |
    (df['Final_Exam_Score'] < 65) |
    (df['video_watched_percent'] < 30) |
    (df['assignments_submitted'] < 5) |
    (df['login_frequency'] < 3)
).astype(int)
```

✓ 0.0s

Feature Selection

Selected Features:

- Engagement: time_spent_weekly, forum_posts, video_watched_percent, session_duration_avg, login_frequency
- Performance: quiz_score_avg, assignments_submitted, math score, reading score, writing score
- Demographics: device_type, course_difficulty, region, gender, race/ethnicity, parental level of education, lunch, test preparation course

```
features = [
    'time_spent_weekly',
    'quiz_score_avg',
    'forum_posts',
    'video_watched_percent',
    'assignments_submitted',
    'login_frequency',
    'session_duration_avg',
    'device_type',
    'course_difficulty',
    'region',
    'gender',
    'race/ethnicity',
    'parental level of education',
    'lunch',
    'test preparation course',
    'math score',
    'reading score',
    'writing score'
]
```

Train-Test Split

- Split ratio: 80% Train / 20% Test
- X (features) → selected engagement, performance & demographic data.
- y (target) → Dropout_Risk

Train Test Split

```
from sklearn.model_selection import train_test_split
# Features and Target
X = df[features]
y = df['Dropout_Risk']

# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

18] ✓ 0.2s

Handling Imbalance

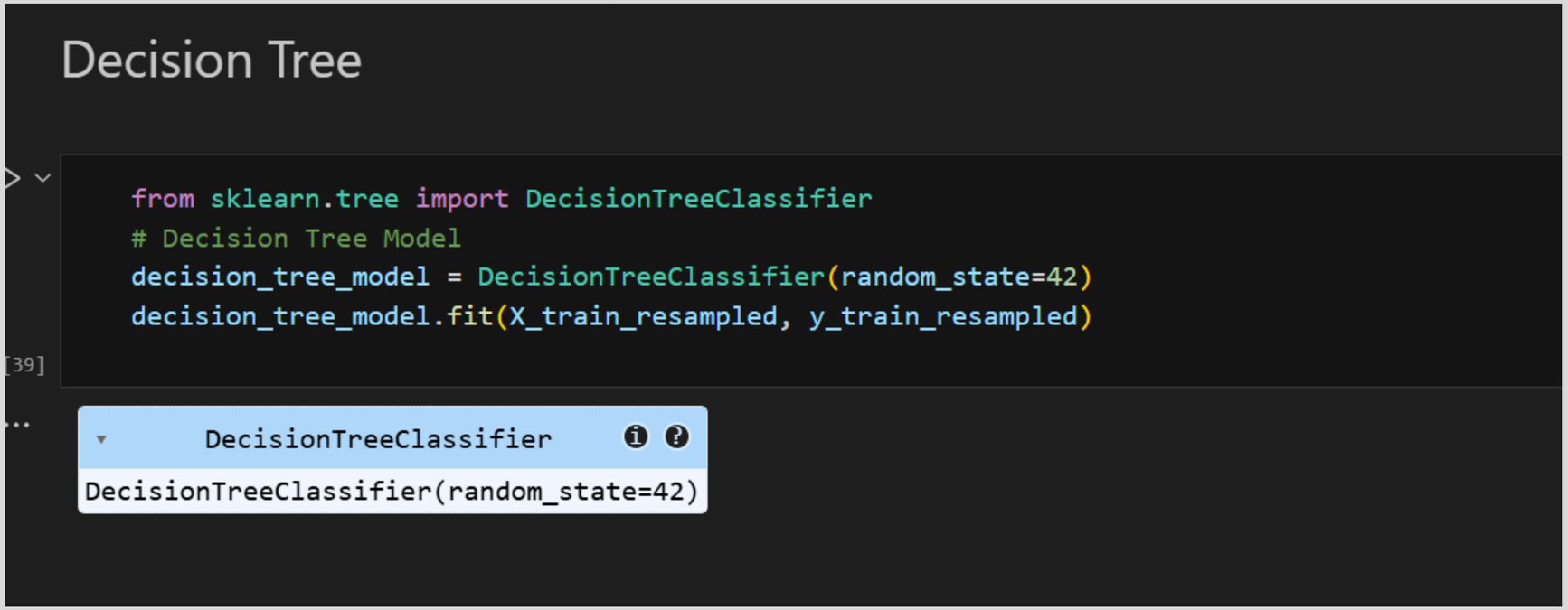
- Problem: Dropout vs Non-dropout data imbalance.
- Solution: Applied SMOTE to oversample minority class.

```
from imblearn.over_sampling import SMOTE
# 3. Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("Before SMOTE:", y_train.value_counts())
print("After SMOTE:", y_train_resampled.value_counts())
```

