Algorithmic Trading Subsystems Data Collection and Transformation Process Automation Using DRI for Machine Learning Modelling and Forecasting

Justin Drenka
Computer Science
UBCO
Kelowna, Canada
0009-0008-6821-8379

Helana Jaraiseh
Computing Science
UFV
Abbotsford, Canada
0009-0000-1239-2820

James Midtdal
Computer Science
Okanagan College
Kelowna, Canada
0009-0005-6312-569X

Kristina Cormier
Computer Science
Okanagan College
Kelowna, Canada
0009-0004-3783-9704

Youry Khmelevsky
Computer Science
Okanagan College
Kelowna, Canada
0000-0002-6837-3490

Brandon Hay
Computer Science
Okanagan College
Kelowna, Canada
0009-0001-6605-6037

***Abstract*—This research paper focuses on developing an automated data collection and transformation framework for Algorithmic Trading (AT) forecasting using machine learning (ML). The implementation of the computational resources is supported by a grant from the Digital Research Alliance of Canada (DRAC), Alliance Cloud Connect Pilot [1]. This enables large-scale data processing, storage, and model training. Historical market data and real-time data streams are collected at 5-minute intervals, stored in staging tables of a PostgreSQL DBMS, and aggregated into 15-minute intervals in a custom-built Data Warehouse. It is essential to ensure that data from multiple sources remains consistent, accurate, and reliably reproducible throughout the data pipeline.**

***Index Terms*—Algorithmic Trading, Machine Learning Modelling, System Integration, Performance Optimization**

I. Introduction

The applied research paper discusses solutions for automatically acquiring complete historical and real-time datasets, as well as the significant storage and computational resources required to train a machine learning model for short-term Algorithmic Trading predictions. The focus during this phase of the project is on automating data collection and transformation.

The Alliance Cloud Connect Pilot allocated 200 core-years on the NIBI-compute system, 20.0 RGU-years on the nibi-gpu system, and 59 TB of project storage on the NIBI storage system for High Performance Computing (HPC). It also allocates 16 VCPU-years, 4 cloud instances, 32 GB of RAM, 7 volumes, 7 snapshots, 2 Floating IP addresses, and 60,000 GB of cloud volume and snapshot storage for the cloud allocations on the Arbutus-persistent-cloud system.

After a project is approved and resources are allocated, a few more steps are required to access them. One of the first steps is to set up Secure Shell (SSH) keys to access the resources. Another critical step was to set up Multi-factor Authentication (MFA), which is used when signing in to DRAC and when using the SSH tunnel to access the server. Furthermore, users must email DRAC to request access to specific resources, specify the required quantity, and wait for a reply before connecting to the dedicated PostgreSQL database.

With the provided resources, the data could be collected for the past thirty years. The previous work only included the three previous years. The new dataset is ten times larger than the previous datasets described in [2]–[8].

One of our biggest challenges is that many previous datasets are incomplete, leading to unexpected results with machine learning models. The current work focuses on updating missing values and expanding the dataset obtained from the data sources automatically.

The main contributions of this paper are (1) a new approach to data collection from external data sources for XGBoost training, testing, and stock forecasting; (2) the new and historical datasets uploaded to a PostgreSQL general purpose DBMS within DRI research project; and (3) the design and testing of a new API for direct access to the DBMS from the core subsystem as part of machine learning (ML) training, testing, and forecasting on the vast datasets collected, transformed, and stored in the DBMS.

This work is a continuation of the previous research projects described in [9]–[24]

II. Existing Works

In our previous papers, we analyzed existing work on algorithmic trading systems and ML algorithms for predicting stock prices, such as Al-Akashi and Hassan [25], Kolte et al. [26], and Li [27], and related statistical methods for prediction accuracy. R. K. Dubey in "Algorithmic Trading: The Intelligent Trading Systems and Its Impact on Trade Size" [28] reported about "lack of progress in developing an algorithmic trading system that is operationally efficient." "The main issues remain in acquiring, transforming, and storing the large datasets required for such a system" [29]. Many algorithmic trading and ML forecasting projects were limited by the size of the data set during model development and by the lack of system support for implementing the developed models [30].

