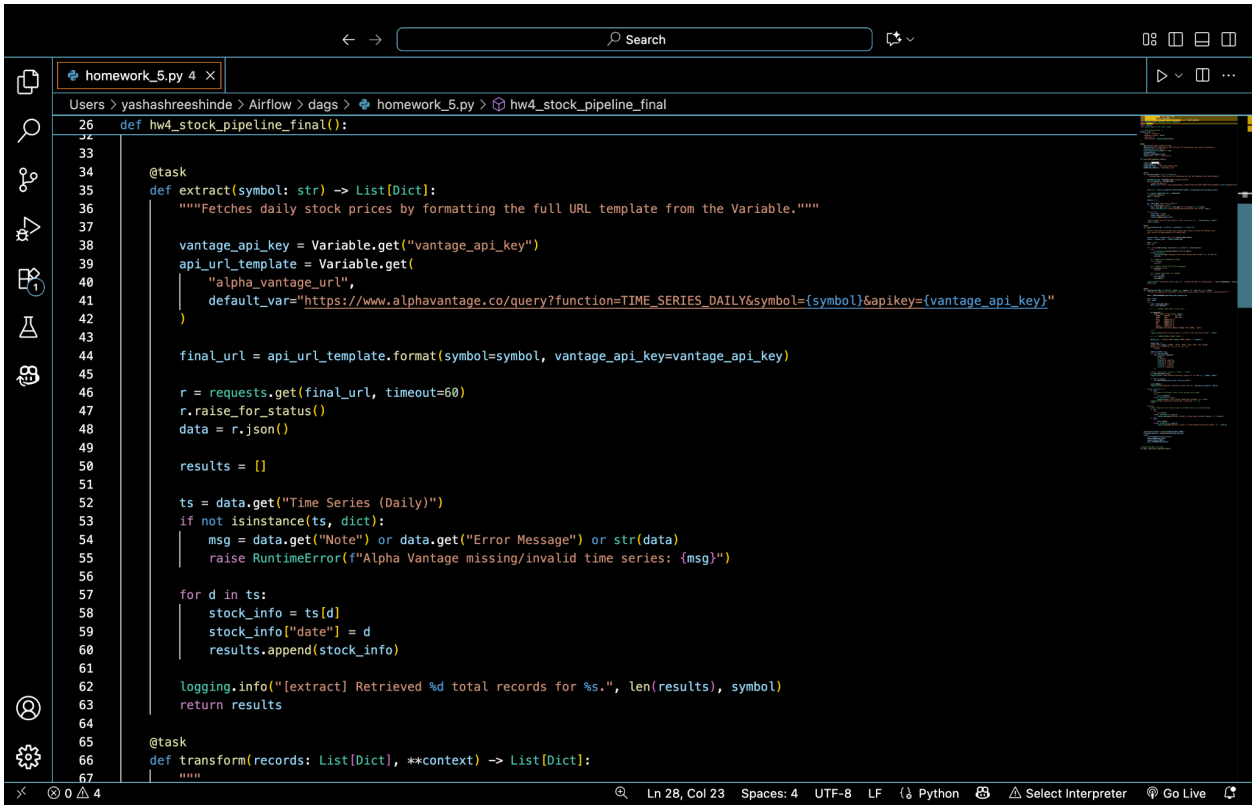
Homework 5

Name - Yashashree Shinde

Porting homework #4 to Airflow

1) **Create tasks using @task decorator**
   TASK-1



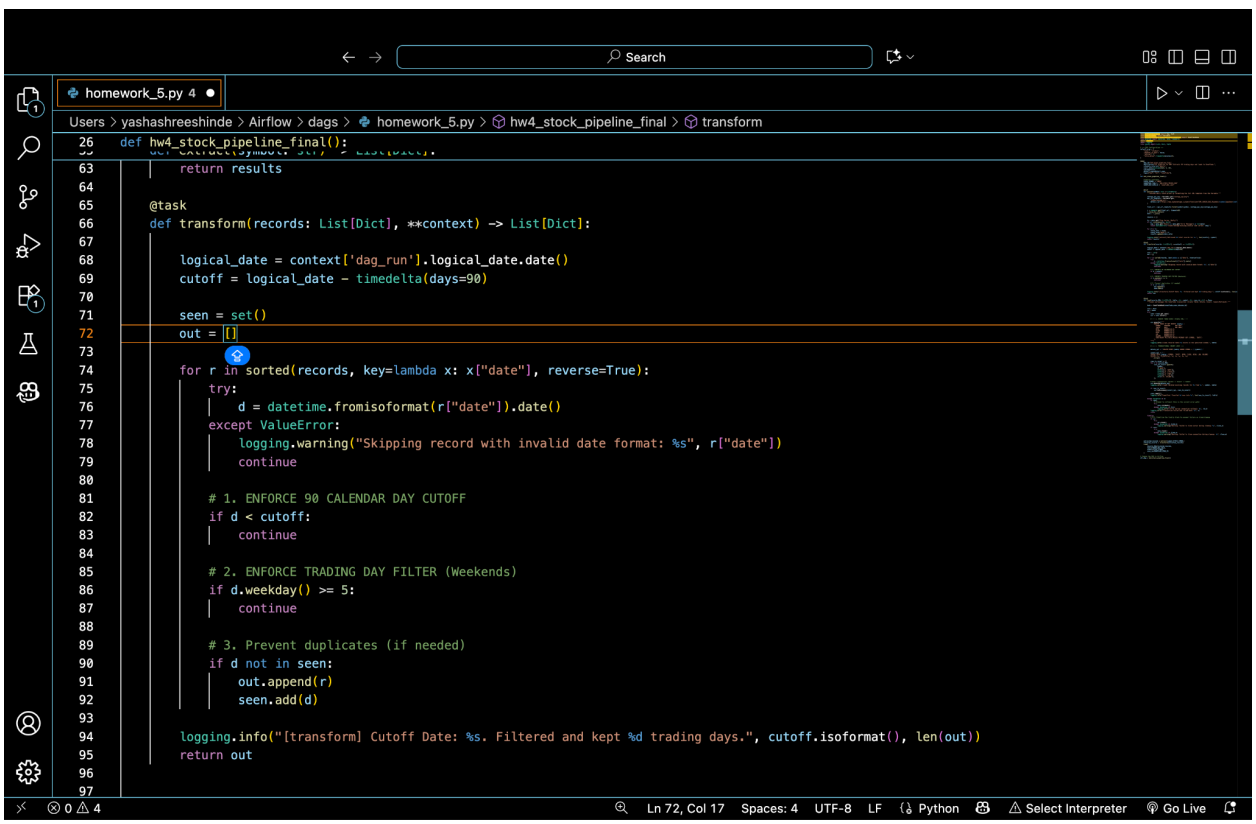

```
def hw4_stock_pipeline_final():


    @task
    def extract(symbol: str) -> List[Dict]:
        """Fetches daily stock prices by formatting the full URL template from the Variable."""

        vantage_api_key = Variable.get("vantage_api_key")
        api_url_template = Variable.get(
            "alpha_vantage_url",
            default_var="https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}"
        )

        final_url = api_url_template.format(symbol=symbol, vantage_api_key=vantage_api_key)

        r = requests.get(final_url, timeout=60)
        r.raise_for_status()
        data = r.json()

        results = []

        ts = data.get("Time Series (Daily)")
        if not isinstance(ts, dict):
            msg = data.get("Note") or data.get("Error Message") or str(data)
            raise RuntimeError(f"Alpha Vantage missing/invalid time series: {msg}")

        for d in ts:
            stock_info = ts[d]
            stock_info["date"] = d
            results.append(stock_info)

        logging.info("[extract] Retrieved %d total records for %s.", len(results), symbol)
        return results

    @task
    def transform(records: List[Dict], **context) -> List[Dict]:
        """
```

