ImportError in Python 3.10

```
Traceback (most recent call last):
  File "C:\Users\Amit\Desktop\Krish Naik\MLOPS\My
Things\PROJECTS\NetworkSecurity\push_data.py", line 2, in <module>
    from pymongo.mongo_client import MongoClient
  File "C:\Users\Amit\Desktop\Krish Naik\MLOPS\My
Things\PROJECTS\NetworkSecurity\network\lib\site-packages\pymongo\__init__.py", line 77, in
<module>
    from pymongo.collection import ReturnDocument
  File "C:\Users\Amit\Desktop\Krish Naik\MLOPS\My
Things\PROJECTS\NetworkSecurity\network\lib\site-packages\pymongo\collection.py", line 21, in
<module>
    from bson.code import Code
  File "C:\Users\Amit\Desktop\Krish Naik\MLOPS\My
Things\PROJECTS\NetworkSecurity\network\lib\site-packages\bson\__init__.py", line 83, in <module>
    from bson.codec_options import (
  File "C:\Users\Amit\Desktop\Krish Naik\MLOPS\My
Things\PROJECTS\NetworkSecurity\network\lib\site-packages\bson\codec_options.py", line 19, in
<module>
    from collections import MutableMapping, namedtuple
ImportError: cannot import name 'MutableMapping' from 'collections'
(C:\Users\Amit\AppData\Local\Programs\Python\Python310\lib\collections\__init__.py)
```

The error you're encountering indicates that the `MutableMapping` class cannot be imported from the `collections` module. In Python 3.10+, `MutableMapping` has been moved from `collections` to `collections.abc`.

To resolve this, you can update your code or the dependent library to use `collections.abc.MutableMapping` instead of `collections.MutableMapping`.

Here's how you can modify your code (if you're able to edit it):

```python
from collections.abc import MutableMapping
```

Alternatively, if the error occurs within the installed libraries (`pymongo` or `bson`), it suggests that the libraries need to be updated to versions that support Python 3.10 and later.

To update the libraries, run:

```
bash


  pip install --upgrade pymongo bson
```

If the error persists after upgrading, it may be worth checking if the library is compatible with Python 3.10 and considering downgrading to Python 3.9 for compatibility.

```
from pymongo.mongo_client import MongoClient

uri = "mongodb+srv://amit_0805:Amit12345@cluster0.acjas.mongodb.net/?
retryWrites=true&w=majority&appName=Cluster0"

# Create a new client and connect to the server
client = MongoClient(uri)

# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)

trhsi is the code
```

The error you're encountering might be due to the `MutableMapping` import issue in the MongoDB driver (`pymongo`). Here's a step-by-step solution to help you resolve this issue and ensure that your code works correctly:

1. **Fix the `MutableMapping` Import Issue**:
    - As mentioned earlier, in Python 3.10 and later, `MutableMapping` has been moved to `collections.abc`. Ensure that the libraries you are using are up-to-date and compatible with Python 3.10.
2. **Update MongoDB Python Driver (`pymongo`)**:
    - If the error is coming from the `pymongo` package or its dependencies, you can try updating `pymongo` and `bson` to versions that are compatible with Python 3.10.

    Run the following command to update:

    ```
    bash


      pip install --upgrade pymongo
      pip install --upgrade bson
    ```

3. **Check MongoDB Connection**: Your code should work if the dependencies are properly installed and your MongoDB URI is correct. If everything is up-to-date, the following code should connect successfully:

```
python


  from pymongo.mongo_client import MongoClient

  uri = "mongodb+srv://amit_0805:Amit12345@cluster0.acjas.mongodb.net/?retryWrites=true&w=majo
  rity&appName=Cluster0"

  # Create a new client and connect to the server
  client = MongoClient(uri)

  # Send a ping to confirm a successful connection
  try:
      client.admin.command('ping')
      print("Pinged your deployment. You successfully connected to MongoDB!")
  except Exception as e:
      print(f"An error occurred: {e}")
```

If this doesn't resolve the issue, you can check the following:

- **Python version**: Ensure you are using a version of `pymongo` compatible with your Python version.
- **Virtual Environment**: If you're using a virtual environment, ensure all dependencies are installed within the correct environment.

