AI Documentation

1.

To read a CSV file from a directory that is two levels inside the `data` folder (which is at the same level as your `scripts` folder), you can use relative paths in Python. Here's how you can construct the correct path:

### Directory Structure:

```

project\_root/

├── scripts/

│ └── your\_script.py

└── data/

└── subfolder1/

└── subfolder2/

└── your\_file.csv

```

### Solution:

In your Python script (`your\_script.py`), you can use the following code to read the CSV file:

```python

import pandas as pd

import os

# Construct the path to the CSV file

csv\_path = os.path.join('..', 'data', 'subfolder1', 'subfolder2', 'your\_file.csv')

# Read the CSV file

df = pd.read\_csv(csv\_path)

# Display the dataframe

print(df.head())

```

### Explanation:

1. `os.path.join('..', 'data', 'subfolder1', 'subfolder2', 'your\_file.csv')`:

- `'..'` moves up one directory level (from `scripts` to `project\_root`).

- `'data'` enters the `data` folder.

- `'subfolder1'` and `'subfolder2'` navigate to the subfolders.

- `'your\_file.csv'` is the target file.

2. `pd.read\_csv(csv\_path)` reads the CSV file into a pandas DataFrame.

### Alternative (using pathlib for better path handling):

If you're using Python 3.4+, you can use `pathlib` for more readable path handling:

```python

from pathlib import Path

import pandas as pd

# Construct the path to the CSV file

csv\_path = Path('..') / 'data' / 'subfolder1' / 'subfolder2' / 'your\_file.csv'

# Read the CSV file

df = pd.read\_csv(csv\_path)

# Display the dataframe

print(df.head())

```

### Notes:

- Ensure that the file `your\_file.csv` exists at the specified path.

- If you're still facing issues, you can check the absolute path being used by printing `os.path.abspath(csv\_path)` and verifying it points to the correct location.

Let me know if you need further assistance!

2.

i am working on bi grams and i have a data frame of bi grams and i want to filter some bi grams which contain some words of my choice

Filtering Bi-grams in a DataFrame

To filter bi-grams in a DataFrame that contain specific words of your choice, you can use pandas in Python. Here are several approaches depending on your exact needs:

Basic Approach

python

Copy

Download

import pandas as pd

# Sample DataFrame

data = {'bi\_gram': [('apple', 'pie'), ('banana', 'bread'), ('apple', 'juice'),

('orange', 'juice'), ('grape', 'fruit')]}

df = pd.DataFrame(data)

# Words you want to filter by

target\_words = ['apple', 'juice']

# Filter bi-grams that contain ANY of the target words

filtered\_df = df[df['bi\_gram'].apply(lambda x: any(word in x for word in target\_words))]

print(filtered\_df)

Alternative Methods

1. **Filter bi-grams where BOTH words are in your target list:**

python

Copy

Download

filtered\_df = df[df['bi\_gram'].apply(lambda x: all(word in target\_words for word in x))]

1. **Filter bi-grams where FIRST word is in your target list:**

python

Copy

Download

filtered\_df = df[df['bi\_gram'].apply(lambda x: x[0] in target\_words)]

1. **Filter bi-grams where SECOND word is in your target list:**

python

Copy

Download

filtered\_df = df[df['bi\_gram'].apply(lambda x: x[1] in target\_words)]

1. **Using set operations for efficiency with large word lists:**

python

Copy

Download

target\_set = set(target\_words)

filtered\_df = df[df['bi\_gram'].apply(lambda x: not set(x).isdisjoint(target\_set))]

Example with String-Formatted Bi-grams

If your bi-grams are stored as strings (like "apple pie") rather than tuples:

python

Copy

Download

# If bi-grams are strings like "apple pie"

df['bi\_gram\_str'] = df['bi\_gram'].apply(' '.join)

# Filter using string operations

filtered\_df = df[df['bi\_gram\_str'].apply(lambda x: any(word in x.split() for word in target\_words))]

Would you like me to adapt any of these solutions to your specific DataFrame structure or filtering requirements?

suppose i am working on articles about gaza and i dont want irrelevant bi grams like "to the" instead i want bi grams with words like "war" " strike" what should i do

Filtering Relevant Bi-grams for Gaza Articles

To filter your bi-grams to keep only relevant terms related to Gaza (like "war", "strike", "conflict") while excluding common/irrelevant phrases ("to the", "in the"), here's a comprehensive approach:

Step 1: Define Your Keyword Lists

python

Copy

Download

# Words that make bi-grams relevant (keep these)

relevant\_words = [

'war', 'strike', 'conflict', 'gaza', 'hamas', 'israel', 'palestine',

'attack', 'bomb', 'raid', 'violence', 'ceasefire', 'hostage',

'border', 'refugee', 'aid', 'siege', 'military', 'terror'

]

# Common bi-grams to exclude (filter these out)

irrelevant\_bi\_grams = {

('to', 'the'), ('in', 'the'), ('of', 'the'), ('it', 'is'),

('as', 'a'), ('at', 'the'), ('for', 'the'), ('on', 'the'),

('and', 'the'), ('to', 'be')

}

Step 2: Filter Your DataFrame

python

Copy

Download

import pandas as pd

# Assuming your DataFrame has a 'bi\_gram' column with tuples

df = pd.DataFrame({

'bi\_gram': [('gaza', 'war'), ('to', 'the'), ('israeli', 'strike'),

('in', 'the'), ('palestinian', 'casualties'), ('ceasefire', 'negotiations')]

})

# Filter function

def is\_relevant(bi\_gram):

# First exclude common irrelevant bi-grams

if bi\_gram in irrelevant\_bi\_grams:

return False

# Then check if either word is in relevant words

return any(word in relevant\_words for word in bi\_gram)

# Apply the filter

filtered\_df = df[df['bi\_gram'].apply(is\_relevant)]

print(filtered\_df)

Step 3: Alternative Approach with String Matching

If you prefer working with strings:

python

Copy

Download

# Convert bi-grams to strings if they aren't already

df['bi\_gram\_str'] = df['bi\_gram'].apply(' '.join)

# Define irrelevant patterns (regex)

irrelevant\_patterns = r'\b(to|in|of|it|as|at|for|on|and)\s+(the|a|be|is)\b'

# Filter

filtered\_df = df[

~df['bi\_gram\_str'].str.contains(irrelevant\_patterns) &

df['bi\_gram\_str'].str.contains('|'.join(relevant\_words), case=False)

]

Step 4: Enhance with Proper Nouns

For Gaza coverage, you might want to automatically include:

* All bi-grams containing place names ("gaza", "jerusalem")
* Organization names ("hamas", "idf")
* People names ("netanyahu", "haniyeh")

python

Copy

Download

proper\_nouns = {'gaza', 'hamas', 'israel', 'palestine', 'idf', 'west bank',

'netanyahu', 'haniyeh', 'jerusalem', 'tel aviv'}

def is\_relevant\_enhanced(bi\_gram):

if bi\_gram in irrelevant\_bi\_grams:

return False

return (any(word in relevant\_words for word in bi\_gram) or

any(word in proper\_nouns for word in bi\_gram))

filtered\_df = df[df['bi\_gram'].apply(is\_relevant\_enhanced)]

Tips for Better Results:

1. **Case sensitivity**: Use .lower() if your text isn't normalized
2. **Stemming/Lemmatization**: Consider processing words to root forms ("strikes" → "strike")
3. **Dynamic expansion**: Automatically add frequent bi-grams to relevant/irrelevant lists after manual review

Would you like me to modify any part of this approach for your specific dataset?