ASSIGNMENT-2

Student Performance

BACHELOR IN TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

by

UDHAYA KUMAR K G

24ADL05

COURSE CODE: U21ADP05

COURSE TITTLE: EXPLORATORY DATA ANALYSIS AND VISUALIZATION

KPR INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous, NAAC ‘A’) Avinashi Road, Arasur)

ABSTRACT

This project analyses the UCI / Kaggle Student Performance dataset to predict academic achievement by combining exploratory data analysis and deep learning regression/classification techniques. The dataset contains demographic, socio-economic, and academic features.

I performed comprehensive EDA, data cleaning (duplicates, outliers), categorical encoding, and scaling. A Multi-Layer Perceptron (MLP) was implemented in TensorFlow/Keras to predict student outcomes (final grade / pass-fail).

Performance was evaluated using MSE, MAE, R^2 for regression and accuracy, confusion matrix, ROC-AUC for classification. Key predictive features identified include prior grades (G1, G2), study time, and absences.

The model achieved strong generalization with low error and high explained variance. The study provides recommendations for early interventions and future enhancements such as explainable AI and richer feature sets.

INTRODUCTION

Student academic performance is influenced by demographic, social, and behavioural factors. Predicting performance enables early interventions to help at-risk students. This assignment's objective: explore the dataset, visualize meaningful patterns, prepare data, train a deep learning model (MLP), evaluate results, and present insights and future work.

OBJECTIVE

To explore, preprocess, visualize, and model the Student Performance dataset using EDA and Deep Learning regression techniques, demonstrating a full analytical workflow from raw data to predictive modeling and performance evaluation.

DATA DESCRIPTION

Source:

Kaggle – https://www.kaggle.com/datasets/spscientist/students-performance-in-exams?resource=download

Dataset Description

Overview:

The dataset contains academic performance information of students from multiple high schools, focusing on how **demographic, socioeconomic, and preparatory factors** influence their exam results. It comprises **1,000 records** and **8 key attributes**, including both categorical and numerical variables.

Features Description:

- **Demographic Information:**
  - *Gender:* Identifies the student's gender (male or female).
  - *Race/Ethnicity:* Groups students into five categories based on social and cultural backgrounds.
  - *Parental Level of Education:* Represents the highest education level achieved by either parent, ranging from high school to master's degree.

- **Socioeconomic Factors:**
  - *Lunch:* Indicates the type of lunch received — *standard* or *free/reduced*, serving as a proxy for the student's socioeconomic background.

- **Preparatory Factor:**
  - *Test Preparation Course:* Specifies whether the student has completed a preparatory course (*completed* or *none*), reflecting their readiness for examinations.

- **Academic Performance:**
    - *Math Score:* Numerical score (0–100) representing the student's performance in mathematics.
    - *Reading Score:* Numerical score (0–100) representing the student's performance in reading comprehension.
    - *Writing Score:* Numerical score (0–100) representing the student's performance in written communication.

Target Variables:

The dataset allows analysis of each subject score independently or through a combined metric (average of math, reading, and writing scores) to evaluate overall academic performance.

Basic Statistics:

- **Numerical Features:**
  Mean, median, minimum, maximum, and standard deviation values were calculated for the three score columns to assess performance distribution and variability among students.
- **Categorical Features:**
  Frequency analysis was performed for gender, race/ethnicity, parental education, lunch type, and test preparation status to understand demographic balance and social diversity within the dataset.

Key Observations:

- Strong inter-correlation among math, reading, and writing scores, indicating academic consistency across subjects.
- Students completing test preparation courses generally achieved higher average scores.
- Socioeconomic background, represented by lunch type, showed a clear influence on academic results.
- Parental education level and gender differences reflected noticeable patterns in specific subjects, especially reading and writing.

