Bank_Project

February 15, 2024

Acquiring and Processing Information on the World's Largest Banks

0.0.1 Project Scenario:

You have been hired as a data engineer by research organization. Your boss has asked you to create a code that can be used to compile the list of the top 10 largest banks in the world ranked by market capitalization in billion USD. Further, the data needs to be transformed and stored in GBP, EUR and INR as well, in accordance with the exchange rate information that has been made available to you as a CSV file. The processed information table is to be saved locally in a CSV format and as a database table.

Your job is to create an automated system to generate this information so that the same can be executed in every financial quarter to prepare the report.

Particulars of the code to be made have been shared below.

0.0.2 ETL

[1]:
```
import pandas as pd
import datetime as datetime
import numpy as numpy
import sqlite3
import requests
import bs4
import logging
import  xml
from bs4 import BeautifulSoup
```

[2]:
```
import datetime

# Specify the full path for consistency
log_file_path = '/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_ENGINEERING/code_log.txt'
```

0.0.3 LOG FILE

<!–"'This function logs the mentioned message of a given stage of the #code execution to a log file. Function returns nothing"'

[ ]:
```python
import datetime

# Specify the full path for consistency
log_file_path = '/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_ENGINEERING/code_log.txt'

def log_progress(message):
    """Logs a message with a timestamp to a specified log file in the new␣
 ↪format."""
    # Use the full path for the log file
    with open(log_file_path, 'a') as log_file:
        # Format the timestamp
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        # Write the formatted log entry using the new format with " : "
        log_file.write(f"{timestamp} : {message}\n")

# Write a test log entry
log_progress("This is a test log entry.")

# Try reading back the log to verify
with open(log_file_path, 'r') as file:
    print(file.read())
```

0.0.4 EXTRACT

<!–def extract(url, table_attribs): ”‘This function aims to extract the required information from the website and save it to a data frame. The function returns the data frame for further processing.”’ return df

[ ]:
```python
#EXTRACT
import requests
from bs4 import BeautifulSoup
# Define the URL
url = 'https://web.archive.org/web/20230908091635/https://en.wikipedia.org/wiki/
 ↪List_of_largest_banks'

# Define the extract function
def extract(url):
    """
    This function aims to extract the required information from the website and␣
 ↪save it to a data frame.
    The function returns the data frame for further processing.
    """
    # Send a GET request to the URL
    response = requests.get(url)

    # Check if the request was successful
```

```
    if response.status_code == 200:
        # Use pandas to read tables directly from the HTML content
        dfs = pd.read_html(response.content)

        # Assuming the first table is the one we want
        # Log progress
        log_progress("Data extracted successfully from the webpage.")
        return dfs
    else:
        # Log an error if the page could not be retrieved
        log_progress(f"Failed to retrieve data from {url}. Status code:␣
 ↪{response.status_code}")
        return []

# Call the extract function with the URL
dfs = extract(url)

# Check if any tables were returned
if dfs:
    # Check the first few rows of each table to identify the one you need
    for i, df in enumerate(dfs):
        print(f"Showing Table {i}")
        print(df.head())
        print("\n")


    # Log after extraction
    log_progress("Data extraction complete. Initiating Transformation process")

else:
    print("No tables were found on the webpage.")

# Assuming extract(url) returns a list of DataFrames
dfs = extract(url)

# Check the first few rows of each table to identify the one you need
for i, df in enumerate(dfs):
    print(f"Table {i}:")
    print(df.head())
    print("\n")
```

0.0.5 TRANSFORM

<!–def transform(df, csv_path): ”‘This function accesses the CSV file for exchange rate information, and adds three columns to the data frame, each containing the transformed version of Market Cap column to respective currencies”’ return df

[95]:
```
import pandas as pd

def transform(df, exchange_rates):
    """
    This function converts the 'Market cap(US$ billion)' column to GBP, EUR,␣
 ↪and INR
    using the provided exchange rates.
    """
    # Convert the market cap to the new currencies and add new columns to df
    df['Market_cap_GBP_Billion'] = round(df['Market cap(US$ billion)'] *␣
 ↪exchange_rates['GBP'], 2)
    df['Market_cap_EUR_Billion'] = round(df['Market cap(US$ billion)'] *␣
 ↪exchange_rates['EUR'], 2)
    df['Market_cap_INR_Billion'] = round(df['Market cap(US$ billion)'] *␣
 ↪exchange_rates['INR'], 2)

    return df

# Assuming dfs is a list of DataFrames returned by the extract function
dfs = extract('https://web.archive.org/web/20230908091635/https://en.wikipedia.
 ↪org/wiki/List_of_largest_banks')
correct_index = 0  # Adjust this based on which DataFrame in the list you need
df = dfs[correct_index]

# Load the exchange rates from CSV into a dictionary
csv_path = '/Users/mariomartinez/Desktop/Data_Engineering/FINAL_DATA_EGINEERING/
 ↪exchange_rate.csv'
exchange_rates_df = pd.read_csv(csv_path)
exchange_rates = exchange_rates_df.set_index('Currency')['Rate'].to_dict()

# Transform the DataFrame using the exchange rates
transformed_df = transform(df, exchange_rates)

# Display the transformed DataFrame
print(transformed_df.head())
```

```
   Rank                                Bank name  Market cap(US$ billion)  \
0     1                           JPMorgan Chase                   432.92
1     2                           Bank of America                  231.52
2     3  Industrial and Commercial Bank of China                   194.56
3     4               Agricultural Bank of China                   160.68
4     5                                 HDFC Bank                  157.91

   Market_cap_GBP_Billion  Market_cap_EUR_Billion  Market_cap_INR_Billion
0                  346.34                  402.62                35910.71
1                  185.22                  215.31                19204.58
2                  155.65                  180.94                16138.75
```

```
3                  128.54                   149.43                 13328.41
4                  126.33                   146.86                 13098.63
```

[130]:
```
#change name so i can run query Table as Largest_Banks
largest_banks=df
```

[133]:
```
largest_banks
```

[133]:
```
   Rank                            Bank name  Market cap(US$ billion)  \
0     1                         JPMorgan Chase                   432.92
1     2                        Bank of America                   231.52
2     3  Industrial and Commercial Bank of China                   194.56
3     4           Agricultural Bank of China                   160.68
4     5                               HDFC Bank                   157.91
5     6                           Wells Fargo                   155.87
6     7                     HSBC Holdings PLC                   148.90
7     8                        Morgan Stanley                   140.83
8     9              China Construction Bank                   139.82
9    10                         Bank of China                   136.81

   Market_cap_GBP_Billion  Market_cap_EUR_Billion  Market_cap_INR_Billion  \
0                  346.34                  402.62                35910.71
1                  185.22                  215.31                19204.58
2                  155.65                  180.94                16138.75
3                  128.54                  149.43                13328.41
4                  126.33                  146.86                13098.63
5                  124.70                  144.96                12929.42
6                  119.12                  138.48                12351.26
7                  112.66                  130.97                11681.85
8                  111.86                  130.03                11598.07
9                  109.45                  127.23                11348.39

   MC_EUR_Billion
0          402.62
1          215.31
2          180.94
3          149.43
4          146.86
5          144.96
6          138.48
7          130.97
8          130.03
9          127.23
```

0.0.6 Quiz question prompt:

5th largest bank Experiment with the statement provided for adding the transformed columns to the dataframe. There will be a question on this in the quiz.

Print the contents of df[‘MC_EUR_Billion’][4], which is the market capitalization of the 5th largest bank in billion EUR. Note this value, as it will be the answer to a question in the final quiz. <!– # Assuming the DataFrame ‘df’ and ‘exchange_rates’ dictionary are already defined # with ‘df’ having the column ‘Market cap (US$ billion)’ and ‘exchange_rates’ containing the EUR exchange rate.

[97]:
```
# let's define a sample exchange rate from our exchange_rate.csv
exchange_rates = {'EUR': 0.93}  # this is an  the actual exchange rate

# Perform the conversion from USD to EUR
df['MC_EUR_Billion'] = df['Market cap(US$ billion)'] * exchange_rates['EUR']

# Assuming you want to round the results to two decimal places
df['MC_EUR_Billion'] = df['MC_EUR_Billion'].round(2)

# Now, print the market capitalization of the 5th largest bank in billion EUR
fifth_largest_bank_mc_eur = df['MC_EUR_Billion'][4]

print(fifth_largest_bank_mc_eur)
```

```
146.86
```

[ ]:
```

```

0.0.7 load_to_csv

<!–def load_to_csv(df, output_path): ”‘This function saves the final data frame as a CSV file in the provided path. Function returns nothing.”’

[98]:
```
# Save the DataFrame to a CSV file
df.to_csv(csv_file_path, index=False)
print(f'DataFrame saved to {csv_file_path}')

# Log after saving to CSV
log_progress("Data saved to CSV file")

# Define the path where you want to save the CSV file
csv_file_path = '/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_EGINEERING/Largest_banks_data.csv'
```

```
DataFrame saved to /Users/mariomartinez/Desktop/Data_Engineering/FINAL_DATA_EGIN
EERING/Largest_banks_data.csv
```

[ ]:
```
import sqlite3

# Specify the path to your SQLite database
db_file_path = '/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_EGINEERING/Banks.db'
```

```
# Establish a connection to the SQLite database
conn = sqlite3.connect('/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_EGINEERING/Banks.db')

# Create a cursor object to execute SQL commands
cur = conn.cursor()

# SQL command to rename the table
rename_table_sql = "ALTER TABLE market_capitalizations RENAME TO Largest_Banks;"

# Execute the SQL command to rename the table
cur.execute(rename_table_sql)

# Commit the changes to the database
conn.commit()

# Close the connection
cur.close()
conn.close()
```

0.0.8 LOAD DB Table

<!–"'This function saves the final data frame to a database table with the provided name. Function returns nothing."'

