

## Project

# **MAANG Stock Prices Analysis**

Milestone 6

Cloud Problem Definition

Group 8

Pooja Ramesh

Sai Varun Kumar Namburi

Ramesh.po@northeastern.edu

Namburi.sai@northeastern.edu

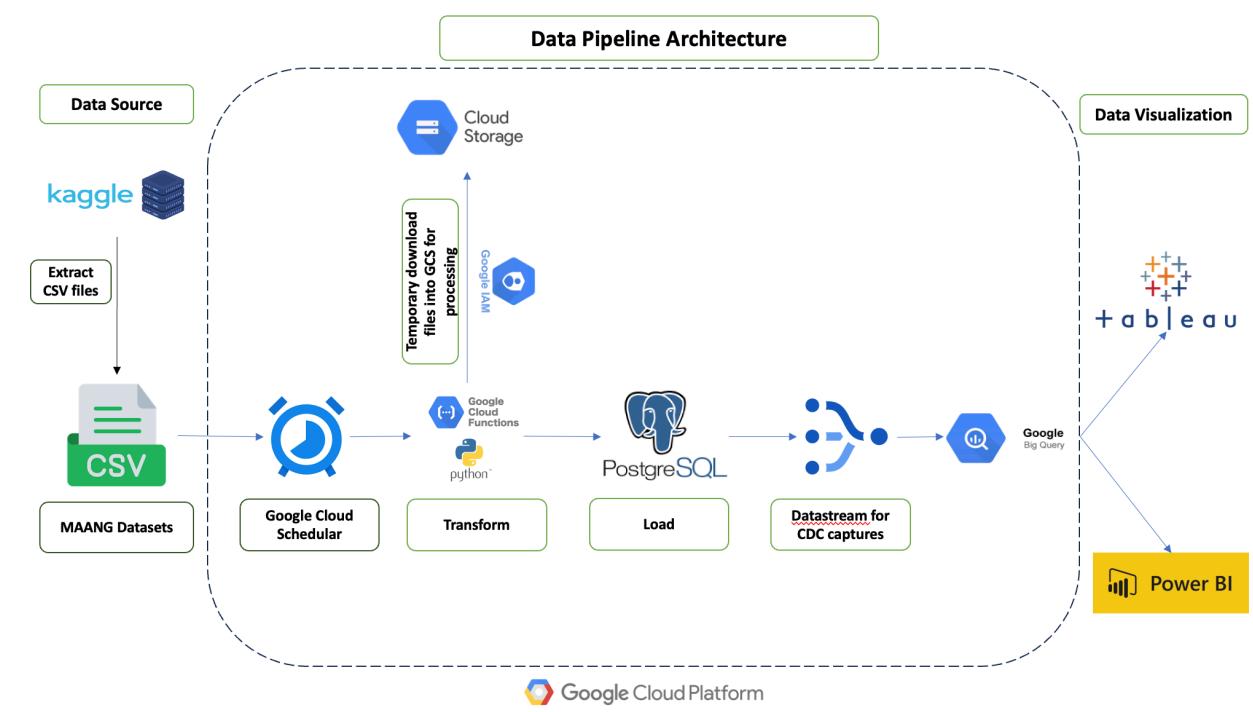
Submission Date: Dec 1<sup>st</sup>, 2023

## Objective:

This project seeks to develop a data driven solution to address the complexities of analyzing the historical stock prices of MAANG companies and identify the market conditions and external factors that could help make informed investment decisions. With this, the project aims to contribute to a broader understanding of MAANG stock prices, providing investors with niche perspectives of market dynamics and risks involved. The project aims to predict the effectiveness of trading strategies by back tracing with historical data insights and provide actionable insights to make data driven decisions about the trends, opportunities, and volatility.

Furthermore, by developing an efficient cloud-based data engineering solution which can handle large volumes of data for real time analysis.

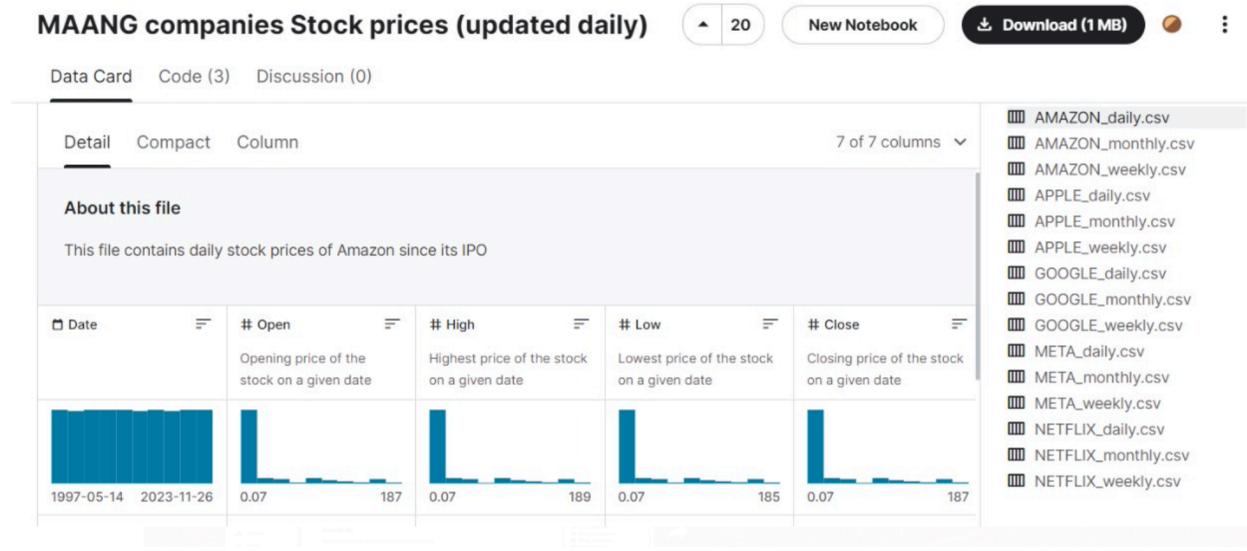
## Data Pipeline Architecture:



The MAANG stocks dataset from Kaggle is updated on a daily, weekly, and monthly basis. Here our aim is to retrieve the data daily and transform it to the database (Postgres) using the Kaggle API. After this, we read the data in cloud functions using python and this is triggered by Goggle cloud scheduler based on the requirements. As the cloud function is serverless, we need a temporary storage which can be used to hold the extracted files in google cloud storage. Then we transform the data (data types, renaming columns, adding columns) and then files are ingested to three tables based on daily, weekly, and monthly into the Postgres DB. Then we use DataStream to capture for the changes (Auto incremental load) and move it into BigQuery.

## Data source:

<https://www.kaggle.com/datasets/nikhil1e9/netflix-stock-price>



## Data Ingestion:

We have adopted py scripts for daily, weekly, and monthly to transform and insert the 15 files into cloud postgres. Here, we used the Kaggle API instead of downloading the 15 files into the local or cloud bucket, as the data is very dynamic and changes daily.

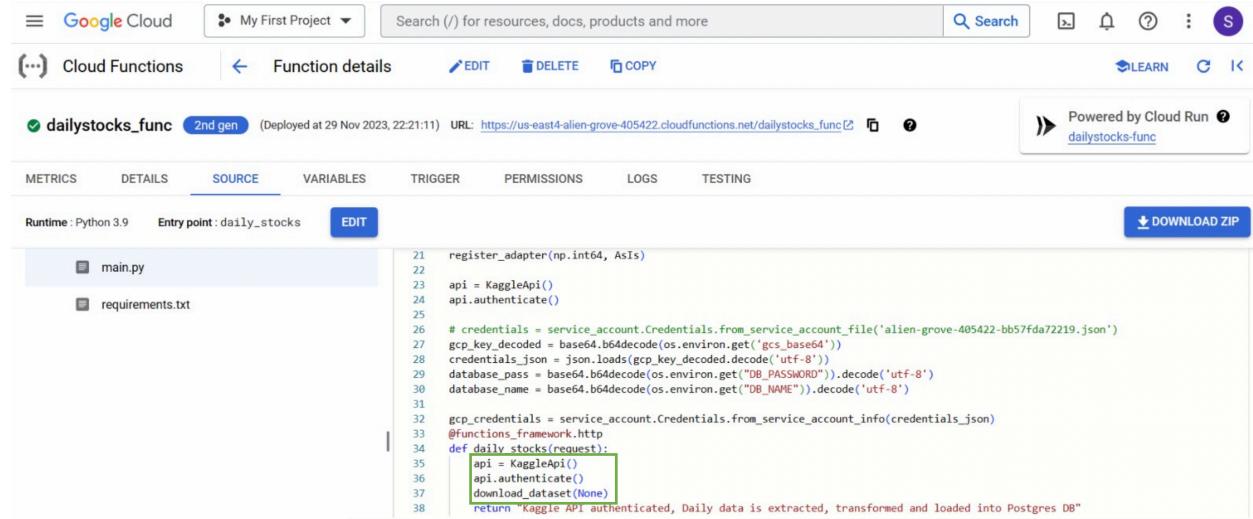
We are using Cloud Scheduler to trigger the cloud functions.

The screenshot shows the Google Cloud Platform Cloud Scheduler interface. The top navigation bar includes 'Google Cloud', 'My First Project', 'sche', 'Search', and other navigation icons. The main section is titled 'SCHEDULER JOBS' and 'APP ENGINE CRON JOBS'. It lists three scheduled jobs: 'dailyscheduler', 'monthlyscheduler', and 'weeklyscheduler'. Each job is shown with its name, status, region, state, description, frequency, target URL, and run history. The 'Actions' column provides options for each job.

Name	Status of last execution	Region	State	Description	Frequency	Target	Last run	Next run	Actions
dailyscheduler	Has not run yet.	us-central1	Enabled	Daily Scheduler to run cloud functions	0 23 * * *	URL : https://us-east4-alien-grove-405422.cloudfunctions.net/dailystocks_func		30 Nov 2023, 23:00:00	⋮
monthlyscheduler	Has not run yet.	us-east4	Enabled	Monthly Scheduler for stocks	0 23 1 * *	URL : https://us-central1-alien-grove-405422.cloudfunctions.net/monthlystocks_func		1 Dec 2023, 23:00:00	⋮
weeklyscheduler	Has not run yet.	us-east1	Enabled	Weekly Scheduler for stocks data	0 23 * * 6	URL : https://us-east1-alien-grove-405422.cloudfunctions.net/weeklystocks_func		2 Dec 2023, 23:00:00	⋮

# IE7374 Data Warehousing and Data Integration

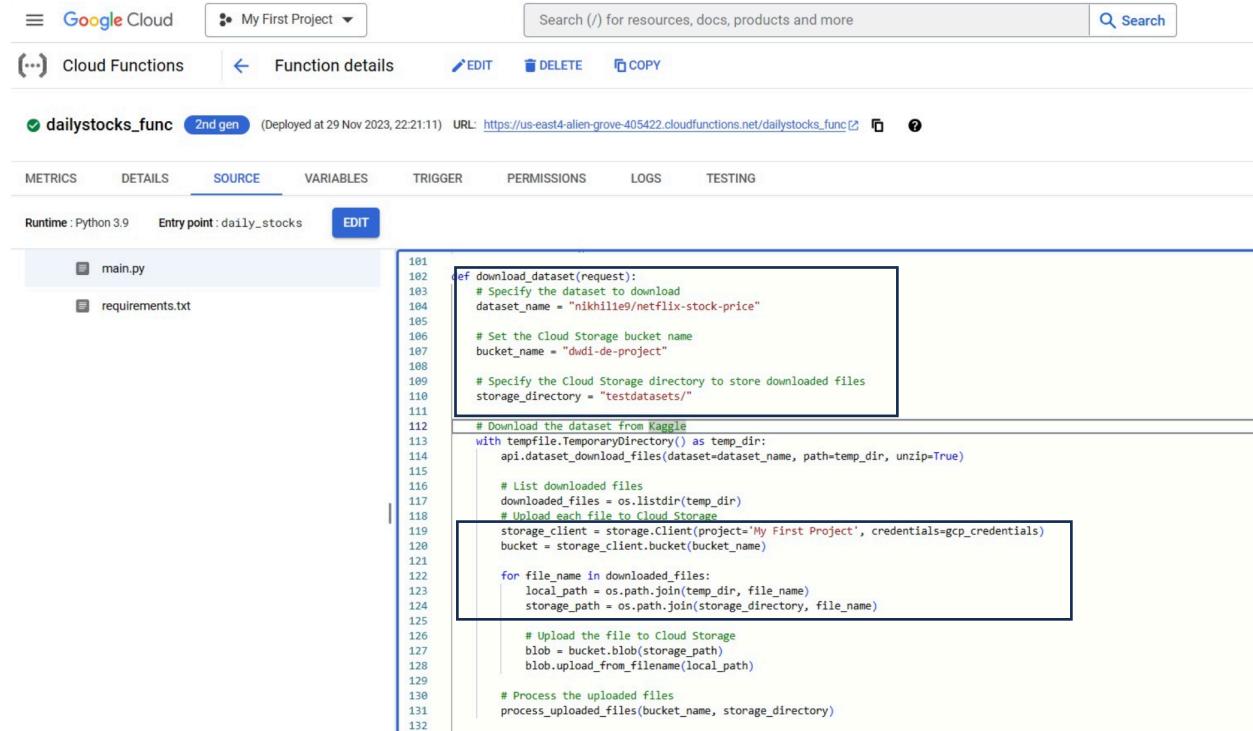
Authentication of Kaggle API by providing credentials:



