```python
import re
import os
import pandas as pd
import multiprocessing
from time import time as timer
from tqdm import tqdm
import numpy as np
from pathlib import Path
from functools import partial
import requests
import urllib

def download_image(image_link, savefolder):
    if(isinstance(image_link, str)):
        filename = Path(image_link).name
        image_save_path = os.path.join(savefolder, filename)
        if(not os.path.exists(image_save_path)):
            try:
                urllib.request.urlretrieve(image_link,
image_save_path)
            except Exception as ex:
                print('Warning: Not able to download - {}\
n{}'.format(image_link, ex))
        else:
            return
    return

def download_images(image_links, download_folder):
    if not os.path.exists(download_folder):
        os.makedirs(download_folder)
    results = []
    download_image_partial = partial(download_image,
savefolder=download_folder)
    with multiprocessing.Pool(100) as pool:
        for result in tqdm(pool.imap(download_image_partial,
image_links), total=len(image_links)):
            results.append(result)
        pool.close()
        pool.join()

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
df=pd.read_csv('/content/drive/MyDrive/amazon_dataset/train.csv')
```

```python
mid_25k_df = df.tail(50000)
mid_25k_df.shape
```

```
(50000, 4)
```

```
print(mid_25k_df.columns)
```

```
Index(['sample_id', 'catalog_content', 'image_link', 'price'],
dtype='object')
```

```
import os

# Path to your image folder
image_folder = '/content/drive/MyDrive/amazon_dataset/train_images'

# Get all filenames in the folder
total_files_names = os.listdir(image_folder)

# Optionally print the total count
print(f"Total files found: {len(total_files_names)}")
```

```
Total files found: 72287
```

```
import os
import random
from PIL import Image
import matplotlib.pyplot as plt
import pandas as pd
from tqdm.auto import tqdm

tqdm.pandas()

# --- 1. Load your training data ---
df = pd.read_csv('/content/drive/MyDrive/amazon_dataset/train.csv')
print(f"Original DataFrame shape: {df.shape}")

# --- 2. Slice the required middle 25k rows ---
df_mid = df.tail(50000).copy()

# --- 3. Define the image directory ---
image_dir = '/content/drive/MyDrive/amazon_dataset/train_images'

# --- 4. Create the path from the 'image_link' column ---
# Extract filename from URL and join with image_dir
df_mid['image_path'] = df_mid['image_link'].apply(lambda link:
os.path.join(image_dir, str(link).split('/')[-1]))

# --- 5. Verify which files actually exist ---
print("\nVerifying that each image file exists on disk...")
df_mid['file_exists'] =
df_mid['image_path'].progress_apply(os.path.exists)

# --- 6. Clean the DataFrame ---
original_rows = len(df_mid)
df_clean = df_mid[df_mid['file_exists']].copy()  # Use copy to avoid
warnings
```

```python
final_rows = len(df_clean)

# Drop the temporary 'file_exists' column
if 'file_exists' in df_clean.columns:
    df_clean = df_clean.drop(columns=['file_exists'])

print(f"\nRemoved {original_rows - final_rows} rows due to missing image files.")
print(f"Clean DataFrame shape: {df_clean.shape}")

# --- 7. Display 8 random images from df_clean ---

# Get the list of existing image paths
image_files = df_clean['image_path'].tolist()

# Randomly sample 8 images
sample_images = random.sample(image_files, k=8)

# Plot images in 2x4 grid
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i, img_path in enumerate(sample_images):
    try:
        img = Image.open(img_path)
        axes[i].imshow(img)
        axes[i].set_title(os.path.basename(img_path)[:15] + '...')
        axes[i].axis('off')
    except Exception as e:
        print(f"Could not display {img_path}. Reason: {e}")

plt.tight_layout()
plt.show()
```

```
Original DataFrame shape: (75000, 4)
```

```
Verifying that each image file exists on disk...
```

{"model_id":"c127a32c5dec43b9bbcfcd5341b8b74d","version_major":2,"version_minor":0}

```

Removed 30129 rows due to missing image files.
Clean DataFrame shape: (19871, 5)
```

61UkglwJsjL.jpg...

615RdK8XzPL.jpg...

71dqA7-KWXL.jpg...

71OW2jFPMpL.jpg...

41ZT7IQiLbL.jpg...

71jyp2o5EPL.jpg...

61dafFvrnPL.jpg...

51HvJEfFlyL.jpg...

```
df_clean.head()
```

```
{"summary":"{\n  \"name\": \"df_clean\",\n  \"rows\": 19871,\n
\"fields\": [\n    {\n      \"column\": \"sample_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
86254,\n        \"min\": 4,\n        \"max\": 299416,\n
\"num_unique_values\": 19871,\n        \"samples\": [\n
147189,\n          285178,\n          103234\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"catalog_content\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 19851,\n        \"samples\": [\n
\"Item Name: Heinz Homestyle Mushroom Gravy (12 oz Jar)\\nBullet
Point: Packaging may vary\\nValue: 12.0\\nUnit: Fl Oz\\n\",\n
\"Item Name: Organic Functional Mushroom Vegan Protein Bars | Made in
USA | Full Dose (1500mg) Lions Mane, Chaga, Reishi & Cordyceps Bars |
Plant Protein Bars w/ Adaptogens & Superfoods | Peanut Butter
Chocolate Flavor\\nBullet Point 1: FULL DOSE OF FUNCTIONAL MUSHROOMS:
Each bar contains the recommended daily serving (1500mg) Cordyceps,
Lion's Mane, Chaga, Reishi mushrooms, unlike the majority of mushroom
products on the market. These ancient superfoods have been found to
boost energy levels, improve cognitive function, mitigate stress,
fight inflammation, support immunity, and (so much) more.\\nBullet
Point 2: USDA ORGANIC, HIGH-QUALITY INGREDIENTS: Unlike most protein
bars, we only use certified organic ingredients of non-animal origin
that are grown responsibly and sustainably. You can taste and feel the
difference. NO ARTIFICIAL SWEETENERS: With only 7g natural sugar
derived from the Cassava Root, Balanced Tiger bars have just the right
amount of sweetness without the negative effects of high sugar or the
synthetic taste of artificial sugar.\\nBullet Point 3: FUNCTIONAL
MUSHROOMS ON-THE-GO: We believe that adaptogens should be easy to take
and take with you. No more capsules or powders \\u00e2 just a
