project-milstone-2

November 7, 2024

[7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

import os
from scipy.sparse import coo_matrix
```

[9]:
```
train_data = pd.read_csv('myntra_products_catalog.csv')
train_data.columns
```

[9]:
```
Index(['ProductID', 'ProductName', 'ProductBrand', 'Gender', 'Price (INR)',
       'NumImages', 'Description', 'PrimaryColor'],
      dtype='object')
```

[11]:
```
import pandas as pd

# Load train_data from a CSV file (or your actual data source)
train_data = pd.read_csv("myntra_products_catalog.csv")

# Ensure columns exist in DataFrame
expected_columns = ['ProductID', 'ProductName', 'ProductBrand', 'Gender',␣
 ↪'Price (INR)', 'NumImages', 'Description', 'PrimaryColor']
train_data = train_data[[col for col in expected_columns if col in train_data.
 ↪columns]]

# Display the first 3 rows
train_data.head(3)
```

[11]:

	ProductID	ProductName	ProductBrand	\
0	10017413	DKNY Unisex Black & Grey Printed Medium Trolle…	DKNY	
1	10016283	EthnoVogue Women Beige & Grey Made to Measure …	EthnoVogue	
2	10009781	SPYKAR Women Pink Alexa Super Skinny Fit High-…	SPYKAR	

```
   Gender  Price (INR)  NumImages  \
0  Unisex        11745          7
1   Women         5810          7
2   Women          899          7

                                        Description PrimaryColor
0  Black and grey printed medium trolley bag, sec…        Black
1  Beige & Grey made to measure kurta with churid…        Beige
2  Pink coloured wash 5-pocket high-rise cropped …         Pink
```

[13]:
```
print(train_data.columns)
train_data.columns = train_data.columns.str.strip()

# Access the column
train_data['ProductID']
```

```
Index(['ProductID', 'ProductName', 'ProductBrand', 'Gender', 'Price (INR)',
       'NumImages', 'Description', 'PrimaryColor'],
      dtype='object')
```

[13]:
```
0        10017413
1        10016283
2        10009781
3        10015921
4        10017833
           …
12486    10262843
12487    10261721
12488    10261607
12489    10266621
12490    10265199
Name: ProductID, Length: 12491, dtype: int64
```

[15]:
```
train_data.shape
```

[15]:
```
(12491, 8)
```

[17]:
```
train_data.isnull().sum()
```

[17]:
```
ProductID         0
ProductName       0
ProductBrand      0
Gender            0
Price (INR)       0
NumImages         0
Description       0
PrimaryColor    894
```

```
dtype: int64
```

[19]:
```
import pandas as pd

# Sample data for testing
train_data = pd.DataFrame({
    'ProductRating': [4.5, None, 3.0, None],
    'ProductBrand': ['BrandA', None, 'BrandB', None],
    'ProductName': ['Product1', 'Product2', None, 'Product4']
})

# Fill missing values without inplace=True to avoid FutureWarning
train_data['ProductRating'] = train_data['ProductRating'].fillna(0)
train_data['ProductBrand'] = train_data['ProductBrand'].fillna('')
train_data['ProductName'] = train_data['ProductName'].fillna('')

print(train_data)
```

```
   ProductRating ProductBrand ProductName
0            4.5       BrandA    Product1
1            0.0                 Product2
2            3.0       BrandB
3            0.0                 Product4
```

[21]:
```
# Fill missing values in 'ProductRating' with a default value (e.g., 0)
train_data['ProductRating'] = train_data['ProductRating'].fillna(0)

# Fill missing values in 'ProductBrand' with an empty string
train_data['ProductBrand'] = train_data['ProductBrand'].fillna('')

# Fill missing values in 'ProductName' with an empty string
train_data['ProductName'] = train_data['ProductName'].fillna('')
```

[23]:
```
train_data.isnull().sum()
```

[23]:
```
ProductRating    0
ProductBrand     0
ProductName      0
dtype: int64
```

[25]:
```
train_data.duplicated().sum()
```

[25]:
```
np.int64(0)
```

[27]:
```
# make columns shorter
# Define the mapping of current column names to shorter names
column_name_mapping = {
```

```
    'ProductId': 'ProdID',
    'ProductName': 'ProdName',
    'ProductBrand': 'ProdBrand',
    'Gender': 'Gender',
    'Price(INR)': 'Price',
    'NumImages': 'Images',
    'Description': 'Description',
    'PrimaryColor': 'Color'