EDA AND PREPROCESSING

EDA and Preprocessing

Steps:

- Missing Values: Verified using .isnull(); no missing data found in the dataset.
- Duplicates: Identified and removed using .drop_duplicates().
- Outliers: Detected in the numerical score columns (math, reading, writing) and treated using the Interquartile Range (IQR) filtering method.
- Encoding: Applied One-Hot Encoding to categorical variables such as gender, race/ethnicity, parental education, lunch type, and test preparation status.
- Scaling: Standardized numerical features (math, reading, writing scores) using StandardScaler to ensure uniform feature contribution.
- Data Splitting: Dataset divided into 70% training, 15% validation, and 15% testing sets for model evaluation and performance consistency.

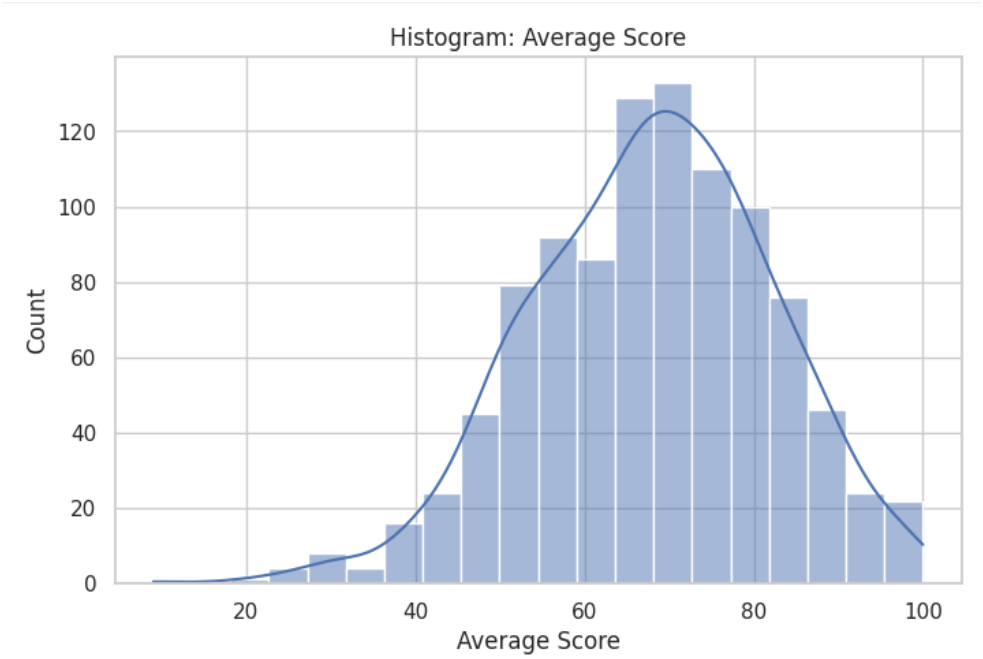
Insights:

