Final Project

Live Data Pipeline for Financial Data Modeling

Harjot Dhaliwal (30051859)

Lukas Escoda (30211208)

Shabbir Khandwala (30219011)

Gurdeep Panag (30101520)

University of Calgary

DATA 608: Developing Big Data Applications

Instructor: Dr. Tyler Bonnell

April 11th, 2024

Table of Contents

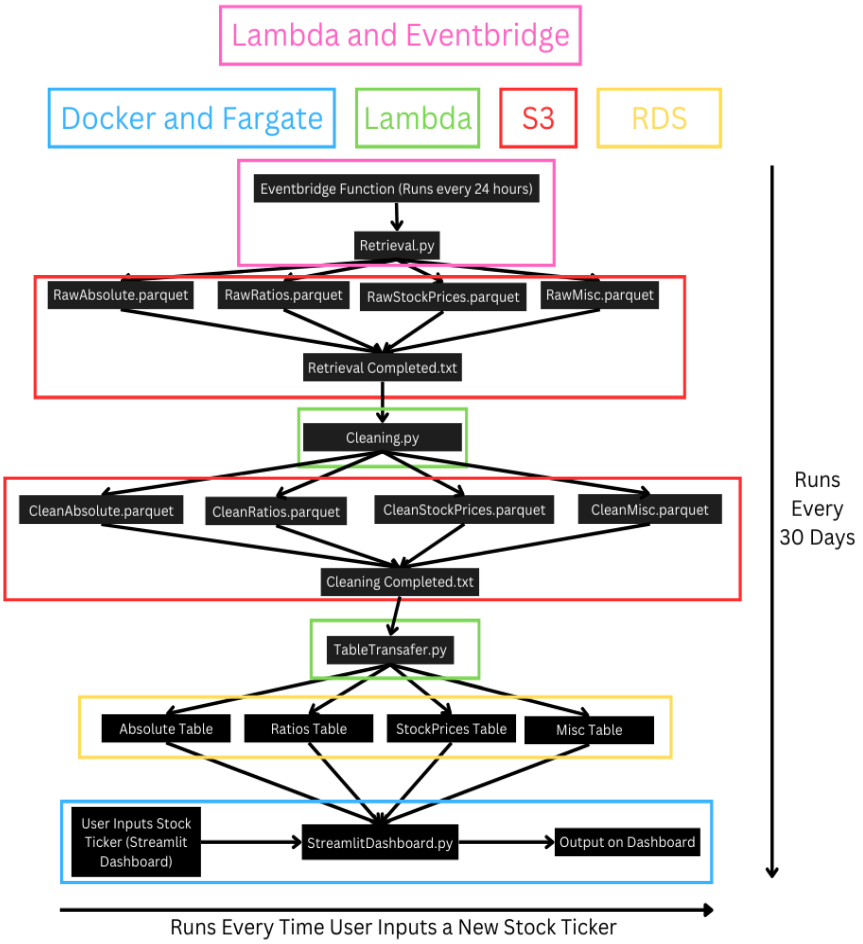
Section 1: Problem Statement and a Concise Summary of Results

Problem Statement: How can a robust data pipeline infrastructure support value stock investing by leveraging real-time financial data updates for individuals with limited financial knowledge?

This project successfully implemented a live data pipeline for financial data modeling, aimed at supporting value stock investing for individuals with limited financial knowledge. Leveraging AWS cloud infrastructure (Lambda, S3, RDS, ECS, Fargate), the pipeline automated data collection, cleaning, transformation, storage, and serving processes. Financial data was scraped from Yahoo Finance using the "yfinance" Python module, ensuring regular updates and accuracy. The resulting Streamlit dashboard, provided users with dynamic and interactive visualizations, facilitating insights into various companies' stock performance. Despite limitations such as expertise gaps and Lambda constraints, the project laid a strong foundation for future enhancements, including deeper visualizations, exploring AWS QuickSight for comprehensive dashboards, and leveraging web scraping for dynamic data acquisition. Overall, the project demonstrated the effective implementation of a robust data pipeline for financial data modeling on the AWS cloud platform, highlighting key learnings in AWS product usage, code optimization, and project management.

Section 2: Data Engineering Lifecycle

Section 2.0: Pipeline Diagram



Section 2.1: Data Generation

The source for our data is the Yahoo Finance website, from which the data is scraped using the "yfinance" Python module which provides us with a range of information in regards with the various characteristics of the company you provide the name of such as its general information, financial statements, stock price, etc.

The manner in which the module works is that by providing the ticker code for the company, it returns an object with all the information scraped from the internet and each piece of this information is present as an attribute within this object in the form of a Pandas DataFrame. The rationale behind opting for this particular type of data stems from its capacity to ensure the continual upkeep of our information. Given that this form of data undergoes daily updates while the website structure would continue to remain the same over the long-run, it afforded us the advantage of maintaining a consistently refreshed dataset that is able to scrape data accurately on regular basis, thereby enhancing the accuracy and relevance of our information over time.

Section 2.2: Data Ingestion

We have a series of Python Scripts that are responsible for the three steps on a general ETL process. The first file, retrieval.py is scheduled to run on a regular basis that pulls all the data using the above mentioned module. The schedule depends on how frequent the data refreshes such as stock price change daily while financial statements change quarterly. We were forced to use an EC2 instance, as opposed to a Lambda/Eventbridge instance, as there is a 15 minute timeout by default for Lambda, due to our extremely large dataset, this was not feasible, and thus we had to make use of an EC2 instance instead. This file simply extracts all the information for a static list of tickers we have identified and places them in the form of Parquet files on S3 as demonstrated in the Pipeline Diagram. This is also an area where we faced huge performance issues as the EC2 instance wasn't capable of executing for all the tickers at once. Therefore, we implemented batching without logic to divide the list of tickers into smaller chunks, generate these files with just the information about the tickers in one chunk, place them on S3 and move to the next chunk. While without batching the code didn't execute to the end, with it incorporated, the entire extraction process took ~30 minutes.