CODE

```
def extract(symbol: str) -> List[Dict]:
    """Fetches daily stock prices by formatting the full URL template from the Variable."""

    vantage_api_key = Variable.get("vantage_api_key")
    api_url_template = Variable.get(
        "alpha_vantage_url",

default_var="https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}"
    )

    final_url = api_url_template.format(symbol=symbol, vantage_api_key=vantage_api_key)
```

```
    r = requests.get(final_url, timeout=60)
    r.raise_for_status()
    data = r.json()

    results = []
    ts = data.get("Time Series (Daily)")
    if not isinstance(ts, dict):
        msg = data.get("Note") or data.get("Error Message") or str(data)
        raise RuntimeError(f"Alpha Vantage missing/invalid time series: {msg}")

    for d in ts:
        stock_info = ts[d]
        stock_info["date"] = d
        results.append(stock_info)
    logging.info("[extract] Retrieved %d total records for %s.", len(results), symbol)
    return results
```

TASK-2

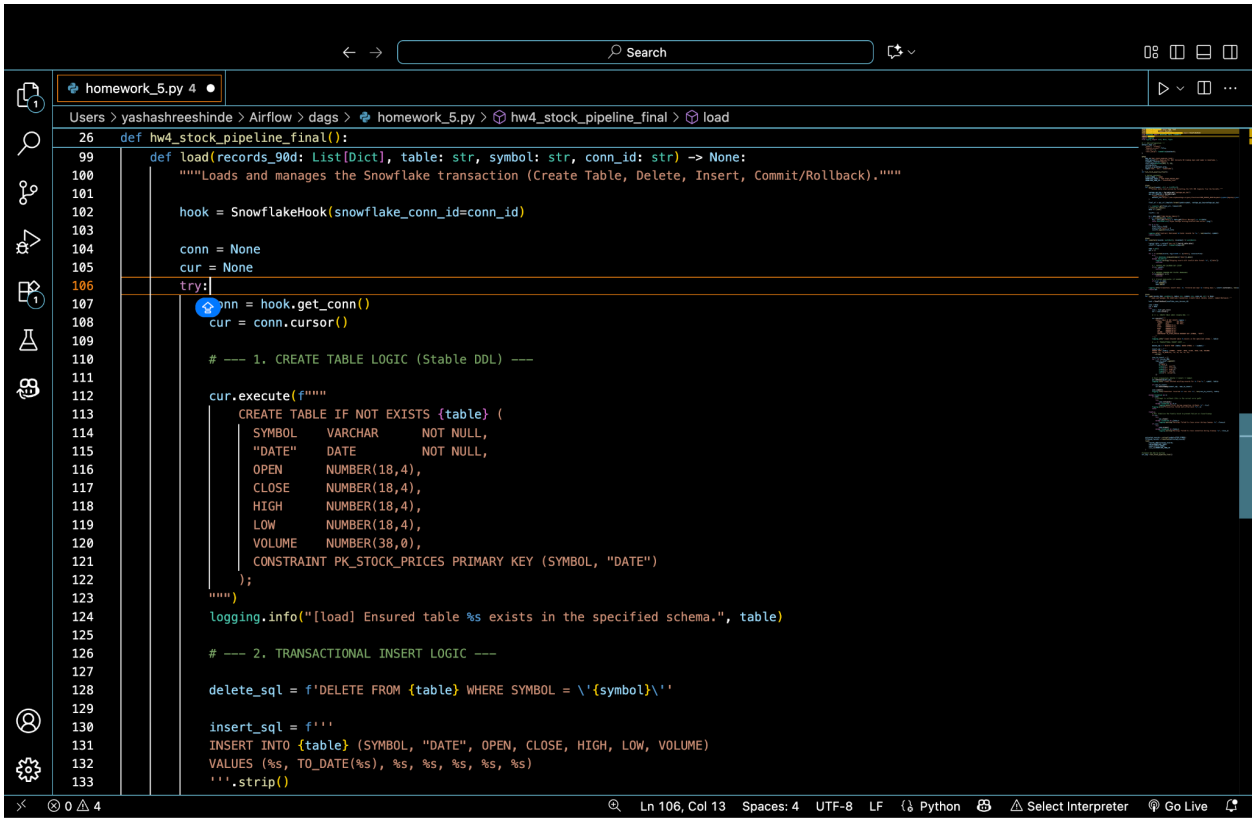


CODE

```
@task
  def transform(records: List[Dict], **context) -> List[Dict]:
    logical_date = context['dag_run'].logical_date.date()
```

```
    cutoff = logical_date - timedelta(days=90)

    seen = set()
    out = []

    for r in sorted(records, key=lambda x: x["date"], reverse=True):
        try:
            d = datetime.fromisoformat(r["date"]).date()
        except ValueError:
            logging.warning("Skipping record with invalid date format: %s", r["date"])
            continue

        # 1. ENFORCE 90 CALENDAR DAY CUTOFF
        if d < cutoff:
            continue

        # 2. ENFORCE TRADING DAY FILTER (Weekends)
        if d.weekday() >= 5:
            continue

        # 3. Prevent duplicates (if needed)
        if d not in seen:
            out.append(r)
            seen.add(d)
    logging.info("[transform] Cutoff Date: %s. Filtered and kept %d trading days.",
cutoff.isoformat(), len(out))
    return out
```