Yulianto in [31] stressed that the "heterogeneity of data from various sources can be dealt with by designing an Extract, Clean, Conform, and Delivery/Load" process. Azeroual et al. [32] stated that implementing the ETL process could overcome many current challenges in data analysis. Oyewale et al. [33] informed that "market complexity and volatility, along with the volume and speed at which financial data is generated, shape the challenges surrounding stock market analysis." On the other hand, Haryono et al. [34] stressed that "ETL (Extract/Transform/Load) and ELT (Extract/Load/Transform) are the primary data processing methods for implementing a DW." Katari and Rodwal [35] identified issues in traditional ETL processes in financial markets. Our research over the last few years has confirmed that the ELT process is preferable to ETL.

Patel et al. in [36] outlined ETL tools, some of which are "code-based, GUI-based, cloud-based, Metadata support, Real-time support, and batch processing. Michael et al. discuss ETL systems for building a DW in their research. Biswas et al. in [37] described commercially available GUI-based products as ETL solutions. Ali [38] created an ETL framework for Big Data. On the other hand, relatively few papers investigate the complete data life cycle [39]. Wu et al. [40] proposed using Apache Airflow with Python scripts to automate ETL tasks. The previous analyses on several implementation techniques were documented in [8].

Based on the described works above and our research team's testing and observations, we use a well-established, proven object-oriented approach to design an efficient ELT process for this research project. The Python scripts, PostgreSQL DW, and a user interface (UI) are used to improve the performance of the ELT process.

III. A New Approach for Data Collection Automation using DRI

A. Overview of the Auto Data Collector System

To achieve accurate short-term forecasting, the model must always be aware of recent market activity. The automated data collection system on Fig. 1 is designed to provide this by continuously updating the model dataset with current market information.

B. Cloud Deployment and Scheduling

The automated collector lives on a virtual machine instance within the Arbutus Cloud, part of Canada's Digital Research Infrastructure (DRI) [1]. The system runs on a p4-6gb instance, which provides 4 virtual CPUs, 6 GB of RAM, and a 20 GB persistent disk. Hosting the process in the cloud enables consistent up-time, remote access, and protection against local hardware or power failures. Arbutus Cloud instances offer persistent storage, allowing the collector to run continuously without manual intervention. After about fifteen days of collection for 500 tickers, the PostgreSQL database uses roughly 100 MB of space, most of which is initial PostgreSQL table and index metadata rather than data records. This shows that the available storage is more than enough to support years of collected market data.

Job scheduling is managed using systemd timers, which run data collection at fixed intervals during standard market hours. This scheduling method ensures that no data is missed and that execution remains reliable in the event of a network or system interruption. Each run creates a status log, forming a history of performance for easy review and debugging. This combination of stable cloud infrastructure with automated scheduling supports a reliable data pipeline that stays synchronized with current market data and requires minimal operator oversight.

As shown in Listing 1, a *systemd* timer deployed on the VM triggers the collector throughout standard stock market hours.

Listing 1. Systemd timer for the Auto Data Collector

```
[Unit]
Description=Auto Data Collector Timer

# Run data collector twice per market hour
# Time in UTC on the VM
[Timer]
OnCalendar=Mon..Fri *-*-* 13:58
# Market open 09:30 EST
OnCalendar=Mon..Fri *-*-* 14:30
OnCalendar=Mon..Fri *-*-* 14:58
...
OnCalendar=Mon..Fri *-*-* 20:58
# Market closed 16:00 EST
OnCalendar=Mon..Fri *-*-* 21:30

Persistent=true
AccuracySec=1s

[Install]
WantedBy=timers.target
```

The timer triggers the service shown in Listing 2, which executes the collector script.

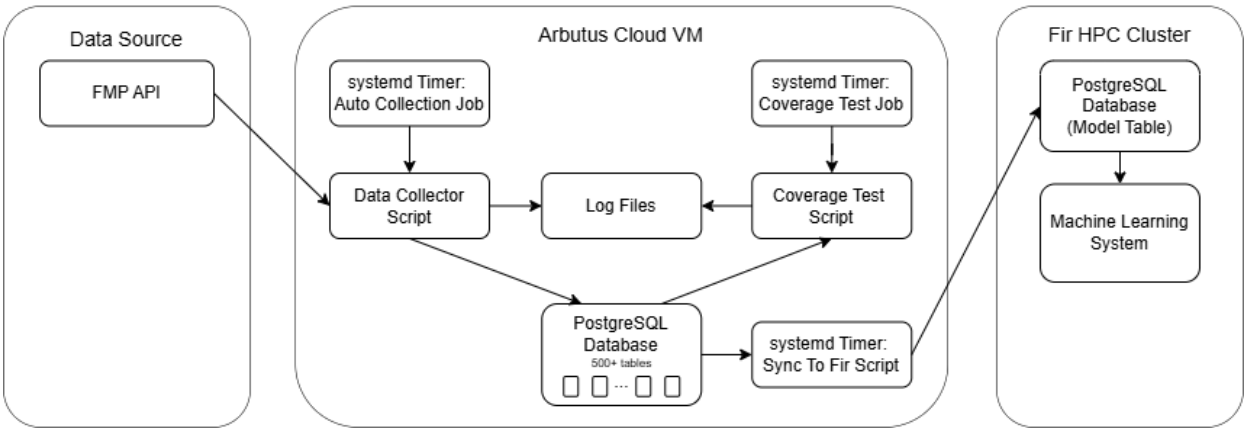

Fig. 1. Automated Data Collector Architecture and Workflow Design

Listing 2. Systemd service that runs the Auto Data Collector

```
[Unit]
Description=AutoDataCollector Service
# Ensure network is ready before running
After=network.target

[Service]
# Run the script once per timer trigger
Type=oneshot
User=almalinux
WorkingDirectory=.../AutoDataCollector/
# Entry script executed by the service
ExecStart=.../bin/run_collector.sh
# Retry if the script exits with an error
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

C. Data Collection Process

The automated data collection process is handled by a Python script that runs on the virtual machine described in Section B. The collection script is designed to be run continuously throughout standard trading hours. During each execution, a time window is created which covers the most recent trading period, with a slight overlap from the previous run to ensure no data is missed. As shown in Listing 3, the script computes a rolling one-hour EST window that ends twenty minutes before the current time, creating deliberate overlap between runs to ensure that no time interval is ever missed.

Listing 3. Time window calculation used by the Auto Data Collector

```
# Determine rolling 1-hour time window
def get_current_time_window():
    now_ny =
        datetime.now(ZoneInfo("America/New_York"))

    # End window 20 minutes before now to
        ensure no missed rows from API
    end_time = now_ny.replace(second=0,
        microsecond=0) -
        timedelta(minutes=20)
    start_time = end_time -
        timedelta(hours=1)

    # Store timestamps in EST
    return start_time.replace(tzinfo=None),
        end_time.replace(tzinfo=None)
```

Data is extracted from the Financial Modelling Prep (FMP) [2] API using this time window, and the schedule is configured so that each collection occurs shortly after new data becomes available. This keeps the database aligned with recent market activity with a slight lag.

The script uses lists of tickers so that symbols we want to track can be added or removed at any time, and the collector will adjust. This makes the system flexible and easier to maintain as data requirements evolve. Because FMP provides timestamps in EST, the script preserves this format when inserting records into the PostgreSQL database. After retrieving recent data from FMP, the results are written to the database with an "upsert" operation, where entries are updated when already present, and inserted when not present yet. Listing 4 shows the portion of the collector that filters each record into the active time window and writes it to the corresponding ticker table using an UPSERT operation.

Listing 4. Filtering and UPSERT insertion of 5-minute records

```
# Fetch, filter, and upsert 5-minute
    market data into the staging table
rows = []
for record in data:
    ts = parse_fmp_timestamp(record["date"])
    if window_start <= ts < window_end and
        record.get("close") is not None:
        rows.append((
            ts,
            record["open"],
            record["high"],
```

```
            record["low"],
            record["close"],
            record["volume"]
        ))

sql = f"""
    INSERT INTO {table} (ts, open, high,
        low, close, volume)
    VALUES %s
    ON CONFLICT (ts) DO UPDATE SET
        open=EXCLUDED.open,
            high=EXCLUDED.high,
        low=EXCLUDED.low,
            close=EXCLUDED.close,
        volume=EXCLUDED.volume;
"""
# Bulk insert all row tuples into DB
execute_values(cursor, sql, rows)
```

Running this process with *systemd* timers on the cloud-based VM ensures a stable environment for collection, keeping the historical dataset up to date, accurate, and ready for training the forecasting model.

D. Database Storage Architecture

The auto collector stores new market data in a set of staging tables in the database, with each ticker getting its own table. Each table uses OHLCV format (open, high, low, close, and volume), matching the structure provided by our data source (FMP API). These values describe how the market price moved during each time interval, and using the same format as the historical dataset keeps both sources consistent.

Every entry in the staging tables represents one 5-minute interval and is identified by its timestamp. Listing 5 shows the structure of a typical staging table, where the timestamp serves as the primary key, ensuring that overlapping collection windows do not introduce duplicates.

Listing 5. Example staging table used for a single ticker