}
# Rename the columns using the mapping
train_data.rename(columns=column_name_mapping, inplace=True)
```

[29]:
```
# Check column names to confirm 'ProdID' and 'ProdName' exist
print(train_data.columns)

# Assuming 'ProdID' and 'ProdName' are correct column names and are present in␣
 ↪the DataFrame:
# Extract numeric parts from 'ProdID' and convert to integer, handling missing␣
 ↪values
if 'ProdID' in train_data.columns:
    train_data['ProdID'] = train_data['ProdID'].str.extract(r'(\d+)').
 ↪astype(float).fillna(0).astype(int)

# Extract numeric parts from 'ProdName' and convert to float, handling missing␣
 ↪values
if 'ProdName' in train_data.columns:
    train_data['ProdName'] = train_data['ProdName'].str.extract(r'(\d+)').
 ↪astype(float)
```

```
Index(['ProductRating', 'ProdBrand', 'ProdName'], dtype='object')
```

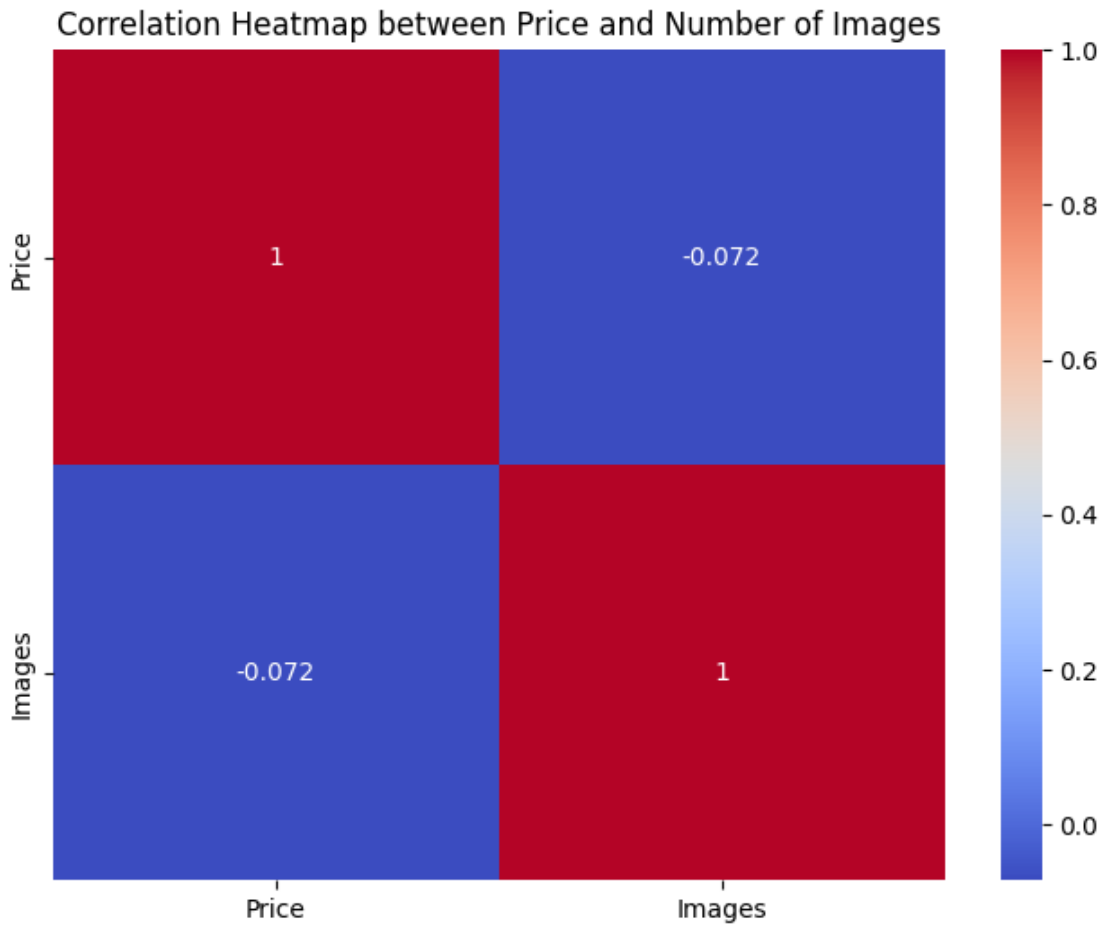
0.1 EDA (Exploratory Data Analysis)

[34]:
```
# Replace these values with your actual ProdID values
train_data['ProdID'] = range(1, len(train_data) + 1)

