

HOUSE PRICE PREDICTION:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import mean_absolute_error, r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import pickle

data_path='/content/drive/MyDrive/house_price_prediction.csv'
data = pd.read_csv(data_path)
data = data.drop(data.columns[8], axis=1)
print(data.head())
print(data.info())
if 'Price in taka' in data.columns:
    data = data.drop('Price in taka', axis=1)
data = data.drop('Occupancy_Status', axis=1)
# Convert Price_in_Taka to numeric (remove commas and convert to float)
data['Price_in_Taka'] = pd.to_numeric(data['Price_in_Taka'].replace('[,]', '', regex=True),
errors='coerce')
if data['Price_in_Taka'].isna().any():
    print("Warning: NaNs found in Price_in_Taka. Dropping rows with NaNs.")
    data = data.dropna(subset=['Price_in_Taka'])
print("Dataset Preview:")
print(data.head())
# One-hot encode Location
location_dummies = pd.get_dummies(data['Location'], prefix='Location')
data = pd.concat([data, location_dummies], axis=1)
data = data.drop('Location', axis=1)
tfidf = TfidfVectorizer(max_features=50, stop_words='english')
title_features = tfidf.fit_transform(data['Title']).toarray()
title_df = pd.DataFrame(title_features, columns=[f'title_{i}' for i in
range(title_features.shape[1])])
data = pd.concat([data, title_df], axis=1)
data = data.drop('Title', axis=1)
# Define features and target
X = data.drop('Price_in_Taka', axis=1)
y = data['Price_in_Taka']
if X.isna().any().any() or y.isna().any():
    print("Warning: NaNs found in features or target. Dropping rows with NaNs.")
    data = data.dropna()
    X = data.drop('Price_in_Taka', axis=1)
    y = data['Price_in_Taka']
scaler = StandardScaler()
```

```

X_scaled = scaler.fit_transform(X)
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1) # Regression output
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.summary()
class TrainingLogger(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        print(f"Epoch {epoch+1}: loss={logs['loss']:.2e}, mae={logs['mae']:.2f}, "
            f"val_loss={logs['val_loss']:.2e}, val_mae={logs['val_mae']:.2f}")

history = model.fit(X_train, y_train, epochs=50, batch_size=16,
                    validation_data=(X_test, y_test), verbose=0,
                    callbacks=[TrainingLogger()])
print("\nHistory keys:", list(history.history.keys()))
print(f"Length of loss: {len(history.history.get('loss', []))}")
print(f"Length of val_loss: {len(history.history.get('val_loss', []))}")
plt.figure(figsize=(8, 6))
if 'loss' in history.history and 'val_loss' in history.history and len(history.history['loss']) > 0:
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Mean Squared Error')
    plt.legend()
else:
    print("Warning: No loss data to plot. Check training step for errors or NaNs.")
plt.show()
y_pred = model.predict(X_test).flatten()
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Test MAE: {mae:.2f} Taka")
print(f"Test R² Score: {r2:.4f}")
test_samples = pd.DataFrame({
    'Title': [
        "Fully Furnished 1200 Sq Ft Residential Apartment",
        "Budget-Friendly 1000 Sq Ft Apartment For Sale"
    ],
    'Bedrooms': [3, 2],
    'Bathrooms': [2, 1],

```

```

    'Floor_Number': [2, 3],
    'Floor_Area': [1200, 1000],
    'Location': ['Kamal Road', 'New Babupara']
})
# Preprocess test samples
test_location_dummies = pd.get_dummies(test_samples['Location'], prefix='Location')
for col in location_dummies.columns:
    if col not in test_location_dummies.columns:
        test_location_dummies[col] = 0
test_location_dummies = test_location_dummies[location_dummies.columns]
test_samples = pd.concat([test_samples, test_location_dummies], axis=1)
test_samples = test_samples.drop('Location', axis=1)

test_title_features = tfidf.transform(test_samples['Title']).toarray()
test_title_df = pd.DataFrame(test_title_features, columns=[f'title_{i}' for i in
range(test_title_features.shape[1])])
test_samples = pd.concat([test_samples, test_title_df], axis=1)
test_samples = test_samples.drop('Title', axis=1)

test_scaled = scaler.transform(test_samples)
test_predictions = model.predict(test_scaled).flatten()

for i, pred in enumerate(test_predictions):
    print(f"Test Sample {i+1}:")
    print(f"Features: Bedrooms={test_samples['Bedrooms'].iloc[i]},
Bathrooms={test_samples['Bathrooms'].iloc[i]}, "
        f"Floor_Number={test_samples['Floor_Number'].iloc[i]},
Floor_Area={test_samples['Floor_Area'].iloc[i]}")
    print(f"Predicted Price: {pred:.2f} Taka\n")

```