On its completion, this script also updates a file called RetrievalCompleted.txt. Subsequently, this triggers an AWS Lambda function which runs a Python script cleaning and organizing them into another set of Parquet files divided based on the information they carry, which are then also stored on S3. Similar to our previous script, this script updates a file called CleaningCompleted.txt which triggers our final Lambda function for our ETL process to be completed that runs a python script pushing all the clean transformed data into an AWS Postgresql RDS. The reason for the use of Parquet data format is the data compression that

comes with it to not only reduce the memory consumption on our free-tier EC2 instance but also incur significantly lesser S3 costs.

Section 2.3: Data Storage

Our data storage strategy leverages the strengths of two AWS services to optimize cost and performance. Amazon S3 serves as our cost-effective workhorse for storing large amounts of raw and intermediate data in CSV/Parquet format. Its scalability ensures we can accommodate growing data volumes, while its durability safeguards our information. Furthermore, S3 acts as a convenient staging ground for processing, as Lambda functions can be triggered by new data uploads to initiate automated cleaning and organization. Once the data is cleaned and transformed, it migrates to Amazon RDS Postgresql. This managed relational database service is ideal for storing our final, usable data due to its efficient querying capabilities via familiar SQL language. This is crucial as users will be querying the data on-demand through the dashboard, requiring fast and structured access. Additionally, RDS Postgresql offers robust data integrity features and advanced security options to safeguard sensitive financial information. By segregating storage based on data purpose and access needs, this approach provides a cost-effective, secure, and performant solution for our stock price dashboard project.

Section 2.4: Data Transformation

The data obtained from “yfinance” while being clean for the most part, we did have to make a few changes for analytical or cost-effective purposes. To ensure seamless functionality and integrity, the first step was to ensure we kept just the relevant columns and provided appropriate names for each column adhering to the SQL standards. While the data was relatively free from missing values, there were instances by virtue of actual missing data on the Yahoo Finance Website which we backfilled using some domain expertise.

Besides the data already formatted in a neat tabular format, features and columns which were directly not present without our datasets were created. The aim of doing this was to reduce the computation time when refreshing our dashboards as the user inputs the ticker names, the script would just extract the information already in a tabular format and plot it in near real-time. Compressing the data in Parquet format for cost and performance effectiveness, the transformed data gets transferred to S3, alleviating memory constraints and allowing smooth processing.

Section 2.5: Data Serving

In our approach to data serving, we chose to present our results through a Streamlit dashboard hosted on AWS Fargate using Elastic Container Service (ECS). We selected Fargate for its ability to dynamically adjust to user demand, anticipating varying levels of traffic to our dashboard. To achieve this, we built a Docker image with a foundation of Amazon Linux, incorporating our frontend code and GIF, along with essential tools such as Python, pip, Streamlit, SQLAlchemy, Pandas, Plotly, and psycopg2. This Docker image was uploaded to Docker Hub and deployed on ECS for reliable and consistent execution, ensuring all necessary packages, dependencies, and files are readily available. In our decision-making process, we prioritized Docker to guarantee the reliability and consistency of our code execution, ensuring all essential components are consistently accessible. For data querying, we opted for SQL to get the data needed from the Amazon RDS stuffed into a Pandas DataFrame using SQLAlchemy. As DataFrames allow us to make the plots quickly while still providing the quick execution that SQL enables us with, it was the way to go for us. Choosing Streamlit for our dashboard was driven by its flexibility, customization options, and seamless integration with Python and Plotly for creating dynamic and interactive visualizations. To enhance user experience, we incorporated a sidebar to separate text input, industry, and company descriptions to separate it from the core financial information/data. We also ensured that our text input box is not case-sensitive and that we included core financial definitions and information on the dashboard for improved usability for beginners. Moreover, we added a motivation button, featuring a GIF, to inspire users and foster a productive mindset. This comprehensive approach ensures our dashboard meets user needs effectively while providing a seamless and engaging experience.

Section 3: Limitations and Possible Next Steps

The approach we adopted for this dashboard was to cover a few basic visualizations that would enable people new to trading to get a feel of the different KPIs they must take a look at to make their decision. Partly due to lack of sufficient expertise in this domain, one key limitation is that our dashboard doesn't delve into the deeper aspects that might make the person confident about their trading decision. While the script does extract all the possible information about the company, we are using approximately 15%-20% of the available data for our visualizations. Getting someone with more domain expertise and collaborating with them to design visualizations catered towards the new traders and creating an alternative dashboard using a toggle for experts would increase the usability of our dashboard since the data is already available. Secondly, we could build a completely AWS-Native Dashboard solution by leveraging AWS QuickSight rather than use an EC2 instance and Streamlit for creating the dashboard. This would have enabled us to cover a lot more visualizations as the AWS offering provides a GUI interface similar to Power BI/Tableau to work on an existing data model, which is already built in AWS RDS in our case. Another limitation we faced was the 15 minute timeout for AWS Lambda, and how it forced us to use an EC2 instance instead of Lambda/Eventbridge, a possible remedy to this problem would be to use a uninhibited AWS account with more credits and compute resources that we could assign to it, as opposed to the learner lab environment. Lastly, one technical limitation is our static list of tickers which is constrained possibly from the "yfinance" module and a lack of a function within it to provide a list of tickers. The best solution would be to move away from this list and use a web-scraping solution for obtaining the list of tickers from the internet.

Section 4: Conclusions

Through this project, we've gained invaluable insights into building an end-to-end data pipeline on AWS cloud infrastructure. We utilized the "yfinance" Python module for continuous data scraping from Yahoo Finance, ensuring our dataset remained accurate and up-to-date. Challenges in data ingestion were overcome by implementing batching and leveraging an EC2 instance due to Lambda timeout constraints. Storage optimization was achieved by segregating data storage between Amazon S3 for raw and intermediate data and Amazon RDS Postgresql for final, usable data, ensuring efficient querying and data integrity. Efficient data transformation processes involved streamlining data, filling missing values, and compressing data in Parquet format for performance and cost-effectiveness. Hosting our dashboard on AWS Fargate using ECS, coupled with Docker for reliability, SQL querying for data retrieval, and Streamlit for dynamic visualizations, ensured seamless data serving and user interaction. Acknowledging limitations such as lack of expertise and AWS Lambda constraints, future steps include deeper visualizations with domain experts, exploring AWS QuickSight for a comprehensive dashboard solution, and utilizing web scraping for obtaining a dynamic list of tickers. Overall, this project provided us with valuable experience in AWS product usage, code optimization for cost reduction, and effective data engineering project management.