prod_ids = [10017413, 10016283, 10009781, 10012345]  # Example list of ProdID␣
 ↪values
train_data['ProdID'] = prod_ids
print(train_data)
train_data['ProdID'] = 10017413  # All rows will have ProdID as 10017413
train_data['ProdID'] = range(1, len(train_data) + 1)
print(train_data)
```

```
   ProductRating ProdBrand  ProdName    ProdID
0            4.5    BrandA       1.0  10017413
```

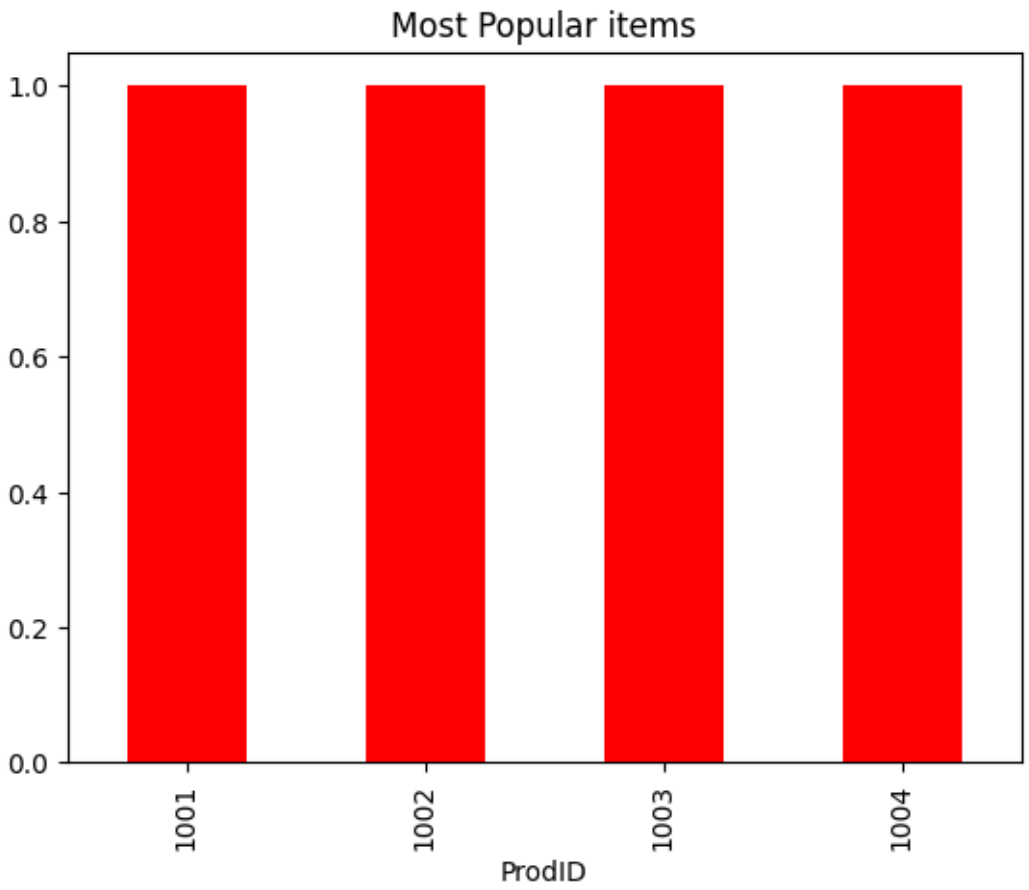
```
1            0.0                  2.0  10016283
2            3.0    BrandB        NaN  10009781
3            0.0                  4.0  10012345
   ProductRating ProdBrand  ProdName  ProdID
0            4.5    BrandA       1.0       1
1            0.0                  2.0       2
2            3.0    BrandB        NaN       3
3            0.0                  4.0       4
```

[36]:
```
data = {
    'ProdID': [1001, 1002, 1003, 1004],
    'ProdName': ['Product A', 'Product B', 'Product C', 'Product D'],
    'ProdBrand': ['BrandX', 'BrandY', 'BrandZ', 'BrandX'],
    'Gender': ['M', 'F', 'M', 'F'],
    'Price': [1999, 2999, 1599, 3999],
    'Images': [5, 2, 3, 4],
    'Description': ['Desc1', 'Desc2', 'Desc3', 'Desc4'],
    'PrimaryColor': ['Red', 'Blue', 'Green', 'Black']
}
train_data = pd.DataFrame(data)