delicious mushroom-infused plant based protein bar for people on the
go. Superfood, meet super convenient.\\nBullet Point 4: DELICIOUS: We
recipe tested for months to get the flavors juuuust right. Our bars
are sweet, satisfying, and \\u00e2 this may come as a surprise \\u00e2
bear no resemblance to your farmer\\u00e2s market fungi. Don't take
our word for it, try a sampler pack and see for yourself.\\nBullet
Point 5: DIET FRIENDLY: Our bars are vegan, gluten-free, soy-free,
dairy-free, and non-GMO certified. With 11g plant-based protein in
only 190 calories, each bar is designed to satisfy all the nutritional
needs of the modern lifestyle.\\nBullet Point 6: NO ARTIFICIAL
SWEETENERS: With only 7g natural sugar derived from the Cassava Root,
Balanced Tiger bars have just the right amount of sweetness without
the negative effects of high sugar or the synthetic taste of
artificial sugar.\\nValue: 18.48\\nUnit: Ounce\\n\",\n          \"Item
Name: Frontera Foods Inc. Salsa, Med Corn & Poblano, 16-Ounce (Pack of
6)\\nBullet Point 1: Made from fresh ingredients\\nBullet Point 2:
Gluten Free\\nBullet Point 3: No preservatives\\nValue: 96.0\\nUnit:
oz\\n\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"image_link\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 18936,\n
\"samples\": [\n
\"https://m.media-amazon.com/images/I/71ne9qzuLbL.jpg\",\n
\"https://m.media-amazon.com/images/I/81kdZCAPGeL.jpg\",\n
\"https://m.media-amazon.com/images/I/71sAf9ips7L.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"price\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 32.59726772564281,\n
\"min\": 0.13,\n        \"max\": 1280.0,\n
\"num_unique_values\": 6227,\n        \"samples\": [\n
52.19,\n          54.89,\n          0.855\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"image_path\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 18936,\n        \"samples\": [\n
\"/content/drive/MyDrive/amazon_dataset/train_images/71ne9qzuLbL.jpg\"
,\n
\"/content/drive/MyDrive/amazon_dataset/train_images/81kdZCAPGeL.jpg\"
,\n
\"/content/drive/MyDrive/amazon_dataset/train_images/71sAf9ips7L.jpg\"
\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_clean"}
```

```
df_clean.shape
```

```
(19871, 5)
```

```python
from sklearn.model_selection import train_test_split
import pandas as pd

# Assuming df_clean is your cleaned DataFrame for the middle 25k rows

# --- 1. Create Price Bins for Stratification using Quantiles ---
df_clean['price_bin'] = pd.qcut(df_clean['price'],
                                q=10,
                                labels=False,
                                duplicates='drop')

# --- 2. Perform the Split ---
train_df, val_df = train_test_split(
    df_clean,
    test_size=0.2,
    random_state=42,
    stratify=df_clean['price_bin']
)

# --- 3. Clean Up ---
train_df = train_df.drop(columns=['price_bin'])
val_df = val_df.drop(columns=['price_bin'])

# --- 4. Verify the Results ---
print("Data splitting complete.")
print(f"Training set shape: {train_df.shape}")
print(f"Validation set shape: {val_df.shape}")

print("\nPrice distribution in the training set (sample):")
print(train_df['price'].describe())

print("\nPrice distribution in the validation set (sample):")
print(val_df['price'].describe())
```

```
Data splitting complete.
Training set shape: (15896, 5)
Validation set shape: (3975, 5)

Price distribution in the training set (sample):
count    15896.000000
mean        24.279291
std         32.995707
min          0.130000
25%          6.985000
50%         14.500000
75%         29.130000
max       1280.000000
Name: price, dtype: float64

Price distribution in the validation set (sample):
```

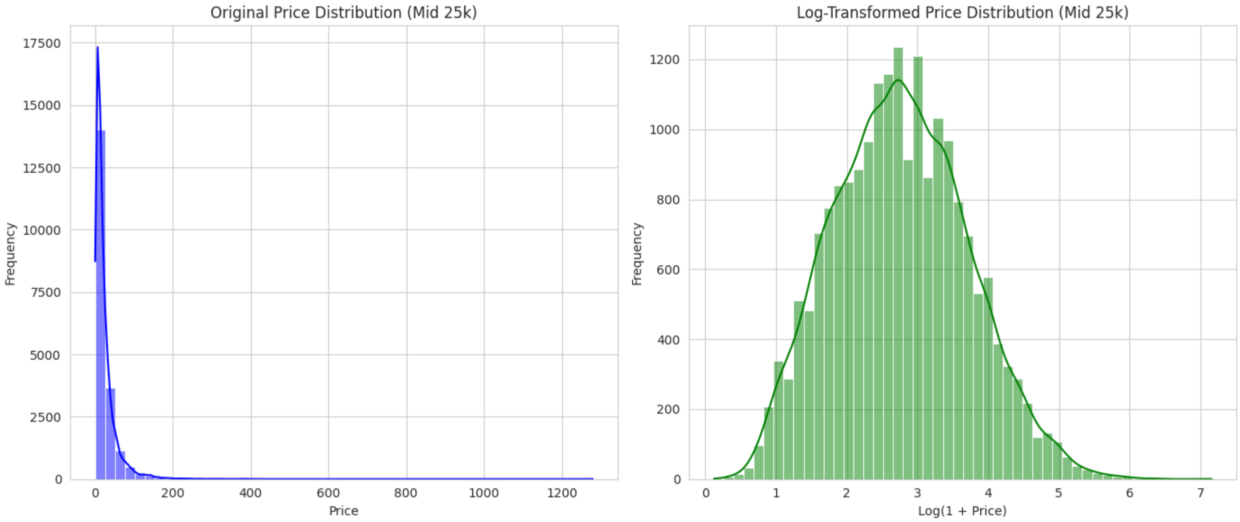
```
count    3975.000000
mean       23.994717
std        30.955640
min         0.500000
25%         6.850000
50%        14.500000
75%        29.380000
max       600.590000
Name: price, dtype: float64
```

```python
import tensorflow as tf

# --- 1. Define Constants ---
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 32
AUTOTUNE = tf.data.AUTOTUNE  # optimal performance

# --- 2. Create a preprocessing function ---
def preprocess_image(image_path, price):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, IMAGE_SIZE)
    image = image / 255.0
    return image, price

# --- 3. Training data pipeline ---
train_ds = tf.data.Dataset.from_tensor_slices((train_df['image_path'],
train_df['price']))
train_ds = train_ds.map(preprocess_image,
num_parallel_calls=AUTOTUNE).shuffle(1024).batch(BATCH_SIZE).prefetch(
AUTOTUNE)

# --- 4. Validation data pipeline ---
val_ds = tf.data.Dataset.from_tensor_slices((val_df['image_path'],
val_df['price']))
val_ds = val_ds.map(preprocess_image,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

print("□ Data pipelines for mid 25k data created successfully!")
```

```
□ Data pipelines for mid 25k data created successfully!
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Use your middle 25k cleaned DataFrame df_clean

# Set the style for the plot
sns.set_style("whitegrid")
```

```python
# Create a figure to hold the plots
plt.figure(figsize=(14, 6))

# --- Plot 1: The Original Price Distribution ---
plt.subplot(1, 2, 1)  # (1 row, 2 columns, 1st plot)
sns.histplot(df_clean['price'], bins=50, kde=True, color='blue')
plt.title('Original Price Distribution (Mid 25k)')
plt.xlabel('Price')
plt.ylabel('Frequency')

# --- Plot 2: The Log-Transformed Price Distribution ---
plt.subplot(1, 2, 2)  # (1 row, 2 columns, 2nd plot)
sns.histplot(np.log1p(df_clean['price']), bins=50, kde=True,
color='green')
plt.title('Log-Transformed Price Distribution (Mid 25k)')
plt.xlabel('Log(1 + Price)')
plt.ylabel('Frequency')

# Show the plots
plt.tight_layout()
plt.show()
```



```python
import numpy as np

# Apply log(1 + x) transformation to the price column in training and
validation sets
train_df['log_price'] = np.log1p(train_df['price'])
val_df['log_price'] = np.log1p(val_df['price'])

print("Log transformation applied to prices for mid 25k data.")
