

```

# =====
# CB&DP Project – Full Summary Notebook
# Dataset: Depression Student Dataset.csv
# =====

# 📦 PHASE 1 – Setup & Imports
!pip install -q missingno

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, RocCurveDisplay
from sklearn.model_selection import GridSearchCV
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import joblib
import os

sns.set(style="whitegrid")
print("✅ Libraries Loaded")

# =====
# PHASE 2 – Data Loading & Understanding
# =====

file_path = "/content/Depression Student Dataset.csv"
df = pd.read_csv(file_path)

print("✅ Dataset Loaded Successfully")
print(df.head())
print("\n🔴 Shape:", df.shape)
print("\n🔴 Data Types:\n", df.dtypes)
print("\n🔴 Missing Values:\n", df.isnull().sum())
print("\n🔴 Duplicates:", df.duplicated().sum())
print("\n🔴 Statistical Summary:\n", df.describe())

# Visualize missing data
msno.matrix(df)
plt.show()

# Numeric feature distributions
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
df[numeric_cols].hist(figsize=(12, 6))
plt.suptitle("Numeric Feature Distributions")
plt.show()

# =====
# PHASE 3 – Data Cleaning
# =====

# Drop rows missing >30% data
df = df.dropna(thresh=int(0.7 * len(df.columns)))

# Fill missing values
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())
categorical_cols = df.select_dtypes(include='object').columns
for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])

# Remove duplicates
df = df.drop_duplicates()

# Clean text columns
for col in categorical_cols:
    df[col] = df[col].astype(str).str.strip().str.lower()

# Remove outliers (IQR)
for col in numeric_cols:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

```

```

Q1 = df[col].quantile(0.25)
Q3 = df[col].quantile(0.75)
IQR = Q3 - Q1
df = df[(df[col] >= Q1 - 1.5*IQR) & (df[col] <= Q3 + 1.5*IQR)]

print("\n✅ Data Cleaning Complete | New Shape:", df.shape)

# =====
# PHASE 4 – Feature Engineering
# =====

# Encode Binary Columns
le = LabelEncoder()
for col in categorical_cols:
    if df[col].nunique() == 2:
        df[col] = le.fit_transform(df[col])

# One-Hot Encode multi-category columns
multi_cat = [c for c in categorical_cols if df[c].nunique() > 2]
df = pd.get_dummies(df, columns=multi_cat, drop_first=True)

# Normalize numeric features
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

print("\n✅ Feature Engineering Complete")
print(df.head())

# =====
# PHASE 5 – Train / Validation / Test Split
# =====

target_col = "depression" # Update if necessary
X = df.drop(columns=[target_col])
y = df[target_col]

X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)

print("\n✅ Data Split Complete")
print(f"Training Set: {X_train.shape}")
print(f"Validation Set: {X_val.shape}")
print(f"Test Set: {X_test.shape}")

# =====
# PHASE 6 – Neural Network Model
# =====

mlp_model = Sequential([
    Dense(128, input_dim=X_train.shape[1], activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

mlp_model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

mlp_model.summary()

history = mlp_model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    batch_size=32,
    verbose=1
)

# Evaluate MLP
loss, accuracy = mlp_model.evaluate(X_test, y_test, verbose=0)
print("\n✅ MLP Test Accuracy:", accuracy)

v_pred_prob_nn = mlp_model.predict(X_test)

```

```

y_pred_nn = (y_pred_prob_nn > 0.5).astype(int)

print("Classification Report (NN):\n", classification_report(y_test, y_pred_nn))
print("Confusion Matrix (NN):\n", confusion_matrix(y_test, y_pred_nn))

# Plot Training History
plt.figure(figsize=(12,5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("MLP Accuracy Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure(figsize=(12,5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("MLP Loss Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

# =====
# PHASE 7 – Model Evaluation & Comparison
# =====

# Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
y_prob_lr = lr.predict_proba(X_test)[:,:1]

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:,:1]

# Compare Models
print("\nLogistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Neural Network Accuracy:", accuracy_score(y_test, y_pred_nn))

# ROC Curves
RocCurveDisplay.from_predictions(y_test, y_prob_lr, name="Logistic Regression")
RocCurveDisplay.from_predictions(y_test, y_prob_rf, name="Random Forest")
RocCurveDisplay.from_predictions(y_test, y_pred_prob_nn, name="Neural Network")
plt.title("ROC Curves Comparison")
plt.show()

# Hyperparameter Tuning Example (Random Forest)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42),
                        param_grid, cv=3, scoring='accuracy', n_jobs=-1)
grid_rf.fit(X_train, y_train)
print("\nBest RF Params:", grid_rf.best_params_)
print("Test Accuracy (Tuned RF):", accuracy_score(y_test, grid_rf.best_estimator_.predict(X_test)))

# =====
# PHASE 8 – Model Deployment
# =====

# Save models
model_dir = "saved_models"
os.makedirs(model_dir, exist_ok=True)

joblib.dump(lr, os.path.join(model_dir, "logistic_regression.pkl"))
joblib.dump(grid_rf.best_estimator_, os.path.join(model_dir, "random_forest.pkl"))
mlp_model.save(os.path.join(model_dir, "neural_network.h5"))

print("\n✅ All models saved successfully!")

```

```
# Example prediction function
def predict_depression(input_data):
    input_scaled = scaler.transform(input_data[numeric_cols])
    pred_lr = lr.predict(input_scaled)
    pred_rf = grid_rf.best_estimator_.predict(input_scaled)
    pred_nn = (mlp_model.predict(input_scaled) > 0.5).astype(int)
    return {"LogisticRegression": pred_lr, "RandomForest": pred_rf, "NeuralNetwork": pred_nn}

# Sample prediction
sample_input = X_test.iloc[:5]
predictions = predict_depression(sample_input)
print("\n✅ Sample Predictions:\n", predictions)
```