```
-- Staging table for a single ticker (AAPL)
CREATE TABLE IF NOT EXISTS market.aapl (
    ts    TIMESTAMP PRIMARY KEY,
    open  DOUBLE PRECISION,
    high  DOUBLE PRECISION,
    low   DOUBLE PRECISION,
    close DOUBLE PRECISION,
    volume BIGINT
);
```

Choosing the timestamp as the primary key allows the collector to easily update rows when collection windows overlap, and separating each ticker into its own table avoids write conflicts when many symbols are updated at the same time. This simple layout makes the storage layer easy to maintain and reliable for continuous data collection.

The staging tables serve as the first stop for new data. Their role is to hold recent market information in the same structure as the historical dataset. In a later step of the pipeline, both the historical data and the auto-collected staging data are combined into a single model table used for training the forecasting model.

E. Testing and Logging

Testing and logging are essential to ensure the auto collector captures 100% of the expected market data for each market day. The system includes a daily coverage test that runs automatically about one hour after the market closes. Its job is to check that every 5-minute interval during standard trading hours is collected for every ticker. Listing 6 shows the core logic of the daily coverage test, which verifies that the expected number of 5-minute intervals was collected for each ticker.

Listing 6. Core logic of the daily coverage test

```
# Single ticker coverage between two dates
def coverage_for_symbol(conn, symbol,
    start_date, end_date):
    expected_per_day =
        expected_intervals_per_day()
    total_expected = 0
    total_actual = 0

    table = format_table_name(symbol)
    current = start_date

    with conn.cursor() as cur:
        while current <= end_date:
            # Skip weekends
            if current.weekday() >= 5:
                current += timedelta(days=1)
                continue

            # Intervals for one trading day
            total_expected += expected_per_day

            # Count found rows for that day
            open_dt =
                datetime.combine(current,
                time(9,30)).replace(tzinfo=None)
            close_dt =
                datetime.combine(current,
                time(16,0)).replace(tzinfo=None)

            cur.execute(
                f"SELECT COUNT(*) FROM {table}
                    WHERE ts >= %s AND ts < %s",
                (open_dt, close_dt),
            )
            total_actual += cur.fetchone()[0]

            current += timedelta(days=1)