The screenshot shows the Google Cloud Functions interface for a 2nd gen function named 'dailystocks\_func'. The 'SOURCE' tab is selected, showing the code in main.py. The code imports 'KaggleApi' and authenticates it. It then defines a function 'daily\_stocks' that creates a 'KaggleApi' object, authenticates it, and downloads a dataset. A callout box highlights the authentication and download logic.

```
21 register_adapter(np.int64, AsIs)
22
23 api = KaggleApi()
24 api.authenticate()
25
26 # credentials = service_account.Credentials.from_service_account_file('alien-grove-405422-bb57fda72219.json')
27 gcp_key_decoded = base64.b64decode(os.environ.get('gcs_base64'))
28 credentials_json = json.loads(gcp_key_decoded.decode('utf-8'))
29 database_pass = base64.b64decode(os.environ.get("DB_PASSWORD")).decode('utf-8')
30 database_name = base64.b64decode(os.environ.get("DB_NAME")).decode('utf-8')
31
32 gcp_credentials = service_account.Credentials.from_service_account_info(credentials_json)
33 @functions_framework.http
34 def daily_stocks(request):
35     api = KaggleApi()
36     api.authenticate()
37     download_dataset(None)
38
39     return "Kaggle API authenticated, Daily data is extracted, transformed and loaded into Postgres DB"
```

Providing the dataset link to download from Kaggle. Furthermore, the details of temporary storage bucket are also provided. It automatically authenticates using GCP default credentials to get the bucket access:



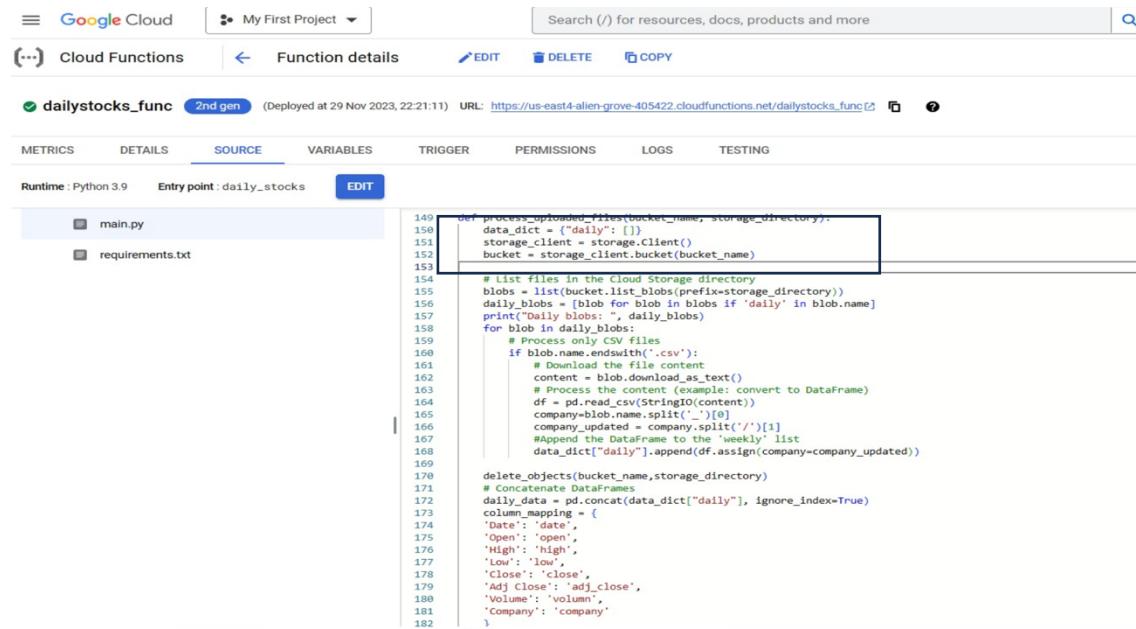
The screenshot shows the Google Cloud Functions interface for the same function. The 'SOURCE' tab is selected, showing the code in main.py. The code includes logic to download a dataset from Kaggle and upload it to a Cloud Storage bucket. Callout boxes highlight the dataset download and Cloud Storage upload sections.

```
101
102     if download_dataset(request):
103         # Specify the dataset to download
104         dataset_name = "nikhil110/netflix-stock-price"
105
106         # Set the Cloud Storage bucket name
107         bucket_name = "widi-de-project"
108
109         # Specify the Cloud Storage directory to store downloaded files
110         storage_directory = "testdatasets/"
111
112         # Download the dataset from Kaggle
113         with tempfile.TemporaryDirectory() as temp_dir:
114             api.dataset_download_files(dataset=dataset_name, path=temp_dir, unzip=True)
115
116             # List downloaded files
117             downloaded_files = os.listdir(temp_dir)
118             # Upload each file to Cloud Storage
119             storage_client = storage.Client(project="My First Project", credentials=gcp_credentials)
120             bucket = storage_client.bucket(bucket_name)
121
122             for file_name in downloaded_files:
123                 local_path = os.path.join(temp_dir, file_name)
124                 storage_path = os.path.join(storage_directory, file_name)
125
126                 # Upload the file to Cloud Storage
127                 blob = bucket.blob(storage_path)
128                 blob.upload_from_filename(local_path)
129
130             # Process the uploaded files
131             process_uploaded_files(bucket_name, storage_directory)
132
```

Now we call the process to upload files into cloud storage by providing the temporary path. Below is the screenshot for Daily, weekly, and monthly. Here the data is read and transformed. Now here we deleted the data stored in the bucket i.e., blobs.

# IE7374 Data Warehousing and Data Integration

## Daily -



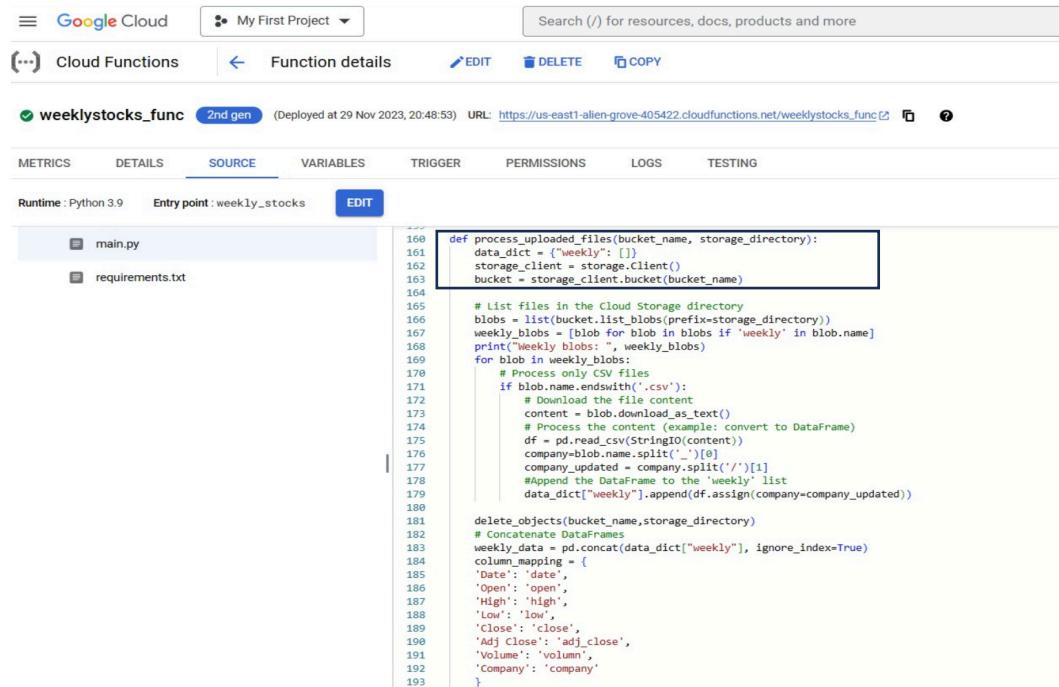
The screenshot shows the Google Cloud Functions interface for a function named "dailystocks\_func". The code is written in Python 3.9 and processes uploaded files from a bucket. It uses the storage client to list blobs and download CSV files. The content is then processed into a DataFrame and appended to a dictionary. The code is annotated with line numbers and a red box highlights the main processing logic.

```
def process_uploaded_files(bucket_name, storage_directory):
    data_dict = {"daily": []}
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)

    # List files in the Cloud Storage directory
    blobs = list(bucket.list_blobs(prefix=storage_directory))
    daily_blobs = [blob for blob in blobs if 'daily' in blob.name]
    print("Daily blobs: ", daily_blobs)
    for blob in daily_blobs:
        # Process only CSV files
        if blob.name.endswith('.csv'):
            # Download the file content
            content = blob.download_as_text()
            # Process the content (example: convert to DataFrame)
            df = pd.read_csv(StringIO(content))
            df['company'] = company_name.split('.')[0]
            company_updated = company_name.split('/')[1]
            #Append the DataFrame to the 'daily' list
            data_dict["daily"].append(df.assign(company=company_updated))

    delete_objects(bucket_name, storage_directory)
    # Concatenate DataFrames
    weekly_data = pd.concat(data_dict["daily"], ignore_index=True)
    column_mapping = {
        'Date': 'date',
        'Open': 'open',
        'High': 'high',
        'Low': 'low',
        'Close': 'close',
        'Adj Close': 'adj_close',
        'Volume': 'volume',
        'Company': 'company'
    }
```

## Weekly -



The screenshot shows the Google Cloud Functions interface for a function named "weeklystocks\_func". The code is written in Python 3.9 and processes uploaded files from a bucket. It uses the storage client to list blobs and download CSV files. The content is then processed into a DataFrame and appended to a dictionary. The code is annotated with line numbers and a red box highlights the main processing logic.

```
def process_uploaded_files(bucket_name, storage_directory):
    data_dict = {"weekly": []}
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)

    # List files in the Cloud Storage directory
    blobs = list(bucket.list_blobs(prefix=storage_directory))
    weekly_blobs = [blob for blob in blobs if 'weekly' in blob.name]
    print("Weekly blobs: ", weekly_blobs)
    for blob in weekly_blobs:
        # Process only CSV files
        if blob.name.endswith('.csv'):
            # Download the file content
            content = blob.download_as_text()
            # Process the content (example: convert to DataFrame)
            df = pd.read_csv(StringIO(content))
            company_name = blob.name.split('.')[0]
            company_updated = company_name.split('/')[1]
            #Append the DataFrame to the 'weekly' list
            data_dict["weekly"].append(df.assign(company=company_updated))

    delete_objects(bucket_name, storage_directory)
    # Concatenate DataFrames
    weekly_data = pd.concat(data_dict["weekly"], ignore_index=True)
    column_mapping = {
        'Date': 'date',
        'Open': 'open',
        'High': 'high',
        'Low': 'low',
        'Close': 'close',
        'Adj Close': 'adj_close',
        'Volume': 'volume',
        'Company': 'company'
    }
```

## Monthly -

The screenshot shows the Google Cloud Functions interface for a function named 'monthlystocks\_func'. The code is written in Python 3.9 and is located in 'main.py'. The code processes uploaded files from a storage bucket, specifically filtering for 'monthly' files. It then downloads each file, converts it to a DataFrame, and appends it to a dictionary. Finally, it deletes the objects from the storage bucket. The code is annotated with line numbers from 173 to 266.

```

173 def process_uploaded_files(bucket_name, storage_directory):
174     data_dict = {"monthly": []}
175     storage_client = storage.Client()
176     bucket = storage_client.bucket(bucket_name)
177
178     # List files in the Cloud Storage directory
179     blobs = list(bucket.list_blobs(prefix=storage_directory))
180     monthly_blobs = [blob for blob in blobs if 'monthly' in blob.name]
181     print("Monthly blobs: ", monthly_blobs)
182
183     for blob in monthly_blobs:
184         # Process only CSV files
185         if blob.name.endswith('.csv'):
186             # Download the file content
187             content = blob.download_as_text()
188             df = pd.read_csv(StringIO(content))
189             company = blob.name.split('/')[-1]
190             company_updated = company.split('.')[0]
191             #Append the DataFrame to the 'monthly' list
192             data_dict["monthly"].append(df.assign(company=company_updated))
193
194     delete_objects(bucket_name,storage_directory)
195
196     # Concatenate DataFrames
197     monthly_data = pd.concat(data_dict["monthly"], ignore_index=True)
198     column_mapping = {
199         'Date': 'date',
200         'Open': 'open',
201         'High': 'high',
202         'Low': 'low',
203         'Close': 'close',
204         'Adj Close': 'adj_close',
205         'Volume': 'volume',
206         'Company': 'company'
207     }

```

Now, after this the data gets downloaded into the cloud storage bucket for temporary purpose. We read the extracted files using the py script and read the files based on the files name and then read it as dataframes to transform such as renaming columns, adding column (Company name) and modifying the datatypes. Then we check now if the data is already present in Postgres Cloud using the py script. In case it is not present, we insert the data required to avoid data redundancy.

After reading the data, we call the connectDB function to insert it into Cloud Postgres. Now the transformed data is fed into cloud postgres.

The screenshot shows a Python script with a function named 'connect\_db\_bulk'. This function takes a DataFrame and a database name ('daily') as arguments. It renames the DataFrame columns according to a mapping and then connects to the database to bulk insert the data. The code is annotated with line numbers from 182 to 199.