TASK-3



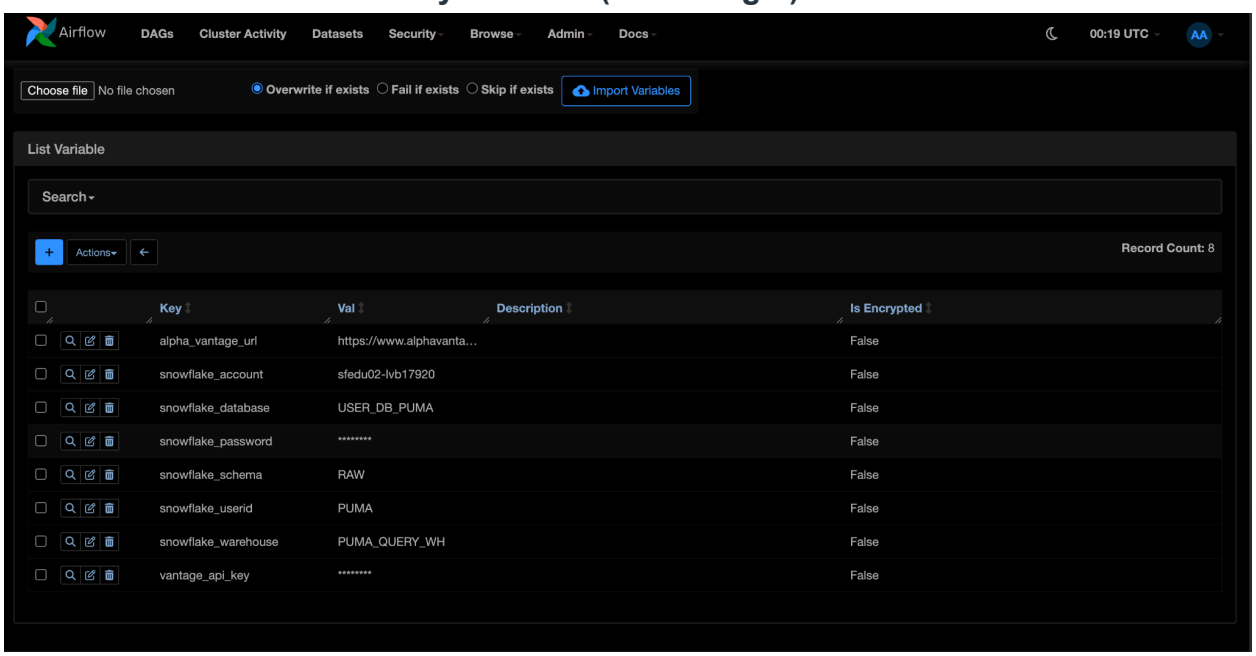
CODE

```
@task
  def load(records_90d: List[Dict], table: str, symbol: str, conn_id: str) -> None:
      """Loads and manages the Snowflake transaction (Create Table, Delete, Insert,
Commit/Rollback)."""

      hook = SnowflakeHook(snowflake_conn_id=conn_id)

      conn = None
      cur = None
      try:
         conn = hook.get_conn()
         cur = conn.cursor()

         # --- 1. CREATE TABLE LOGIC (Stable DDL) ---

         cur.execute(f"""
            CREATE TABLE IF NOT EXISTS {table} (
              SYMBOL    VARCHAR      NOT NULL,
              "DATE"    DATE         NOT NULL,
              OPEN      NUMBER(18,4),
              CLOSE     NUMBER(18,4),
              HIGH      NUMBER(18,4),
```

```
        LOW       NUMBER(18,4),
        VOLUME    NUMBER(38,0),
        CONSTRAINT PK_STOCK_PRICES PRIMARY KEY (SYMBOL, "DATE")
      );
    """)
    logging.info("[load] Ensured table %s exists in the specified schema.", table)

    # --- 2. TRANSACTIONAL INSERT LOGIC ---

    delete_sql = f'DELETE FROM {table} WHERE SYMBOL = \'{symbol}\'

    insert_sql = f'''
    INSERT INTO {table} (SYMBOL, "DATE", OPEN, CLOSE, HIGH, LOW, VOLUME)
    VALUES (%s, TO_DATE(%s), %s, %s, %s, %s, %s)
    '''.strip()
    rows_to_insert = []
    for r in records_90d:
        rows_to_insert.append((
            symbol,
            r["date"],
            float(r["1. open"]),
            float(r["4. close"]),
            float(r["2. high"]),
            float(r["3. low"]),
            int(r["5. volume"]),
        ))

    # Start transaction: Delete -> Insert -> Commit
    cur.execute(delete_sql)
    logging.info("[load] Deleted existing records for %s from %s.", symbol, table)
    if rows_to_insert:
        cur.executemany(insert_sql, rows_to_insert)

    conn.commit()
    logging.info("Committed. Inserted %d rows into %s", len(rows_to_insert), table)

except Exception as e:
    if conn:
        # Attempt to rollback (this is the correct error path)
        try:
            conn.rollback()
        except Exception as rb_e:
            logging.error("Error during connection rollback: %s", rb_e)
    logging.error("Transaction failed and rolled back: %s", e)
    raise

finally:
    if cur:
        try:
```