Section 5: Code and a Static Link to the Project

Static Link: http://100.26.146.168:8501/

Retrieval.py: Appendix A

Cleaning.py: Appendix B

TableTransfer.py: Appendix C

StreamlitDashboard.py: Appendix D.01

Dockerfile: Appendix D.02

Requirements.txt: Appendix D.03

References

yfinance. (2024) y*finance pypi*. https://pypi.org/project/yfinance/

Appendix

Appendix A: Retrieval.py

```
import yfinance as yf
import numpy as np
import pandas as pd
import datetime
from dateutil.relativedelta import relativedelta
import boto3
import os, shutil
import argparse

def getDescription(tickers: list[str]):
    '''
    Input: List of tickers
    Output: DataFrame with instantaneous values of each ticker
    Example: getDescription(["AAPL", "GOOG"])
    '''
    description = []
    for i in tickers:
        ticker = yf.Ticker(i)
        info = pd.Series(ticker.info, name=i).astype("string")
        description.append(pd.Series(info))

    df = pd.concat(description, axis=1).transpose()
    df.columns = list(map(lambda x: ''.join(e for e in
x.lower().replace(" ", "_") if e.isalnum() or e == "_"), df.columns))
    df = df.rename_axis("ticker").reset_index()
    return df

def getStock(tickers: list[str], num_years: int, interval: str):
    '''
    Input: List of tickers, look back period in terms of number of
years, interval across which the metrics are computed
    Output: DataFrame with time-series data regarding the stock of
each ticker - high, low, close, open, etc.
    Example: getStock(["AAPL", "GOOG"], 2, "1mo")
    '''
    stocks = []
    for i in tickers:
```

```
        ticker = yf.Ticker(i)
        df = ticker.history(start=datetime.datetime.today() -
relativedelta(years = num_years),
                            end=datetime.datetime.today(),
                            interval=interval
                            )
        df["ticker"] = i
        stocks.append(df)

    stocks = pd.concat(stocks)
    stocks.index = list(map(lambda x: datetime.date.strftime(x,
"%Y-%m"), stocks.index))
    stocks.columns = list(map(lambda x: ''.join(e for e in
x.lower().replace(" ", "_") if e.isalnum() or e == "_"),
stocks.columns))
    stocks = stocks.rename_axis("month").reset_index()
    return stocks

def getQuarterlyInformation(tickers):
    '''
    Input: List of tickers
    Output: DataFrame with time-series data obtained from the
financial statements for each ticker
    Example: getQuarterlyInformation(["AAPL", "GOOG"])
    '''
    quarterly_metrics = []
    for i in tickers:
        ticker = yf.Ticker(i)
        df = pd.concat([ticker.quarterly_financials.transpose(),
ticker.quarterly_balance_sheet.transpose()], axis=1)
        df["ticker"] = i
        quarterly_metrics.append(df)

    quarterly_metrics = pd.concat(quarterly_metrics)
    quarterly_metrics.index = list(map(lambda x:
datetime.date.strftime(x, "%Y-%m"), quarterly_metrics.index))
    quarterly_metrics.columns = list(map(lambda x: ''.join(e for e in
x.lower().replace(" ", "_") if e.isalnum() or e == "_"),
quarterly_metrics.columns))
```

```
    quarterly_metrics =
quarterly_metrics.rename_axis("month").reset_index()
    return quarterly_metrics

def getTickers(bucket_name: str, key: str, s3=None):
    '''
    Input: The bucket and key for the list of tickers
    Output: A list of tickers
    '''
    if s3 is None:
        s3 = boto3.client('s3')

    try:
        obj = s3.get_object(Bucket=args.bucket_name,
Key=args.ticker_list_location)
        tickers = list(map(lambda x: str(x).upper(),
pd.read_csv(obj['Body']).ticker_name.values))
        return tickers
    except Exception as e:
        print("Failed to obtain ticker list")
        return []

def uploadFiles(source_path: str, bucket_name: str, key: str,
dest_path: str, s3=None):
    '''
    Input: Path on Local with all the files, the S3 bucket where it
must be uploaded and the key for the files
    Output: No Output
    '''
    if s3 is None:
        s3 = boto3.client('s3')

    try:
        # If the directory is already existing, empty it first
        response = s3.list_objects_v2(Bucket=bucket_name, Prefix=key)
        if response["KeyCount"]:
            for object in response['Contents']:
                s3.delete_object(Bucket=bucket_name,
Key=object['Key'])
```

```
        # Uploading all the files from the source path to specified
key on S3
        for root, _, files in os.walk(source_path):
            for file in files:
                s3.upload_file(os.path.join(root, file), bucket_name,
os.path.join(root, file).replace(dump_path, dest_path).strip("/"))
        return 1

    except Exception as e:
        print(e)
        return 0
# Consuming user-defined arguments
# Example: python extract.py --num_batches 10 "stock-price-dashboard"
"ticker_list/ticker_list.csv" "raw_datasets"
parser = argparse.ArgumentParser()
parser.add_argument("--num_batches", type=int, default=10)      #
Setting batch size for quicker processing
parser.add_argument("bucket_name")                              #
Source & destination bucket name
parser.add_argument("ticker_list_location")                     # Key
for the ticker list
parser.add_argument("raw_file_destination")                     # Key
for the destination
args = parser.parse_args()

# Obtaining the list of tickers
tickers = getTickers(args.bucket_name, args.ticker_list_location)
print("Obtained the list of Tickers")

# Creating empty directories where these files will be stored
dump_path = os.path.join(os.getcwd(), "raw_datasets")
if os.path.isdir(dump_path):
    shutil.rmtree(dump_path)

os.mkdir(dump_path)
os.mkdir(os.path.join(dump_path, "info"))
os.mkdir(os.path.join(dump_path, "stock_price"))
os.mkdir(os.path.join(dump_path, "financial_statements"))
```

```
# Implementing batching to reduce the RAM utilization
batches = int(args.num_batches)
batches = list(map(lambda x: int(x), np.linspace(0, len(tickers) + 1,
batches + 1)))
batches = zip(batches[:-1], batches[1:])
for index, (i, j) in enumerate(batches):
    # For each batch, we will execute the three functions
    print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')} :
Started Batch {index + 1} - [{i}, {j-1}]")
    try:
        print(f"Extracting Instantaneous Information")

getDescription(tickers[i:j]).to_parquet(os.path.join(os.path.join(dum
p_path, "info"), f"part-{index}.parquet"), index=False)
        print(f"Extracting Monthly Stock Summary")
        getStock(tickers[i:j], 2,
"1mo").to_parquet(os.path.join(os.path.join(dump_path,
"stock_price"), f"part-{index}.parquet"), index=False)
        print(f"Extracting Quarterly Financial Document Summary")

getQuarterlyInformation(tickers[i:j]).to_parquet(os.path.join(os.path
.join(dump_path, "financial_statements"), f"part-{index}.parquet"),
index=False)
        print(f"{datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')} : Completed Batch {index + 1}\n")
    except Exception as e:
        # In case of any failures, we will now to the next batch
        print(f"{datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')} : Failed Batch {index + 1}\n")
        print(e)

if uploadFiles(dump_path, args.bucket_name,
args.raw_file_destination, args.raw_file_destination):
    print("Successfully uploaded files to S3")
    print("Step 1 of ETL Completed")
    with open("extract_step_completed.txt", "w+") as lambda_file:
        lambda_file.write(f"Completed on: {datetime.datetime.now()}")
```

```
    s3 = boto3.client('s3')
    s3.upload_file("extract_step_completed.txt", args.bucket_name,
f"{args.raw_file_destination}/extract_step_completed.txt")
else:
    print("S3 Upload has failed")
```

Appendix B: Cleaning.py

```
import numpy as np
import pandas as pd
import argparse
import os, shutil
import boto3
import datetime

parser = argparse.ArgumentParser()
parser.add_argument("bucket_name")                          # Source
& destination bucket name
parser.add_argument("raw_file_location")                    # Key for
the source
parser.add_argument("transform_file_destination")           # Key for
the destination
args = parser.parse_args()

print("Starting with Loading Datasets")
info =
pd.read_parquet(f"s3://{args.bucket_name}/{args.raw_file_location}/in
fo")
stock_price =
pd.read_parquet(f"s3://{args.bucket_name}/{args.raw_file_location}/st
ock_price")
financial_statements =
pd.read_parquet(f"s3://{args.bucket_name}/{args.raw_file_location}/fi
nancial_statements")
print("Loaded Datasets")

# Transforming company details
column_mapping = {
    "ticker": "ticker",
    "shortname": "company_nm",
    "website": "website",
    "industry": "industry",
    "longbusinesssummary": "company_info",
    "fulltimeemployees": "full_time_employees"
}
```

```
company_info =
info[column_mapping.keys()].rename(columns=column_mapping)
company_info.full_time_employees =
company_info.full_time_employees.astype(np.float64)

# Transforming stocks details
column_mapping = {
    "ticker": "ticker",
    "month": "month",
    "open": "opening_price",
    "close": "closing_price",
    "high": "month_high",
    "low": "month_low"
}
stock_price =
stock_price[column_mapping.keys()].rename(columns=column_mapping)
stock_price[[col for col in stock_price.columns if col not in
["ticker", "month"]]] = stock_price[[col for col in
stock_price.columns if col not in ["ticker",
"month"]]].astype(np.float64)

# Transforming financial statement metrics
column_mapping = {
    "ticker": "ticker",
    "month": "month",
    "cash_and_cash_equivalents": "cash_and_cash_equivalents",
    "ebitda": "ebitda",
    "net_income": "net_income",
    "net_debt": "net_debt",
    "total_debt": "total_debt",
    "current_assets": "current_assets",
    "current_liabilities": "current_liabilities"
}

financial_statements =
financial_statements[column_mapping.keys()].rename(columns=column_map
ping)
financial_statements[[col for col in financial_statements.columns if
col not in ["ticker", "month"]]] = financial_statements[[col for col
```

```
in financial_statements.columns if col not in ["ticker",
"month"]]].astype(np.float64)
financial_statements["current_ratio"] =
financial_statements.current_assets /
financial_statements.current_liabilities

# Find the latest month of available records
temp =
financial_statements.groupby(by="ticker")["month"].max().rename_axis(
"ticker").reset_index()
financial_statements = financial_statements.merge(temp,
how="inner").drop(columns=["month"])

# Generating all ratios
column_mapping = {
    "ticker": "ticker",
    "sharesoutstanding": "outstanding_shares",
    "previousclose": "latest_closing_price",
    "freecashflow": "free_cash_flow",
    "operatingcashflow": "operating_cash_flow",
    "dividendyield": "dividend_yield",
    "trailingpe": "trailing_pe",
    "debttoequity": "debt_to_equity",
    "returnonassets": "return_on_assets",
    "returnonequity": "return_on_equity",
}

ratios = info[column_mapping.keys()].rename(columns=column_mapping)
ratios.loc[:, ratios.columns != "ticker"] = ratios.loc[:,
ratios.columns != "ticker"].astype(np.float64)

ratios["market_cap"] = ratios.outstanding_shares *
ratios.latest_closing_price

# Add the EV/EBITDA & Current Ratio to the Ratios table
temp = temp.merge(financial_statements[["total_debt",
"cash_and_cash_equivalents", "ebitda", "current_ratio", "ticker"]],
how="inner")
temp = temp.merge(ratios[["ticker", "market_cap"]], how="inner")
```

```
temp["ev_to_ebitda"] = (temp.market_cap + temp.total_debt -
temp.cash_and_cash_equivalents) / temp.ebitda
ratios = ratios.merge(temp[["ticker", "current_ratio",
"ev_to_ebitda"]], how="left")

# Creating empty directories where these files will be stored
dump_path = os.path.join(os.getcwd(), "transformed_datasets")
if os.path.isdir(dump_path):
    shutil.rmtree(dump_path)

os.mkdir(dump_path)
os.mkdir(os.path.join(dump_path, "company_info"))
os.mkdir(os.path.join(dump_path, "stock_price"))
os.mkdir(os.path.join(dump_path, "financial_statements"))
os.mkdir(os.path.join(dump_path, "ratios"))

company_info.to_parquet(os.path.join(os.path.join(dump_path,
"company_info"), f"part-0.parquet"), index=False)
stock_price.to_parquet(os.path.join(os.path.join(dump_path,
"stock_price"), f"part-0.parquet"), index=False)
financial_statements.to_parquet(os.path.join(os.path.join(dump_path,
"financial_statements"), f"part-0.parquet"), index=False)
ratios.to_parquet(os.path.join(os.path.join(dump_path, "ratios"),
f"part-0.parquet"), index=False)

try:
    s3 = boto3.client('s3')
    # If the directory is already existing, empty it first
    response = s3.list_objects_v2(Bucket=args.bucket_name,
Prefix=args.transform_file_destination)
    if response["KeyCount"]:
        for object in response['Contents']:
            s3.delete_object(Bucket=args.bucket_name,
Key=object['Key'])

    # Uploading all the files from the source path to specified key
on S3
    for root, _, files in os.walk(dump_path):
        for file in files:
```

```
            s3.upload_file(os.path.join(root, file),
args.bucket_name, os.path.join(root, file).replace(dump_path,
args.transform_file_destination).strip("/"))

    print("Files successfully uploaded to S3")
    print("Step 2 Completed")
    with open("transform_step_completed.txt", "w+") as lambda_file:
        lambda_file.write(f"Completed on: {datetime.datetime.now()}")

    s3 = boto3.client('s3')
    s3.upload_file("transform_step_completed.txt", args.bucket_name,
f"{args.transform_file_destination}/transform_step_completed.txt")
except Exception as e:
    print(e)
    print("S3 upload was unsuccessful")
```

Appendix C: TableTransfer.py

```
import pandas as pd
import argparse
from sqlalchemy.engine.url import URL
from sqlalchemy import create_engine
import json
import boto3

parser = argparse.ArgumentParser()
parser.add_argument("bucket_name")                          # Source
& destination bucket name
parser.add_argument("transformed_file_location")            # Key for
the source
parser.add_argument("db_config_json")                       # Key for
the JSON file containing db configuration
args = parser.parse_args()

try:
    # Obtain the configurational details
    s3 = boto3.client("s3")
    response = s3.get_object(Bucket=args.bucket_name,
Key=args.db_config_json)
    db_info =
dict(json.loads(response['Body'].read().decode('utf-8')))
    db_url = URL.create(**db_info)
    print("Configuration details provided correctly")

    # Connect to your postgres DB
    engine = create_engine(db_url)

    # Execute a query

pd.read_parquet(f"s3://{args.bucket_name}/{args.transformed_file_loca
tion}/company_info").to_sql("company_info", con=engine,
if_exists="replace", index=False)

pd.read_parquet(f"s3://{args.bucket_name}/{args.transformed_file_loca
tion}/stock_price").to_sql("stock_price", con=engine,
```

```
if_exists="replace", index=False)

pd.read_parquet(f"s3://{args.bucket_name}/{args.transformed_file_loca
tion}/financial_statements").to_sql("financial_statements",
con=engine, if_exists="replace", index=False)

pd.read_parquet(f"s3://{args.bucket_name}/{args.transformed_file_loca
tion}/ratios").to_sql("ratios", con=engine, if_exists="replace",
index=False)

    print("All the files have been pushed to the database")
    print("Step 3 Completed")

except Exception as e:
    print(f"Error while executing Load step: {e}")
```

Appendix D: Frontend

Appendix D.01: StreamlitDashboard.py

```
# importing packages
import pandas as pd
from sqlalchemy import create_engine
import plotly.graph_objects as go
import streamlit as st
from sqlalchemy.engine.url import URL

# connecting to the postgresql database engine
db_info = {
   "drivername": "postgresql+psycopg2",
   "username": "postgres",
   "password": "postgres123",
   "host":
"super-open-stocks-rds.c9ko248kcabw.us-east-1.rds.amazonaws.com",
   "port": "5432",
   "database": "stock_data",
}
db_url = URL.create(**db_info)
engine = create_engine(db_url)

# setting the dashboard configuration to wide
st.set_page_config(layout="wide")
# initializing the input field for the stock ticker in the dashboard
ticker = st.sidebar.text_input("## Enter Stock Ticker:").upper()

# checking if stock ticker is not empty
if ticker != "":
   # getting the company info as a dataframe from amazon rds
   company_info_df = pd.read_sql_query(
       f"""
                           SELECT *
                           FROM company_info
                           WHERE ticker = '{ticker}';
                       """,
       engine,
```

```
    )
    # storing the company name, industry, description and company
website as strings
    name = company_info_df["company_nm"][0]
    industry = company_info_df["industry"][0]
    description = company_info_df["company_info"][0]
    company_website = company_info_df["website"][0]
    # getting the company financial statements from amazon rds
    financial_statements_df = pd.read_sql_query(
        f"""
                            SELECT *
                            FROM financial_statements
                            WHERE ticker = '{ticker}';
                        """,
        engine,
    )
    # getting the company financial ratios from amazon rds
    ratios_df = pd.read_sql_query(
        f"""
                            SELECT *
                            FROM ratios
                            WHERE ticker = '{ticker}';
                        """,
        engine,
    )
    # getting the company stock price values from amazon rds
    stock_price_df = pd.read_sql_query(
        f"""
                            SELECT *
                            FROM stock_price
                            WHERE ticker = '{ticker}';
                        """,
        engine,
    )
    # converting the date feature into a datetime datatype
    stock_price_df["month"] = pd.to_datetime(stock_price_df["month"])
    stock_price_df["month"] = stock_price_df["month"].dt.strftime("%b
%Y")
    # getting a dataframe of industry average values
```

```
   industry_averages = pd.read_sql_query(
       f"""
                           SELECT
AVG(financial_statements.cash_and_cash_equivalents) AS
cash_and_cash_equivalents, AVG(financial_statements.ebitda) AS
ebitda, AVG(financial_statements.net_income) AS net_income,
AVG(financial_statements.net_debt) AS net_debt,
AVG(ratios.current_ratio) AS current_ratio,
AVG(ratios.free_cash_flow) AS free_cash_flow,
AVG(ratios.operating_cash_flow) AS operating_cash_flow,
AVG(ratios.debt_to_equity) AS debt_to_equity,
AVG(ratios.return_on_assets) AS return_on_assets,
AVG(ratios.return_on_equity) AS return_on_equity,
AVG(ratios.ev_to_ebitda) AS ev_to_ebitda, AVG(ratios.trailing_pe) AS
trailing_pe
                           FROM company_info
                               LEFT JOIN financial_statements
                                   ON company_info.ticker =
financial_statements.ticker
                                       LEFT JOIN ratios
                                           ON
financial_statements.ticker = ratios.ticker
                           WHERE industry = '{industry}'
                           GROUP BY industry;
                       """,
       engine,
   )
   # getting the average stock price values for the industry for each
month
   industry_average_stock_price = pd.read_sql_query(
       f"""
                       SELECT stock_price.month,
AVG(stock_price.closing_price) AS closing_price
                       FROM company_info
                           LEFT JOIN stock_price
                               ON company_info.ticker =
stock_price.ticker
                       WHERE company_info.industry = '{industry}'
                       GROUP BY stock_price.month
```

```
                        ORDER BY stock_price.month;
                        """,
        engine,
    )
    # converting the date feature into a datetime datatype
    industry_average_stock_price["month"] = pd.to_datetime(
        industry_average_stock_price["month"]
    )
    industry_average_stock_price["month"] =
industry_average_stock_price[
        "month"
    ].dt.strftime("%b %Y")
    # joining the ratios and financial statement values into one
dataframe
    financial_statements_and_ratios = pd.merge(
        financial_statements_df,
        ratios_df.drop(columns="current_ratio"),
        left_index=True,
        right_index=True,
        how="left",
    )
    # conducting a union between the industry averages and the company
values for financial statements and ratios
    industry_and_company = pd.concat(
        [financial_statements_and_ratios, industry_averages]
    )
    # setting the index to be labelled as either the company ticker or
the industry average
    industry_and_company.index = [ticker, "Industry Average"]
    # dropping columns which are not needed for analysis
    industry_and_company.drop(
        columns=[
            "ticker_x",
            "total_debt",
            "current_assets",
            "current_liabilities",
            "ticker_y",
            "outstanding_shares",
            "latest_closing_price",
```

```
        "dividend_yield",
        "market_cap",
    ],
    inplace=True,
)
# filtering for only financial statement values
financial_statements_df = industry_and_company[
    [
        "cash_and_cash_equivalents",
        "ebitda",
        "net_income",
        "net_debt",
        "free_cash_flow",
        "operating_cash_flow",
    ]
]
# filtering for only financial ratio values
ratios_df = industry_and_company[
    [
        "current_ratio",
        "trailing_pe",
        "debt_to_equity",
        "return_on_assets",
        "return_on_equity",
        "ev_to_ebitda",
    ]
]
# transposing the dataframes above
financial_statements_df = financial_statements_df.T
ratios_df = ratios_df.T

# inserting the dashboard sidebar information
st.sidebar.title(f"{name}")
st.sidebar.markdown("## Industry")
st.sidebar.markdown(f"{industry}")
st.sidebar.markdown("## Company Website")
st.sidebar.markdown(f"{company_website}")
st.sidebar.markdown("## Description")
st.sidebar.markdown(f"{description}")
```

```
    # inserting the main streamlit dashboard title
    st.title(f"{name} Stock Investing Guide")
    # creating the introductory text for the dashboard
    st.markdown(
        """
                The purpose of this dashboard is to provide financial
information about companies for individuals who are not familiar with
finance. This dashboard contains information about stock prices and
the most important metrics for evaluating a company's financial
situation compared to the industry.
                """
    )
    # creating the disclaimer text
    st.markdown(
        "*Disclaimer: This is not financial advice and is solely for
educational purposes.*"
    )

    # creating a line chart for stock price changes over time
    trace_stock = go.Scatter(
        x=stock_price_df["month"],
        y=stock_price_df["closing_price"],
        mode="lines",
        name=ticker,
        line=dict(color="#D6B4FC", width=3),
    )
    trace_industry_average = go.Scatter(
        x=industry_average_stock_price["month"],
        y=industry_average_stock_price["closing_price"],
        mode="lines",
        name="Industry Average",
        line=dict(color="#FBF07B", width=3),
    )
    layout = go.Layout(showlegend=True)
    fig = go.Figure(data=[trace_stock, trace_industry_average],
layout=layout)
    fig.update_layout(
        barmode="group",
```

```
        title=f"{ticker} Stock Price Over Time",
        xaxis=dict(title="Date", showgrid=False),
        yaxis=dict(title="USD ($)", showgrid=False),
        title_x=0.3,
        title_font=dict(size=24),
    )
    st.plotly_chart(fig, use_container_width=600)

    # creating a bar chart for financial statement comparisons between
industry and the stock
    trace1 = go.Bar(
        x=financial_statements_df.index,
        y=financial_statements_df[ticker],
        name=ticker,
        marker=dict(color="#D6B4FC"),
    )
    trace2 = go.Bar(