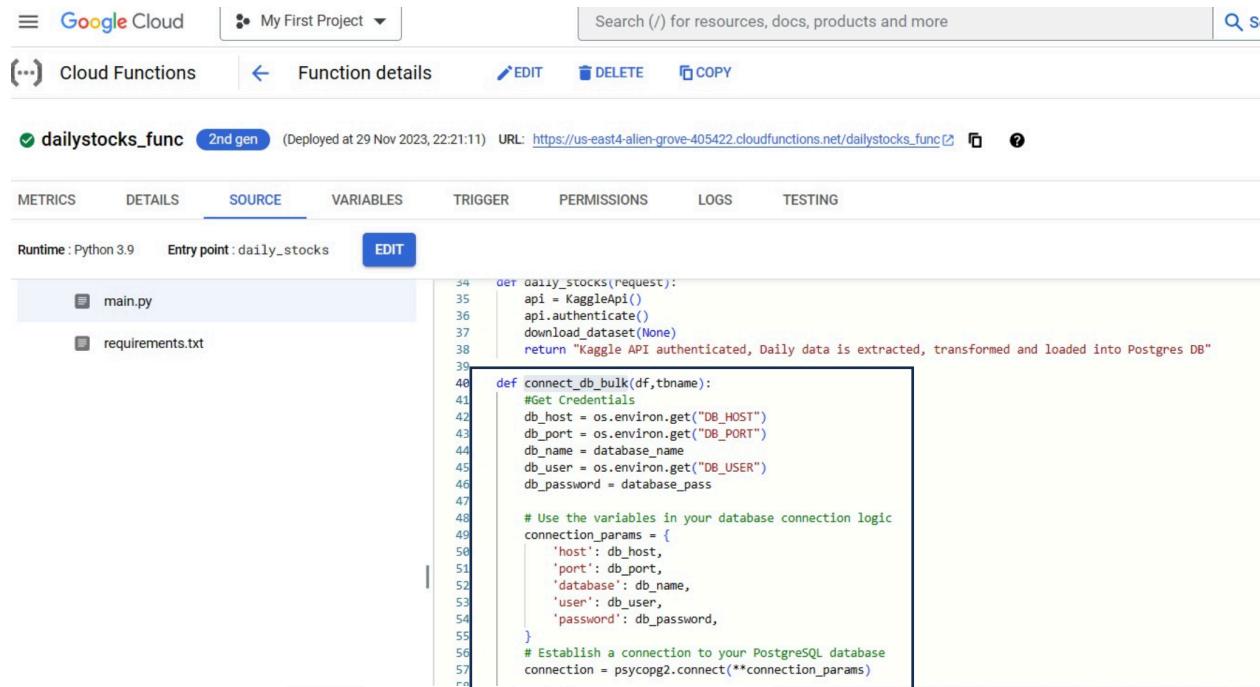
```

182
183     }
184
185     # Map DataFrame columns to PostgreSQL columns
186     df_daily_mapped = daily_data.rename(columns=column_mapping)
187     connect_db_bulk(df_daily_mapped,'daily')
188
189     return len(df_daily_mapped)
190

```

Now we provide the DB connections to connect to postgres DB and the password and DB name has been encoded in Base 64 format to ensure security.

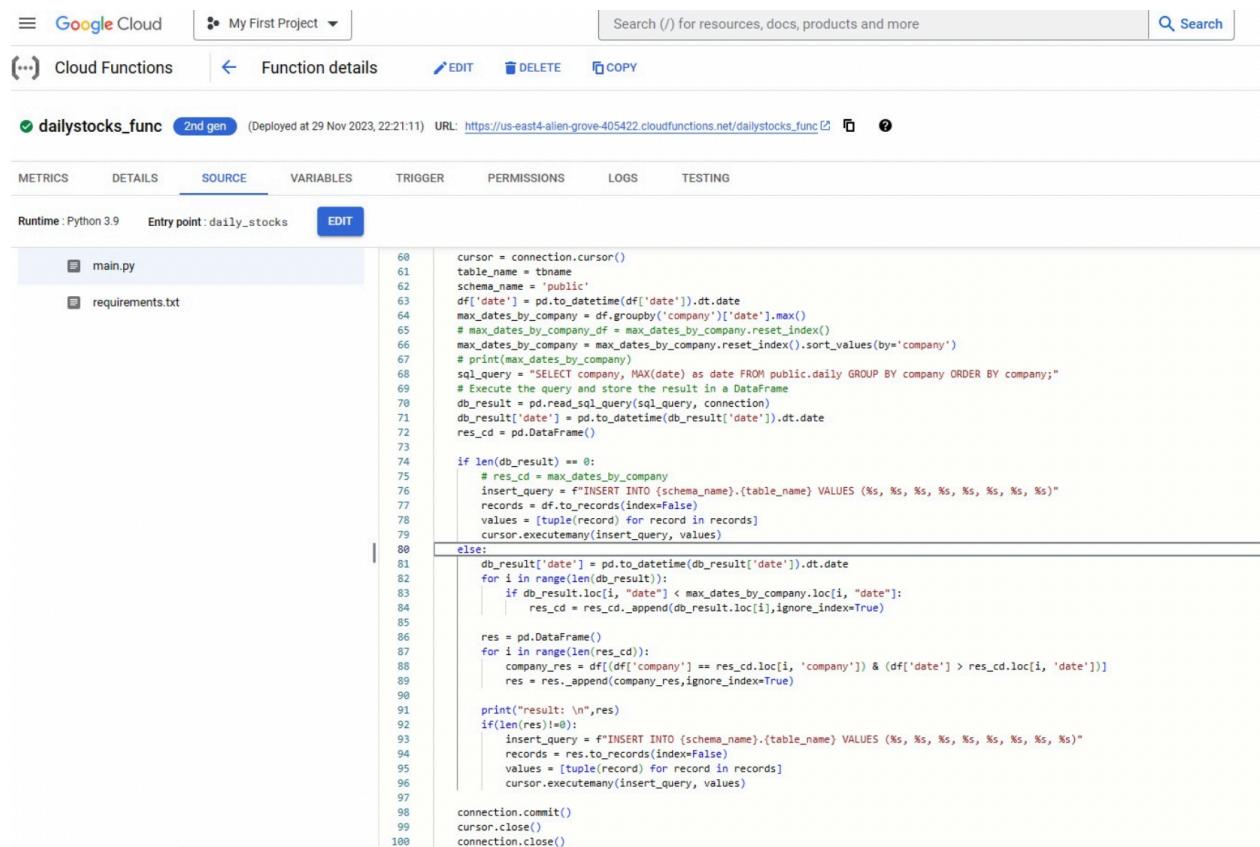
# IE7374 Data Warehousing and Data Integration



The screenshot shows the Google Cloud Functions interface for a function named 'dailystocks\_func'. The 'SOURCE' tab is selected. The code in main.py is as follows:

```
34     def daily_stocks(request):
35         api = KaggleApi()
36         api.authenticate()
37         download_dataset(None)
38         return "Kaggle API authenticated, Daily data is extracted, transformed and loaded into Postgres DB"
39
40     def connect_db_bulk(df,tbname):
41         #Get Credentials
42         db_host = os.environ.get("DB_HOST")
43         db_port = os.environ.get("DB_PORT")
44         db_name = database_name
45         db_user = os.environ.get("DB_USER")
46         db_password = database_pass
47
48         # Use the variables in your database connection logic
49         connection_params = {
50             'host': db_host,
51             'port': db_port,
52             'database': db_name,
53             'user': db_user,
54             'password': db_password,
55         }
56
57         # Establish a connection to your PostgreSQL database
58         connection = psycopg2.connect(**connection_params)
```

We are now checking of the data is present or not, if not we insert the data in order to avoid data redundancy.