# Selecting the columns of interest
heatmap_data = train_data[['Price', 'Images']]

# Calculating correlation
correlation_matrix = heatmap_data.corr()

# Plotting the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap between Price and Number of Images')
plt.show()
```



[38]:
```
# Most popular items
popular_items = train_data['ProdID'].value_counts().head(5)
popular_items.plot(kind='bar',color='red')
plt.title("Most Popular items")
```

[38]: Text(0.5, 1.0, 'Most Popular items')



0.2 Data cleaning and tags creation

[ ]:
```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS

nlp = spacy.load("en_core_web_sm")

def clean_and_extract_tags(text):
    doc = nlp(text.lower())
    tags = [token.text for token in doc if token.text.isalnum() and token.text␣
 ↪not in STOP_WORDS]
    return ', '.join(tags)

columns_to_extract_tags_from = ['Category', 'Brand', 'Description']

for column in columns_to_extract_tags_from:
    train_data[column] = train_data[column].apply(clean_and_extract_tags)
```

[ ]:
```
train_data['Tags'] = train_data[columns_to_extract_tags_from].apply(lambda row:␣
↪', '.join(row), axis=1)
```

0.3 Model Creation

[65]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load your dataset (modify this according to your data path)
df = pd.read_csv("myntra_products_catalog.csv")  # Adjust to your file location

# Assuming the 'category' column holds the labels for the dataset
labels = df['ProductName']  # Or use another column like 'subcategory'

# Encode labels to integers (since neural networks require numerical labels)
encoder = LabelEncoder()
labels = encoder.fit_transform(labels)

# Reshape labels to 2D for categorical cross-entropy (one-hot encoding)
from tensorflow.keras.utils import to_categorical
labels = to_categorical(labels)
```

[69]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, Input
from tensorflow.keras.optimizers import Adam

# Define the model
model = Sequential()

# Use the Input layer to specify the input shape
model.add(Input(shape=(64, 64, 3)))  # Input shape for RGB images (64x64)

# Add layers to the model
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout layer to prevent overfitting

# Output layer
model.add(Dense(labels.shape[1], activation='softmax'))  # Number of categories␣
↪in labels

# Compile the model
```

```
model.compile(optimizer=Adam(), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Print model summary
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_7 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten_2 (Flatten)	(None, 12544)	0
dense_3 (Dense)	(None, 128)	1,605,760
dropout_2 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 10761)	1,388,169

Total params: 3,013,321 (11.49 MB)

Trainable params: 3,013,321 (11.49 MB)

Non-trainable params: 0 (0.00 B)

[80]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Input
from tensorflow.keras.optimizers import Adam

# Build the CNN model
model = Sequential()

# Add an Input layer with the desired input shape (e.g., 64x64x3 for RGB images␣
 ↪of size 64x64)
```

```
model.add(Input(shape=(64, 64, 3)))  # Adjust input shape as per your dataset

# First convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolutional layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Third convolutional layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the 3D output to 1D
model.add(Flatten())

# Fully connected layer (dense layer)
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout layer to prevent overfitting

# Output layer (number of categories = 3 for this example)
model.add(Dense(3, activation='softmax'))  # Change '3' to the number of classes

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Summary of the model
model.summary()
```

```
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_11 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_12 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_12 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_13 (Conv2D)	(None, 12, 12, 128)	73,856
max_pooling2d_13 (MaxPooling2D)	(None, 6, 6, 128)	0

```
flatten_4 (Flatten)              (None, 4608)                  0

dense_7 (Dense)                  (None, 128)             589,952

dropout_4 (Dropout)              (None, 128)                   0

dense_8 (Dense)                  (None, 3)                   387
```

Total params: 683,587 (2.61 MB)

Trainable params: 683,587 (2.61 MB)

Non-trainable params: 0 (0.00 B)

[ ]: