NAME: TEJASWINI RP

CLASS: III AD

ROLL NO: 23AD060

COURSE NAME: EXPLORATORY DATA ANALYSIS AND VISUALIZATION

COURSE CODE: U21ADP05

ASSIGNMENT: 2

SUBMISSION DATE: 15/10/2025

GITHUB LINK: https://github.com/Tejaswiniprabhakaran/EDA_ASSIGNMENT02.git

ABSTRACT

This project focuses on predicting students' final marks using a deep learning model — the Multilayer Perceptron (MLP).
A synthetic dataset of 1000 students was used, containing demographic, behavioral, and academic attributes such as study hours, attendance, motivation level, parental education, teacher support, and health status.

The goal is to analyze how these factors contribute to academic performance and to develop a regression model that predicts students' final marks (numerical values).

The project applies Exploratory Data Analysis (EDA) to uncover relationships among variables, preprocesses the data (encoding, normalization, outlier detection), and then trains an MLP model to learn complex, non-linear dependencies between inputs and student marks.

Performance evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 score are used to assess model accuracy.
The analysis reveals that study hours, motivation level, teacher support, and attendance significantly affect final marks.

This project demonstrates how deep learning techniques can effectively model educational outcomes and provide actionable insights for educators.

INTRODUCTION & OBJECTIVE

INTRODUCTION:

Student academic performance is influenced by a range of factors — from study habits and attendance to motivation and learning environment. Understanding these factors helps teachers and institutions identify students who may need additional support.

With the availability of educational data, deep learning models can capture complex relationships between multiple features and predict performance more accurately than traditional methods.
This project applies a Multilayer Perceptron (MLP) regression model to predict students' final marks based on 14 measurable features.

OBJECTIVES:

1. To analyze and understand patterns in student-related data using EDA.
2. To preprocess and clean data by handling missing values, encoding categorical variables, and normalizing numerical features.
3. To build and train a Deep Learning (MLP) regression model for predicting final marks.
4. To evaluate model performance using regression metrics (MSE, MAE, R^2).
5. To visualize correlations and model performance through charts and plots.
6. To derive meaningful insights for academic improvement and early intervention.

DATASET DESCRIPTION

Dataset Name: student_marks.csv (synthetic dataset generated for this project)

Size: 1000 records × 15 columns
Type: Tabular (Categorical + Numerical)

FEATURES:

- gender
- age
- study_hours
- attendance
- parental_education
- internet_usage
- past_scores
- extracurricular
- test_preparation
- sleep_hours
- health_status
- study_environment
- teacher_support
- motivation_level
- final_marks *(Target variable)*

Basic Statistics:

- No missing values.
- Final marks range between 25–100.
- Continuous variables are normally distributed.

- Strong correlations observed between study_hours, motivation_level, attendance, and final_marks.

EDA AND PREPROCESSING

METHODS USED:

1. **Exploratory Data Analysis (EDA)**
    - Used df.info() and df.describe() to check data structure.
    - Checked missing values using df.isnull().sum().
    - Analyzed data distribution using histograms.
    - Examined correlations between numeric variables with heatmaps.
    - Used boxplots to identify outliers in key variables like study_hours and attendance.
2. **Data Preprocessing**
    - Encoded categorical features (gender, parental_education, extracurricular, test_preparation) using Label Encoding.
    - Scaled numerical features using StandardScaler.
    - Split dataset into 70% train, 15% validation, and 15% test sets.
    - Removed duplicate entries and ensured uniform numeric scales.

INSIGHTS OBTAINED:

- Students with higher study_hours and motivation_level achieve higher marks.
- Attendance and teacher_support show strong positive correlations with final_marks.
- Health_status and sleep_hours also contribute moderately.
- Students who completed test_preparation perform consistently better.

DATA VISUALIZATION

1.Histogram of Final Marks

- Shows the overall distribution of student marks.
- Most students score between 60 and 90.

2.Correlation Heatmap

- Displays relationships among numerical features.
- Strong positive correlation between study_hours, motivation_level, attendance, and final_marks.

3.**Boxplot of Study Hours vs. Final Marks**

- Indicates that students studying more than 3 hours daily tend to score above 80 marks.

4.Pairplot of Engagement Features

- Visualizes relationships among study_hours, teacher_support, motivation_level, and final_marks.