The screenshot shows the Google Cloud Functions interface for the same function. The 'SOURCE' tab is selected. The code in main.py has been updated to include data insertion logic:

```
60     cursor = connection.cursor()
61     table_name = tbname
62     schema_name = 'public'
63     df['date'] = pd.to_datetime(df['date']).dt.date
64     max_dates_by_company = df.groupby('company')['date'].max()
65     max_dates_by_company_df = max_dates_by_company.reset_index()
66     max_dates_by_company = max_dates_by_company.reset_index().sort_values(by='company')
67     # print(max_dates_by_company)
68     sql_query = "SELECT company, MAX(date) as date FROM public.daily GROUP BY company ORDER BY company;"
69     # Execute the query and store the result in a DataFrame
70     db_result = pd.read_sql_query(sql_query, connection)
71     db_result['date'] = pd.to_datetime(db_result['date']).dt.date
72     res_cd = pd.DataFrame()
73
74     if len(db_result) == 0:
75         # res_cd = max_dates_by_company
76         insert_query = f"INSERT INTO {schema_name}.{table_name} VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
77         records = df.to_records(index=False)
78         values = [tuple(record) for record in records]
79         cursor.executemany(insert_query, values)
80     else:
81         db_result['date'] = pd.to_datetime(db_result['date']).dt.date
82         for i in range(len(db_result)):
83             if db_result.loc[i, "date"] < max_dates_by_company.loc[i, "date"]:
84                 res_cd = res_cd.append(db_result.loc[i], ignore_index=True)
85
86         res = pd.DataFrame()
87         for i in range(len(res_cd)):
88             company_res = df[df['company']] == res_cd.loc[i, 'company'] & (df['date'] > res_cd.loc[i, 'date'])
89             res = res.append(company_res, ignore_index=True)
90
91             print("result: \n",res)
92             if len(res)>0:
93                 insert_query = f"INSERT INTO {schema_name}.{table_name} VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
94                 records = res.to_records(index=False)
95                 values = [tuple(record) for record in records]
96                 cursor.executemany(insert_query, values)
97
98         connection.commit()
99         cursor.close()
100        connection.close()
```

# IE7374 Data Warehousing and Data Integration

## Loading into Postgres:

The screenshot shows the Google Cloud SQL Overview page for the instance `instancepostgresql`. The left sidebar lists various management options like Overview, System insights, Query insights, Connections, Users, Databases, Backups, Replicas, and Operations. The main area displays a chart titled "CPU utilisation" showing usage over the last 24 hours. A notable peak is visible around 22:00 UTC-5. Below the chart, there's a link to "Go to Query insights for more in-depth info on queries and performance". On the right, there are sections for "Connect to this instance" (with a Public IP address of `35.224.2.253`) and "Configuration" (showing vCPUs, Memory, and SSD storage). A "Release notes" section is also present.

## Daily data:

The screenshot shows the DBeaver SQL Editor interface. The top bar includes File, Edit, Navigate, Search, SQL Editor, Database, Window, and Help. The central area displays the results of a SQL query: `select * from daily d;`. The results are presented in a grid titled "daily 1 x". The columns include date, open, high, low, close, adj\_close, volume, and net company. The data spans from November 27, 2023, to November 17, 2023. The results show daily price movements for several companies, with Apple, Meta, Google, and Netflix being the most prominent. The left sidebar shows the Database Navigator with the connection to `35.224.2.253` and the schema `dwdl-cloud-db`.

# IE7374 Data Warehousing and Data Integration

## Weekly data:

The screenshot shows the DBeaver interface with the following details:

- Database Navigator:** Shows the database structure with the `public` schema containing `daily`, `monthly`, and `weekly` tables.
- SQL Editor:** Contains the SQL query: `select * from weekly w;`
- Results Grid:** Displays the results for the `weekly` table. The columns are: date, open, high, low, close, adj\_close, volume, and company. The data includes rows for Google, Netflix, Apple, Amazon, and Meta from November 2023.
- Project - General:** Shows a single record named `DataSource`.

date	open	high	low	close	adj_close	volume	company
2023-11-27	137.57	139.63	137.54	138.05	138.05	17,868,000	GOOGLE
2023-11-27	479.03	482	475.35	479.17	479.17	3,623,800	NETFLIX
2023-11-27	189.92	190.67	188.9	189.79	189.79	40,500,500	APPLE
2023-11-27	147.53	149.26	146.88	147.73	147.73	53,666,700	AMAZON
2023-11-27	336.18	339.9	334.2	334.7	334.7	15,646,300	META
2023-11-20	465.4	482.7	465.4	479.56	479.56	10,861,600	NETFLIX
2023-11-20	334.92	342.92	334.19	338.23	338.23	45,158,600	META
2023-11-20	135.5	141.1	135.49	138.22	138.22	63,352,500	GOOGLE
2023-11-20	189.89	192.93	189.25	189.97	189.97	148,305,600	APPLE
2023-11-20	145.13	147.74	141.5	146.74	146.74	181,224,700	AMAZON
2023-11-13	326.2	338.4	325.7	334.19	334.19	67,552,100	META
2023-11-13	133.36	138.88	132.77	138.7	138.7	72,183,200	GOOGLE
2023-11-13	447.25	467.28	442.6	466.95	466.95	15,648,300	NETFLIX
2023-11-13	185.82	190.96	184.21	189.71	189.71	211,939,300	APPLE
2023-11-13	142.08	147.29	139.52	142.83	142.83	205,884,400	AMAZON
2023-11-06	130.22	134.27	129.93	134.06	134.06	88,527,200	GOOGLE
2023-11-06	315.98	329.1	314.45	328.77	328.77	75,752,300	META
2023-11-06	176.38	186.57	176.21	186.4	186.155	303,608,500	APPLE
2023-11-06	434.38	447.48	429.61	447.24	447.24	15,827,200	NETFLIX
2023-11-06	138.76	143.65	138.36	143.56	143.56	228,568,800	AMAZON
2023-10-30	299.09	318.82	296.86	314.6	314.6	106,689,800	META
2023-10-30	169.02	177.78	167.9	176.65	176.418	310,010,400	APPLE
2023-10-30	129.72	139.49	128.56	138.6	138.6	281,848,200	AMAZON
2023-10-30	402.35	434.82	399.41	432.36	432.36	22,141,600	NETFLIX
2023-10-30	124.46	130.73	123.88	130.37	130.37	115,435,200	GOOGLE
2023-10-23	309.5	318.35	279.4	296.73	296.73	175,795,200	META

## Monthly data:

The screenshot shows the DBeaver interface with the following details:

- Database Navigator:** Shows the database structure with the `public` schema containing `daily`, `monthly`, and `weekly` tables.
- SQL Editor:** Contains the SQL query: `select * from monthly m;`
- Results Grid:** Displays the results for the `monthly` table. The columns are: date, open, high, low, close, adj\_close, volume, and company. The data includes rows for Netflix, Google, Meta, Amazon, and Apple from November 2023.
- Project - General:** Shows a single record named `DataSource`.

date	open	high	low	close	adj_close	volume	company
2023-11-01	414.77	482.7	414.18	479.17	479.17	61,706,300	NETFLIX
2023-11-01	125.34	141.1	124.925	138.05	138.05	337,642,400	GOOGLE
2023-11-01	301.85	342.92	301.85	334.7	334.7	277,424,200	META
2023-11-01	133.96	149.26	133.71	147.73	147.73	876,754,600	AMAZON
2023-11-01	171	192.93	170.12	189.79	189.54	969,310,000	APPLE
2023-10-01	377.48	418.84	344.73	411.69	411.69	164,021,900	NETFLIX
2023-10-01	132.155	142.38	121.46	125.3	125.3	514,877,100	GOOGLE
2023-10-01	302.74	330.54	279.4	301.27	301.27	511,307,900	META
2023-10-01	171.22	182.34	165.67	170.77	170.545	1,172,719,600	APPLE
2023-10-01	127.28	134.48	118.35	133.09	133.09	1,224,564,700	AMAZON
2023-09-01	139.46	145.86	123.04	127.12	127.12	1,120,271,900	AMAZON
2023-09-01	189.98	189.98	167.62	171.21	170.985	1,337,586,600	APPLE
2023-09-01	437.73	453.45	371.1	377.6	377.6	100,278,600	NETFLIX
2023-09-01	299.37	312.87	286.79	300.21	300.21	406,686,600	META
2023-09-01	138.43	139.93	128.19	131.85	131.85	389,593,900	GOOGLE
2023-08-01	133.55	143.63	126.41	138.01	138.01	1,210,426,200	AMAZON
2023-08-01	437.37	445.25	398.15	433.68	433.68	107,298,900	NETFLIX
2023-08-01	317.54	324.14	274.38	295.89	295.89	423,147,800	META
2023-08-01	130.855	138.4	127	137.35	137.35	463,482,000	GOOGLE
2023-08-01	196.24	196.73	171.96	187.87	187.37	1,322,439,400	APPLE
2023-07-01	193.78	198.23	186.6	196.45	195.927	996,066,400	APPLE
2023-07-01	120.32	134.07	115.83	133.11	133.11	525,456,900	GOOGLE
2023-07-01	286.7	326.2	284.85	318.6	318.6	624,605,100	META
2023-07-01	130.82	136.65	125.92	133.68	133.68	1,058,754,800	AMAZON
2023-07-01	439.76	485	411.88	438.97	438.97	168,720,200	NETFLIX
2023-06-01	120.69	121.49	110.93	120.26	120.26	1,212,419,900	AMAZON

## Data Ingestion into BigQuery:

Now the updated data in postgres will be inserted into BigQuery using DataStream.

The screenshot shows the Google Cloud Datastream interface. The left sidebar has 'DataStream' selected. Under 'Streams', there is one entry: 'postrestobq' (PostgreSQL/BigQuery). The main panel shows 'Stream details' for this stream. Key information includes:

- Stream ID:** postrestobq
- Source profile:** pgconnection
- Destination profile:** dsbigqueryconnection
- Created:** 22 Nov 2023, 17:09:10
- Updated:** 30 Nov 2023, 18:37:58

Below the details, there are tabs for 'OVERVIEW' (which is selected), 'MONITORING', and 'OBJECTS'. Under 'Properties', the following configuration is shown:

Region	us-central1 (Iowa)
Labels	No labels set
Objects to include	1 schema
Objects to exclude	None
Backfill mode	Automatic
Destination data set	MAANGdata
Staleness limit	5 minutes
Encryption	Google-managed
Tags	None

Below is the connection profile in DataStream:

The screenshot shows the Google Cloud Datastream interface. The left sidebar has 'DataStream' selected. Under 'Connection profiles', there is a table listing three profiles:

Profile name	Activity	Connection profile ID	Type	Connection details	Created	Region	Labels
dsbigqueryconnection	1 stream	dsbigqueryconnection	BigQuery	—	22 Nov 2023	us-central1 (Iowa)	⋮
pgconnection	1 stream	pgconnection	PostgreSQL	35.224.2.253 : dwdi-cloud-db	22 Nov 2023	us-central1 (Iowa)	⋮

In order to capture the changes in the postgres database, we implemented DataStream to capture the incremental load in data in BigQuery.

# IE7374 Data Warehousing and Data Integration

## Daily:

The screenshot shows the Google Cloud BigQuery interface. On the left, the sidebar includes sections for Analysis, BigQuery Studio, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Migration, Assessment, SQL translation, Administration, Monitoring, Capacity management, BI Engine, and Release notes. The main area displays the 'public\_daily' table under the 'public' dataset. The table has columns: Row, date, open, high, low, and close. The data shows historical price movements from 2013-12-18 to 1994-02-11. The table view includes options for QUERY, SHARE, COPY, SNAPSHOT, DELETE, EXPORT, and REFRESH.

Row	date	open	high	low	close
1	2013-12-18	19.632143020629883	19.694643020629883	19.24285697937012	19.67035675048828
2	2014-07-31	28.950515747070312	29.102598190307617	28.42196655273437	28.501747131347656
3	2013-12-19	19.625	19.64285659790039	19.418928146362305	19.4449996482422
4	1997-05-15	0.1218750029802322	0.125	0.0963540002703666	0.0979169979691505
5	2014-08-01	28.441913604736328	28.719152450561523	28.0654685363769	28.2260055419922
6	1994-02-09	0.3191959857490476	0.3258930146694183	0.3147319853305816	0.3236609995365143
7	2017-03-29	42.95249938964844	43.821998569191406	42.95100021362305	43.71599960327149
8	2022-12-09	115.3000030517578	117.54000091552734	113.87000274658205	115.9000015258789
9	2013-12-20	19.479642868041992	19.7003743713379	19.457857131958008	19.60785675048828
10	1997-05-16	0.0984380021691322	0.0989580005407333	0.0854170024394989	0.0864579975605011
11	2014-08-04	28.37409731445312	28.68873405456543	28.127775192260746	28.579036712646484
12	1994-02-10	0.3236609995365143	0.3348209857940674	0.3214290142059326	0.3258930146694183
13	2017-03-30	43.747501373291016	43.85300064086914	43.58300018310547	43.81700134277344
14	2022-12-12	115.18000030517578	115.72000122070312	113.13999938964844	114.70999908447266
15	2013-12-23	20.28571319580078	20.38265369018555	20.0985717734375	20.3603572845459
16	1997-05-19	0.0880210027098655	0.0885419994592666	0.0812499970197677	0.0854170024394989
17	2014-08-05	28.424461364746094	28.52069664001465	28.0534782409668	28.1761417388916
18	1994-02-11	0.3236609995365143	0.3348209857940674	0.3236609995365143	0.3303569853305816

## Weekly:

The screenshot shows the Google Cloud BigQuery interface. The sidebar and table structure are identical to the 'public\_daily' screenshot above, but the data represents weekly price movements from 1997-05-01 to 1997-09-08. The table view includes options for QUERY, SHARE, COPY, SNAPSHOT, DELETE, EXPORT, and REFRESH.

Row	date	open	high	low	close
1	1997-05-12	0.1218750029802322	0.125	0.0854170024394989	0.0864579975605011
2	1997-05-19	0.0880210027098655	0.0885419994592666	0.0656249970197677	0.0750000029802322
3	1997-05-26	0.0755209997296333	0.0822919979691505	0.072919994592666	0.0750000029802322
4	1997-06-02	0.0755209997296333	0.0854170024394989	0.0687500014901161	0.0828130021691322
5	1997-06-09	0.0882130021691322	0.0854170024394989	0.0765630006790161	0.0791670009493827
6	1997-06-16	0.08820800035209655	0.08820800035209655	0.0747000006911413	0.0763019993901252
7	1997-06-23	0.0770829990506172	0.0770829990506172	0.0739580020308494	0.0744789987802505
8	1997-06-30	0.0755209997296333	0.095830035209655	0.0739580020308494	0.095573000698747
9	1997-07-07	0.0916669964790344	0.1286460012197494	0.0916669964790344	0.1145830005407333
10	1997-07-14	0.1161459982395172	0.1247399970889091	0.10572899878025051	0.1078130006790161
11	1997-07-21	0.1088540032505989	0.1166670024394989	0.1031249985098838	0.1114580035209655
12	1997-07-28	0.1114580035209655	0.1252599954605102	0.1109379976987838	0.12083300203084939
13	1997-08-04	0.1187499965098838	0.1205729991197586	0.10520800203084939	0.1145830005407333
14	1997-08-11	0.1145830005407333	0.1166670024394989	0.0968749970197677	0.10572899878025051
15	1997-08-18	0.10260400176048272	0.1104170009498287	0.09843800216913219	0.10625000029802322
16	1997-08-25	0.10520800203084939	0.1197919945926661	0.10520800203084939	0.1169269979000091
17	1997-09-01	0.1171879918890001	0.133329975605011	0.11562500149011609	0.125
18	1997-09-08	0.1265629976987838	0.1848960071802139	0.125	0.1843750029802322

## Monthly:

The screenshot shows the Google Cloud BigQuery Explorer interface. On the left, the sidebar includes sections for Analysis, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Migration, Assessment, SQL translation, Administration, Monitoring, Capacity management, BI Engine, and Release notes. The main area displays the 'public\_monthly' dataset under the 'MAANGdata' project. The schema table shows columns: Row, volume, company, datastream\_metadata.uuid, dat\_source\_timestamp, and is\_deleted. The data table lists 34 rows of data for the 'public\_monthly' table, with the first few rows shown below:

Row	volume	company	datastream_metadata.uuid	dat_source_timestamp	is_deleted
1	108431948	AMAZON	11740cc0-896f-4a7f-b586-35a9...	1700691122375	false
18	13841858	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
19	14484360000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
20	12926976000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
21	6362920000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
22	6620200000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
23	16102295	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
24	6075356000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
25	7294948000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
26	21316284	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
27	77730080000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
28	9181716000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
29	17683716	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
30	6432336000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
31	5086230000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
32	5621404000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
33	4929522000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
34	5367432	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
35	5244180000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
36	4149154000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false
37	3131692000	AMAZON	77f4cd0-896f-4a7f-b586-35a9...	1700691122375	false

## Analytical Queries:

- Calculate the average closing price for last week for Apple:

The screenshot shows the Google Cloud BigQuery Explorer interface. The sidebar includes sections for Analysis, Data transfers, Scheduled queries, Analytics Hub, Dataform, Partner Centre, Migration, Assessment, SQL translation, Administration, Monitoring, Capacity management, BI Engine, and Release notes. The main area displays an analytical query named 'Untitled 2' in the 'public\_weekly' dataset under the 'MAANGdata' project. The query is as follows:

```

1 SELECT company, AVG(CAST(close AS FLOAT64)) AS average_closing_price
2 FROM `alien-grove-405422.MAANGdata.public_weekly`
3 WHERE company = 'APPLE'
4 AND date >= DATE_SUB(CURRENT_DATE(), INTERVAL EXTRACT(DAYOFWEEK FROM CURRENT_DATE()) + 6 DAY)
5 AND date < DATE_SUB(CURRENT_DATE(), INTERVAL EXTRACT(DAYOFWEEK FROM CURRENT_DATE()) - 1 DAY)
6 group by 1;
7

```

The results table shows the average closing price for Apple:

company	average_closing_price
APPLE	189.9700012207...

2. What is the opening stock price for Meta in the month of October?

```

16
17 SELECT company, open
18 FROM `alien-grove-405422.MAANGdata.public_monthly`
19 WHERE company = 'META'
20 AND EXTRACT(MONTH FROM date) = 10
21 AND EXTRACT(YEAR FROM date) = EXTRACT(YEAR FROM CURRENT_DATE());
22

```

**Query results**

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION
Row	company	open				
1	META	302.739990234375				

3. Which is the lowest trading price and company in the month of November?

```

14
15 WITH MonthlyLowestPrices AS (
16   SELECT company, MIN(low) AS lowest_trading_price
17   FROM `alien-grove-405422.MAANGdata.public_monthly`
18   WHERE EXTRACT(MONTH FROM date) = 11
19   AND EXTRACT(YEAR FROM date) = EXTRACT(YEAR FROM CURRENT_DATE())
20   GROUP BY company
21 )
22
23 SELECT company, lowest_trading_price
24 FROM MonthlyLowestPrices
25 ORDER BY lowest_trading_price ASC
26 LIMIT 1;
27
28

```

### Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUT
Row	company	lowest_trading_price				
1	GOOGLE	124.9250030517578				

4. On the 27<sup>th</sup> of November, what were the top 3 stocks sold and by which company?

The screenshot shows a data warehousing tool interface. At the top, there are buttons for 'RUN', 'SAVE QUERY', 'DOWNLOAD', 'SHARE', and a help icon. Below this is a code editor window containing the following SQL query:

```
30
31 SELECT company, SUM(volume) AS total_volume
32 FROM `alien-grove-405422.MAANGdata.public_daily`
33 WHERE DATE(date) = DATE('2023-11-27')
34 GROUP BY company
35 ORDER BY total_volume DESC
36 LIMIT 3;
37
38
39
40
41
42
43
44
```

Below the code editor is a section titled 'Query results' with tabs for 'JOB INFORMATION', 'RESULTS', 'CHART', 'PREVIEW' (which is selected), and 'JSON'. The results table displays the following data:

Row	company	total_volume
1	AMAZON	53666700
2	APPLE	40500500
3	GOOGLE	17868000