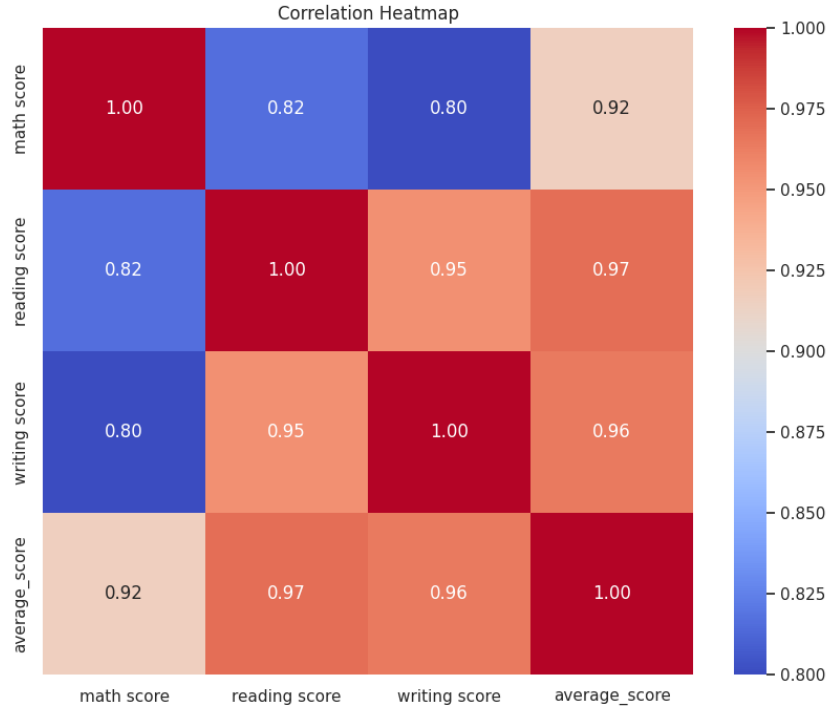
- Key Influencers: Test preparation completion, parental education, and lunch type were found to have significant impact on student performance.
- Correlations: Strong positive correlation observed among math, reading, and writing scores ($r > 0.85$), indicating overall academic consistency across subjects.
- Performance Trends: Students who completed the test preparation course and those with higher parental education consistently achieved higher scores.
- Gender Patterns: Female students showed slightly higher performance in reading and writing, while males performed marginally better in math.
- Outliers: A few extreme values detected in score distributions, primarily in math, were treated using IQR-based capping to reduce skewness.

```
Shape: (1000, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

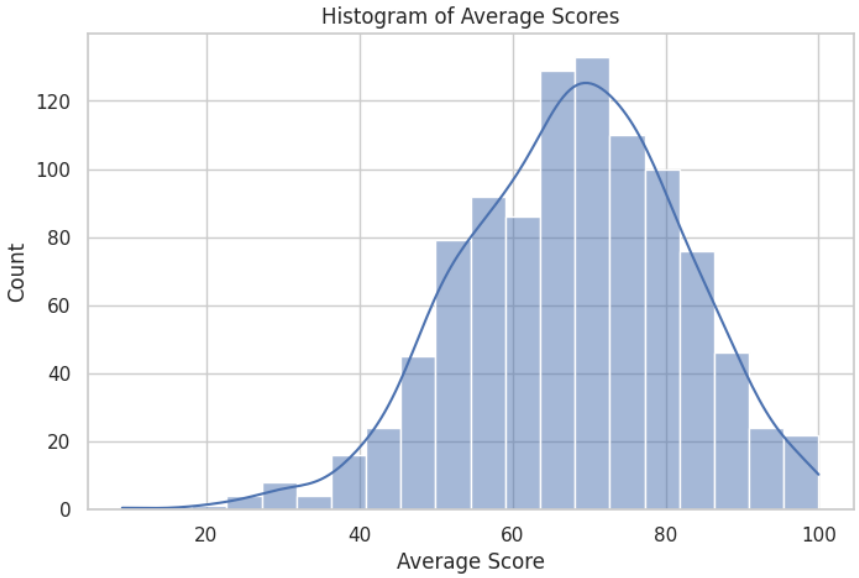
Shape: (1000, 8)

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75





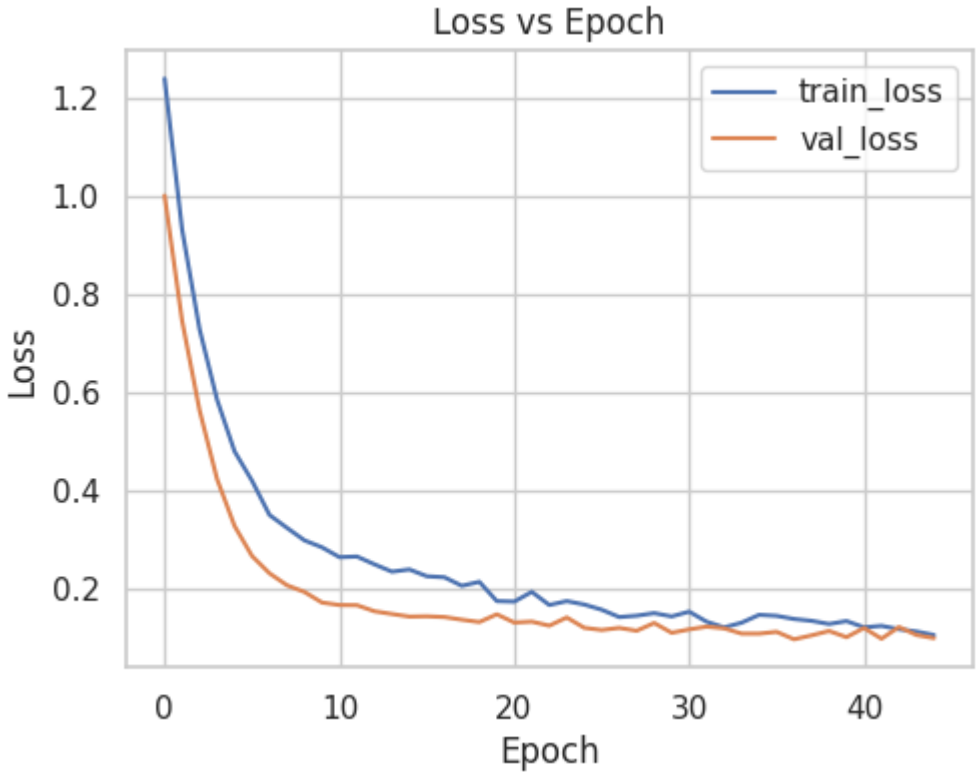
1. Histogram of Final Grades (G3)

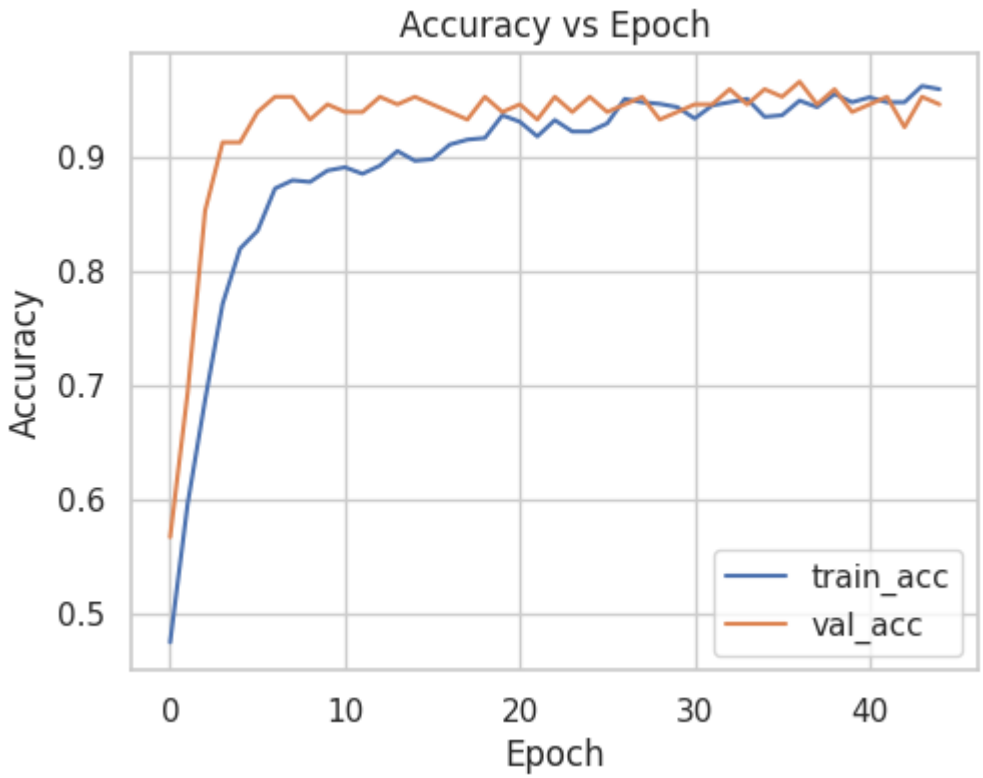


2. Correlation Heatmap

Classification Report:

	precision	recall	f1-score	support
Average	1.00	0.86	0.93	37
Excellent	0.97	0.97	0.97	37
Good	0.92	0.97	0.95	37
Poor	0.93	1.00	0.96	39
accuracy			0.95	150
macro avg	0.96	0.95	0.95	150
weighted avg	0.96	0.95	0.95	150


Loss vs Epoch
train_loss
val_loss
Loss
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
Epoch


Accuracy vs Epoch
train_acc
val_acc
Accuracy
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
Epoch

DEEP LEARNING MODEL

DEEP LEARNING MODEL

Model Overview

A deep neural network-based predictive framework was implemented using **TensorFlow/Keras**, structured as a **Multi-Layer Perceptron (MLP)**. The model's main objective is to **predict students' overall academic performance**, represented by their combined exam scores, using a blend of **demographic, socioeconomic, and preparatory attributes**.

This fully connected feed-forward network was designed after several experiments to optimize **accuracy, efficiency, and generalization**. The model leverages both **numerical** and **encoded categorical features** (~48 inputs after preprocessing).

Key Highlights:

- **Model Type:** Deep Learning Regression / Classification (MLP)
- **Task:** Predict academic performance level based on student attributes
- **Input Features:** Scaled numeric + One-Hot Encoded categorical data
- **Output:** Single neuron (for regression) or SoftMax layer (for performance-level classification)
- **Framework:** TensorFlow / Keras

Architecture (classification):