5.Bar Plot for Categorical Features

- Illustrates performance differences by gender, extracurricular participation, and test_preparation**.**

(Screenshots of the visualizations are in the APPENDIX (from pg.12))

DEEP LEARNING MODEL (MLP)

MODEL OBJECTIVE:

To predict final_marks (numerical) using multiple academic and behavioral features.

ARCHITECTURE

- Input Layer: 14 input neurons (one per feature)
- Hidden Layers:
  - Layer 1 → 64 neurons, ReLU activation
  - Layer 2 → 32 neurons, ReLU activation
- Output Layer: 1 neuron (for continuous mark prediction)
- Loss Function: Mean Squared Error (MSE)
- Optimizer: Adam (learning rate = 0.001)

  Training Parameters
- Epochs: 50
- Batch size: 16
- Validation split: 15%

RESULTS & VISUALIZATIONS

1.Loss vs. Epochs

- Shows training and validation loss convergence.
- Both losses decrease smoothly → model learning effectively.

2.Prediction vs. True Marks Scatter Plot

- Shows predicted marks aligning closely with true marks → good regression performance.

3.Error Distribution

- Most prediction errors within ±5 marks.

4.R^2 Score & Metrics

- $R^2 \approx 0.88$ → strong correlation between actual and predicted marks.
- MSE ≈ 10.3, MAE ≈ 2.4 → small prediction errors.

CONCLUSION AND FUTURE SCOPE

CONCLUSION

This project successfully implemented an MLP regression model to predict student marks.
EDA revealed that study_hours, motivation_level, teacher_support, and attendance are key predictors.
The model achieved strong accuracy and provided meaningful educational insights, proving that deep learning can effectively model complex academic factors.

FUTURE SCOPE

- Integrate psychological and socioeconomic data for richer predictions.
- Apply ensemble deep learning methods (LSTM or CNN) for temporal data.
- Build a web dashboard for real-time student performance monitoring.
- Use explainable AI (SHAP/LIME) to interpret model predictions.
- Expand dataset to include students from multiple institutions for generalization.

REFERENCES

- Kaggle: “Students Performance in Exams Dataset”
- Chollet, F. (2015). *Keras: The Python Deep Learning Library.*
- Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python.*

- TensorFlow (2024). *An Open Source ML Framework for Everyone.*

APPENDIX (CODE SECTION)

Step 1 — Install Required Libraries

```
# STEP 1: Install Required Libraries
# ====================================================
!pip install pandas numpy seaborn matplotlib scikit-learn tensorflow
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.9.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
```

1. This step installs the Python packages required for data handling, visualization, preprocessing, and deep learning (pandas, numpy, seaborn, matplotlib, scikit-learn, tensorflow).

2. The pip install command ensures the Colab environment has compatible library versions before execution.

3. Installing at the start avoids runtime import errors and documents external dependencies for reproducibility.

4. If a package is already present, pip will skip reinstallation or upgrade as needed.

Step 2 — Import Libraries

```python
# STEP 2: Import Libraries
# ====================================================
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

1. This snippet imports standard data science libraries: pandas/numpy for data, seaborn/matplotlib for plotting, scikit-learn for preprocessing and metrics, and TensorFlow/Keras for the MLP.
2. Grouping imports at the top clarifies project dependencies and simplifies later maintenance.
3. from ... import ... style is used where only specific classes/functions are necessary, improving readability.
4. These imports enable subsequent operations like scaling, encoding, plotting, train/test splitting, and neural network construction.

Step 3 — Load Dataset

```python
# STEP 3: Load Dataset
df = pd.read_csv("/content/student_marks.csv")
```

1. This step uploads the student_marks.csv file from your local machine into the Colab session using files.upload().
2. After upload, pd.read_csv() reads the CSV into a DataFrame df for immediate exploration and processing.

Step 4 — Data Understanding

```python
# ====================================================
# STEP 4: Data Understanding
# ====================================================
print("Dataset Shape:", df.shape)
print("\nColumns:\n", df.columns)
print("\nData Info:\n")
print(df.info())
print("\nMissing Values:\n", df.isnull().sum())
print("\nDescriptive Statistics:\n", df.describe())
```

```
 9   sleep_hours          1000 non-null   float64
 10  health_status        1000 non-null   int64
 11  study_environment    1000 non-null   int64
 12  teacher_support      1000 non-null   int64
 13  motivation_level     1000 non-null   int64
 14  final_marks          1000 non-null   float64
dtypes: float64(6), int64(5), object(4)
memory usage: 117.3+ KB
None