```

```
Log transformation applied to prices for mid 25k data.
```

```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split

print(f"Starting with {len(df_clean)} rows.")

# Engineer Brand Feature (Improved Version)
def extract_full_brand_v2(text):
    """Extracts a multi-word brand name by collecting consecutive
capitalized words."""
    try:
        if isinstance(text, str) and text.startswith("Item Name:"):
            words = text.replace("Item Name:", "").strip().split()
            brand_parts = []
            for word in words:
                if word and word[0].isupper():
                    brand_parts.append(word)
                else:
                    break
            if brand_parts:
                return " ".join(brand_parts)
    except:
        pass
    return "Unknown"

print("Extracting brand names...")
df_clean['brand'] =
df_clean['catalog_content'].apply(extract_full_brand_v2)

# Engineer Numerical Features
def extract_numerical_features(df):
    df['pack_count'] = df['catalog_content'].str.extract(r'\(Pack of
(\d+)\)').astype(float).fillna(1)
    df['item_size'] = df['catalog_content'].str.extract(r'(\d+\.?\d*)\
s*(?:Ounce|oz|OZ)').astype(float).fillna(0)
    df['total_size'] = df['pack_count'] * df['item_size']
    df['unit_price'] = df['price'] / (df['total_size'] + 1e-6)  #
Epsilon for safety
    df = df.drop(columns=['total_size'])
    df[['pack_count', 'item_size', 'unit_price']] = df[['pack_count',
'item_size', 'unit_price']].fillna(0)
    return df

print("Engineering numerical features...")
df_clean = extract_numerical_features(df_clean)

# Apply Log Transformation to Price
df_clean['log_price'] = np.log1p(df_clean['price'])
```

```
# Stratified Split
print("Splitting data into training and validation sets...")
df_clean['price_bin'] = pd.qcut(df_clean['price'], q=10, labels=False,
duplicates='drop')

train_df, val_df = train_test_split(
    df_clean,
    test_size=0.2,
    random_state=42,
    stratify=df_clean['price_bin']
)

train_df = train_df.drop(columns=['price_bin'])
val_df = val_df.drop(columns=['price_bin'])

print(f"Training set shape: {train_df.shape}")
print(f"Validation set shape: {val_df.shape}")
print("□ Feature engineering and data splitting complete for mid 25k
data.")
```

```
Starting with 19871 rows.
Extracting brand names...
Engineering numerical features...
Splitting data into training and validation sets...
Training set shape: (15896, 10)
Validation set shape: (3975, 10)
□ Feature engineering and data splitting complete for mid 25k data.
```

```
import tensorflow as tf
from tqdm.auto import tqdm
from sklearn.model_selection import train_test_split

def is_image_valid(image_path):
    try:
        img_bytes = tf.io.read_file(image_path)
        tf.image.decode_jpeg(img_bytes, channels=3)
        return True
    except tf.errors.InvalidArgumentError:
        return False

print("Pre-filtering middle 25k dataset to remove corrupted images.
This may take a few minutes...")
tqdm.pandas(desc="Verifying images")

df_clean['is_valid'] =
df_clean['image_path'].progress_apply(is_image_valid)

df_fully_clean =
df_clean[df_clean['is_valid']].copy().drop(columns=['is_valid'])
```

```
print(f"\nRemoved {len(df_clean) - len(df_fully_clean)} corrupted
images from mid 25k.")
print(f"Your final mid 25k dataset has {len(df_fully_clean)} valid
images.")

df_fully_clean['price_bin'] = pd.qcut(df_fully_clean['price'], q=10,
labels=False, duplicates='drop')

train_df, val_df = train_test_split(
    df_fully_clean,
    test_size=0.2,
    random_state=42,
    stratify=df_fully_clean['price_bin']
)

train_df = train_df.drop(columns=['price_bin'])
val_df = val_df.drop(columns=['price_bin'])

print("\nRe-split of mid 25k data using only valid images complete.")
```

```
Pre-filtering middle 25k dataset to remove corrupted images. This may
take a few minutes...
```

{"model_id":"ba111a8813ed4986acab0c7484c79506","version_major":2,"version_minor":0}

```
Removed 19 corrupted images from mid 25k.
Your final mid 25k dataset has 19852 valid images.

Re-split of mid 25k data using only valid images complete.
```

```
import tensorflow as tf

# --- Define Constants ---
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 32
AUTOTUNE = tf.data.AUTOTUNE

# --- Preprocessing Function (no changes here) ---
def preprocess_image(image_path, price):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, IMAGE_SIZE)
    image = image / 255.0
    return image, price

# --- 3. Training data pipeline ---
# This is the corrected part. We use train_df['price'].
train_ds = tf.data.Dataset.from_tensor_slices((train_df['image_path'],
train_df['price']))
```

```
train_ds = train_ds.map(preprocess_image,
num_parallel_calls=AUTOTUNE).shuffle(1024).batch(BATCH_SIZE).prefetch(
AUTOTUNE)

# --- 4. Validation data pipeline ---
# This is the corrected part. We use val_df['price'].
val_ds = tf.data.Dataset.from_tensor_slices((val_df['image_path'],
val_df['price']))
val_ds = val_ds.map(preprocess_image,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

print("□ Data pipelines created successfully! They are now using the
raw 'price'.")
```

```
□ Data pipelines created successfully! They are now using the raw
'price'.
```

```
from tensorflow.keras import layers

# --- 1. Load a pre-trained base model ---
base_model = tf.keras.applications.EfficientNetB0(
    include_top=False,
    weights='imagenet',
    input_shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3)
)
base_model.trainable = False # Freeze the base

# --- 2. Build our custom model on top ---
inputs = layers.Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3),
name="input_layer")
x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D(name="global_avg_pooling")(x)
x = layers.Dense(128, activation="relu", name="dense_1")(x)
x = layers.Dropout(0.3, name="dropout_layer")(x)
outputs = layers.Dense(1, name="output_layer")(x)
model = tf.keras.Model(inputs, outputs)

# --- 3. Compile the model ---
# NOTE: The loss and metrics will now be in dollars. For example, a
# 'mean_absolute_error' of 15 means the model's predictions are, on
average,
# $15 off from the actual price.
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='mean_squared_error',
    metrics=['mean_absolute_error']
)

print("□ Model built and compiled successfully.")
model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
□ Model built and compiled successfully.

Model: "functional"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ input_layer (InputLayer)        │ (None, 224, 224, 3)    │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ efficientnetb0 (Functional)     │ (None, 7, 7, 1280)     │
4,049,571 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ global_avg_pooling              │ (None, 1280)           │
0 │
│ (GlobalAveragePooling2D)        │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dense_1 (Dense)                 │ (None, 128)            │
163,968 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout_layer (Dropout)         │ (None, 128)            │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ output_layer (Dense)            │ (None, 1)              │
129 │
└─────────────────────────────────┴────────────────────────┴──────────
─────┘

 Total params: 4,213,668 (16.07 MB)

 Trainable params: 164,097 (641.00 KB)

 Non-trainable params: 4,049,571 (15.45 MB)

# --- 1. Set up Early Stopping ---
# This callback monitors the validation loss and stops training if it
doesn't improve
# for 3 consecutive epochs. This saves time and prevents overfitting.
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True # Automatically restores the best model
weights
)

# --- 2. Train the model ---
# This is where the learning happens. We feed it our prepared data.
print("\nStarting model training...")
history = model.fit(
    train_ds,
    epochs=20, # Train for a maximum of 20 epochs
    validation_data=val_ds,
    callbacks=[early_stopping] # Add our early stopping callback
)
print("□ Model training complete.")
