import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

#from sklearn.preprocessing import StandardScaler

# Load the dataset

df = pd.read\_csv('AmesHousingdataset.csv')

# Separate features (X) and target (y)

X = df.drop("SalePrice", axis=1) # Replace 'SalePrice' with the actual target column name

y = df["SalePrice"]

# Identify numerical and categorical columns

numerical\_features = X.select\_dtypes(include=["int64", "float64"]).columns

categorical\_features = X.select\_dtypes(include=["object"]).columns

# Preprocessing for numerical and categorical features

preprocessor = ColumnTransformer([

("num", SimpleImputer(strategy="mean"), numerical\_features), # Fill missing values with the mean

("cat", OneHotEncoder(handle\_unknown="ignore"), categorical\_features)]) # One-hot encode categorical features])

# Apply preprocessing

X\_preprocessed = preprocessor.fit\_transform(X)

# Scale the features

scaler = StandardScaler(with\_mean=False)

X\_scaled = scaler.fit\_transform(X\_preprocessed)

# Log-transform the target to handle skewness

y = np.log1p(y)

# Split the dataset

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Define the neural network

model = Sequential([

Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)),

Dropout(0.2), # Add dropout to prevent overfitting

Dense(64, activation='relu'),

Dropout(0.2),

Dense(1) # Output layer for regression

])

# Compile the model

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_val, y\_val))

# Evaluate the model

loss, mae = model.evaluate(X\_val, y\_val)

print(f"Validation MAE: {mae}")

# Predict house prices

predictions = model.predict(X\_val)

print(f"prediction: {predictions}")

# Convert predictions back to the original scale

predictions\_original\_scale = np.expm1(predictions)

print(f"originalprediction: {predictions\_original\_scale}")