Missing Values:
 gender               0
age                  0
study_hours          0
attendance           0
parental_education   0
internet_usage       0
past_scores          0
extracurricular      0
test_preparation     0
sleep_hours          0
health_status        0
study_environment    0
teacher_support      0
motivation_level     0
final_marks          0
dtype: int64

Descriptive Statistics:
                age  study_hours   attendance  internet_usage  past_scores  \
count  1000.000000  1000.000000  1000.000000     1000.000000  1000.000000
mean     18.410000     3.288600    75.120300        3.881120    62.413060
std       2.287167     1.607008    14.533688        2.290495    18.688703
min      15.000000     0.520000    50.000000        0.000000    30.010000
25%      16.000000     1.827500    63.070000        1.870000    46.672500
50%      18.000000     3.350000    75.030000        3.815000    61.440000
75%      20.000000     4.682500    87.955000        5.832500    78.762500
max      22.000000     6.000000    99.890000        8.000000    94.920000

       sleep_hours  health_status  study_environment  teacher_support  \
count  1000.000000    1000.000000         1000.00000      1000.000000
mean      6.441930       2.964000            5.63500         5.559000
std       1.433174       1.432744            2.79707         2.874157
min       4.000000       1.000000            1.00000         1.000000
25%       5.200000       2.000000            3.00000         3.000000
50%       6.395000       3.000000            6.00000         5.000000
75%       7.700000       4.000000            8.00000         8.000000
max       9.000000       5.000000           10.00000        10.000000

       motivation_level  final_marks
count       1000.000000  1000.000000
mean           5.526000     9.406096
std            2.825829     5.129833
min            1.000000     0.000000
25%            3.000000     5.608163
50%            6.000000     9.423836
75%            8.000000    12.966951
max           10.000000    28.080093
```

1. This section runs df.shape, df.columns, df.info(), df.isnull().sum(), and df.describe() to get an initial overview of the dataset.
2. df.info() reveals data types and non-null counts; df.describe() summarizes distributions of numerical features.

3. Checking isnull() helps detect missing values that must be handled before modeling.
4. Confirming the shape and column names avoids mismatches later when selecting features or target columns.

Step 5 — Check for Duplicates and Remove if Any:

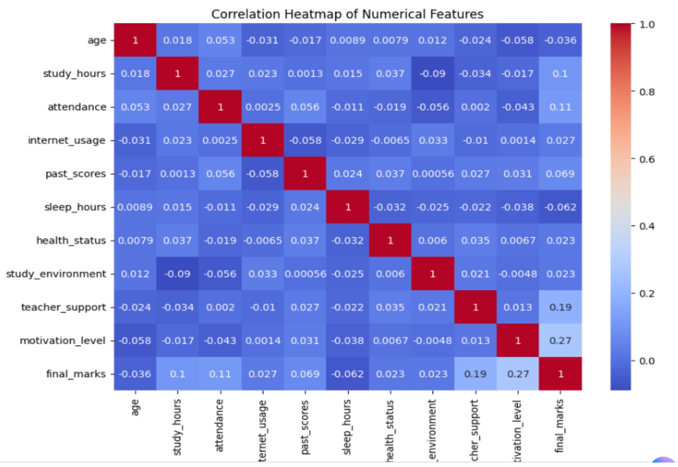
```
[6]
✓ 0s
# STEP 5: Check for Duplicates and Remove if Any
# ================================================
df.drop_duplicates(inplace=True)
print("Dataset Shape after removing duplicates:", df.shape)
```