[183]:
```
db_file_path = '/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_EGINEERING/Banks.db'

# Connect to the SQLite database
conn = sqlite3.connect('/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_EGINEERING/Banks.db')

# Example SQL query to retrieve everything from a table
def run_query(query, sql_connection):

# Run the query and return the result
   return pd.read_sql_query(query, sql_connection)
    #Print only the names of the top 5 banks
query = "SELECT `Bank name` FROM Largest_banks LIMIT 5"
result_df = run_query(query, conn)
result_df
   # Define the SQL query to retrieve everything from the table

query = "SELECT * FROM Largest_banks"
result_df= run_query(query,conn)
#print(result_df
result_df
```

```
 #query = "SELECT name FROM sqlite_master WHERE type='table';"

# Load the DataFrame from the CSV if it's not already loaded
# Uncomment the next line if you need to load the DataFrame from the CSV file

# Write the data to a sqlite table
df.to_sql('Largest_banks', conn, if_exists='replace', index=False)
# Log after loading to the database
log_progress("Data loaded to Database as a table, Executing queries")

# Close the connection to the database
conn.close()

log_progress("Server Connection closed")

print(f'DataFrame loaded into the SQLite database at {db_file_path}')
```

```
DataFrame loaded into the SQLite database at
/Users/mariomartinez/Desktop/Data_Engineering/FINAL_DATA_EGINEERING/Banks.db
```

0.0.9 RUN QUERIES

[190]:
```
query = "SELECT * FROM Largest_banks"
result_df= run_query(query,conn)
#print(result_df
result_df
```

[190]:

	Rank	Bank name	Market cap(US$ billion)	\
0	1	JPMorgan Chase	432.92	
1	2	Bank of America	231.52	
2	3	Industrial and Commercial Bank of China	194.56	
3	4	Agricultural Bank of China	160.68	
4	5	HDFC Bank	157.91	
5	6	Wells Fargo	155.87	
6	7	HSBC Holdings PLC	148.90	
7	8	Morgan Stanley	140.83	
8	9	China Construction Bank	139.82	
9	10	Bank of China	136.81	

	Market_cap_GBP_Billion	Market_cap_EUR_Billion	Market_cap_INR_Billion	\
0	346.34	402.62	35910.71	
1	185.22	215.31	19204.58	
2	155.65	180.94	16138.75	
3	128.54	149.43	13328.41	
4	126.33	146.86	13098.63	
5	124.70	144.96	12929.42	
6	119.12	138.48	12351.26	

```
7                  112.66                  130.97                  11681.85
8                  111.86                  130.03                  11598.07
9                  109.45                  127.23                  11348.39

   MC_EUR_Billion
0          402.62
1          215.31
2          180.94
3          149.43
4          146.86
5          144.96
6          138.48
7          130.97
8          130.03
9          127.23
```

[191]:
```
#Print the average market capitalization of all the banks in Billion USD.

query = "SELECT AVG(Market_cap_GBP_Billion)FROM Largest_banks"
result_df= run_query(query,conn)
result_df
```

[191]:
```
   AVG(Market_cap_GBP_Billion)
0                      151.987
```

[192]:
```
#Print only the names of the top 5 banks
query = "SELECT `Bank name` FROM Largest_banks LIMIT 5"
result_df = run_query(query, conn)
result_df
```

[192]:
```
                                Bank name
0                          JPMorgan Chase
1                         Bank of America
2  Industrial and Commercial Bank of China
3              Agricultural Bank of China
4                                HDFC Bank
```

0.0.10 LOG

[ ]:
```
# Minimal test to write to the log file
log_file_path = '/Users/mariomartinez/Desktop/Data_Engineering/
 ↪FINAL_DATA_ENGINEERING/code_log.txt'
log_progress("This is a test log entry.")

# Then try reading it back
with open(log_file_path, 'r') as file:
    print(file.read())
```

<!–"'This function runs the query on the database table and prints the output on the terminal. Function returns nothing."'"'Here, you define the required entities and call the relevant functions in the correct order to complete the project. Note that this portion is not inside any function."'

<!–# Log after declaring known values log_progress("Preliminaries complete. Initiating ETL process")

1 ... your code for extracting data ...

2 Log after extraction

log_progress("Data extraction complete. Initiating Transformation process")

3 ... your code for transforming data ...

4 Log after transformation

log_progress("Data transformation complete. Initiating Loading process")

5 ... your code for loading to CSV ...

6 Log after saving to CSV

log_progress("Data saved to CSV file")

7 ... your code to initiate SQLite3 connection ...

8 Log after initiating connection

log_progress("SQL Connection initiated")

9 ... your code to load data to the database ...

10 Log after loading to the database

log_progress("Data loaded to Database as a table, Executing queries")

11 ... your code to run queries ...

12 Log after running queries

log_progress("Process Complete")

13 ... your code to close SQLite3 connection ...

14 Log after closing the connection

log_progress(“Server Connection closed”)

<!– import os

15 Define the path to the log file

log_file_path = ‘/Users/mariomartinez/Desktop/Data_Engineering/FINAL_DATA_ENGINEERING/code_log
# actual file path

16 Check if the file exists

if os.path.exists(log_file_path): # Remove the file os.remove(log_file_path) print(f“Removed the file: {log_file_path}”) else: print(f“The file does not exist: {log_file_path}”)