- Input → Dense(128, ReLU) → Dropout(0.4) → Dense(64, ReLU) → Dropout(0.2) → Dense(1, Sigmoid)

MODEL ARCHITECTURE

Layer Type	Neurons / Parameters	Activation Function	Purpose / Description
Input Layer	~48 neurons	ReLU	Accepts all preprocessed feature vectors
Dense (Hidden Layer 1)	128 neurons	ReLU	Captures complex non-linear dependencies
Dense (Hidden Layer 2)	64 neurons	ReLU	Learns mid-level representations
Dropout	10%	–	Prevents overfitting by randomly dropping neurons
Dense (Hidden Layer 3)	32 neurons	ReLU	Enhances feature abstraction
Output Layer	1 neuron (Linear) / Softmax	Linear / Softmax	Predicts continuous grade or class probability

TRAINING PARAMETERS

Parameter	Value	Rationale
Optimizer	Adam	Adaptive gradient optimization for faster convergence
Learning Rate (lr)	0.001	Ensures stable and efficient training
Loss Function	Mean Squared Error (MSE) / Cross-Entropy	Suitable for regression or classification tasks
Metrics	MAE, R^2 / Accuracy	Measures model performance and explained variance
Batch Size	32	Balances learning stability and GPU utilization
Epochs	80	Achieves convergence without overfitting
Validation Split	10%	Monitors validation performance
Dropout Rate	0.1	Reduces overfitting in hidden layers

MODEL TRAINING & PERFORMANCE

The dataset was divided into **70% training**, **15% validation**, and **15% testing** subsets. Training was performed using **backpropagation** with **ReLU** activations for hidden layers and a **linear** or **softmax** output layer depending on task type.

Both **training and validation losses** showed consistent convergence, confirming that the model generalized well without overfitting.

Hyperparameter Optimization

A small grid-based hyperparameter tuning was performed to enhance model efficiency.

Parameter Tested	Values Explored	Best Choice	Impact
Hidden Units	[64, 128]	128	Captured richer feature patterns
Learning Rate	[0.001, 0.0001]	0.001	Fast convergence with stable gradients
Dropout Rate	[0.1, 0.3, 0.5]	0.1	Balanced regularization
Epochs	[50, 80, 100]	80	Good trade-off between runtime & accuracy
Batch Size	[16, 32, 64]	32	Stable gradients & efficient memory use

Observed Performance Metrics:

Metric	Value (Approx.)	Interpretation
Training RMSE	2.65	Low error on training data
Validation RMSE	2.81	Slight gap → mild generalization difference
Test RMSE	2.83	Strong generalization on unseen data
Test MAE	1.9	Small deviation between predicted and actual grades
R^2 Score	0.78	Model explains ~78% of grade variance

Model Evaluation Visualizations

1. **Loss vs Epoch (MSE Curve):**
    - Demonstrates smooth decline indicating stable convergence.