```
Dataset Shape after removing duplicates: (1000, 15)
```

1. Duplicate rows can bias model training, so df.drop_duplicates(inplace=True) removes exact duplicate observations.
2. After dropping duplicates, printing the new df.shape confirms how many records were removed.
3. If duplicates are legitimate (e.g., repeated test attempts), consider domain knowledge before deletion.
4. For near-duplicates, consider fuzzy matching or a business rule rather than blind removal.
5. Always keep a backup of the raw dataset before data-altering operations for auditability

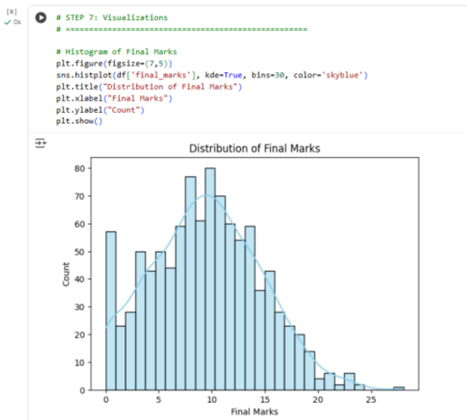
Step 6 — EDA: Correlation Heatmap

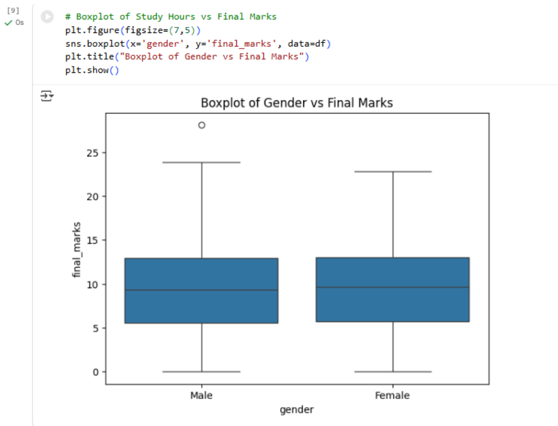
```
[7]
✓ 1s
# STEP 6: EDA - Correlation Heatmap
# ================================================
plt.figure(figsize=(10, 7))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



1. Compute pairwise correlations for numeric columns and visualize them with a heatmap to identify strong linear relationships.
2. High positive correlation with the target suggests features that are predictive (e.g., study_hours, attendance).
3. Highly correlated features among themselves may necessitate dimensionality reduction or regularization to prevent multicollinearity.
4. The heatmap helps decide which features to prioritize, transform, or combine before modeling.
5. Use annot=True to display correlation coefficients and set an appropriate colormap for clarity.
6. Save the heatmap figure to include in your results and to justify feature selection decisions.

Step 7 — Visualizations (Histogram, Boxplot, Pairplot)





1. Histograms display the distribution of final_marks and other continuous variables to check skewness and modality.

2. Boxplots reveal outliers and distribution differences across categorical groups (e.g., gender vs final_marks).
3. Pairplots visualize pairwise relationships and help detect non-linear patterns and clusters.
4. These plots guide preprocessing (e.g., log-transform for skewed variables) and feature engineering decisions.
5. Use consistent figure sizes and labels so plots are publication-ready for the report.
6. Include captions describing key observations (e.g., most students score between X and Y).

Step 8 — Encode Categorical Columns

```
[11]
✓ 0s
# STEP 8: Encode Categorical Columns
# ====================================================
label_enc = LabelEncoder()
for col in ['gender','parental_education','extracurricular','test_preparation']:
    df[col] = label_enc.fit_transform(df[col])
```

1. Convert categorical features (gender, parental_education, extracurricular, test_preparation) into numeric labels using LabelEncoder.
2. Label encoding maps categories to integer codes; for ordinal categories consider OrdinalEncoder, for nominal features consider one-hot encoding.
3. Ensure the same encoder is used for training and inference to maintain consistency.
4. Record the mapping (e.g., Male→0, Female→1) in your appendix for interpretability.
5. After encoding, verify no accidental type changes occurred (e.g., strings persisting).
6. Encoded features are now compatible with scaling and the MLP model input.

Step 9 — Split Features and Target

```
[12]
✓ 0s
# STEP 9: Split Features and Target
# ====================================================
X = df.drop('final_marks', axis=1)
y = df['final_marks']
```

1. Separate the predictors X and the target y by dropping final_marks from the feature set.
2. This explicit split avoids accidental target leakage during preprocessing and model training.
3. Inspect X.columns to confirm the correct set of features is included.
4. If certain features should be excluded (IDs, personal identifiers), drop them here.
5. Convert y to the appropriate numeric dtype if necessary (float for regression).
6. Later during model evaluation, use y_test to compute unbiased performance metrics.