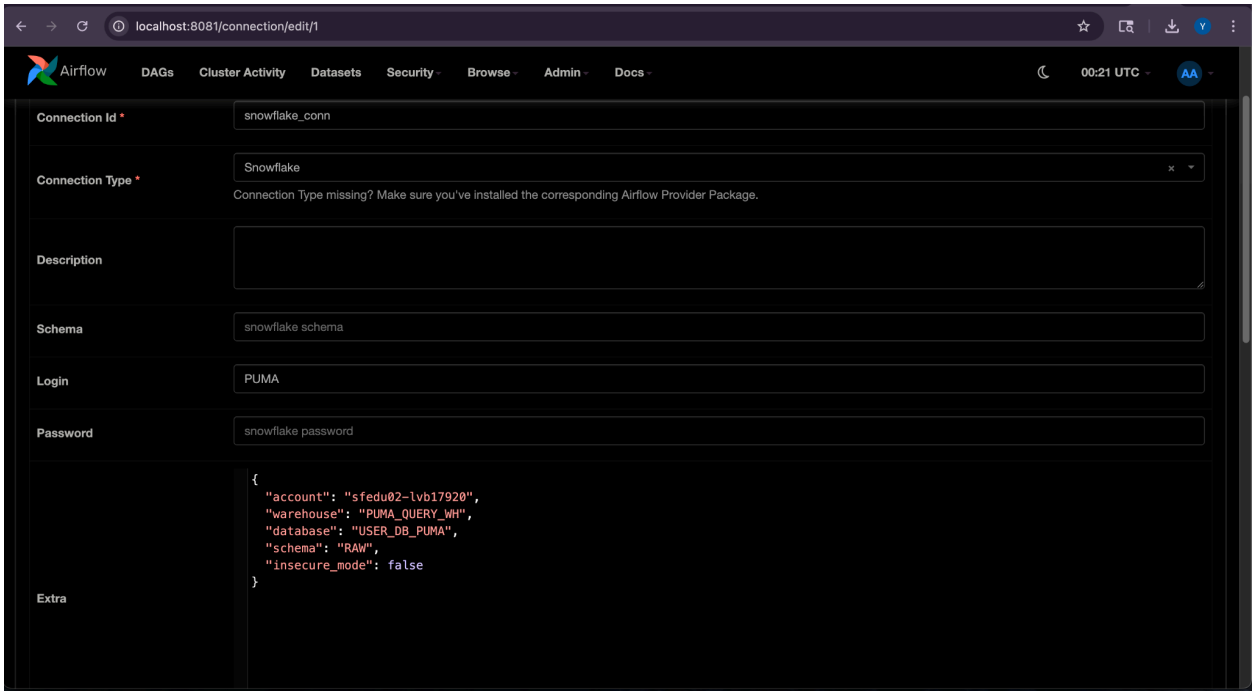
```
            cur.close()
        except Exception as close_e:
            logging.warning("Warning: Failed to close cursor during cleanup: %s", close_e)
    if conn:
        try:
            conn.close()
        except Exception as close_e:
            logging.warning("Warning: Failed to close connection during cleanup: %s", close_e)


extracted_records = extract(symbol=STOCK_SYMBOL)
filtered_records = transform(extracted_records)
load(
    records_90d=filtered_records,
    table=SNOWFLAKE_TABLE,
    symbol=STOCK_SYMBOL,
    conn_id=SNOWFLAKE_CONN_ID
)
```

2) Set up a variable for Alpha Vantage API key
   - Use the variable in your code (Variable.get)



```python
vantage_api_key = Variable.get("vantage_api_key")
api_url_template = Variable.get(
    "alpha_vantage_url",
    default_var="https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}"
)
```

3) Set up Snowflake Connection
   - Use the connection in your code
   - Capture the Connection detail page screenshot

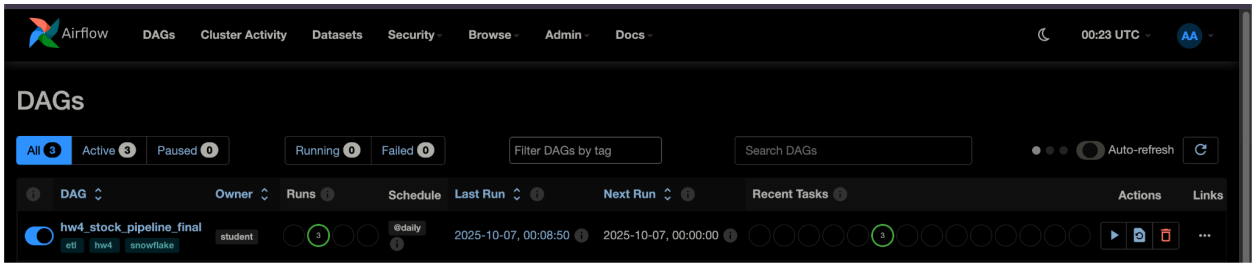



```python
@dag(
    dag_id="hw4_stock_pipeline_final",
    description="ETL pipeline for HW4: Extracts 90 trading days and loads to Snowflake.",
    schedule_interval="@daily",
    start_date=datetime(2025, 5, 30),
    catchup=False,
    default_args=default_args,
    tags=["hw4", "etl", "snowflake"],
)
def hw4_stock_pipeline_final():

    # Define constants
    STOCK_SYMBOL = "AMZN"
    SNOWFLAKE_TABLE = "RAW.STOCK_PRICES_DAG"
    SNOWFLAKE_CONN_ID = "snowflake_conn"
```