    # Return coverage percentage
    return (total_actual / total_expected)
        * 100 if total_expected else 0
```

The test knows the collection start date and adjusts its expectations as days pass, so it continuously logs the coverage percentage for each ticker. This helped us catch several issues early, such as missing intervals, FMP API changes, and unexpected execution times. Finding these

problems quickly made it much easier for us to ensure the dataset is complete and clean.

In addition to the coverage test, the collector writes a status log for every scheduled run, noting when the job started, when it finished, how long it took and whether any errors occurred. The coverage test also records its results in the logs. Together, these tests and logs provide essential visibility into the system's performance and ensure its data remains accurate over time.

IV. API Architecture, Design and Testing to DRI DBMS

As you can see in Fig. 2, the data from Financial Model Prep was acquired into Staging Tables: Extract raw stock, index, bond, and commodity data from the Financial Model Prep API. Load raw data to individual staging tables—used the automatic collector described in Section III.

From Staging Tables, data is transferred to the Model Table: Extract, transform, and merge data, then store it in the model table. Used a Jupyter Notebook.

From Model Table to Model / User Interface: Connect Model to Table; the Model will query the Table to fetch training data. The user interface will be built in the future.

From Staging Tables to Model Table in Detail: A Jupyter Notebook will extract the raw data from the staging tables, merge and transform the data and load them into a single table for model training.

Data Cleaning: FMP's raw data contains missing entries and duplicates. The Jupyter Notebook script creates a dataframe for each dataset, removes duplicates, and uses an *** appropriate imputation method (needs further research) to fill in the missing time entries.

Data Transformation: Only categorical values are transformed during this process. Numerical values are normalized during model training.

The data collection subsystem allows downloading historical data from CSV files and directly from the FMP API, and uploading them to the staging database in PostgreSQL, thereby bypassing the CSV file (Fig. 2).

V. The Automation Process Implementation

Due to limitations of the Fir server at DRI, the virtual machine on DRI's Arbutus Cloud must execute several special scripts. Fir's *crontab* tool has been disabled, and Fir has minimum requirements on job length; Arbutus Cloud provides access to scheduled tasks and has no limitations on job length. The regularity and short runtime of many scripts require large pieces of work to be done outside the Fir database, split between Arbutus Cloud and local (personal) machines. Because the databases on Fir and Arbutus Cloud are disjoint, Fir's PostgreSQL database must be initialized with historical data, then updated with current stock information from the automatic data collector running on Arbutus Cloud.

A. Historical Stock Data Collection

The FMP API, configured for 5-minute intraday intervals, return up to 10 days of data per stock symbol per API call. Due to the large number of API calls required, a script is needed to automate data collection across the desired date range. A Python script, running on a local machine, automates this process. The script defines a start date, an end date, and stock symbols. Then, for each stock symbol, an inner loop iterates through the target range, calling the API with distinct 5-day windows. Each time the inner loop terminates, the individual stock's results are saved locally into a CSV file named with the corresponding stock symbol. The outer loop terminates when all stock symbols have been processed. The resulting CSV files serve as input to a script that checks for missing entries. This Python date validation script loads each file based on the stock symbols defined in the data collector. Exclusion dates are inserted into a list (holidays, etc.) to avoid false alerts. The start date, end date, opening time, and closing time are also defined, matching the definitions in the historical data collection script. Expected 5-minute intervals are calculated, and then actual results are compared against the generated list. Any missing values are logged to the console for the operator to investigate. To confirm the API results were correct, missing entries are double-checked with a manual API call in a web browser. Occasionally, FMP is missing data for an interval. Still, it is critical to validate that the missing entry is a fault in FMP's data collection process and is unrelated to script execution. The raw data CSVs are ready to be patched (missing entries filled with previous entries' information, interpolated, etc.) and uploaded to the database.

B. CSV File to Database Upload Automation

The CSV stock data is inserted into the Fir-hosted database using automated Python scripts, which are publicly available on GitHub [41]. An SSH Tunnel to Fir and multi-factor authentication were required to access the Fir PostgreSQL RDBMS [42], so the database is accessible. A stock CSV file is read into memory and is cast into the appropriate variable format for the database. Because using multiple INSERT commands can take over 10 minutes per stock due to overhead, the script uses PostgreSQL's COPY command, which performs bulk inserts and takes approximately 5 seconds per stock. The script then iterates through the remaining stock files until all data has been copied into their respective tables.

C. Current Stock Data Collection

To update the database with the latest stock information, the FMP API must be called repeatedly throughout the day. The virtual machine on Arbutus Cloud, running a Bash shell on AlmaLinux, provides the mechanism for automating the current data collection script. A systemd timer is configured to run every 30 minutes, covering the period from 4:00 am to 8:00 pm Eastern Standard Time,

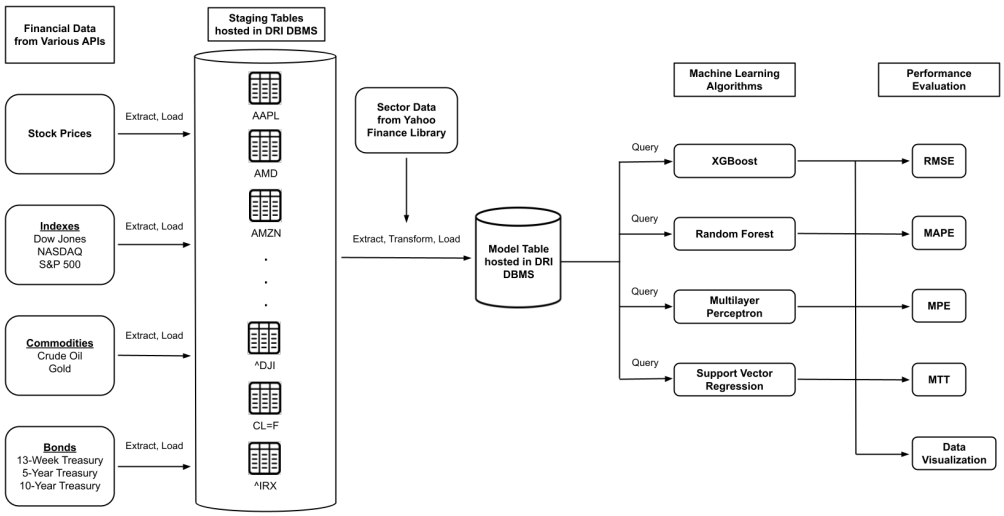

Fig. 2. Automated ELT Architecture of the Algorithmic Trading System

Monday through Friday. The timer's service executes a Bash script that performs the following activities in this order: activates the Python virtual environment, initializes environment variables (API key and database connection), and runs the Python data collection script described in Section III.

D. Data Transformation for DW

The raw data in the staging area is transformed into a single table, appropriate for the ML model. This table contains only numerical data, as ML models are highly mathematical in nature. The number of stocks in the staging area requires a repeatable transformation process.

The raw data is queried from the database and sent to DRI's Arbutus Cloud for Processing with a Python script. Processing is performed in Python due to the extensive libraries that facilitate data transformation. These libraries include pandas, holidays, and yfinance. Sector data is queried from yfinance for each stock and stored in a table. Each stock's data is read from the table and joined with the sector data using pandas. Commodity, bond, and index data are read in and joined to each stock on the date column. The holidays library retrieves a list of holiday dates to assess how pre-holiday and post-holiday periods affect the market. The date column is split into month, week, day, hour, and minute columns. To prevent ML models from inferring a relationship between different integer values in a single column, the script uses one-hot encoding. This technique allows the script to split the original column into a set of Boolean columns for each value present in the data.

Listing 7. One-hot Encoding