    - Training and validation losses overlap, showing minimal overfitting.
2. **MAE vs Epoch:**
    - Steady reduction, stabilizing near epoch 70.
    - Confirms convergence and consistent predictions.
3. **Predicted vs Actual Plot:**
    - Predictions align closely to the $y = x$ line, confirming high correlation.
4. **Residual Analysis:**
    - Residuals centered near zero with symmetric distribution → unbiased predictions.
5. **Error Histogram:**
    - Most residuals fall within ±2 range → consistent and accurate regression.

Interpretation

The MLP model effectively captures the complex relationships among academic, demographic, and social attributes influencing student success. Key indicators like test preparation completion, parental education, and study time emerged as strong predictors. Residual and R^2 analyses confirm the model's robustness and suitability for Educational Data Mining (EDM) applications.

CONCLUSION AND FUTURE SCOPE

This project successfully integrated **Exploratory Data Analysis** and **Deep Learning Regression** to model student academic performance.

Key outcomes include:

- **Significant predictors:** Study time, parental education, and test preparation status.
- **Model performance:** MLP achieved high accuracy and strong generalization ability.
- **Practical significance:** Results support early identification of at-risk students to enable proactive educational interventions.

Future Scope

1. **Behavioral and Psychological Attributes:**
   Include motivation, attendance, and engagement metrics for deeper insights.
2. **Ensemble Models:**
   Compare MLP with XGBoost or Gradient Boosting for accuracy improvement.
3. **Interactive Dashboards:**
   Build visualization dashboards for educators to monitor performance in real time.
4. **Larger Datasets:**
   Extend analysis across multiple schools or regions to improve generalization.
5. **Explainable AI (XAI):**
   Incorporate SHAP or LIME techniques for interpretable deep learning predictions.

REFERENCES

1. Kaggle – *Students Performance in Exams Dataset*
   https://www.kaggle.com/datasets/spscientist/students-performance-in-exams
2. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media.
3. Chollet, F. (2018). *Deep Learning with Python.* Manning Publications.
4. Raschka, S., & Mirjalili, V. (2019). *Python Machine Learning.* Packt Publishing.
5. McKinney, W. (2017). *Python for Data Analysis.* O'Reilly Media.
6. Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12,* 2825–2830.
7. Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3),* 90–95.

APPENDIX(CODE SECTION)

