Final Group Project

Introduction

For your group project, you will write/integrate various scripts and functionalities. The final project comprises five main parts. Each part is explained in its respective section, and each part will have its own code scripts that need to be prepared before moving on to the next step. A successfully completed project means you have completed all the parts, and the pre-processed and tagged news articles can be viewed through some medium, such as a webpage.

- Part I

1. Create a New Script called *FullRSSList_1_2.py* or something similar and Import *posts* from *RssArticles_1.py*

What you need to do

1. **Review the code from** *RssArticles_1.py* and note how *posts* is created.
2. **Create a new script** (you can call it something like *FullRSSList_1_2.py* or any name your group decides) where you import *posts* from *RssArticles_1.py.*

Step-by-step Instructions

1. **Set up your file structure** so that your *FullRSSList_1_2.py* (the new script) and *RssArticles_1.py.* are in the same folder (or ensure correct relative imports).
2. **Run the new script** to ensure you see the expected output (e.g., the length of MyTheFinalList and the first item).¨

Potential Output:

[[...]...['Liberalerna: Terrorstämpla Foxtrot', 'Liberalerna vill att EU terrorstämplar det kriminella nätverket Foxtrot. Det säger partiledare Johan Pehrson till journalister.', 'https://www.svt.se/nyheter/inrikes/liberalerna-terrorstampla-foxtrot', '2025-01-30 13:26:34']]

240

Tip: If you encounter import errors, verify that your Python path is configured correctly and that both scripts (FullRSSList_1_2.py and your new script) are in the same folder or a properly referenced module/package.

- The Script FullRSSList_1_2.py with pseudo code is available.

- Part II

2. Modify Your MLModelMLC_3.py to Allow Imports for the Final Project

What you need to do

- You need to import **categories, train, x_train, vectorizer, best_clf_pipeline** from your MLModelMLC_3.py script.
- The script should be **modified** in such a way that it **exposes** or **returns** these variables (or simply avoids code blocks that prevent them from being accessible).

Step-by-step Instructions

1. **Locate your best-performing model** (the one you used to achieve the highest accuracy in your second assignment). This is likely in your MLModelMLC_3.py script.
2. At the **end of your MLModelMLC_3.py** or after you have defined categories, x_train, vectorizer, and best_clf_pipeline, **add export lines** similar to the following (see code):

Note on second assignment: In your second assignment, you likely tested multiple models or parameter grids. For this final project, **use the model with the highest accuracy**. If needed, you can rename or reorganize your code to clarify that you are using the "best model."

- Part III

3. Create a New Script Called MLModelReturns_4.py

In this script, you will:

- Import the **final list** (MyTheFinalList) from FullRSSList_1_2.py.
- Import the **trained model** (best_clf_pipeline) and supporting objects (categories, vectorizer, etc.) from MLModelMLC_3.py.
- Use your model to make predictions on the new RSS feed articles.

Step-by-step Instructions

1. **Set up imports** in MLModelReturns_4.py. For example:
2. from FullRSSList_1_2 import MyTheFinalList
3. from MLModelMLC_3 import categories, x_train, vectorizer, best_clf_pipeline

4. from RssFeedNewArticle_2 import printdepositlist # If you need to combine summary + title text
5. **Preprocess the new articles** as needed:
    - You might already have them preprocessed in printdepositlist, which contains titles + summaries combined.
    - Confirm that MyTheFinalList is the final structured data (title, summary, link, etc.).
6. **Vectorize your new articles** using the **same** vectorizer from MLModelMLC_3.py:
7. **Combine** the resulting categories with the structured data (MyTheFinalList).
8. **Validate** (optional): Use JSON schema or other checks to ensure the final dictionary is consistent (as shown in the sample code).
9. **Print** or **return** this final dictionary (often stored in a variable like validDict) from MLModelReturns_4.py.

- **You can find the pseudo-code in the group project folder.**

- Part IV

4. Create a New Script Called DbTransfer_5.py

This script will **transfer** the output from MLModelReturns_4.py into your chosen database.

Step-by-step Instructions

1. **Import the final dictionary** (e.g., validDict or whatever you named it) from MLModelReturns_4.py:
2. from MLModelReturns_4 import validDict
3. **Establish a database connection**. You can find **pseudo-code** for a MySQL connection in the group project folder.
4. **Run DbTransfer_5.py** to confirm that your data is inserted into the database.
    - Check your DB table (e.g., via a MySQL client) to verify new rows.

- Part V

5. Display the Results

After you have inserted the data into your database, you will **display** those results on a platform of your choosing. Some popular options:

1. **Flask (Python micro-framework)**

- Create a simple Flask app. In a new Python file (e.g., app.py), connect to your database, retrieve the news items, and render them in an HTML template.