```
def one_hot(og_df, feature_to_encode):
    "Function that takes a dataframe, and a
        feature to one-hot encode and
        returns the dataframe with that
        feature encoded"
    dummies =
        pd.get_dummies(og_df[feature_to_encode],
        prefix = feature_to_encode,
        prefix_sep = "_")
    df = pd.concat([og_df, dummies], axis=1)
    df = df.drop([feature_to_encode],
        axis=1)
    return df
```

The transformed data gets inserted back into the database.

E. DW and Staging Database Synchronization

AlmaLinux provides the *firewalld* service to manage port access. After opening the default PostgreSQL port, a public and private SSH key pair was generated and saved locally on the Arbutus VM. The ssh-copy-id command copies the public SSH key to Fir. A secure SSH tunnel to the Fir database is opened, using a non-dedicated port on the Arbutus VM as a local port. Finally, once the port forwarding is set up, the Arbutus terminal runs psql via the now-open SSH tunnel. This completes the connection, and the VM has command-line access to the PostgreSQL database on Fir. A limiting factor of this method is the requirement for two-factor authentication for every Fir database connection. Future work for this section includes discussions with the Digital Research Alliance of Canada to resolve the two-factor authentication issue. Preferably, a connection from Arbutus to Fir will be established exclusively through internal DRAC tools or networks, rather than via SSH, which always requires Duo authentication. Another possibility is a persistent SSH tunnel, although a connection timeout that forces re-authentication is also possible. The script to automatically insert rows has not yet

been completed, but database access has been established, enabling fully automated database synchronization.

F. Testing Data Analysis Using new API from DW to Jupyter Notebook

The Fir cluster, which offers high-performance computing using GPUs, provides access to JupyterHub and JupyterLab for short tasks such as model training and testing. To access JupyterHub through the Fir server and use its GPUs, login through this link: https://jupyterhub.fir.alliancecan.ca/.

The model Notebook is uploaded to JupyterHub through the Fir Server. The model table is located in the Fir cluster to ensure the model can easily access it. The model uses the psycopg2 and sqlalchemy packages to connect to the database and query the model table.

Listing 8. Database to Notebook Connection

