SOC Midterm Report

Movie Recommendation System : 129

Krisha Biradar
24b3957

Week 1

In week 1, we were required to write a program to count how many movies belong to each genre and plot it with horizontal bars.

1. Creating DataFrames

- `pd.DataFrame(data)`: Create a DataFrame from dict, list of dicts, numpy arrays, etc.
- `pd.read_csv('file.csv')`: Read a CSV file into a DataFrame.

---

2. Viewing and Inspecting Data

- `df.head(n)`: View first *n* rows.
- `df.tail(n)`: View last *n* rows.
- `df.describe()`: Summary statistics for numeric columns.
- `df.shape`: Get (rows, columns).
- `df.columns`: Get column names.
- `df.index`: Get index information.
- `df.dtypes`: Data types of each column.

---

3. Handling Missing Data

- `df.isnull()`: Detects missing values.
- `df.notnull()`: Detects non-missing values.
- `df.dropna()`: Drop rows with missing values.
- `df.fillna(value)`: Fill missing values.

---

4. Sorting and Reordering

- `df.sort_values(by='col')`: Sort by column.
- `df.sort_index()`: Sort by index.
- `df.reset_index()`: Reset index to default.
- `df.set_index('col')`: Set a column as index.

---

5. Aggregation and Grouping

- `df.sum()`, `df.mean()`, `df.max()`, etc.: Aggregations.
- `df.groupby('col')`: Group data by column.
- `df.groupby('col').agg({'col2':'sum'})`: Custom aggregation.

---

6. Iteration

- ```
  for index, row in df.iterrows():
      print(row['column_name'])
  ```
- ```
  for row in df.itertuples():
      print(row.column_name)
  ```
- ```
  for col in df.columns:
      print(col, df[col].mean())
  ```
- ```
  i = 0
  while i < len(df):
      print(df.iloc[i]['column_name'])
      i += 1
  ```

---

7. Plotting with matplotlib

- `plt.figure(figsize=(8, 5))` : optional: set width and height
- `plt.plot(x, y, marker='o')` : line plot
- `plt.title("My First Plot")` : title
- `plt.xlabel("X-axis label")` : x-axis label

- plt.ylabel("Y-axis label") : y-axis label
- plt.grid(True) : optional: add grid
- plt.show() : display the plot

Line Plot	`plt.plot(x, y)`
Bar Plot	`plt.bar(x, y)`
Horizontal Bars	`plt.barh(x, y)`
Scatter Plot	`plt.scatter(x, y)`
Histogram	`plt.hist(data)`

Week 2

In week 2 we were required to merge 4 datasets and create a master data set which would be used for further analysis.

To filter the main dataset (master_dataset) so that it only contains rows for movies that have a valid mapping in links.csv (i.e. a valid 'tmdbId').

```
# Clean the key column in the filtering DataFrame
valid_keys = df_filter[df_filter['foreign_key'].notnull()]['foreign_key'].astype(int)

# Filter the main DataFrame using .isin()
filtered_df = main_df[main_df['primary_key'].isin(valid_keys)]
```

1. Selecting and Filtering Data

- `df['col']` or `df.col`: Access single column.
- `df[['col1', 'col2']]`: Access multiple columns.
- `df.loc[row_label, col_label]`: Access by label.
- `df.iloc[row_idx, col_idx]`: Access by integer position.
- `df[df['col'] > 5]`: Filter rows with condition.

---

2. Modifying Data

- `df['new_col'] = value`: Add new column.
- `df.rename(columns={'old':'new'})`: Rename columns.
- `df.drop(columns=['col'])`: Remove column(s).
- `df.drop(index=5)`: Remove row(s).
- `df.insert(loc, column, value)`: Insert column at a specific position.
- `df.replace(to_replace, value)`: Replace values.

3. Merging and Joining

- `pd.concat([df1, df2])`: Concatenate along a given axis.
- `pd.merge(df1, df2, on='key')`: Merge on key column.
- `df1.join(df2)`: Join using index or key (General Syntax)
- Inner Join (Only Matching Rows)

  ```
  merged_df = df1.merge(df2, on='id', how='inner')
  ```

  Keeps only rows with matching keys in both DataFrames. Drops unmatched rows. Useful when you need strict matches only.

- Left Join (Keep All From Left)

  ```
  merged_df = df1.merge(df2, on='id', how='left')
  ```

  Keeps all rows from `df1` (left). Rows from `df2` are added only where matching. Unmatched values from `df2` become `NaN`. Great for filtering like: “only include info from df2 if it exists.”

- Right Join (Keep All From Right)

  ```
  merged_df = df1.merge(df2, on='id', how='right')
  ```

  Keeps all rows from `df2` (right). Rows from `df1` are added only where matching. Often used if `df2` is your “main” dataset and you're adding info from `df1`.

- Outer Join (All Rows From Both)

  ```
  merged_df = df1.merge(df2, on='id', how='outer')
  ```

  Keeps all rows from both DataFrames. Missing values filled with `NaN`. Useful when you want a full picture, even with missing matches.

Week 3&4

Required to focus on pre-processing and vectorization which helps distill the summary of a movie into a structured format which can be compared and analyzed.

1. Tokenization

Tokenization is the process of breaking text into smaller pieces, called tokens.