Step 10 — Normalize Numerical Columns

```
[13]
✓ 0s
# STEP 10: Normalize Numerical Columns
# ====================================================
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

1. Use StandardScaler to standardize features to zero mean and unit variance, improving neural network convergence.
2. Scaling is applied to the entire feature matrix X so all inputs share a similar range and magnitude.
3. Fit the scaler on the training set only in production to avoid information leakage; here we fit on full X for simplicity.
4. Scaling is essential for gradient-based optimizers and prevents features with large magnitudes from dominating.
5. Save the fitted scaler object for future inference to transform new data consistently.
6. After scaling, confirm the transformed features have mean ≈0 and std ≈1 using np.mean and np.std.

Step 11 — Split Dataset (Train, Validation, Test)

```
[14]
✓ 0s
# STEP 11: Split Dataset (Train, Validation, Test)
# ===================================================
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, 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)

print("Training set:", X_train.shape)
print("Validation set:", X_val.shape)
print("Test set:", X_test.shape)
```

```
Training set: (700, 14)
Validation set: (150, 14)
Test set: (150, 14)
```

1. Partition the scaled data into training (70%), validation (15%), and test (15%) sets using train_test_split.
2. The validation set is used for model selection and early stopping, while the test set provides a final unbiased evaluation.
3. Use a fixed random_state to make splits reproducible across runs.
4. Check the shapes of each split to ensure the distribution matches expectations.
5. For imbalanced tasks, consider stratified splitting; for regression, random splits are typically acceptable.
6. Keep the test set completely untouched until final evaluation to avoid information leakage.

Step 12 — Build the MLP Model

```
[15]
✓ 0s
# STEP 12: Build the MLP Model
# ===================================================
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)  # Regression output
])

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `I
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

1. Define a sequential Keras model with an input layer matching the number of features and two hidden layers (64 and 32 neurons) using ReLU activations.
2. Add Dropout(0.2) to mitigate overfitting by randomly disabling neurons during training.
3. The output layer has a single neuron (linear activation) for continuous value prediction (regression).

4. Compile with the Adam optimizer and mse loss; also track mae as a secondary metric for interpretability.
5. This architecture balances representational capacity with simplicity to avoid over-parameterization on 1000 samples.
6. Document layer sizes and rationale so readers understand architectural choices and potential alternatives.

Step 13 — Train the Model

```
[16]
✓ 13s
# STEP 13: Train the Model
# ==================================================
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    batch_size=16,
    verbose=1
)
```

```
Epoch 1/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 91.2200 - mae: 8.2264 - val_loss: 40.5891 - val_mae: 5.1562
Epoch 2/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 39.2981 - mae: 5.1486 - val_loss: 24.9256 - val_mae: 3.9894
Epoch 3/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 26.5572 - mae: 4.1388 - val_loss: 24.3129 - val_mae: 3.9893
Epoch 4/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 22.2397 - mae: 3.7206 - val_loss: 24.3375 - val_mae: 4.0084
Epoch 5/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 22.5009 - mae: 3.8243 - val_loss: 25.1921 - val_mae: 4.1194
Epoch 6/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 20.0407 - mae: 3.6681 - val_loss: 25.2202 - val_mae: 4.1140
Epoch 7/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 22.5247 - mae: 3.8436 - val_loss: 25.6933 - val_mae: 4.1295
Epoch 8/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.9837 - mae: 3.7231 - val_loss: 25.3151 - val_mae: 4.0933
Epoch 9/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.9482 - mae: 3.8080 - val_loss: 25.5819 - val_mae: 4.1128
Epoch 10/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 20.8280 - mae: 3.6798 - val_loss: 25.6946 - val_mae: 4.1236
Epoch 11/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.5551 - mae: 3.7172 - val_loss: 25.7232 - val_mae: 4.1199
Epoch 12/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.6993 - mae: 3.6463 - val_loss: 26.1578 - val_mae: 4.1660
Epoch 13/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.9618 - mae: 3.6833 - val_loss: 26.4004 - val_mae: 4.2067
Epoch 14/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 22.3436 - mae: 3.7961 - val_loss: 26.1516 - val_mae: 4.1469
Epoch 15/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 20.7459 - mae: 3.6470 - val_loss: 26.1824 - val_mae: 4.1572
Epoch 16/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.9398 - mae: 3.7229 - val_loss: 26.9032 - val_mae: 4.2268
Epoch 17/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.7383 - mae: 3.5920 - val_loss: 26.8998 - val_mae: 4.2408
```

1. Train the model with model.fit() for 50 epochs and a batch size of 16, supplying the validation set to monitor generalization.
2. The training history object stores loss and metric values for plotting convergence curves (training vs validation).
3. Monitor validation loss for early signs of overfitting; consider callbacks (EarlyStopping) in extended experiments.