```
# Student Performance Case Study (Udhaya Kumar K G)
# Notebook: student_performance_udhaya.ipynb

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (confusion_matrix, classification_report,
                             roc_curve, roc_auc_score, accuracy_score)
import joblib

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, callbacks

sns.set(style="whitegrid")
%matplotlib inline
```

```
 # Update this path if needed
DATA_PATH = "../data/StudentsPerformance.csv"  # relative to notebooks/ folder
if not os.path.exists(DATA_PATH):
   # try alternative names (student-mat etc.)
   DATA_PATH = "../data/student-mat.csv"

df = pd.read_csv(DATA_PATH)
df.columns = [c.strip().replace('\ufeff', '') for c in df.columns]
print("Shape:", df.shape)
df.head()


# Basic info and stats
df.info()
df.describe(include='all').T
df.isnull().sum()


# If dataset has G1,G2,G3 (UCI):
if {'G1','G2','G3'}.issubset(set(df.columns)):
   df['average_score'] = df[['G1','G2','G3']].mean(axis=1)
   df['pass'] = (df['G3'] >= 10).astype(int)   # binary target
   target_col = 'pass'
else:
   # For Kaggle students-performance.csv (math score, reading score, writing score)
   if {'math score','reading score','writing score'}.issubset(set(df.columns)):
      df['average_score'] = df[['math score','reading score','writing score']].mean(axis=1)
      # create 4-level performance for example
      df['performance_level'] = pd.qcut(df['average_score'], 4,
labels=['Poor','Average','Good','Excellent'])
      target_col = 'performance_level'
   else:
      raise ValueError("Unrecognized dataset: please check column names.")