```

```
Starting model training...
Epoch 1/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1846s 4s/step - loss: 1194.6115 -
mean_absolute_error: 19.6008 - val_loss: 1051.7643 -
val_mean_absolute_error: 19.3035
Epoch 2/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1841s 4s/step - loss: 1165.8469 -
mean_absolute_error: 19.6819 - val_loss: 1052.6249 -
val_mean_absolute_error: 18.6583
Epoch 3/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1738s 3s/step - loss: 1125.6727 -
mean_absolute_error: 19.4533 - val_loss: 1051.2549 -
val_mean_absolute_error: 19.2910
Epoch 4/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1694s 3s/step - loss: 1115.0055 -
mean_absolute_error: 19.4121 - val_loss: 1050.6559 -
val_mean_absolute_error: 19.1443
Epoch 5/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1741s 3s/step - loss: 1137.7013 -
mean_absolute_error: 19.3772 - val_loss: 1050.5201 -
val_mean_absolute_error: 19.2276
Epoch 6/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1747s 3s/step - loss: 1153.1946 -
mean_absolute_error: 19.4539 - val_loss: 1051.0536 -
val_mean_absolute_error: 18.7347
Epoch 7/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1843s 4s/step - loss: 1133.8771 -
mean_absolute_error: 19.4938 - val_loss: 1052.2225 -
val_mean_absolute_error: 18.5513
Epoch 8/20
497/497 ━━━━━━━━━━━━━━━━━━━━ 1754s 3s/step - loss: 1131.4709 -
mean_absolute_error: 19.4530 - val_loss: 1053.2708 -
```

```
val_mean_absolute_error: 18.4277
□ Model training complete.
```

```
# --- Define a path to your Google Drive to save the model ---
model_shiv =
'/content/drive/MyDrive/amazon_dataset/amazon_price_model.keras'

# --- Save the entire model ---
model.save(model_shiv)

print(f"□ Model saved successfully to: {model_shiv}")
```

```
□ Model saved successfully to:
/content/drive/MyDrive/amazon_dataset/amazon_price_model.keras
```

```
import pandas as pd

# Load the saved DataFrames
train_df =
pd.read_pickle('/content/drive/MyDrive/amazon_dataset/train_df_final.p
kl')
val_df =
pd.read_pickle('/content/drive/MyDrive/amazon_dataset/val_df_final.pkl
')

print("□ Final training and validation data loaded successfully!")
print(f"Training set shape: {train_df.shape}")
print(f"Validation set shape: {val_df.shape}")
```

```
□ Final training and validation data loaded successfully!
Training set shape: (15881, 10)
Validation set shape: (3971, 10)
```

```
import tensorflow as tf

# Load the saved model
model =
tf.keras.models.load_model('/content/drive/MyDrive/amazon_dataset/
amazon_price_model.keras')

print("□ Trained model loaded successfully!")
```

```
□ Trained model loaded successfully!
```

```
train_df.head()
```

```
{"summary":"{\n  \"name\": \"train_df\",\n  \"rows\": 15881,\n
\"fields\": [\n    {\n      \"column\": \"sample_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
86293,\n        \"min\": 4,\n        \"max\": 299416,\n
\"num_unique_values\": 15881,\n        \"samples\": [\n
33402,\n          281947,\n          247216\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"catalog_content\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15868,\n        \"samples\": [\n
\"Item Name: Sheel Chick Peas/Kabuli Chana 2 lbs\\nBullet Point 1:
Wholesome & Nutritious\\nBullet Point 2: Procured from Selected
Sources\\nBullet Point 3: Carefully Cleaned & Processed\\nBullet Point
4: Tested for Consistent Superior Quality\\nBullet Point 5:
Hygienically Handled & Packed\\nProduct Description: The chickpea or
chick pea is an annual legume of the family Fabaceae, subfamily
Faboideae. Its different types are variously known as gram or Bengal
gram, garbanzo or garbanzo bean, Egyptian pea. Chickpea seeds are high
in protein.\\nValue: 32.0\\nUnit: Ounce\\n\",\n          \"Item Name:
Shake 'n Bake Seasoned Coating Mix - Parmesan Crusted - 4.75 Oz\\
nBullet Point 1: Product Type:Grocery\\nBullet Point 2: Item Package
Dimension:3.3 cm L X15.9 cm W X21.5 cm H X\\nBullet Point 3: Item
Package Weight:0.183 kg\\nBullet Point 4: Country Of Origin: United
States\\nValue: 4.75\\nUnit: Ounce\\n\",\n          \"Item Name:
Kerrygold Pure Irish Butter - Unsalted (8 ounce)\\nBullet Point 1:
Pack of eight ounces\\nBullet Point 2: Kerrygold's higher fat content
gives its butter a distinctive richness\\nBullet Point 3: The foil
wrapper preserves freshness and premium quality\\nBullet Point 4: Pure
Irish butter\\nValue: 8.0\\nUnit: Ounce\\n\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"image_link\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15240,\n        \"samples\": [\n
\"https://m.media-amazon.com/images/I/51a211Ln5mL.jpg\",\n
\"https://m.media-amazon.com/images/I/81XEbkh9A3L.jpg\",\n
\"https://m.media-amazon.com/images/I/81CRqvJVQ1L.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"price\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 32.64278149939822,\n
\"min\": 0.36,\n        \"max\": 1280.0,\n
\"num_unique_values\": 5469,\n        \"samples\": [\n
31.96,\n          95.99,\n          4.109999999999999\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"image_path\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15240,\n        \"samples\": [\n
\"/content/drive/MyDrive/amazon_dataset/train_images/51a211Ln5mL.jpg\"
,\n
\"/content/drive/MyDrive/amazon_dataset/train_images/81XEbkh9A3L.jpg\"
,\n
\"/content/drive/MyDrive/amazon_dataset/train_images/81CRqvJVQ1L.jpg\"
\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"brand\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13832,\n        \"samples\": [\n
\"Hula Market Maffles Mochi Waffle Mix\",\n          \"Mom Brand
Frosted Flakes,\",\n          \"Hormel Chili No Beans,\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pack_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.652462517846176,\n        \"min\": 1.0,\n        \"max\": 1000.0,\n
\"num_unique_values\": 51,\n        \"samples\": [\n          416.0,\n
25.0,\n          21.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"item_size\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16.05942237000445,\n        \"min\":
0.0,\n        \"max\": 512.0,\n        \"num_unique_values\": 729,\n
\"samples\": [\n          22.8,\n          12.72,\n          30.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"unit_price\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
29664747.861407224,\n        \"min\": 0.0017934782576205105,\n
\"max\": 1280000000.0,\n        \"num_unique_values\": 10885,\n
\"samples\": [\n          2.030862943911763,\n
0.056919642730090085,\n          6130000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"log_price\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.945125313055822,\n        \"min\": 0.3074846997479606,\n
\"max\": 7.155396301896734,\n        \"num_unique_values\": 5397,\n
\"samples\": [\n          3.2449333591874905,\n
4.935408747853786,\n          2.560709613203501\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"train_df"}
```

```python
import tensorflow as tf
import pandas as pd
import os
from tqdm.auto import tqdm

# --- 1. Define the image health-check function ---
# This function will check if an image file is valid and can be
opened.