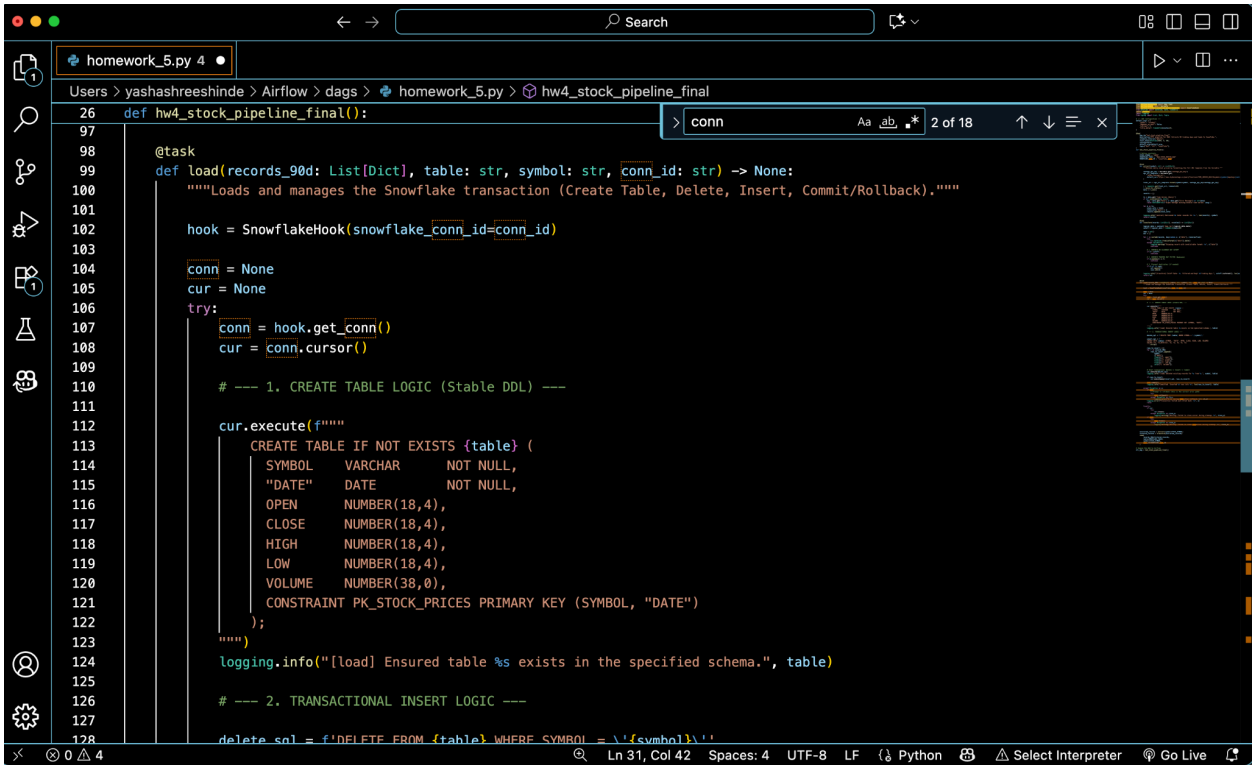
4) Ensure the overall DAG is implemented properly and runs successfully

- **A github link with the entire code needs to be submitted**
- **Implement the same full refresh using SQL transaction**

GITHUB LINK - https://github.com/yashashree5/DATA-226-Data-Warehouse/blob/main/Homeworks/Homework5/



Sql transactions



homework_5.py 4

Users > yashashreeshinde > Airflow > dags > homework_5.py > hw4_stock_pipeline_final

conn 2 of 18

```python
def hw4_stock_pipeline_final():
    def load(records_90d: List[Dict], table: str, symbol: str, conn_
        # --- 2. TRANSACTIONAL INSERT LOGIC ---

        delete_sql = f'DELETE FROM {table} WHERE SYMBOL = \'{symbol}\''

        insert_sql = f'''
        INSERT INTO {table} (SYMBOL, "DATE", OPEN, CLOSE, HIGH, LOW, VOLUME)
        VALUES (%s, TO_DATE(%s), %s, %s, %s, %s, %s)
        '''.strip()

        rows_to_insert = []
        for r in records_90d:
            rows_to_insert.append((
                symbol,
                r["date"],
                float(r["1. open"]),
                float(r["4. close"]),
                float(r["2. high"]),
                float(r["3. low"]),
                int(r["5. volume"]),
            ))

        # Start transaction: Delete -> Insert -> Commit
        cur.execute(delete_sql)
        logging.info("[load] Deleted existing records for %s from %s.", symbol, table)

        if rows_to_insert:
            cur.executemany(insert_sql, rows_to_insert)

        conn.commit()
        logging.info("Committed. Inserted %d rows into %s", len(rows_to_insert), table)
```

0 4 Ln 31, Col 42 Spaces: 4 UTF-8 LF Python Select Interpreter Go Live

homework_5.py 4

Users > yashashreeshinde > Airflow > dags > homework_5.py > hw4_stock_pipeline_final > load

conn 2 of 18

```python
def hw4_stock_pipeline_final():
    def load(records_90d: List[Dict], table: str, symbol: str, conn_id: str) -
        cur.executemany(insert_sql, rows_to_insert)

        conn.commit()
        logging.info("Committed. Inserted %d rows into %s", len(rows_to_insert), table)

    except Exception as e:
        if conn:
            # Attempt to rollback (this is the correct error path)
            try:
                conn.rollback()
            except Exception as rb_e:
                logging.error("Error during connection rollback: %s", rb_e)
        logging.error("Transaction failed and rolled back: %s", e)
        raise

    finally:
        if cur:
            try:
                cur.close()
            except Exception as close_e:
                logging.warning("Warning: Failed to close cursor during cleanup: %s", close_e)
        if conn:
            try:
                conn.close()
            except Exception as close_e:
                logging.warning("Warning: Failed to close connection during cleanup: %s", close_e)



extracted_records = extract(symbol=STOCK_SYMBOL)
filtered_records = transform(extracted_records)
load(
    records_90d=filtered_records,
    table=SNOWFLAKE_TABLE,
```