        x=financial_statements_df.index,
        y=financial_statements_df["Industry Average"],
        name="Industry Average",
        marker=dict(color="#FBF07B"),
    )
    fig_financial_statements_chart = go.Figure(data=[trace1, trace2])
    fig_financial_statements_chart.update_layout(
        barmode="group",
        title=f"{ticker} vs. Industry Average Financial Statement
Values",
        yaxis=dict(title="USD ($)", showgrid=False),
        xaxis=dict(
            showgrid=False,
            tickvals=list(range(6)),
            ticktext=[
                "Cash and Cash Equivalents",
                "EBITDA",
                "Net Income",
                "Net Debt",
                "Free Cash Flow",
                "Operating Cash Flow",
            ],
```

```
        ),
        title_x=0.2,
        title_font=dict(size=24),
    )
    st.plotly_chart(fig_financial_statements_chart,
use_container_width=600)

    # creating descriptions for financial statements values and
valuable insights for them
    st.markdown("#### Cash and Cash Equivalents")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company holding cash and cash equivalents above the industry
average suggests a strong liquidity position, offering financial
flexibility, enhanced risk management, and the ability to capitalize
on growth opportunities without relying on external financing. This
can signal to the market and investors that the company is
financially stable and strategically poised for future investments or
shareholder value enhancement. However, it's essential to balance
this against potential perceptions of missed investment opportunities
or inefficient capital use, as excessively high cash reserves may
also raise concerns about the company's growth prospects and
operational efficiency.
        """
    )
    st.markdown("#### EBITDA")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company reporting an EBITDA (Earnings Before Interest,
Taxes, Depreciation, and Amortization) above the industry average
typically indicates operational efficiency and profitability. This
superior EBITDA margin suggests that the company is generating
substantial earnings from its operations before accounting for
interest, taxes, and non-cash expenses, highlighting its core
business strength. Such financial performance can attract investors
```

```
by showcasing the company's ability to generate cash flow, invest in
growth opportunities, and withstand economic fluctuations. However,
while high EBITDA is often a positive sign, it's crucial to consider
it alongside other financial metrics to fully assess a company's
financial health and long-term sustainability.
        """
    )
    st.markdown("#### Net Income")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company reporting a net income above the industry average
demonstrates a strong profitability and financial health, indicating
it effectively manages its expenses and operations to maximize
earnings. This superior profitability can signal to investors and
stakeholders the company's success in translating revenues into
actual profits, after accounting for all costs, including taxes and
interest. Such financial performance suggests not only efficient
operations but also the potential for sustainable growth, dividend
payouts, and investment in new opportunities. High net income can
enhance the company's appeal to investors, potentially leading to a
higher stock price. However, it's essential to analyze this metric in
the context of the company's overall financial strategy and market
conditions, as singular emphasis on net income without considering
other financial health indicators might not provide a complete
picture.
        """
    )
    st.markdown("#### Net Debt")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company reporting a net debt below the industry average
suggests a strong balance sheet characterized by low leverage, which
indicates the company has managed its borrowing prudently and
maintains a healthy ratio of debt to equity. This position can signal
financial stability and flexibility, as the company relies less on
```

```
external financing and faces lower interest obligations, potentially
freeing up more resources for investment, growth opportunities, and
returns to shareholders. A lower net debt level can also provide a
buffer against economic downturns and financial stress, enhancing the
company's ability to navigate adverse conditions without jeopardizing
its operational integrity. However, it's crucial to consider the
industry context and the company's growth strategy, as too little
debt might also suggest an overly conservative approach that could
limit growth opportunities and the efficient use of capital.
        """
    )
    st.markdown("#### Free Cash Flow")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company with a free cash flow (FCF) above the industry
average showcases its ability to generate cash from its operations
after accounting for capital expenditures, which is critical for
sustaining growth, paying dividends, and reducing debt. This strong
FCF indicates not only operational efficiency but also financial
flexibility, allowing the company to invest in new opportunities,
innovate, and return value to shareholders without relying on
external financing. High free cash flow is often seen by investors as
a sign of a company's health and potential for long-term success, as
it reflects the company's capability to generate surplus cash that
can be used strategically. However, it's important to analyze this in
conjunction with the company's investment plans, as consistently high
FCF could also suggest underinvestment in the business, potentially
stifling future growth.
        """
    )
    st.markdown("#### Operating Cash Flow")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company that reports operating cash flow above the industry
average demonstrates strong operational efficiency and an effective
```

```
cash conversion cycle. This indicates the company's core business
activities generate more cash than its peers, providing essential
liquidity for day-to-day operations, debt repayment, investment, and
growth initiatives without depending on external financing. High
operating cash flow signals to investors and stakeholders the
company's ability to sustain and grow its operations profitably,
highlighting a solid foundation for financial health and potential
value creation. However, it's crucial to view this metric in the
context of the company's overall financial strategy and investment
needs, as reinvestment in the business is essential for long-term
growth, and excessively high operating cash flow might reflect
underinvestment.
        """
    )

    # creating a bar chart for financial ratio comparisons between
industry and the stock
    trace3 = go.Bar(
        x=ratios_df.index,
        y=ratios_df[ticker],
        name=ticker,
        marker=dict(color="#D6B4FC"),
    )
    trace4 = go.Bar(
        x=ratios_df.index,
        y=ratios_df["Industry Average"],
        name="Industry Average",
        marker=dict(color="#FBF07B"),
    )
    fig_ratio_chart = go.Figure(data=[trace3, trace4])
    fig_ratio_chart.update_layout(
        barmode="group",
        title=f"{ticker} vs. Industry Average Financial Ratio Values",
        yaxis=dict(title="USD ($)", showgrid=False),
        xaxis=dict(
            showgrid=False,
            tickvals=list(range(6)),
            ticktext=[
                "Current Ratio",
```

```
                "PE Ratio",
                "Debt to Equity",
                "Return on Assets (ROA)",
                "Return on Equity (ROE)",
                "EV to EBITDA",
            ],
        ),
        title_x=0.2,
        title_font=dict(size=24),
    )
    st.plotly_chart(fig_ratio_chart, use_container_width=600)

    # creating descriptions for financial ratio values and valuable
insights for them
    st.markdown("#### Current Ratio")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company with a current ratio above the industry average
indicates strong liquidity, meaning it possesses more than enough
short-term assets (like cash, inventory, and receivables) to cover
its short-term liabilities (such as accounts payable and other
upcoming financial obligations). This higher current ratio suggests
the company is well-positioned to meet its short-term financial
commitments, reflecting financial stability and operational
efficiency. It can also signal to investors and creditors that the
company manages its working capital effectively and has a buffer
against financial distress, enhancing its creditworthiness. However,