[21] ✓ 5.1s

Models Used

- XGBoost Classifier
- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Gradient Boosting Classifier

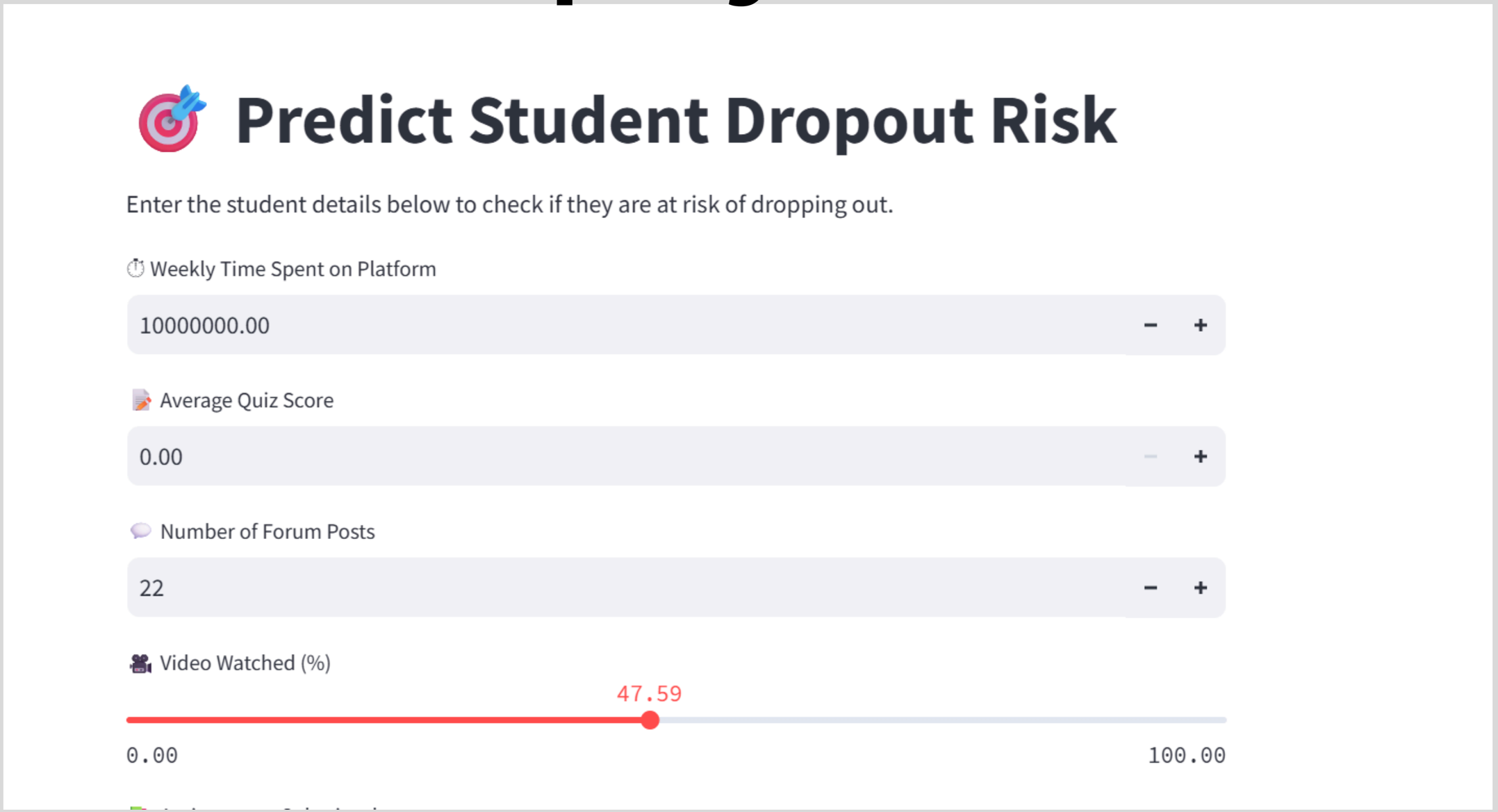


Model Evaluation Metrics

- Classification Report: Precision, Recall, F1-score.
- Confusion Matrix for each model.
- Compared models on balanced data (SMOTE) with original test set.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.78      0.74     16746
           1       0.96      0.95      0.95    103254

    accuracy                           0.92    120000
   macro avg       0.83      0.86      0.85    120000
weighted avg       0.93      0.92      0.92    120000