Let me know if you need further assistance!

list(json.loads(data.T.to_json()).values())

Give an example how it convers a sample csv to json in super simple words

Sure! Let's break it down with an example, keeping it super simple.

Goal:

You want to take data from a CSV file, convert it to JSON format, and then extract the values as a list using `json.loads` and `data.T.to_json()`.

Step-by-step:

1. **CSV Data Example**: Imagine we have a simple CSV file called `sample.csv` with the following content:

```
sql


  Name,Age,City
  John,28,New York
```

```
Alice,30,Los Angeles
Bob,25,Chicago
```

2. **Step 1: Load the CSV Data**: First, you would load the CSV file into a pandas DataFrame:

```
python


import pandas as pd

# Load the CSV file
data = pd.read_csv("sample.csv")
```

After loading, the `data` DataFrame looks like this:

```
markdown


    Name  Age         City
0    John   28     New York
1   Alice   30  Los Angeles
2     Bob   25      Chicago
```

3. **Step 2: Convert DataFrame to JSON**: Use the `to_json()` method on the DataFrame to convert it into a JSON-like string. The `.T` transposes the DataFrame (rows become columns and vice versa).

```
python


json_data = data.T.to_json()
```

Now, `json_data` will look like this (it's a string that represents a JSON object):

```
json


{
    "0": {"Name": "John", "Age": 28, "City": "New York"},
    "1": {"Name": "Alice", "Age": 30, "City": "Los Angeles"},
    "2": {"Name": "Bob", "Age": 25, "City": "Chicago"}
}
```

4. **Step 3: Parse JSON and Extract Values**: Now, you use `json.loads()` to parse the JSON string into a Python dictionary and then call `.values()` to get the values (which are the rows).

```
python


import json

# Parse the JSON string
parsed_data = json.loads(json_data)

# Get the values as a list
values = list(parsed_data.values())
print(values)
```

This will output:

```
python


  [
      {"Name": "John", "Age": 28, "City": "New York"},
      {"Name": "Alice", "Age": 30, "City": "Los Angeles"},
      {"Name": "Bob", "Age": 25, "City": "Chicago"}
  ]
```

In Simple Words:

- We start with a CSV file (table format).
- We load it into a pandas DataFrame.
- We convert the DataFrame into JSON format (transpose it, then convert).
- Finally, we parse the JSON and extract the rows as a list of dictionaries.

Let me know if you need more clarity on any part!

```
def insert_data_mongodb(self, records, database, collection):
        try:
            self.database = database
            self.collection = collection
            self.records = records

            # Create a MongoClient instance
            self.mongo_client = pymongo.MongoClient(MONGO_DB_URL)
            self.database = self.mongo_client[self.database]

            self.collection=self.database[self.collection]
            self.collection.insert_many(self.records)
            return(len(self.records))
        except Exception as e:
            raise NetworkSecurityException(e, sys)
```

Explain whats happening in this function in super simple words

Let's break it down step by step in super simple words:

Purpose of the Function:

This function is designed to insert records (data) into a specified collection in a MongoDB database.

Steps:

1. **Setting Up Variables**:

```
python


  self.database = database
  self.collection = collection
  self.records = records
```

- `self.database`: This is the name of the database where we want to insert data.
- `self.collection`: This is the collection (a table-like structure) in the MongoDB database where we will store the data.
- `self.records`: These are the actual data records (usually a list of dictionaries) that we want to insert into the MongoDB collection.

2. **Creating a MongoClient**:

```
python


  self.mongo_client = pymongo.MongoClient(MONGO_DB_URL)
```

- This line creates a connection to the MongoDB server using the `MongoClient` class from the `pymongo` library.
- `MONGO_DB_URL` is a placeholder for the MongoDB connection string, which is the URL used to connect to your MongoDB instance (it could include credentials and database server details).

3. **Connecting to the Database**:

```
python


  self.database = self.mongo_client[self.database]
```

- Now that we have the MongoDB client (`mongo_client`), we access the specific database we want by using `self.database`, which is set earlier.