def is_image_valid(image_path):
    """Checks if an image can be successfully decoded by
TensorFlow."""
    try:
        # First, check if the file exists and is not empty
        if not os.path.exists(image_path) or
os.path.getsize(image_path) == 0:
            return False
        # Then, try to decode it
        img_bytes = tf.io.read_file(image_path)
        tf.image.decode_jpeg(img_bytes, channels=3)
        return True
    except Exception:
        # If any error occurs (e.g., file not found, corrupted), it's
invalid
        return False

# --- 2. Load the test data and create image paths ---
test_df =
pd.read_csv('/content/drive/MyDrive/amazon_dataset/test.csv')
test_image_dir = '/content/drive/MyDrive/amazon_dataset/test_images'
test_df['image_path'] = test_df['image_link'].apply(lambda link:
os.path.join(test_image_dir, str(link).split('/')[-1]))

# --- 3. NEW: Filter the test_df for valid images ---
print("Pre-filtering test dataset to find valid images. This may take
a moment...")
tqdm.pandas(desc="Verifying test images")
test_df['is_valid'] =
test_df['image_path'].progress_apply(is_image_valid)

# Create a clean DataFrame with only the rows that have valid images
test_df_clean = test_df[test_df['is_valid']].copy()
print(f"\nFound {len(test_df_clean)} valid images out of
{len(test_df)} to predict on.")

# --- 4. Create a data pipeline using ONLY the clean data ---
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 32
AUTOTUNE = tf.data.AUTOTUNE

def preprocess_test_image(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, IMAGE_SIZE)
    image = image / 255.0
    return image

# Build the pipeline from the clean DataFrame's image paths
test_ds_clean =
tf.data.Dataset.from_tensor_slices(test_df_clean['image_path'])
test_ds_clean = test_ds_clean.map(preprocess_test_image,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

# --- 5. Get predictions on the CLEAN data ---
print("\nMaking predictions on the clean test set...")
# Make sure you have your trained model loaded, e.g., model =
tf.keras.models.load_model(...)
predictions = model.predict(test_ds_clean)
predicted_prices = predictions.flatten()
print("□ Predictions complete.")
```

```
# --- 6. Create the final submission file ---
# Add the predictions as a new 'price' column to our clean DataFrame
test_df_clean['price'] = predicted_prices

# Now, merge these predictions back into the original full test_df.
# This ensures we have a row for EVERY sample_id.
submission_df =
test_df[['sample_id']].merge(test_df_clean[['sample_id', 'price']],
on='sample_id', how='left')

# Fill any missing prices (from bad images) with a neutral value, like
the mean or median of your predictions.
# Using the mean is a safe and common strategy.
mean_price = submission_df['price'].mean()
submission_df['price'].fillna(mean_price, inplace=True)

# As a final safety check, make sure all prices are positive
submission_df['price'] = submission_df['price'].apply(lambda p: max(0,
p))

# --- 7. Save the final, complete submission file ---
submission_path = '/content/drive/MyDrive/amazon_dataset/test_out.csv'
submission_df.to_csv(submission_path, index=False)

print(f"\n□ Submission file with {len(submission_df)} rows created
successfully at: {submission_path}")
```

```
Pre-filtering test dataset to find valid images. This may take a
moment...
```

{"model_id":"efe400468bd0464c9e394ca606ee71ff","version_major":2,"vers
ion_minor":0}

```
Found 0 valid images out of 75000 to predict on.
```

```
Making predictions on the clean test set...
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/
epoch_iterator.py:160: UserWarning: Your input ran out of data;
interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to
use the `.repeat()` function when building your dataset.
  self._interrupted_warning()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)
/tmp/ipython-input-2802077611.py in <cell line: 0>()
     53 print("\nMaking predictions on the clean test set...")
     54 # Make sure you have your trained model loaded, e.g., model =
tf.keras.models.load_model(...)
---> 55 predictions = model.predict(test_ds_clean)
     56 predicted_prices = predictions.flatten()
     57 print(" Predictions complete.")

/usr/local/lib/python3.12/dist-packages/keras/src/utils/traceback_util
s.py in error_handler(*args, **kwargs)
    120             # To get the full stack trace, call:
    121             # `keras.config.disable_traceback_filtering()`
--> 122             raise e.with_traceback(filtered_tb) from None
    123         finally:
    124             del filtered_tb

/usr/local/lib/python3.12/dist-packages/keras/src/utils/progbar.py in
update(self, current, values, finalize)
    117
    118             if self.target is not None:
--> 119                 numdigits = int(math.log10(self.target)) + 1
    120                 bar = ("%" + str(numdigits) + "d/%d") %
(current, self.target)
    121                 bar = f"\x1b[1m{bar}\x1b[0m "

ValueError: math domain error
```

```python
import tensorflow as tf
import pandas as pd
import os
from tqdm.auto import tqdm

# --- 1. Load the official test data ---
test_df =
pd.read_csv('/content/drive/MyDrive/amazon_dataset/test.csv')

# --- 2. CORRECTED: Create image paths using the sample_id ---
# This matches how your download script saved the files.
test_image_dir = '/content/drive/MyDrive/amazon_dataset/test_images' #
Or whatever your folder is named
test_df['image_path'] = test_df['sample_id'].apply(lambda sid:
os.path.join(test_image_dir, f'{sid}.jpeg'))

print("Building image paths based on sample_id...")

# --- 3. Filter for valid/existing images ---
# (This step is still important to handle any failed downloads)
tqdm.pandas(desc="Verifying test images")
test_df['is_valid'] =
test_df['image_path'].progress_apply(os.path.exists)
test_df_clean = test_df[test_df['is_valid']].copy()
print(f"\nFound {len(test_df_clean)} valid images out of
{len(test_df)}.")

# --- 4. Create a data pipeline using the clean data ---
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 32
AUTOTUNE = tf.data.AUTOTUNE

def preprocess_test_image(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, IMAGE_SIZE)
    image = image / 255.0
    return image

test_ds_clean =
tf.data.Dataset.from_tensor_slices(test_df_clean['image_path'])
test_ds_clean = test_ds_clean.map(preprocess_test_image,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

# --- 5. Get predictions ---
print("\nMaking predictions on the clean test set...")
predictions = model.predict(test_ds_clean)
predicted_prices = predictions.flatten()
print("□ Predictions complete.")

# --- 6. Create the final submission file ---
test_df_clean['price'] = predicted_prices
submission_df =
test_df[['sample_id']].merge(test_df_clean[['sample_id', 'price']],
on='sample_id', how='left')
mean_price = submission_df['price'].mean()
submission_df['price'].fillna(mean_price, inplace=True)
submission_df['price'] = submission_df['price'].apply(lambda p: max(0,
p))

# --- 7. Save the submission file ---
submission_path = '/content/drive/MyDrive/test_out.csv'
submission_df.to_csv(submission_path, index=False)

print(f"\n□ Submission file created successfully at:
{submission_path}")
```

```
Building image paths based on sample_id...