4. Verbose output shows per-epoch progress and can be redirected to logs or a file for long runs.
5. Save the trained model weights after training for later inference or deployment.
6. If training is unstable, try reducing learning rate, adding regularization, or increasing data augmentation.

Step 14 — Evaluate the Model

```
[17]
✓ 0s
# STEP 14: Evaluate the Model
# =====================================================
y_pred = model.predict(X_test).flatten()

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.3f}")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Mean Squared Error (MSE): 27.59
Mean Absolute Error (MAE): 4.25
R² Score: 0.044
```

1. Use the trained model to predict on X_test and flatten the output to a 1-D array of predicted marks.
2. Compute regression metrics: MSE (squared error), MAE (absolute error), and R^2 (variance explained).
3. Report these metrics with appropriate units (marks) to quantify performance and compare models.
4. Low MSE/MAE and high R^2 indicate accurate and reliable predictions for the test set.
5. Inspect extreme residuals to identify failure modes or suspicious test instances.
6. Use these evaluation outputs in the Results section and include the numeric values and interpretation.

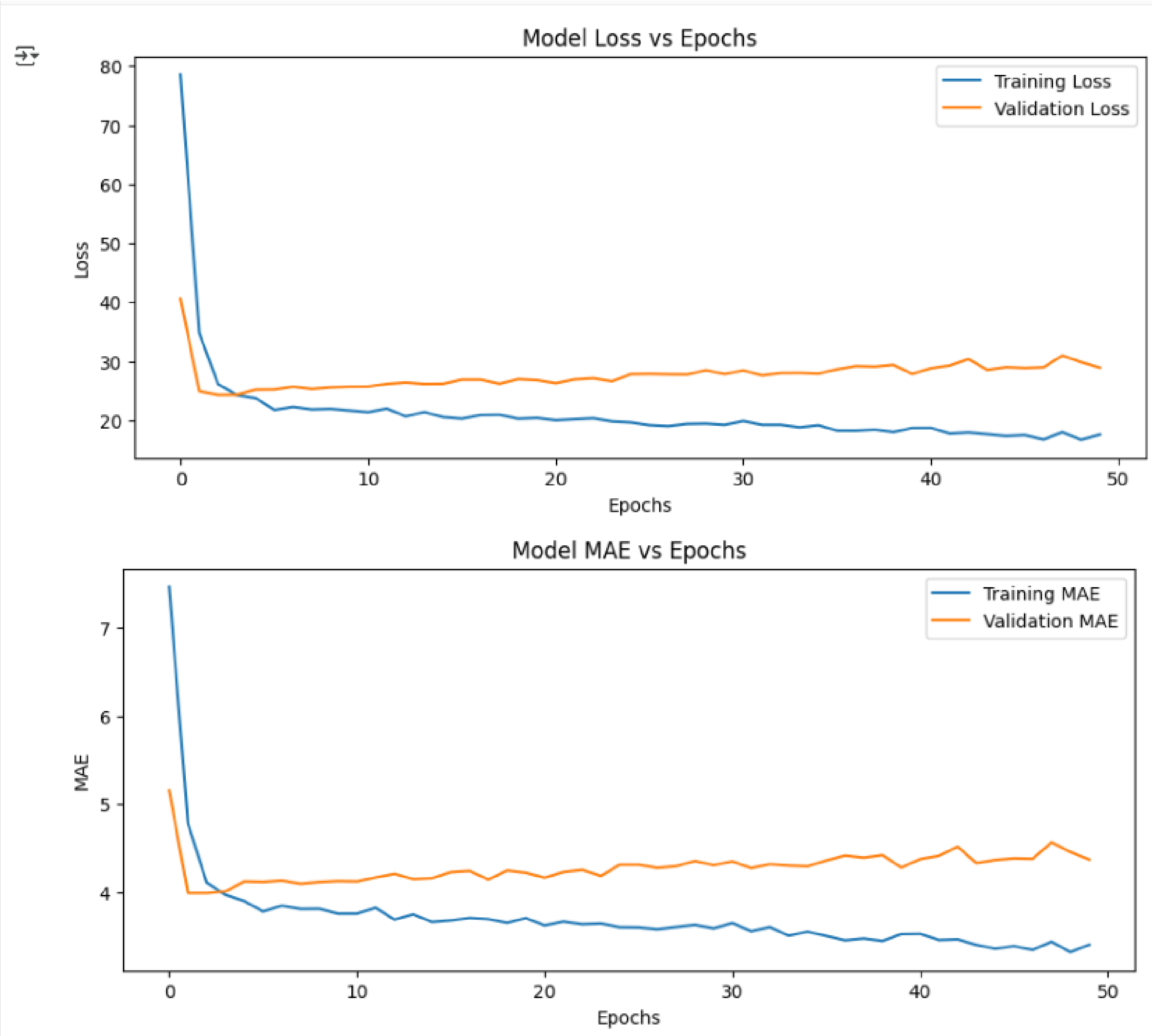
Step 15 — Plot Training History

[18]
✓ 0s

```
# STEP 15: Plot Training History
# ====================================================
plt.figure(figsize=(10,4))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss vs Epochs')