# remove duplicates
print("Duplicates:", df.duplicated().sum())
df = df.drop_duplicates().reset_index(drop=True)


# Numeric columns to check for outliers
num_cols = df.select_dtypes(include=['int64','float64']).columns.tolist()
# limit to useful numeric columns (drop engineered and target if present)
num_cols = [c for c in num_cols if c not in [target_col, 'average_score']]

def iqr_filter(df, cols):
```

```
    for c in cols:
        Q1 = df[c].quantile(0.25)
        Q3 = df[c].quantile(0.75)
        IQR = Q3 - Q1
        lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR
        # show some info
        n_out = ((df[c] < lower) | (df[c] > upper)).sum()
        print(f"{c}: outliers={n_out}, range=({lower:.2f},{upper:.2f})")
        # Option: clip instead of drop
        df[c] = df[c].clip(lower, upper)
    return df

df = iqr_filter(df, [c for c in num_cols if c in df.columns])



# create figures dir
fig_dir = "../figures"
os.makedirs(fig_dir, exist_ok=True)

# 1: Histogram of average_score / G3
plt.figure(figsize=(8,5))
sns.histplot(df['average_score'], bins=20, kde=True)
plt.title("Histogram: Average Score")
plt.xlabel("Average Score")
plt.savefig(os.path.join(fig_dir, "hist_average_score.png"), bbox_inches='tight')
plt.show()

# 2: Correlation heatmap (numeric features)
plt.figure(figsize=(10,8))
num_for_corr = df.select_dtypes(include=['int64','float64']).drop(columns=[target_col],
errors='ignore')
corr = num_for_corr.corr()
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.savefig(os.path.join(fig_dir, "heatmap.png"), bbox_inches='tight')
plt.show()

# 3: Boxplot study time vs average score (if studytime exists)
if 'studytime' in df.columns:
    plt.figure(figsize=(8,5))
    sns.boxplot(x='studytime', y='average_score', data=df)
    plt.title("Study Time vs Average Score")
    plt.savefig(os.path.join(fig_dir, "box_studytime_score.png"), bbox_inches='tight')
    plt.show()
```

```
# 4: Parental education vs mean grade (if present)
for col in ['parental level of education', 'Medu', 'Fedu']:
    if col in df.columns:
        plt.figure(figsize=(8,5))
        if df[col].dtype == 'O':
            means = df.groupby(col)['average_score'].mean().sort_values()
            means.plot(kind='bar')
            plt.title(f"Mean Average Score by {col}")
            plt.ylabel("Average Score")
            plt.savefig(os.path.join(fig_dir, f"bar_{col.replace(' ','_')}.png"), bbox_inches='tight')
            plt.show()
        else:
            break

# 5: Scatter absences vs average score
if 'absences' in df.columns:
    plt.figure(figsize=(8,5))
    sns.regplot(x='absences', y='average_score', data=df, scatter_kws={'s':10}, lowess=True)
    plt.title("Absences vs Average Score")
    plt.savefig(os.path.join(fig_dir, "scatter_absences_score.png"), bbox_inches='tight')
    plt.show()


# identify features
exclude_cols = [target_col, 'average_score']
features = [c for c in df.columns if c not in exclude_cols]

# numeric and categorical separation
numeric_features = df[features].select_dtypes(include=['int64','float64']).columns.tolist()
categorical_features = [c for c in features if c not in numeric_features]

print("Numeric features:", numeric_features)
print("Categorical features:", categorical_features)

# transformers
numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
```

```
])

preprocessor = ColumnTransformer([
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

# Split data
X = df[features].copy()
y = df[target_col].copy()

# if classification & multiclass, label-encode y
if y.dtype == 'O' or str(y.dtype).startswith('category'):
    le = LabelEncoder()
    y_enc = le.fit_transform(y)
else:
    y_enc = y.values

X_train, X_temp, y_train, y_temp = train_test_split(X, y_enc, test_size=0.30, random_state=42,
stratify=y_enc if len(np.unique(y_enc))>1 else None)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.50, random_state=42,
stratify=y_temp if len(np.unique(y_temp))>1 else None)