```

{"model_id":"068dd48edad74674af4514653956b0da","version_major":2,"vers
ion_minor":0}

```
Found 65681 valid images out of 75000.

Making predictions on the clean test set...
```

```
2053/2053 ━━━━━━━━━━━━━━━━━━━━ 18867s 9s/step
□ Predictions complete.

/tmp/ipython-input-2041998222.py:48: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  submission_df['price'].fillna(mean_price, inplace=True)


□ Submission file created successfully at:
/content/drive/MyDrive/test_out.csv
```

```python
import pandas as pd
import os
from PIL import Image
import matplotlib.pyplot as plt

# --- 1. Load your submission and original test data ---
submission_path = '/content/drive/MyDrive/test_out.csv'
test_csv_path = '/content/drive/MyDrive/amazon_dataset/test.csv'
test_image_dir = '/content/drive/MyDrive/amazon_dataset/test_images' #
Your test images folder

submission_df = pd.read_csv(submission_path)
test_df = pd.read_csv(test_csv_path)

# --- 2. Merge the data ---
display_df = pd.merge(submission_df, test_df, on='sample_id')

# --- 3. Select 5 new random samples to display ---
samples_to_show = display_df.sample(5)

print("--- Displaying 5 new random samples ---")

# --- 4. Loop through the samples and display everything ---
for index, row in samples_to_show.iterrows():
    sample_id = row['sample_id']
    predicted_price = row['price']
    catalog_content = row['catalog_content']

    # Build the image path
    image_path = os.path.join(test_image_dir, f"{sample_id}.jpeg")

    print("--------------------------------------------------")
    print(f"Sample ID: {sample_id}")
    print(f"PREDICTED PRICE: ${predicted_price:.2f}")
    print(f"Catalog Content: {catalog_content}")
    print("--------------------------------------------------")

    # Display the image
    try:
        img = Image.open(image_path)
        plt.figure(figsize=(4, 4))
        plt.imshow(img)
        plt.axis('off')
        plt.show()
    except Exception as e:
        print(f"Could not display image. Reason: {e}")
```

```
--- Displaying 5 new random samples ---
--------------------------------------------------
Sample ID: 286959
PREDICTED PRICE: $24.45
Catalog Content: Item Name: Frontier Co-op Organic Celery Salt 1lb
Bullet Point 1: Certified Organic
Bullet Point 2: 1 pound bulk bag (16 ounces)
Product Description: Many cooks consider salt a flavor enhancer;
celery adds another dimension. Use this blend of sea salt and ground
celery seed in place of salt in most any dish.
Value: 16.0
Unit: Ounce

--------------------------------------------------
```

```
--------------------------------------------------
Sample ID: 260051
PREDICTED PRICE: $24.12
Catalog Content: Item Name: Finn Crisps Original Rye Crispbread 7oz |
Thin, Crispy Rye Flatbread Crackers | Multipack of Authentic Finnish
Sourdough Crispbread | Whole Grain Rye Crackers [2 Boxes x 7oz]
Bullet Point 1: RYE CRISPBREAD: Enjoy the crispy toasted grain crunch
of our Original Rye Crispbreads. Full of fiber and fresh crunch, our
thin crisps are ideal as a base for your lunch or a light snack
between meals. Dress up with different toppings for snack platters,
stack up alongside a charcuterie board or simply spread with butter
for breakfast.
Bullet Point 2: LOW CALORIE, HIGH FIBER: With 1.3grams of fiber per
cracker, these rye crispbreads will keep you full and satisfied
whether you’re on a low fat, low carb, low sugar, keto or vegan diet.
There are approximately 30 fresh and crunchy rye crackers in each box
of Finn Crisps Rye Crispbread.
Bullet Point 3: SO VERSATILE: Consider each crispbread a blank canvas
for your culinary creations. Best topped with a spread of protein and
a layer of fresh fruit or vegetables, the possibilities are endless.
Smoked salmon, radish and dill, avocado and salsa, cream cheese and
strawberries, cheddar and lingonberry jam or PB&J with cucumbers- try
something different every time. For more ryefull inspiration, check
out our recipes page. This 2 pack will keep you going for a while.
Bullet Point 4: MADE IN FINLAND: Baked with wholegrain rye flour and
sourdough, we prepare these rye thins in Finland using traditional
Nordic recipes and techniques. We keep things as simple as possible
with no additives, trans fat or GMO, and minimal sugar or salt, just
like our ancestors have been doing for years. Our delicious, crispy
rye crispbreads are also certified Kosher.
```

```
Bullet Point 5: FINN CRISP: After years of rye flour and crispbread
production, the Finn Crisp brand was officially established and
launched at the Helsinki Summer Olympics. We produce our rye
crispbreads, favorite staples in every Finnish home, with simple, pure
ingredients and authentic Finnish baking traditions and recipes.
Value: 14.0
Unit: Ounce

--------------------------------------------------
```

```
--------------------------------------------------
Sample ID: 234
PREDICTED PRICE: $24.70
Catalog Content: Item Name: Purina Beneful Originals With Real Beef
Adult Dry Dog Food - 15.5 Lb. Bag
Bullet Point 1: Purina Beneful Originals With Real Beef Adult Dry Dog
Food
Bullet Point 2: Real Farm-Raised Beef Is The #1 Ingredient
Bullet Point 3: Accents Of Real Spinach, Peas And Carrots Add Variety
To His Diet
Bullet Point 4: Antioxidant-Rich Nutrition To Help Support A Healthy
Immune System
Bullet Point 5: Proudly Produced In Purina-Owned U.S. Facilities To
Ensure Safe, Quality Food
Value: 15.5
Unit: pound

--------------------------------------------------
```

```
--------------------------------------------------
Sample ID: 292155
PREDICTED PRICE: $24.94
Catalog Content: Item Name: Trani flavor syrup Pink Grapefruit 750ml
Bullet Point: Unit Sold By: Bottle (750mL)
Product Description: Take a quick little jaunt to Florida. Flavor your
drink with this Ruby Red Grapefruit Syrup, and you'll swear you're in
the Sunshine State. The perfect balance of sweet and tart welcomes you
to a citrus heaven. Ingredients: Pure cane sugar, water, natural
flavors, citric acid, tartaric acid, sodium benzoate (to preserve
freshness), FD&C red # 40
Value: 1.0
Unit: Count

--------------------------------------------------
```

```
--------------------------------------------------
Sample ID: 217600
PREDICTED PRICE: $25.01
Catalog Content: Item Name: Tide Free And Gentle Laundry Detergent,
100 Loads, 132 FL OZ
Bullet Point: LQ HE SCNT FREE
Value: 132.0
Unit: Fl Oz

--------------------------------------------------
```

```python
import pandas as pd
import os
from PIL import Image
import matplotlib.pyplot as plt

# --- 1. Define the paths to your files ---
submission_path = '/content/drive/MyDrive/test_out.csv'
test_csv_path = '/content/drive/MyDrive/amazon_dataset/test.csv'
test_image_dir = '/content/drive/MyDrive/amazon_dataset/test_images' #
Your test images folder

# --- 2. Load your submission file and the original test data ---
print("Loading your final submission and original test data...")
submission_df = pd.read_csv(submission_path)
test_df = pd.read_csv(test_csv_path)

# --- 3. Merge them to link predictions with the original product info
---
# This uses 'sample_id' to match the rows from both files.
display_df = pd.merge(submission_df, test_df, on='sample_id')
print("□ Data merged successfully.")