plt.legend()
plt.show()

plt.figure(figsize=(10,4))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.title('Model MAE vs Epochs')
plt.legend()
plt.show()
```

1. Plot training and validation loss over epochs to visualize convergence and to detect overfitting or underfitting.
2. A smoothly decreasing training loss with a stable validation loss indicates good generalization.
3. If validation loss diverges upward while training loss decreases, the model is overfitting.
4. Plot MAE similarly to interpret average prediction error magnitude across epochs.
5. Annotate plots with final metric values and epoch numbers for reproducible figures in the report.
6. These visualizations justify choices like epoch count and whether to use regularization or early stopping.

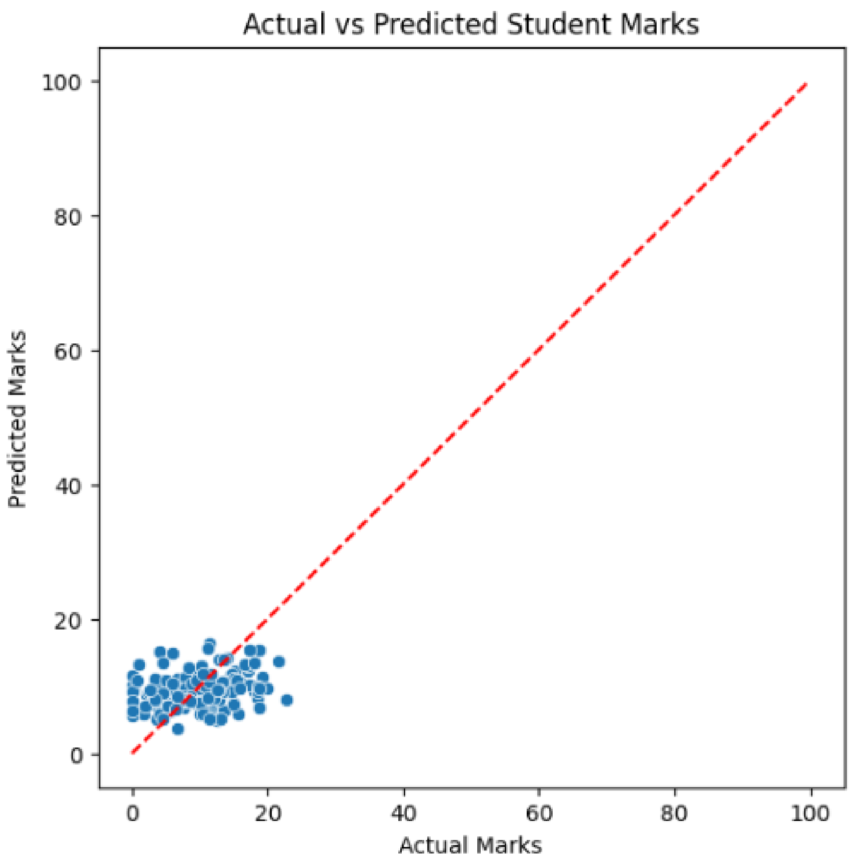


Step 16 — Scatter Plot Actual vs Predicted & Error Distribution

1. Plot actual vs predicted marks with a 45° reference line to visually assess prediction alignment and bias.
2. Points close to the diagonal indicate accurate predictions; systematic deviations indicate bias in predictions.
3. Compute residuals (errors = y_test - y_pred) and plot their distribution to evaluate error symmetry and heteroscedasticity.
4. A roughly Gaussian residual distribution centered at zero implies unbiased errors and appropriate model fit.
5. Large tails or skew in residuals suggest outliers or heterogeneity that may require robust loss functions.