2. **Streamlit**
   - Use Streamlit's straightforward syntax to quickly display your database content.
   - Example:

```
import streamlit as st
import mysql.connector
def fetch_data():
 cnxn = mysql.connector.connect(host="...", user="...", password="...", database="...")
cursor = cnxn.cursor(dictionary=True)
cursor.execute("SELECT * FROM news")
rows = cursor.fetchall()
cursor.close()
cnxn.close()
return rows
def main():
st.title("News Articles")
data = fetch_data()
for row in data:
st.subheader(row['title'])
st.write(row['summary'])
st.write("Published:", row['published'])
st.write("Topics:", row['topic'])
st.write("---")
if __name__ == "__main__":
main()
```

3. **Django**
   - Set up a simple Django project, configure your models (with columns matching news table), run migrations, and then retrieve and display articles in a template.

At minimum, your **final step** must show the DB data on **at least** one chosen platform.

Grading Requirements

Grade Pass (G)

1. **Functional End-to-End Pipeline**
    - Your pipeline must:
        1. **Fetch and Clean** RSS feeds (title, summary, link, published date).
        2. **Train/Import and Apply** a text classification model (e.g., Logistic Regression) to label the articles.
        3. **Store** the labeled articles in a database table with minimal errors (title, summary, link, published, category).
2. **Basic Data Analysis**
    - Perform **at least one** simple analysis on the RSS data or classification results:
        - For example, **count** how many articles fall under each predicted category and print or plot a simple bar chart.
        - Another option: show **basic statistics** (e.g., earliest and latest publication date, how many articles total).
    - Present these results in **tabular or basic chart** form (e.g., using pandas `.value_counts()` or a simple matplotlib bar chart).
3. **Basic Web (or Console) Display of the Articles**
    - Provide a simple way for a user (or yourself) to **view** the final articles and their classifications:
        - **Option A**: A minimal website (Flask, Streamlit, Django, etc.) that lists the articles (title, date, category) and **basic sorting** (e.g., newest to oldest).
        - **Option B**: If web is too large a step, a **console-based** or **Jupyter-based** table output is acceptable—but must at least allow a sorted or filtered view (e.g., by date).

Note: For Pass (G), the **analysis** can be quite basic—just enough to show you've looked at the data numerically or graphically.

Grade Pass with Distinction (VG)

Must fulfill **all** requirements for **Grade Pass (G) plus at least two** enhancements from the list below. You can pick the ones you find most suitable, or propose your own **equivalent** additions:

1. **Extended Data Analysis**
    - Example enhancements:
        - **Time-Based Insights**: Show how many articles fall under each category **by day** or **by week** (line plot, stacked bar, etc.).
        - **Common Words or Word Cloud**: Identify the **most frequent words** (excluding stopwords) in titles/summaries, present a top-10 word frequency table or a word cloud.

        - **Additional Accuracy/Threshold Exploration**: Compare **two or more** threshold values (e.g., 0.3 vs. 0.5) for the classifier; discuss the trade-offs in precision vs. recall.
2. **Enhanced Web Display (if a website is created)**
    - At least **one** extra feature that makes the user interface more data-analyst-friendly:
        - *Examples:*
            - **Filter or Sort** by multiple parameters (e.g., category + publication date).
            - **Basic Search**: Let users enter a keyword to see matching articles.
            - **Better UI/UX**: Some styling or layout that organizes articles more clearly.
3. **Clear Conclusions & Insights**
    - Provide **brief written commentary** (in markdown, Jupyter notebook cells, or a readme) explaining what your analysis shows:
        - Mention any **patterns** (e.g., "We see the majority of articles are from the last 7 days" or "Category A is more prevalent than Category B").
        - Suggest **possible improvements** or **next steps** (e.g., collecting more data, refining model categories).

Overall Goal: Show a **deeper** analytical or visual exploration of your data, and/or present it in a more **user-friendly** manner.

Summary

- **Pass (G)**: Minimal but functional data pipeline, a simple analysis (e.g., category counts), and a basic display of results (via website or structured console/Jupyter output).
- **Pass with Distinction (VG)**: Demonstrate additional analytical depth, possibly with at least one advanced feature in the display (filtering, searching), and at least one more comprehensive data exploration or visualization.

Best of luck and enjoy integrating both **data analysis** and a **basic user display** into your final assignment!

Final Notes

- **File Organization** is important. Keep track of where each .py file resides and how they import each other.
- **Version Control**: If you are working in a group, use Git or another version-control system to merge changes efficiently.
- **Testing**: Test each step **individually** before moving to the next. This will help isolate issues.
- **Documentation**: Add comments and docstrings to clarify your code for future reference.

- Feel free to **tailor** each script to match your existing code, naming conventions, or specific data-handling logic.

Good luck with your final project!