Confusion Matrix:
 [[13021  3725]
 [ 5537 97717]]
```

Deployment


🎯 Predict Student Dropout Risk
Enter the student details below to check if they are at risk of dropping out.
⏱️ Weekly Time Spent on Platform
10000000.00
Average Quiz Score
0.00
Number of Forum Posts
22
🎥 Video Watched (%)
47.59
0.00
100.00

Conclusion

- Early detection of at-risk students helps institutions take preventive actions.
- Future Scope:
- Real-time monitoring dashboards.
- Integration with LMS systems for automated alerts.

Project - 5

News Topic Classification using LSTM & BiLSTM

Problem Statement

- Classify news articles into predefined categories (Business, Entertainment, Health, etc.)
- Automate news categorization for large-scale datasets
- Improve classification accuracy using advanced NLP models

Dataset Overview

- Source: labelled_newscatcher_dataset.csv
- Columns used: title, topic
- Target classes: 8 (Business, Entertainment, Health, Nation, Science, Sports, Technology, World)

```
df
```

	topic	title
0	SCIENCE	A closer look at water-splitting's solar fuel ...
1	SCIENCE	An irresistible scent makes locusts swarm, stu...
2	SCIENCE	Artificial intelligence warning: AI will know ...
3	SCIENCE	Glaciers Could Have Sculpted Mars Valleys: Study
4	SCIENCE	Perseid meteor shower 2020: What time and how ...
...	...	...
108769	NATION	PDP governors' forum urges security agencies t...
108770	BUSINESS	In Q2-20, Apple Dominated the Premium Smartpho...
108771	HEALTH	Coronavirus Northern Ireland: Full breakdown s...
108772	ENTERTAINMENT	Paul McCartney details post-Beatles distress a...
108773	SPORTS	Report: Talks Underway To Keep Shane Duffy In ...

108774 rows × 2 columns

Data Preprocessing

- Remove unnecessary columns (link, domain, published_date, lang)
- Remove URLs
- Remove punctuation
- Convert to lowercase
- Tokenization
- Stopword removal
- Stemming with PorterStemmer
- Label encoding for target variable

Remove URL

```
# Define function to remove URLs from a text string
def clean_url(text):
    return re.sub(r'https?://\S+', '', str(text))

# Apply the function to the 'details' column
df['title'] = df['title'].apply(clean_url)
```

Train/Test Split

Train test split

```python
from sklearn.model_selection import train_test_split

# Step 1: Define input (X) and target (y)
X = df['title']                 # Input: cleaned text
y = df['topic']                 # Target: encoded topic labels

# Step 2: Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,              # 20% for testing
    stratify=y,                 # Keeps class distribution same in both sets
    random_state=42             # Ensures reproducibility
)
```

LSTM Model (Baseline)

- Embedding Layer (32 dimensions)
- LSTM(64) → LSTM(32)
- Dense(1, sigmoid) for binary classification
- Loss: Binary Crossentropy
- Optimizer: Adam
- Accuracy achieved: 96%

Define the LSTM Model