0 4 Ln 144, Col 41 Spaces: 4 UTF-8 LF Python Select Interpreter Go Live

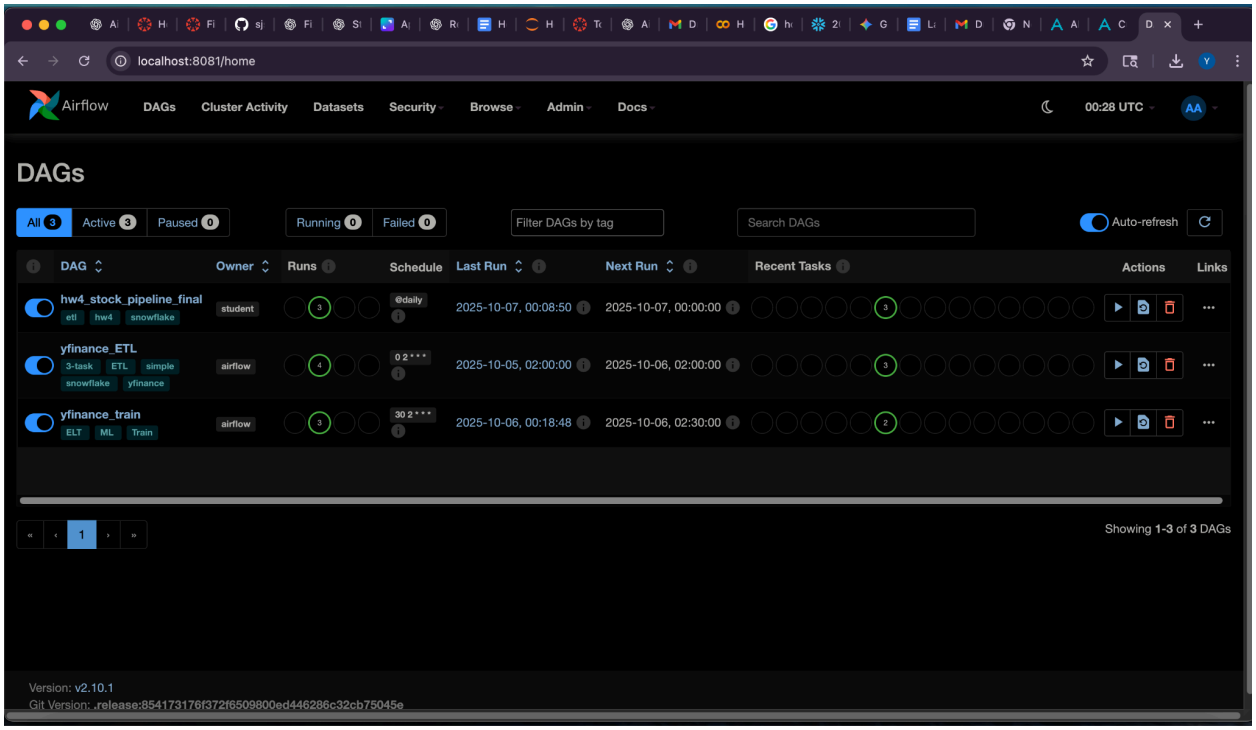
CODE

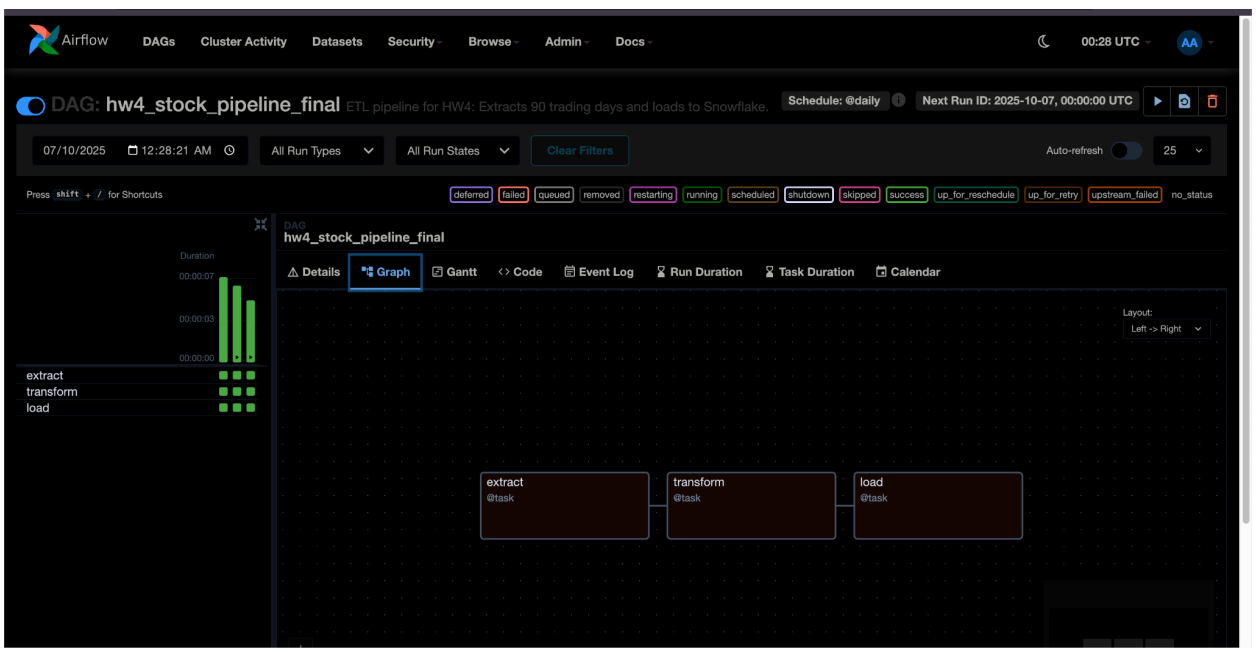
```
def load(records_90d: List[Dict], table: str, symbol: str, conn_id: str) -> None:
    """Loads and manages the Snowflake transaction (Create Table, Delete, Insert,
Commit/Rollback)."""

    hook = SnowflakeHook(snowflake_conn_id=conn_id)

    conn = None
    cur = None
    try:
        conn = hook.get_conn()
        cur = conn.cursor()

        # --- 1. CREATE TABLE LOGIC (Stable DDL) ---

        cur.execute(f"""
            CREATE TABLE IF NOT EXISTS {table} (
              SYMBOL    VARCHAR      NOT NULL,
              "DATE"    DATE         NOT NULL,
              OPEN      NUMBER(18,4),
              CLOSE     NUMBER(18,4),
              HIGH      NUMBER(18,4),
              LOW       NUMBER(18,4),
              VOLUME    NUMBER(38,0),
              CONSTRAINT PK_STOCK_PRICES PRIMARY KEY (SYMBOL, "DATE")
            );
        """)
        logging.info("[load] Ensured table %s exists in the specified schema.", table)

        # --- 2. TRANSACTIONAL INSERT LOGIC ---

        delete_sql = f'DELETE FROM {table} WHERE SYMBOL = \'{symbol}\'

        insert_sql = f'''
        INSERT INTO {table} (SYMBOL, "DATE", OPEN, CLOSE, HIGH, LOW, VOLUME)
        VALUES (%s, TO_DATE(%s), %s, %s, %s, %s, %s)
        '''.strip()
        rows_to_insert = []
        for r in records_90d:
            rows_to_insert.append((
                symbol,
                r["date"],
                float(r["1. open"]),
                float(r["4. close"]),
                float(r["2. high"]),
                float(r["3. low"]),
                int(r["5. volume"]),
            ))

        # Start transaction: Delete -> Insert -> Commit
```