- Makes text easier to process by machine learning models
- Lets you count words, calculate frequency, or build vectorizers
- Needed for tasks like search, translation, sentiment analysis, etc.

General Syntax :

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

text = "I love movies!"
tokens = word_tokenize(text)
print(tokens)  # ['I', 'love', 'movies', '!']
```

---

2. Lowercase Conversion

Lowercase conversion means changing all uppercase letters to lowercase in a string or a column.
This is often done in text preprocessing to ensure uniformity, since `"Movie"` and `"movie"` should be treated as the same word in most NLP tasks.

- To standardize text for analysis or machine learning.
- Prevent duplicates like "Action" and "action" from being treated as different.
- Helps with tokenization, vectorization, and search matching.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(lowercase=True)  # Default is True
```

---

3. Stopwords Removal

Stopword removal means deleting common, unimportant words (called stopwords) from your text data — like:

"is", "the", "and", "a", "in", "of", "to", etc.

These words appear often but add little meaning, so we remove them to focus on more useful or unique words when analyzing text.

- To reduce noise in data
- Improve the performance of:
  - Text classification
  - Clustering
  - Recommendation systems
- Helps CountVectorizer, TfidfVectorizer, etc. focus on important words

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(stop_words='english')

corpus = ["I love watching movies on weekends", "Movies are good"]
X = vectorizer.fit_transform(corpus)

print(vectorizer.get_feature_names_out())
# Output: ['good', 'love', 'movies', 'watching', 'weekends']
```

---

4. Stemming

Stemming is the process of reducing a word to its root or base form (called a stem) by chopping off suffixes.

Stemming does not always return real words, but it helps group similar words together during text analysis. (e.g. "happily" → "happili")

- Reduces dimensionality of text data
- Groups related words together (e.g. *"run", "running", "ran"* → "run")
- Useful for search, text classification, topic modeling

```
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

nltk.download('punkt')

text = "He was running and eating at same time. He has bad habits"
words = word_tokenize(text)

# Create stemmer
stemmer = PorterStemmer()

# Apply stemming
stemmed_words = [stemmer.stem(word) for word in words]

print(stemmed_words)
```

---

5. Lemmatization

Lemmatization is the process of converting a word to its base or dictionary form (called a lemma), using proper linguistic rules (like part of speech, grammar, etc.).

Unlike stemming, lemmatization returns real words.

- More accurate and readable than stemming
- Important for text classification, search, summarization, etc.
- Especially useful in NLP pipelines

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
import nltk

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

lemmatizer = WordNetLemmatizer()
text = "The children are running in the gardens"
tokens = word_tokenize(text)
```

```
lemmas = [lemmatizer.lemmatize(token) for token in tokens]
print(lemmas)
```

---

6. CountVectorizer

CountVectorizer is a feature extraction tool in scikit-learn that converts a collection of text documents into a matrix of token counts.

- To convert raw text into numbers for machine learning
- Useful for:
  - Text classification
  - Clustering
  - Recommendation systems
  - Sentiment analysis

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    "I love movies",
    "I watch movies",
    "Movies are great"
]

# Create the vectorizer
vectorizer = CountVectorizer()

# Learn the vocabulary and transform the data into a count matrix
X = vectorizer.fit_transform(corpus)

# To see the vocabulary
print(vectorizer.get_feature_names_out())

# To see the numeric matrix
print(X.toarray())
```

OUTPUT :

```
['are' 'great' 'love' 'movies' 'watch']
[[0 0 1 1 0]
 [0 0 0 1 1]
 [1 1 0 1 0]]
```

7. Cosine Similarity

Cosine Similarity is a way to measure how similar two vectors are, based on the angle between them — not their magnitude.

It's most commonly used to compare documents, especially in text analysis and recommendation systems.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity


corpus = [
    "I love movies",
    "I enjoy watching movies",
    "I hate horror films"
]

# Convert text to vectors
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

# Compute cosine similarity
cos_sim = cosine_similarity(X)

# Print similarity matrix
print(cos_sim)
```

OUTPUT:

```
[[1.        0.77       0.  ]
 [0.77       1.        0.2 ]
 [0.        0.2        1. ]]
```

In week 3's code we focus on preprocessing a movie dataset (`master_dataset.csv`) to:

- Extract meaningful features like `cast`, `keywords`, and `director`
- Convert complex string data (stored as lists of dictionaries) into usable Python objects

New concepts learnt :

`literal_eval` from `ast`:
Safely evaluates a string representation of a Python object (like list/dict) and converts it back to that type.

`apply(literal_eval)`:
These columns were stored as strings that *look like* lists of dictionaries. This line converts them into real lists of dictionaries for further processing.

`get_director`:
Loops through the `crew` list (which contains people and their jobs) to find the director's name.

`lambda x: [i['name'] for i in x]`:
Extracts just the `'name'` field from each dictionary in the list.
`x[:3]`:
Keeps only the top 3 cast members.
`isinstance(x, list)`:
Safeguard to ensure the value is a list before trying to loop over it.

`np.nan`
Inserts a missing value if not a float

In week 4's code the goal is to clean and reduce the dataset (`master_dataset_new.csv`) by:

- Removing irrelevant or unused columns
- Ensuring that only valid and usable data remains (like float-type popularity, proper director names)
- Preparing it for building a content-based recommender system