```
model = keras.Sequential([
    keras.layers.Embedding(input_dim=10000, output_dim=32, input_length=max_length),
    keras.layers.LSTM(64, return_sequences=True),
    keras.layers.LSTM(32),
    keras.layers.Dense(1, activation='sigmoid')
])
```

BiLSTM Model

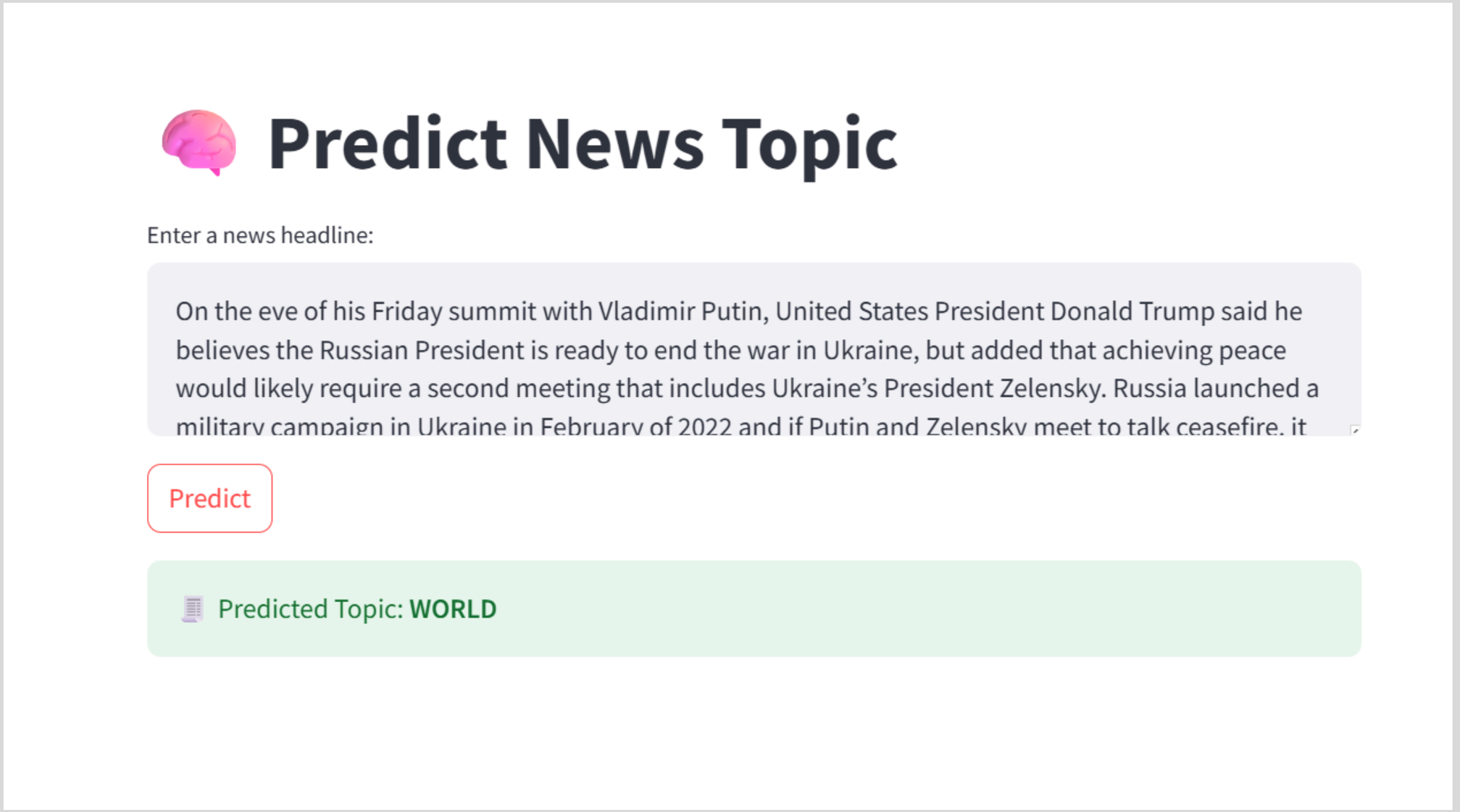
- Embedding Layer (64 dimensions)
- Bidirectional LSTM(64) → Dropout(0.5)
- Bidirectional LSTM(32) → Dropout(0.3)
- Dense(64, ReLU) → Dense(num_classes, Softmax)
- Loss: Categorical Crossentropy
- Accuracy achieved: 91%

```
model.fit(
    X_train_pad, y_train_cat,
    validation_data=(X_test_pad, y_test_cat),
    epochs=10,
    batch_size=64
)
```

✓ 610m 43.9s