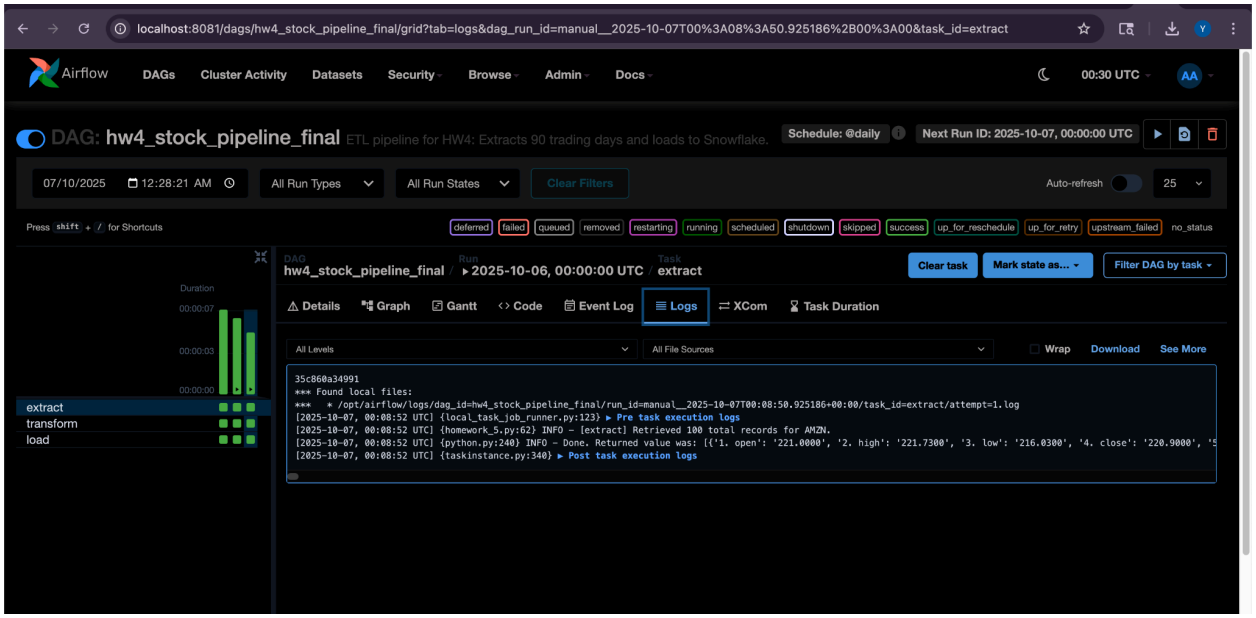
```
        cur.execute(delete_sql)
        logging.info("[load] Deleted existing records for %s from %s.", symbol, table)
        if rows_to_insert:
            cur.executemany(insert_sql, rows_to_insert)

        conn.commit()
        logging.info("Committed. Inserted %d rows into %s", len(rows_to_insert), table)

    except Exception as e:
        if conn:
            # Attempt to rollback (this is the correct error path)
            try:
                conn.rollback()
            except Exception as rb_e:
                logging.error("Error during connection rollback: %s", rb_e)
        logging.error("Transaction failed and rolled back: %s", e)
        raise

    finally:
        if cur:
            try:
                cur.close()
            except Exception as close_e:
                logging.warning("Warning: Failed to close cursor during cleanup: %s", close_e)
        if conn:
            try:
                conn.close()
            except Exception as close_e:
                logging.warning("Warning: Failed to close connection during cleanup: %s", close_e)
```

5) Capture two screenshot of your Airflow Web UI (examples to follow)

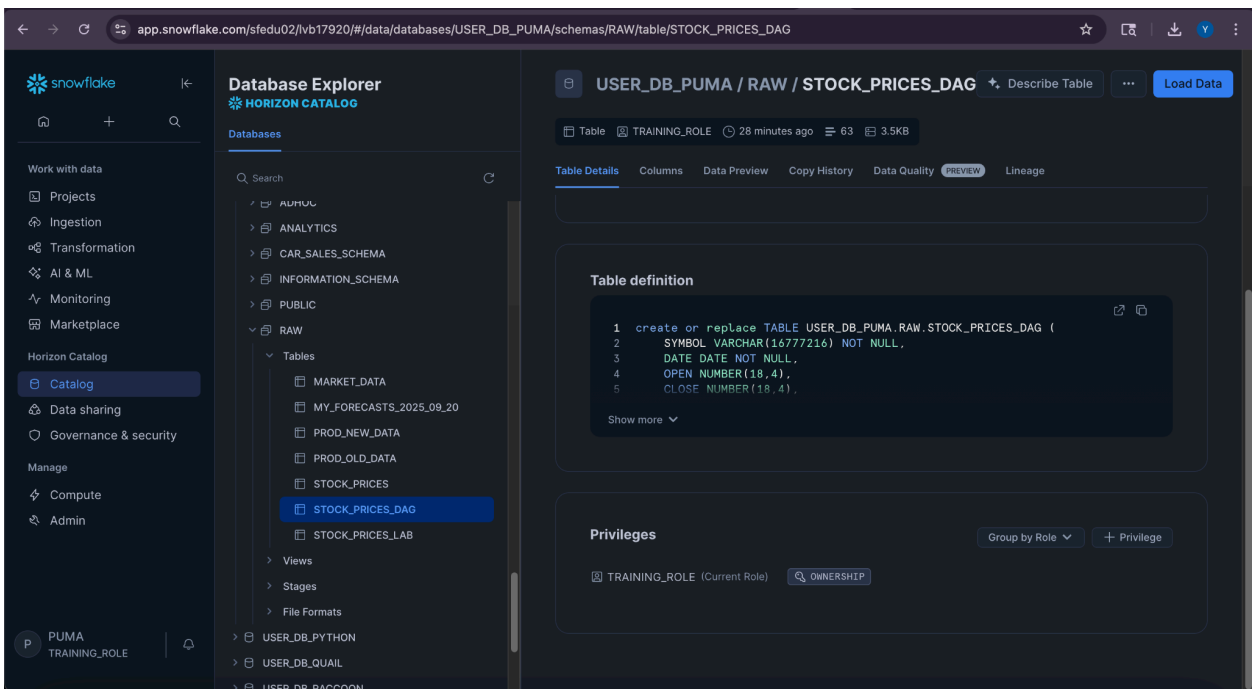
- One with the Airlow homepage showing the DAG
- The other with the log screen of the DAG