it's essential to balance; a too-high current ratio might indicate an
excessive accumulation of assets or underutilization of resources
that could otherwise be invested for growth. Therefore, while a
higher current ratio is generally positive, it should be considered
alongside other financial metrics and operational goals to assess the
company's overall financial health and strategic direction
accurately.
        """
    )
    st.markdown("#### PE Ratio")
```

```
   st.markdown(
       """
       If the company values are greater than the industry average:

       A company with a Price-to-Earnings (P/E) ratio above the
industry average suggests that its stock is priced higher relative to
its earnings than its peers, which can imply that investors expect
higher growth rates, profitability, or stability from this company in
the future. This elevated P/E ratio indicates a market perception of
the company as having strong future prospects, possibly due to
innovative products, market leadership, or efficient operations. It
reflects investor confidence in the company's ability to generate
profits above and beyond the industry norm. However, a high P/E ratio
also raises questions about overvaluation and the sustainability of
the growth rates required to justify such valuation. It's important
for investors to consider this metric in the context of the company's
growth potential, sector dynamics, and broader market conditions to
evaluate whether the stock is a worthwhile investment or if it poses
a risk of being overvalued.
       """
   )
   st.markdown("#### Debt to Equity Ratio")
   st.markdown(
       """
       If the company values are greater than the industry average:

       A company with a debt-to-equity (D/E) ratio above the industry
average indicates it relies more heavily on debt financing relative
to equity to fund its operations and growth. This higher leverage can
suggest the company is aggressive in its growth strategy, potentially
offering higher returns on equity due to the use of borrowed funds.
However, a high D/E ratio also implies greater risk, as the company
must ensure it can cover its debt obligations, even in adverse
economic conditions. This reliance on debt can make the company more
vulnerable to interest rate increases or downturns in its business
cycle, affecting its profitability and financial stability. Investors
and creditors often scrutinize a high D/E ratio to assess the balance
between seeking higher growth through leverage and the risks
associated with increased debt levels. Therefore, while a high D/E
```

```
ratio may signal growth ambitions, it also calls for careful
evaluation of the company's ability to manage its debt responsibly
and maintain financial health.
        """
    )
    st.markdown("#### Return on Assets (ROA)")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company with a Return on Assets (ROA) ratio above the
industry average demonstrates superior efficiency in utilizing its
assets to generate profits. This higher ROA indicates that the
company is effectively converting its investments in assets into net
income, showcasing operational excellence and strategic management of
resources. It reflects not just the company's profitability but also
its ability to leverage its asset base for maximum financial
performance. Such efficiency can attract investors looking for
businesses that deliver strong returns on their investments. However,
while a high ROA is a positive indicator of financial health and
operational efficiency, it's essential to consider this metric in the
broader context of the company's financial strategies, industry
conditions, and growth prospects. A significantly high ROA, compared
to peers, may also prompt further investigation to ensure it is
sustainable and not the result of aggressive accounting practices.
        """
    )
    st.markdown("#### Return on Equity (ROE)")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company with a Return on Equity (ROE) ratio above the
industry average is seen as achieving superior efficiency in
generating profits from its shareholders' equity. This indicates that
the company is effectively using its invested capital to produce
earnings, reflecting strong management performance and financial
health. A high ROE suggests that the company is capable of generating
significant income on the investment made by its shareholders, which
```

```
can be particularly appealing to investors looking for businesses
that offer high returns on their investments. However, it's important
to analyze the components and sources of a high ROE to ensure it's
driven by genuine operational efficiency and growth, rather than by
high levels of debt which can inflate ROE but also introduce higher
financial risk. A sustainable high ROE, not overly reliant on debt,
indicates a robust business model and operational effectiveness,
positioning the company favorably for future growth and
profitability.
        """
    )
    st.markdown("#### EV to EBITDA Ratio")
    st.markdown(
        """
        If the company values are greater than the industry average:

        A company with an Enterprise Value to EBITDA (EV/EBITDA) ratio
above the industry average might be perceived as having a higher
valuation compared to its earnings before interest, taxes,
depreciation, and amortization. This can suggest that the market
values the company more highly, possibly due to expectations of
future growth, operational efficiencies, or strong market position. A
higher EV/EBITDA ratio indicates investors are willing to pay more
for each dollar of EBITDA generated by the company, which could
reflect its competitive advantages, potential for market expansion,
or superior management practices. However, it's also important to be
cautious, as a significantly high ratio compared to industry peers
might indicate overvaluation, where the price paid for the company's
earnings and cash flow is excessively high relative to its actual
financial performance. This metric should therefore be considered in
conjunction with other financial analyses and market conditions to
gauge whether the company's higher valuation is justified by its
growth prospects and operational strengths.
        """
    )

    # initializing the footer text for the dashboard
    st.markdown(
        "**Created by Shabbir Khandwala, Gurdeep Panag, Lukas Escoda
```

```
and Harjot Dhaliwal for DATA 608. All rights reserved.**"
    )

else:
    # if there is no stock ticker inputted, show this message
    st.title(f"Please Input a Valid Stock Ticker")
    # putting an introduction paragraph
    st.markdown(
        """
                The purpose of this dashboard is to provide financial
information about companies for individuals who are not familiar with
finance. This dashboard contains information about stock prices and
the most important metrics for evaluating a company's financial
situation compared to the industry.
                """
    )
    # putting the text for the disclaimer
    st.markdown(
        "*Disclaimer: This is not financial advice and is solely for
educational purposes.*"
    )

    #
------------------------------------------------------------------------
------------------------------------------------
    # creating the motivation button
    gif_path = "motivation.gif"
    if "show_gif" not in st.session_state:
        st.session_state.show_gif = False
    button = st.button("Click for Motivation")
    if button:
        st.session_state.show_gif = not st.session_state.show_gif
    if st.session_state.show_gif:
        st.image(gif_path, width=800)
    #
------------------------------------------------------------------------
------------------------------------------------

    # initializing the footer text for the dashboard
```

```
   st.markdown(
       "**Created by Shabbir Khandwala, Gurdeep Panag, Lukas Escoda
and Harjot Dhaliwal for DATA 608. All rights reserved.**"
   )
```

Appendix D.02: Dockerfile

```
# use a base python image
FROM amazonlinux:latest

# set a working directory
WORKDIR /app

# expose port
ENV PORT 8501

# install required python packages
COPY requirements.txt /app
RUN yum update -y && \
  yum install -y python3 && \
  yum install -y python3-pip && \
  yum install -y gcc python-setuptools python3-devel postgresql-devel && \
  pip install -r requirements.txt

# copy the rest of your application code
COPY . /app

# run the streamlit application
CMD ["streamlit", "run", "Frontend.py"]
```

Appendix D.03: Requirements.txt

```
psycopg2
sqlalchemy
pandas
plotly
streamlit
```