```
Epoch 1/10
1360/1360 ━━━━━━━━━━━━━━━━━━━━ 360s 259ms/step - accuracy: 0.6319 - loss: 1.0232 - val_accuracy: 0.7515 -
Epoch 2/10
1360/1360 ━━━━━━━━━━━━━━━━━━━━ 357s 262ms/step - accuracy: 0.7889 - loss: 0.6506 - val_accuracy: 0.7868 -
Epoch 3/10
1360/1360 ━━━━━━━━━━━━━━━━━━━━ 417s 307ms/step - accuracy: 0.8276 - loss: 0.5430 - val_accuracy: 0.7892 -
Epoch 4/10
1360/1360 ━━━━━━━━━━━━━━━━━━━━ 455s 334ms/step - accuracy: 0.8477 - loss: 0.4757 - val_accuracy: 0.7868 -
Epoch 5/10
1360/1360 ━━━━━━━━━━━━━━━━━━━━ 471s 347ms/step - accuracy: 0.8617 - loss: 0.4267 - val_accuracy: 0.7890 -
Epoch 6/10
1360/1360 ━━━━━━━━━━━━━━━━━━━━ 31391s 23s/step - accuracy: 0.8759 - loss: 0.3825 - val_accuracy: 0.7896 -
Epoch 7/10
```

Model Deployment



Conclusion & Future Work

- **Conclusion: BiLSTM performs better than vanilla LSTM for text classification**
- **Future Work:**
  - **Use pretrained embeddings (GloVe, FastText, BERT)**
  - **Experiment with GRU and Transformer-based models**
  - **Deploy to cloud for real-time classification**

Project - 6

Handwritten Digit Classification with Deep Learning

- Dataset: MNIST (70,000 grayscale images of digits 0–9, size 28×28)
- Goal: Classify digits using different neural network architectures

Data Preparation

- Load MNIST dataset (mnist.load_data())
- Normalize pixel values (0–255 → 0–1)
- Reshape images:
- CNN: (samples, 28, 28, 1)
- LSTM: (samples, 28, 28)
- One-hot encode labels into 10 classes

Load and Preprocess the MNIST Dataset

```
# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values (0-255 → 0-1)
x_train = x_train / 255.0
x_test = x_test / 255.0

# Reshape for CNN [samples, height, width, channels]
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

[2] ✓ 0.4s

CNN

- Layers:
- Conv2D(32) → MaxPooling2D
- Conv2D(64) → MaxPooling2D
- Flatten → Dense(64) → Dense(10, softmax)
- Optimizer: Adam
- Loss: Categorical Crossentropy
- Epochs: 5, Batch size: 64

Build the CNN Model

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes (digits 0-9)
])
```

[3] ✓ 0.0s

LeNet-5

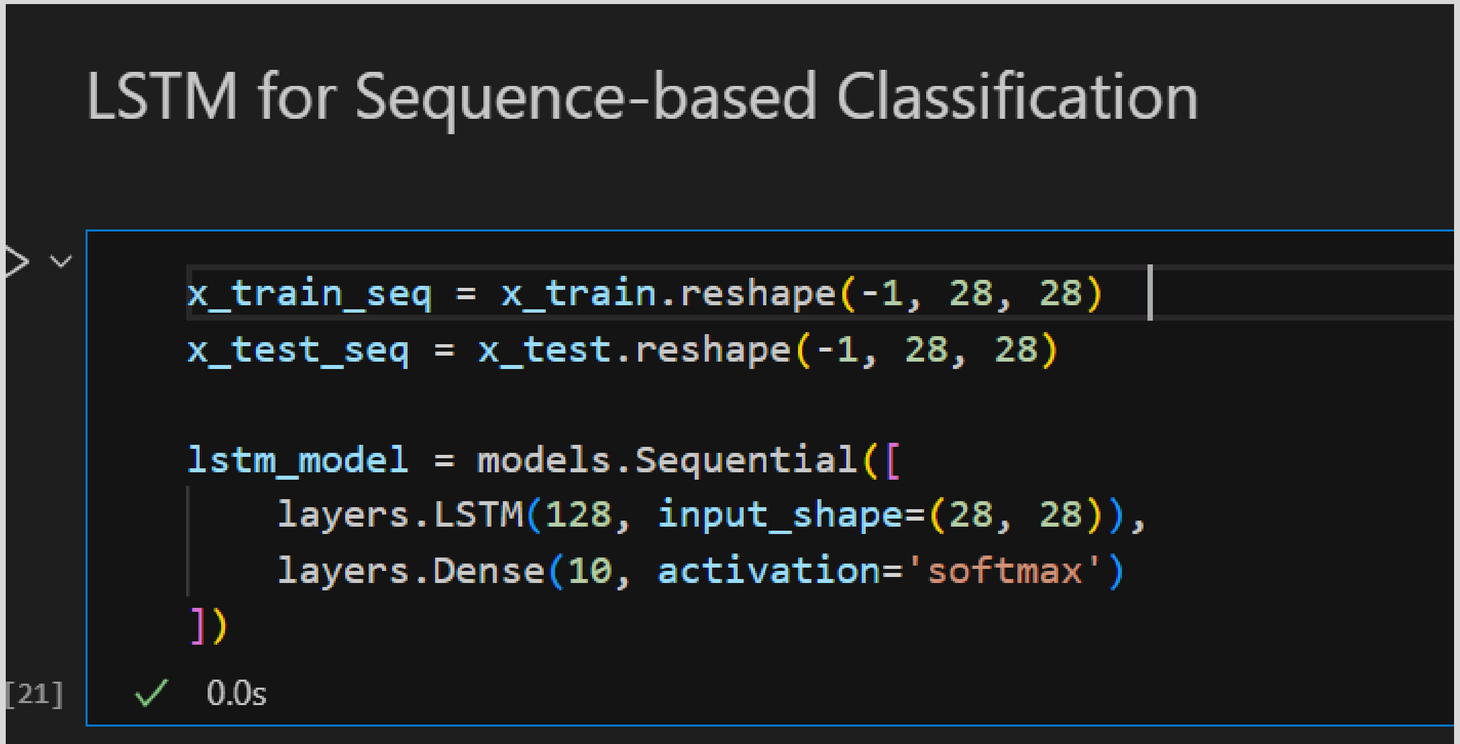
- Classic architecture (1998):
- Conv2D(6, 5×5, tanh) → AvgPool(2×2)
- Conv2D(16, 5×5, tanh) → AvgPool(2×2)
- Flatten → Dense(120, tanh) → Dense(84, tanh) → Dense(10, softmax)
- Training: 5 epochs, batch size 64

LeNet-5