# --- 4. Select 5 random samples to display ---
samples_to_show = display_df.sample(5)

print("\n--- Displaying 5 random samples from your final submission
---")

# --- 5. Loop through the samples and display everything ---
for index, row in samples_to_show.iterrows():
    sample_id = row['sample_id']
    predicted_price = row['price']
    catalog_content = row['catalog_content']

    # Build the image path using the sample_id (which matches how they
were saved)
    image_path = os.path.join(test_image_dir, f"{sample_id}.jpeg")

    print("\n--------------------------------------------------")
    print(f"Sample ID: {sample_id}")
    print(f"FINAL PREDICTED PRICE: ${predicted_price:.2f}")
    print(f"Catalog Content: {catalog_content}")
    print("--------------------------------------------------")

    # Display the corresponding image
    try:
        img = Image.open(image_path)
        plt.figure(figsize=(4, 4))
        plt.imshow(img)
        plt.axis('off')
        plt.show()
    except Exception as e:
        print(f"Could not display image. Reason: {e}")
```

```
Loading your final submission and original test data...
□ Data merged successfully.

--- Displaying 5 random samples from your final submission ---

--------------------------------------------------
Sample ID: 170429
FINAL PREDICTED PRICE: $24.99
Catalog Content: Item Name: Mario Camacho Foods Pitted Queen Party
Colossal Olives, 9 Ounce
Bullet Point 1: Gluten free
Bullet Point 2: Item Package Length: 4.572cm
Bullet Point 3: Item Package Width: 7.366cm
Bullet Point 4: Item Package Height: 18.288cm
Value: 9.0
Unit: ounce

--------------------------------------------------
```

```

--------------------------------------------------
Sample ID: 293694
FINAL PREDICTED PRICE: $25.03
Catalog Content: Item Name: Frontier Co-op Minced Garlic, 1-Pound Bulk
Bag, Aromatic and Flavorful, Great For Savory Dishes
Bullet Point 1: Origin: China
Bullet Point 2: May add directly to food or rehydrate by soaking in
```

```
cool water for 30 minutes
Bullet Point 3: It's compatible with virtually every savory food
Value: 32.0
Unit: Fl Oz

--------------------------------------------------
```

```
--------------------------------------------------
Sample ID: 128282
FINAL PREDICTED PRICE: $24.77
Catalog Content: Item Name: Maple Grove Farms Sugar Free Salad
Dressing, Raspberry Vinaigrette, 8 Ounce (Pack of 12)
Bullet Point 1: Nothing can be sweeter than a delicious Raspberry
Vinaigrette, well except when that Raspberry Vinaigrette is one our
Sugar Free dressings
Bullet Point 2: Made with SPLENDA No Calorie Sweetener
Bullet Point 3: These delicious salad dressings are great on your
favorite salads and pasta salads, as marinades, basting sauces and
more
Bullet Point 4: So go ahead, indulge in our great tasting sugar free
dressings
Bullet Point 5: Delicious flavors with no preservatives or artificial
ingredients - just the way nature intended, simple
Value: 96.0
Unit: Ounce

--------------------------------------------------
```

```
--------------------------------------------------
Sample ID: 144601
FINAL PREDICTED PRICE: $24.39
Catalog Content: Item Name: Mahatma Jasmine Rice, 2 lb.
Bullet Point 1: Mahatma Jasmine Long Grain Thai Fragrant Rice
Bullet Point 2: Mahatma Jasmine Long Grain Thai Fragrant Rice
Value: 32.0
Unit: Ounce

--------------------------------------------------
```

```
--------------------------------------------------
Sample ID: 5749
FINAL PREDICTED PRICE: $24.64
Catalog Content: Item Name: BRYAN Vienna Sausage, 4.6 Ounce Pull-Top
Can (Pack of 48) | Canned Meat | Keto Food, Keto Snacks | Low Carb
High Protein Snacks | Compare to Other Brands of Vienna Sausages &
Smoked Sausages
Bullet Point 1: BRYAN VIENNA SAUSAGE: Made with a great-tasting
combination of chicken, pork and spices, BRYAN Vienna Sausage is a
convenient pantry staple that's perfect for appetizers, snacks or a
quick meal
Bullet Point 2: QUALITY IN EVERY CAN: 48 4.6-ounce pull-top cans of
BRYAN Vienna Sausage
Bullet Point 3: PACKED WITH PROTEIN: Each serving contains 7g of
protein, 150 calories and less than 1g of carbohydrates—these canned
Vienna sausages are low carb high protein snacks are delicious keto
snacks for individuals following a keto diet
Bullet Point 4: CONVENIENT PACKAGING: Sturdy shelf-stable packaging is
perfect for outdoor activities like hunting, fishing, and camping as
well as stocking your pantry at home
Bullet Point 5: THE FLAVOR OF THE SOUTH: Bryan has been an established
and trusted brand for 80+ years!
Value: 220.8
Unit: Ounce

--------------------------------------------------
```

```python
import os
```

```
def get_file_size(file_path):
    """Returns the file size in a human-readable format."""
    try:
        size_bytes = os.path.getsize(file_path)
        if size_bytes < 1024:
            return f"{size_bytes} Bytes"
        elif size_bytes < 1024**2:
            return f"{size_bytes/1024:.2f} KB"
        else:
            return f"{size_bytes/1024**2:.2f} MB"
    except FileNotFoundError:
        return "File not found."

# --- List of your important files ---
files_to_check = {
    "Keras Model":
'/content/drive/MyDrive/amazon_dataset/amazon_price_model.keras',
    "Pickled Model":
'/content/drive/MyDrive/amazon_dataset/amazon_price_pickel.pkl',
    "Final Train DF":
'/content/drive/MyDrive/amazon_dataset/train_df_final.pkl',
    "Final Val DF":
'/content/drive/MyDrive/amazon_dataset/val_df_final.pkl',
    "Submission File": '/content/drive/MyDrive/test_out.csv'
}

print("--- File Sizes ---")
for name, path in files_to_check.items():
    size = get_file_size(path)
    print(f"{name}: {size}")
```

```
--- File Sizes ---
Keras Model: 18.14 MB
Pickled Model: 18.14 MB
Final Train DF: 17.28 MB
Final Val DF: 4.39 MB
Submission File: 1.81 MB
```

```
import pandas as pd

# Define the path to your saved submission file
submission_path = '/content/drive/MyDrive/test_out.csv'

# Load the CSV file into a DataFrame
try:
    submission_df = pd.read_csv(submission_path)
    num_rows = len(submission_df)
    print(f"□ Your submission file 'test_out.csv' has {num_rows}
rows.")
    if num_rows == 75000:
        print("This is the correct number required for submission.
Perfect!")
except FileNotFoundError:
    print("□ File not found. Please make sure the prediction script
that creates 'test_out.csv' has completed successfully.")
```

```
□ Your submission file 'test_out.csv' has 75000 rows.