4. **Accessing the Collection**:

```
python


  self.collection = self.database[self.collection]
```

- Here, we access the collection (a specific "table" inside the database) where we want to insert the data.

5. **Inserting the Records**:

```
python


  self.collection.insert_many(self.records)
```

- This is where the magic happens: the `insert_many()` method inserts multiple records (in the form of dictionaries) into the collection.
- `self.records` is the list of records you want to insert into the collection.

6. **Returning the Number of Inserted Records**:

```
python


  return(len(self.records))
```

   - This returns the number of records that were inserted into the collection. It's just the length of the `self.records` list.

7. **Handling Errors (Exception)**:

```
python


  except Exception as e:
      raise NetworkSecurityException(e, sys)
```

   - If anything goes wrong during the process (like connection failure or data issue), an exception is raised.
   - This is caught by the `except` block, and a custom error (`NetworkSecurityException`) is raised with the error message and system information.

In Simple Words:

- The function connects to a MongoDB database using a connection string.
- It accesses the specified database and collection.
- It inserts the given records (data) into the MongoDB collection.
- It returns how many records were successfully inserted.
- If anything goes wrong, it raises an error with the details.

Let me know if any part of this needs further clarification!

a simple example

Let's go through a **simple example** with the `insert_data_mongodb` function.

Example Scenario:

- You have a MongoDB database called `TestDB`.
- You have a collection within that database called `users`.
- You want to insert some records (data) into that `users` collection.

Sample Data:

You want to insert the following records (list of dictionaries):

```
python


  records = [
      {"name": "John", "age": 28, "city": "New York"},
      {"name": "Alice", "age": 30, "city": "Los Angeles"},
      {"name": "Bob", "age": 25, "city": "Chicago"}
  ]
```

Code Example:

```
python


  import pymongo

  # MongoDB URL (this would be your connection string)
  MONGO_DB_URL = "mongodb+srv://username:password@cluster0.mongodb.net/test"

  class NetworkSecurityException(Exception):
      pass

  class MongoDBHandler:
      def insert_data_mongodb(self, records, database, collection):
          try:
              self.database = database
              self.collection = collection
              self.records = records

              # Create a MongoClient instance
              self.mongo_client = pymongo.MongoClient(MONGO_DB_URL)
              self.database = self.mongo_client[self.database]

              self.collection = self.database[self.collection]
              self.collection.insert_many(self.records)
              return len(self.records)  # Return the number of records inserted
          except Exception as e:
              raise NetworkSecurityException(f"Error: {e}", sys)

  # Create an instance of MongoDBHandler
  db_handler = MongoDBHandler()

  # Data you want to insert
  records = [
      {"name": "John", "age": 28, "city": "New York"},
      {"name": "Alice", "age": 30, "city": "Los Angeles"},
      {"name": "Bob", "age": 25, "city": "Chicago"}
  ]

  # Call the insert function
  result = db_handler.insert_data_mongodb(records, "TestDB", "users")

  # Output the result (number of records inserted)
  print(f"{result} records inserted into the 'users' collection.")
```

Explanation of the Code:

1. **MongoDB URL**: You need to replace the `MONGO_DB_URL` with the actual MongoDB connection string (you can get this from your MongoDB instance).
2. **MongoDBHandler Class**: We define the class `MongoDBHandler` that has the `insert_data_mongodb` function, which inserts records into the specified MongoDB database and collection.
3. **Insert Data**:

- The `insert_data_mongodb` method takes three arguments:
  - `records`: The list of data to be inserted.
  - `database`: The MongoDB database (here, it's `"TestDB"`).
  - `collection`: The MongoDB collection where the data should go (here, it's `"users"`).

4. **Result**:
   - After inserting, the function returns the number of records inserted.
   - We print out how many records were inserted into the `users` collection.

Example Output:

```
plaintext


  3 records inserted into the 'users' collection.
```

This simple example demonstrates how to use the `insert_data_mongodb` function to insert a few records into a MongoDB database. Let me know if you need more help!