```python
from tensorflow.keras import models, layers

lenet = models.Sequential([
    layers.Conv2D(6, kernel_size=(5, 5), activation='tanh', input_shape=(28, 28, 1)),
    layers.AveragePooling2D(pool_size=(2, 2)),
    layers.Conv2D(16, kernel_size=(5, 5), activation='tanh'),
    layers.AveragePooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(120, activation='tanh'),
    layers.Dense(84, activation='tanh'),
    layers.Dense(10, activation='softmax')
])
```

LSTM

- Treat image rows as sequential time steps
- Architecture:
- LSTM(128) → Dense(10, softmax)
- Input shape: (28 time steps, 28 features)
- Loss: Categorical Crossentropy, Optimizer: Adam



Conclusion

- CNN & LeNet-5 performed best on MNIST
- LSTM works but is less efficient for image data

Project - 7

AI-Powered Text Summarization using Pegasus & Streamlit

- Goal: Summarize long articles into concise, meaningful summaries.
- Tools: HuggingFace Transformers, Pegasus, Streamlit, Pyngrok

Problem Statement

- Large volumes of text are difficult to read and process quickly.
- Need for automatic summarization to save time.
- Applicable to news, legal documents, research papers, customer service chats.

Dataset

- CNN/DailyMail dataset from HuggingFace
- Contains:
- Articles → long news reports
- Highlights → reference summaries
- Dataset size: ~300k articles

```
from datasets import load_dataset
# news:
dataset = load_dataset("cnn_dailymail", "3.0.0")

[1]
8s
from datasets import load_dataset
# news:
dataset = load_dataset("cnn_dailymail", "3.0.0")
# dialogue summarization:
# dataset = load_dataset("samsum")