```
engine =
    create_engine(f'postgresql://{username}:
    {password}@{host}:{port}/{database_name}')

query = "SELECT * FROM
    market.modelfeatures WHERE symbol IN
    ('AAPL', 'TSLA', 'MSFT')"

df = pd.read_sql(query, engine)
```

The model uses the pandas library to store the query data into a dataframe and uses the dataframe to train and predict the stock prices. The prediction results are stored and displayed on graphs.

VI. CONCLUSION

This research paper described and discussed a new approach to data collection from FMP data sources for training, testing, and stock forecasting using XGBoost (eXtreme Gradient Boosting), an open-source, high-performance machine learning library that implements optimized gradient boosted decision trees. The new and historical datasets were uploaded to a PostgreSQL general-purpose DBMS within the DRI research project. The research team designed and tested a new API that provides direct access to the DBMS from the core subsystem for machine learning (ML) training, testing, and forecasting on the vast datasets collected, transformed, and stored in the DBMS.

ACKNOWLEDGEMENTS

We thank Okanagan College and the OC's GIA Committee for funding and financial support of the applied student research projects. Additionally, we would like to thank the DRA of Canada for the DRI Champions Award.

REFERENCES

[1] "Alliance Cloud Connect Pilot." [Online]. Available: https://www.alliancecan.ca/en/accp

[2] "Financial Modeling Prep - FinancialModelingPrep — FMP." [Online]. Available: https://site.financialmodelingprep.com/

[3] A. Wong, J. Figini, A. Raheem, G. Hains, Y. Khmelevsky, and P. C. Chu, "Forecasting of stock prices using machine learning models," *2023 IEEE International System Conferences (SYSCON)*, 2023.

[4] A. Wong, J. Figini, A. Raheem, G. Hains, Y. Khmelevsky, and P. Chu, "Forecasting of Stock Prices Using Machine Learning Models," in *IEEE International Systems Conference (SysCon) 2023*, 2023.

[5] A. Wong, S. Whang, E. Sagre, N. Sachin, G. Dutra, Y.-W. Lim, G. Hains, Y. Khmelevsky, and F. Zhang, "Short-term stock price forecasting using exogenous variables and machine learning algorithms," *arXiv preprint arXiv:2309.00618*, 2023.

[6] A. Wong, S. Whang, E. Sagre, N. Sachin, G. Dutra, Y.-W. Lim, G. Hains, Y. Khmelevsky, and F. Chang Zhang, "Short-Term Stock Price Forecasting using Exogenous Variables and Machine Learning Algorithms," in *2023 3rd International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA)*, 2023, pp. 260–265.

[7] A. Wong, J. Figini, A. Raheem, G. Hains, Y. Khmelevsky, and P. Chu, "Forecasting of Stock Prices Using Machine Learning Models," in *IEEE SYSCON*. Vancouver: IEEE, 4 2023.

[8] N. Ebadifard, A. Parihar, Y. Khmelevsky, G. Hains, A. Wong, and F. Zhang, "Data Extraction, Transformation, and Loading Process Automation for Algorithmic Trading Machine Learning Modelling and Performance Optimization," 2023.

[9] Y. Khmelevsky, "SW Development Projects in Academia," *WCCCE 2009 - Proceedings of the 14th Western Canadian Conference on Computing Education*, vol. 1, no. 250, pp. 60–64, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1536292

[10] Y. Khmelevsky, M. Govorov, and L. Burge, "Okanagan College and Vancouver Island University educational joint projects results," in *Proceedings of the 14th Western Canadian Conference on Computing Education - WCCCE '09*, 2009, pp. 65–69. [Online]. Available: http://portal.acm.org/citation.cfm?id=1536274.1536293

[11] P. Sharma, M. Govorov, Y. Khmelevsky, and S. Dhanjal, "Oracle 9iAS Portal as a platform for Geographic Information Science distance and flexible learning at the University of the South Pacific," *WIT Transactions on Information and Communication Technologies*, vol. 31, 2004.

[12] Y. Khmelevsky and V. Voytenko, "Cloud computing infrastructure prototype for university education and research," in *Proceedings of the 15th Western Canadian Conference on Computing Education*, ser. WCCCE '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1806512.1806524

[13] L. C. Pérez, S. Cooper, E. K. Hawthorne, S. Wetzel, J. Brynielsson, A. G. Gökce, J. Impagliazzo, Y. Khmelevsky, K. Klee, M. Leary, A. Philips, N. Pohlmann, B. Taylor, and S. Upadhyaya, "Information assurance education in two- and four-year institutions," in *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Education - Working Group Reports*, ser. ITiCSE-WGR '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 39–53. [Online]. Available: https://doi.org/10.1145/2078856.2078860