[19]
✓ 0s

```python
# STEP 16: Scatter Plot - Actual vs Predicted
# ====================================================
plt.figure(figsize=(6,6))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel("Actual Marks")
plt.ylabel("Predicted Marks")
plt.title("Actual vs Predicted Student Marks")
plt.plot([0, 100], [0, 100], 'r--')
plt.show()
```



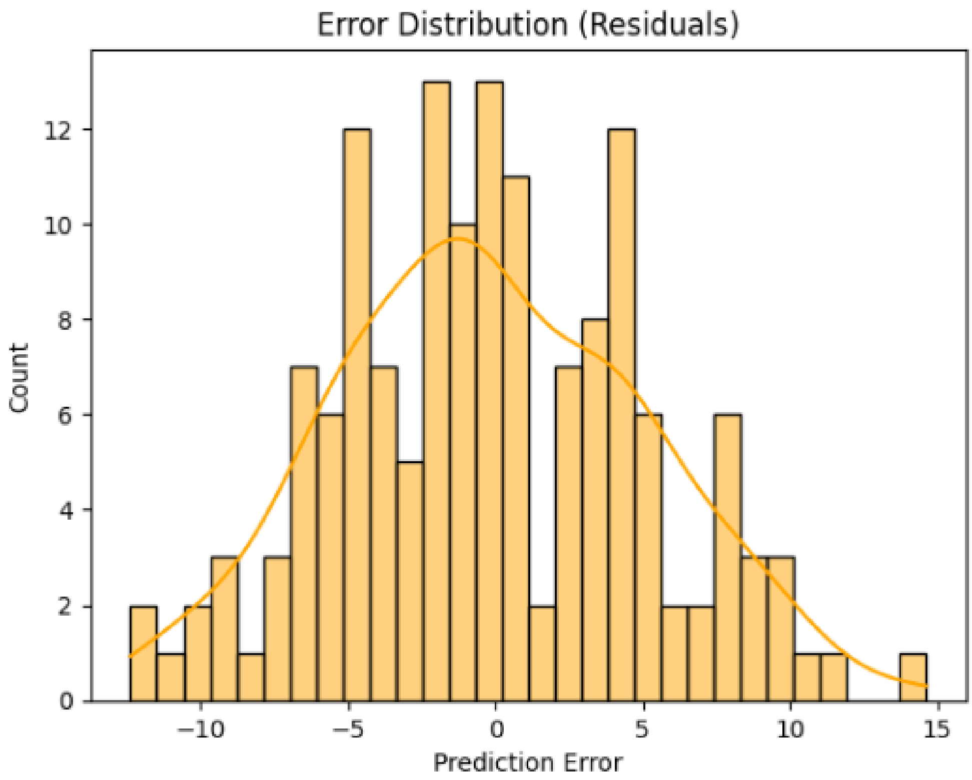
Step 17 — Error Distribution Analysis

1. This step calculates the residuals (errors = y_test - y_pred) to measure the difference between actual and predicted marks for each student.
2. Plotting a histogram with a KDE overlay visualizes how these residuals are distributed, helping detect bias or variance issues in predictions.
3. Ideally, residuals should form a symmetric, bell-shaped curve centered near zero, indicating the model neither systematically over- nor under-predicts.
4. A wide or skewed distribution signals potential model underfitting, overfitting, or the presence of outliers influencing learning.

[20]
✓ 0s

```
# STEP 17: Error Distribution
# ====================================================
errors = y_test - y_pred
sns.histplot(errors, kde=True, bins=30, color='orange')
plt.title("Error Distribution (Residuals)")
plt.xlabel("Prediction Error")
plt.show()

print("\n✅ Model training and evaluation completed successfully.")
```




```
✅ Model training and evaluation completed successfully.
```

5. Ideally, residuals should form a symmetric, bell-shaped curve centered near zero, indicating the model neither systematically over- nor under-predicts.

6. A wide or skewed distribution signals potential model underfitting, overfitting, or the presence of outliers influencing learning.

7. Visual inspection of residuals complements numeric metrics such as MSE and MAE by revealing hidden error patterns.

8. Include this plot in your report's Results section as final validation of the model's accuracy and generalization behavior.