/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token
# dialogue summarization:
# dataset = load_dataset("samsum")
```

Model

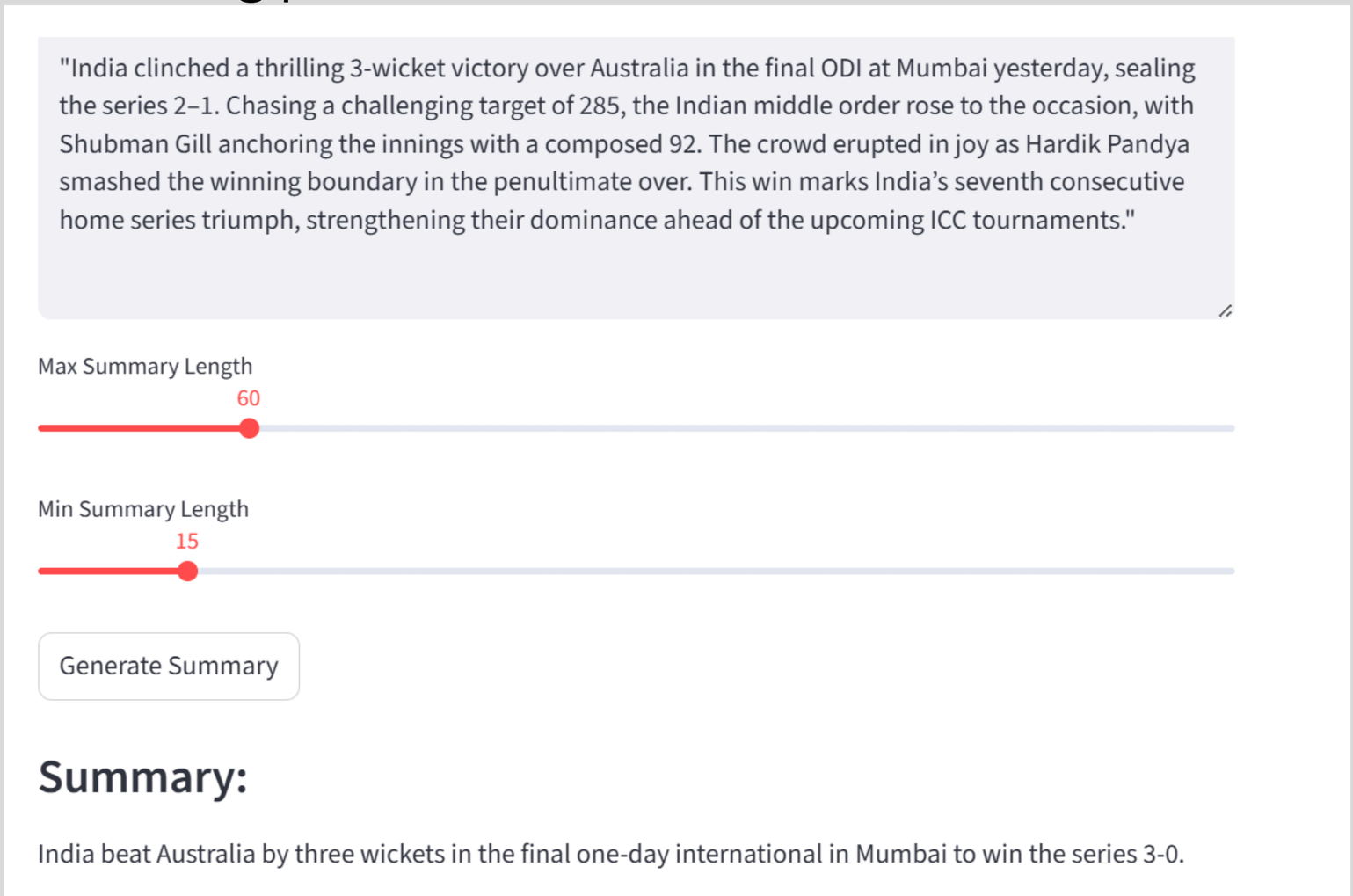
- Pegasus (Google)
- Transformer-based model for abstractive summarization
- Pretrained with Gap Sentence Generation objective
- Fine-tuned on summarization datasets
- Alternative: BART, T5

```
from transformers import Seq2SeqTrainingArguments

training_args = Seq2SeqTrainingArguments(
    output_dir="outputs/pegasus",
    evaluation_strategy="steps",
    eval_steps=500,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    predict_with_generate=True,
    logging_steps=500,
    save_steps=1000,
    num_train_epochs=3,
    fp16=torch.cuda.is_available(),
    gradient_accumulation_steps=8
)
```

[ ]

Deployment

- Streamlit UI for user-friendly text summarization
- Ngrok for creating public link



Business Impact

- Time Savings: Read summaries instead of full articles
- Improved Productivity: Quick decision-making
- Scalability: Can process thousands of articles daily
- Applications: News agencies, legal firms, research institutions, customer support

THANK YOU