This is the correct number required for submission. Perfect!
```

```
import pandas as pd

# Load your final, feature-rich DataFrames
train_df =
pd.read_pickle('/content/drive/MyDrive/amazon_dataset/train_df_final.p
kl')
val_df =
pd.read_pickle('/content/drive/MyDrive/amazon_dataset/val_df_final.pkl
')

print("□ Final training and validation data loaded successfully!")
```

```
□ Final training and validation data loaded successfully!
```

```
import tensorflow as tf

# --- Define Constants ---
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 32
AUTOTUNE = tf.data.AUTOTUNE

# --- Preprocessing Function (no changes) ---
def preprocess_image(image_path, target_price):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, IMAGE_SIZE)
    image = image / 255.0
    return image, target_price

# --- 3. Build the CORRECTED Training Data Pipeline ---
# This is the key change: use the 'log_price' column
train_ds = tf.data.Dataset.from_tensor_slices((train_df['image_path'],
train_df['log_price']))
train_ds = train_ds.map(preprocess_image,
num_parallel_calls=AUTOTUNE).shuffle(1024).batch(BATCH_SIZE).prefetch(
AUTOTUNE)

# --- 4. Build the CORRECTED Validation Data Pipeline ---
# Also use 'log_price' here
val_ds = tf.data.Dataset.from_tensor_slices((val_df['image_path'],
val_df['log_price']))
val_ds = val_ds.map(preprocess_image,
num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)

print("□ Data pipelines created successfully! The model will now learn
to predict 'log_price'.")
```

```
□ Data pipelines created successfully! The model will now learn to
predict 'log_price'.
```

```
train_df
```

{"summary":"{\n \"name\": \"train_df\",\n \"rows\": 15881,\n
\"fields\": [\n   {\n   \"column\": \"sample_id\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
86293,\n   \"min\": 4,\n   \"max\": 299416,\n
\"num_unique_values\": 15881,\n   \"samples\": [\n
33402,\n   281947,\n   247216\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"catalog_content\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 15868,\n   \"samples\": [\n
\"Item Name: Sheel Chick Peas/Kabuli Chana 2 lbs\\nBullet Point 1:
Wholesome & Nutritious\\nBullet Point 2: Procured from Selected
Sources\\nBullet Point 3: Carefully Cleaned & Processed\\nBullet Point
4: Tested for Consistent Superior Quality\\nBullet Point 5:
Hygienically Handled & Packed\\nProduct Description: The chickpea or
chick pea is an annual legume of the family Fabaceae, subfamily
Faboideae. Its different types are variously known as gram or Bengal
gram, garbanzo or garbanzo bean, Egyptian pea. Chickpea seeds are high
in protein.\\nValue: 32.0\\nUnit: Ounce\\n\",\n   \"Item Name:
Shake 'n Bake Seasoned Coating Mix - Parmesan Crusted - 4.75 Oz\\
nBullet Point 1: Product Type:Grocery\\nBullet Point 2: Item Package
Dimension:3.3 cm L X15.9 cm W X21.5 cm H X\\nBullet Point 3: Item
Package Weight:0.183 kg\\nBullet Point 4: Country Of Origin: United
States\\nValue: 4.75\\nUnit: Ounce\\n\",\n   \"Item Name:
Kerrygold Pure Irish Butter - Unsalted (8 ounce)\\nBullet Point 1:
Pack of eight ounces\\nBullet Point 2: Kerrygold's higher fat content
gives its butter a distinctive richness\\nBullet Point 3: The foil
wrapper preserves freshness and premium quality\\nBullet Point 4: Pure
Irish butter\\nValue: 8.0\\nUnit: Ounce\\n\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"image_link\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 15240,\n   \"samples\": [\n
\"https://m.media-amazon.com/images/I/51a211Ln5mL.jpg\",\n
\"https://m.media-amazon.com/images/I/81XEbkh9A3L.jpg\",\n
\"https://m.media-amazon.com/images/I/81CRqvJVQ1L.jpg\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"price\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 32.64278149939822,\n
\"min\": 0.36,\n   \"max\": 1280.0,\n

```
\"num_unique_values\": 5469,\n        \"samples\": [\n
31.96,\n          95.99,\n          4.109999999999999\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"image_path\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15240,\n        \"samples\": [\n
\"/content/drive/MyDrive/amazon_dataset/train_images/51a211Ln5mL.jpg\"
,\n
\"/content/drive/MyDrive/amazon_dataset/train_images/81XEbkh9A3L.jpg\"
,\n
\"/content/drive/MyDrive/amazon_dataset/train_images/81CRqvJVQ1L.jpg\"
\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"brand\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13832,\n        \"samples\": [\n
\"Hula Market Maffles Mochi Waffle Mix\",\n          \"Mom Brand
Frosted Flakes,\",\n          \"Hormel Chili No Beans,\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pack_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.652462517846176,\n        \"min\": 1.0,\n        \"max\": 1000.0,\n
\"num_unique_values\": 51,\n        \"samples\": [\n          416.0,\n
25.0,\n          21.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"item_size\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16.05942237000445,\n        \"min\":
0.0,\n        \"max\": 512.0,\n        \"num_unique_values\": 729,\n
\"samples\": [\n          22.8,\n          12.72,\n          30.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"unit_price\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
29664747.861407224,\n        \"min\": 0.0017934782576205105,\n
\"max\": 1280000000.0,\n        \"num_unique_values\": 10885,\n
\"samples\": [\n          2.030862943911763,\n
0.056919642730090085,\n          6130000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"log_price\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.945125313055822,\n        \"min\": 0.3074846997479606,\n
\"max\": 7.155396301896734,\n        \"num_unique_values\": 5397,\n
\"samples\": [\n          3.2449333591874905,\n
4.935408747853786,\n          2.560709613203501\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"train_df"}
```

```
from tensorflow.keras import layers
import tensorflow as tf

# --- 1. Build a fresh model (same architecture as before) ---
base_model = tf.keras.applications.EfficientNetB0(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3)
)
base_model.trainable = False # Freeze the pre-trained layers

inputs = layers.Input(shape=(224, 224, 3))
x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dropout(0.3)(x) # Dropout helps prevent overfitting
outputs = layers.Dense(1)(x) # Final output layer

# We'll name the new model 'model_log' to avoid confusion
model_log = tf.keras.Model(inputs, outputs)

# --- 2. Compile the model ---
model_log.compile(
    optimizer='adam',
    loss='mean_squared_error',
    metrics=['mean_absolute_error']
)

# --- 3. Set up Early Stopping ---
# This stops training when the validation loss stops improving
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=3, # Stop after 3 epochs with no improvement
    restore_best_weights=True # Keep the best version of the model
)

# --- 4. Re-train the model on the 'log_price' data ---
print("\nStarting NEW model training on 'log_price'. This will take
some time...")
history_log = model_log.fit(
    train_ds,
    epochs=20, # Maximum number of epochs
    validation_data=val_ds,
    callbacks=[early_stopping]
)
print("□ New model training complete.")

# --- 5. IMPORTANT: Save your new, smarter model! ---
new_model_path =
'/content/drive/MyDrive/amazon_dataset/amazon_price_log_model.keras'
model_log.save(new_model_path)
print(f"□ New 'log_price' model saved successfully to:
{new_model_path}")
```