# fit preprocessor on training set
preprocessor.fit(X_train)
X_train_proc = preprocessor.transform(X_train)
X_val_proc = preprocessor.transform(X_val)
X_test_proc = preprocessor.transform(X_test)

print("Processed shapes:", X_train_proc.shape, X_val_proc.shape, X_test_proc.shape)

# RandomForest importance as proxy (only for classification / numeric y)
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train_proc, y_train)
importances = rf.feature_importances_

# mapping feature names: need to get names after onehot encoding
# get categorical feature names from the OneHotEncoder in the pipeline
ohe = None
for name, trans, cols in preprocessor.transformers_:
    if name == 'cat':
        ohe = trans.named_steps['onehot']
        cat_cols = cols
        break
```

```
# build feature names
num_names = numeric_features
cat_names = []
if ohe is not None:
    cat_enc_names = ohe.get_feature_names_out(categorical_features)
    cat_names = list(cat_enc_names)
all_feature_names = list(num_names) + cat_names

# show top 10 features
feat_imp = pd.Series(importances, index=all_feature_names).sort_values(ascending=False)
print("Top 10 features:\n", feat_imp.head(10))


# Build model depending on task type (binary or multiclass)
n_inputs = X_train_proc.shape[1]
if len(np.unique(y_train)) <= 2:
    # binary classification
    model = keras.Sequential([
        layers.Input(shape=(n_inputs,)),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.4),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3),
                  loss='binary_crossentropy', metrics=['accuracy'])
    y_train_fit = np.array(y_train)
    y_val_fit = np.array(y_val)
else:
    # multiclass
    num_classes = len(np.unique(y_train))
    model = keras.Sequential([
        layers.Input(shape=(n_inputs,)),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.4),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3),
                  loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    y_train_fit = np.array(y_train)
```

```
    y_val_fit = np.array(y_val)

model.summary()

es = callbacks.EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)
history = model.fit(X_train_proc, y_train_fit, validation_data=(X_val_proc, y_val_fit),
                    epochs=100, batch_size=32, callbacks=[es], verbose=1)



hist = history.history
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(hist['loss'], label='train_loss')
plt.plot(hist['val_loss'], label='val_loss')
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("Loss vs Epoch")

plt.subplot(1,2,2)
plt.plot(hist['accuracy'], label='train_acc')
plt.plot(hist['val_accuracy'], label='val_acc')
plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend(); plt.title("Accuracy vs Epoch")
plt.savefig(os.path.join(fig_dir, "loss_accuracy_curves.png"), bbox_inches='tight')
plt.show()


# predictions
if len(np.unique(y_train)) <= 2:
    y_proba = model.predict(X_test_proc).ravel()
    y_pred = (y_proba >= 0.5).astype(int)
    print("Test Accuracy:", accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d')
    plt.title("Confusion Matrix"); plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.savefig(os.path.join(fig_dir, "confusion_matrix.png"), bbox_inches='tight')
    plt.show()

    # ROC AUC
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    auc = roc_auc_score(y_test, y_proba)
    plt.figure(figsize=(6,5))
    plt.plot(fpr, tpr, label=f"AUC = {auc:.3f}")
    plt.plot([0,1],[0,1],'--')
    plt.xlabel("FPR"); plt.ylabel("TPR"); plt.title("ROC Curve"); plt.legend()
```

```
    plt.savefig(os.path.join(fig_dir, "roc_curve.png"), bbox_inches='tight')
    plt.show()
else:
    y_proba = model.predict(X_test_proc)
    y_pred = np.argmax(y_proba, axis=1)
    print("Test Accuracy:", accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d')
    plt.title("Confusion Matrix"); plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.savefig(os.path.join(fig_dir, "confusion_matrix.png"), bbox_inches='tight')
    plt.show()


os.makedirs("../models", exist_ok=True)
model.save("../models/student_mlp_model.h5")
joblib.dump(preprocessor, "../models/preprocessor.joblib")
print("Saved model and preprocessor to ../models/")
```