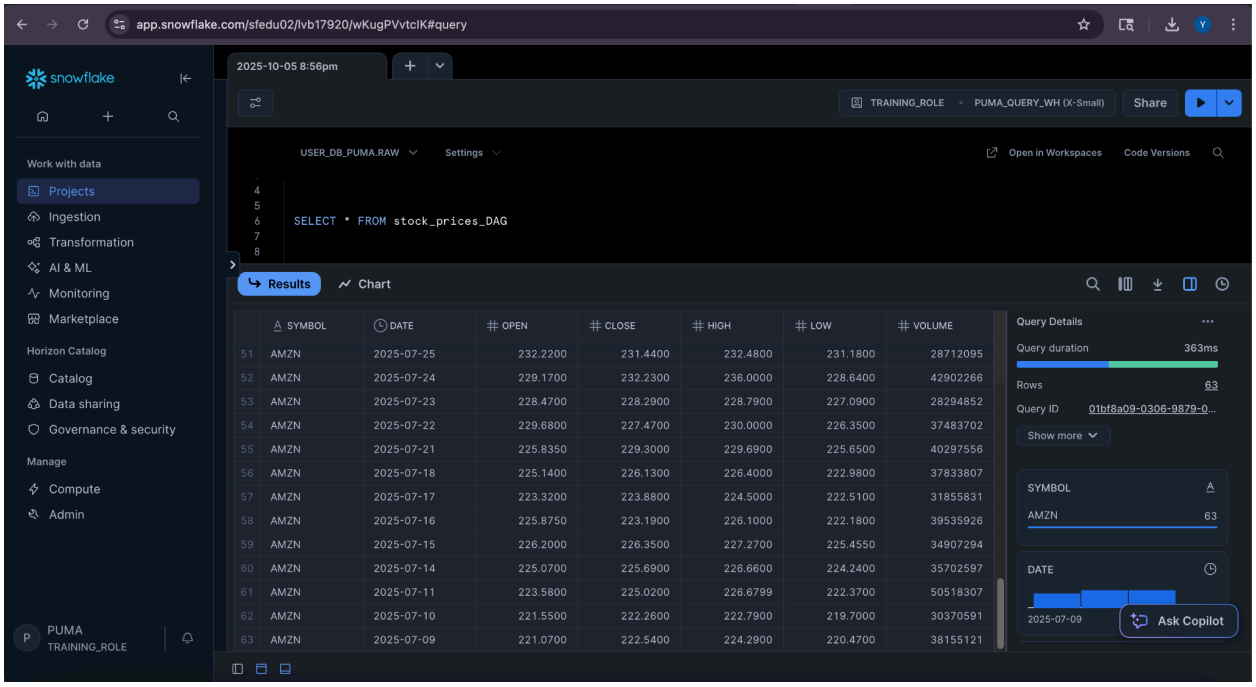

localhost:8081/home
Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
00:28 UTC
DAGs
All 3
Active 3
Paused 0
Running 0
Failed 0
Filter DAGs by tag
Search DAGs
Auto-refresh
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
Actions
Links
hw4_stock_pipeline_final
etl
hw4
snowflake
student
@daily
2025-10-07, 00:08:50
2025-10-07, 00:00:00
yfinance_ETL
3-task
ETL
simple
snowflake
yfinance
airflow
0 2 * * *
2025-10-05, 02:00:00
2025-10-06, 02:00:00
yfinance_train
ELT
ML
Train
airflow
30 2 * * *
2025-10-06, 00:18:48
2025-10-06, 02:30:00
Showing 1-3 of 3 DAGs
Version: v2.10.1
Git Version: .release:854173176f372f6509800ed446286c32cb75045e


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
00:28 UTC
DAG: hw4_stock_pipeline_final
ETL pipeline for HW4: Extracts 90 trading days and loads to Snowflake.
Schedule: @daily
Next Run ID: 2025-10-07, 00:00:00 UTC
07/10/2025
12:28:21 AM
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:00:07
00:00:03
00:00:00
extract
transform
load
DAG
hw4_stock_pipeline_final
Details
Graph
Gantt
Code
Event Log
Run Duration
Task Duration
Calendar
Layout:
Left -> Right
extract
@task
transform
@task
load
@task



SNOWFLAKE VALIDATIONS





	SYMBOL	DATE	OPEN	CLOSE	HIGH	LOW	VOLUME
51	AMZN	2025-07-25	232.2200	231.4400	232.4800	231.1800	28712095
52	AMZN	2025-07-24	229.1700	232.2300	236.0000	228.6400	42902266
53	AMZN	2025-07-23	228.4700	228.2900	228.7900	227.0900	28294852
54	AMZN	2025-07-22	229.6800	227.4700	230.0000	226.3500	37483702
55	AMZN	2025-07-21	225.8350	229.3000	229.6900	225.6500	40297556
56	AMZN	2025-07-18	225.1400	226.1300	226.4000	222.9800	37833807
57	AMZN	2025-07-17	223.3200	223.8800	224.5000	222.5100	31855831
58	AMZN	2025-07-16	225.8750	223.1900	226.1000	222.1800	39535926
59	AMZN	2025-07-15	226.2000	226.3500	227.2700	225.4550	34907294
60	AMZN	2025-07-14	225.0700	225.6900	226.6600	224.2400	35702597
61	AMZN	2025-07-11	223.5800	225.0200	226.6799	222.3700	50518307
62	AMZN	2025-07-10	221.5500	222.2600	222.7900	219.7000	30370591
63	AMZN	2025-07-09	221.0700	222.5400	224.2900	220.4700	38155121