[14] Y. Khmelevsky, V. Ustimenko, G. Hains, C. Kluka, E. Ozan, and D. Syrotovsky, "International collaboration in SW engineering research projects," in *Proceedings of the 16th Western Canadian Conference on Computing Education*, 2011, pp. 52–56.

[15] Y. Khmelevsky and V. Voytenko, "Strategies for teaching mobile application development," in *18th Western Canadian Conference on Computing Education*, vol. 18, 2013, pp. 8–13.

[16] ——, "Hybrid Cloud Computing Infrastructure in Academia." in *WCCCE 2015 - the 20th Western Canadian Conference on Computing Education, At May 8-9, 2015*. Vancouver Island University (VIU), Nanaimo, British Columbia, Canada., 2015.

[17] Y. Khmelevsky, "Ten Years of Capstone Projects at Okanagan College: A Retrospective Analysis," in *Proceedings of the 21st Western Canadian Conference on Computing Education*. New York, NY, USA: ACM, 2016, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2910925.2910949

[18] N. McDonald, D. Atkinson, Y. Khmelevsky, and S. McMillan, "Sport wearable biometric data encrypted emulation and storage in cloud," in *Canadian Conference on Electrical and Computer Engineering*, 2016.

[19] N. Mcdonald, D. Leader, C. K. Chiang, Y. Khmelevsky, R. Bartlett, and A. Needham, "A new online tool for gamer network performance analysis," in *2016 IEEE International Conference on Cybercrime and Computer Forensic (ICCCF)*, 2016, pp. 1–6.

[20] N. Mcdonald, D. Atkinson, C. Frank, Y. Khmelevsky, and S. McMillan, "Biometric data emulation and encryption for sport wearable devices (A case study)," in *2016 Annual IEEE Systems Conference (SysCon)*, 2016, pp. 1–6.

[21] Y. Khmelevsky, K. Chidlow, K. Sugihara, and K. Zhang, "Engaging and Motivating Students Through Programming Competitions and GIS Applied Research Projects," *Proceedings of the 22nd Western Canadian Conference on Computing Education*, 5 2017. [Online]. Available: http://dx.doi.org/10.1145/3085585.3088491

[22] B. Ward, Y. Khmelevsky, G. Hains, R. Bartlett, A. Needham, and T. Sutherland, "Gaming network delays investigation and collection of very large-scale data sets," in *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings*, 2017.

[23] M. Cocar, R. Harris, and Y. Khmelevsky, "Utilizing Minecraft bots to optimize game server performance and deployment," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017, pp. 1–5.

[24] D. Joiner, M. Clement, S. T. Chan, K. Pereira, A. Wong, Y. Khmelevsky, J. Mahony, and M. Ferri, "DW vs OLTP Performance Optimization in the Cloud on PostgreSQL (A Case Study)," in *2022 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2022, pp. 1–8.

[25] A. A.-A. Falah Hassan, "Stock market index prediction using artificial neural network," *Journal of Information Technology Research (JITR)*, vol. 15, pp. 1–16, 2022.

[26] H. N. P. S. B. K. Geeta Kolte, Varadraj Kini, "Stock market prediction using deep learning," *International Journal for Reseeaarch in Applied Science & Engineering Technology*, vol. 10, no. 4, pp. 26–32, 2022.

[27] Y. Li, "Strategy analysis of financial neural network model in bond investment prediction," *Frontiers in Business, Economics and Management*, vol. 12, pp. 36–39, 2023.

[28] R. K. Dubey, "Algorithmic Trading: The Intelligent Trading Systems and Its Impact on Trade Size," *Expert Systems with Applications*, vol. 202, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0957417422006479

[29] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner, "Software Engineering for AI-Based Systems: A Survey," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 2, 4 2022.

[30] T. Théate and D. Ernst, "An application of deep reinforcement learning to algorithmic trading," *Expert Systems with Applications*, vol. 173, p. 114632, 2021.

[31] A. A. Yulianto, "Extract transform load (etl) process in distributed database academic data warehouse," *APTIKOM Journal on Computer Science and Information Technologies*, vol. 4, no. 2, pp. 61–68, 2019.

[32] O. Azeroual, G. Saake, and M. Abuosba, "Etl best practices for data quality checks in ris databases," in *Informatics*, vol. 6, no. 1. MDPI, 2019, p. 10.

[33] W. A. A. C. C. O. O. C. O. . C. E. U. Adedoyin Tolulope Oyewole, Omotayo Bukola Adeoye, "Predicting stock market movements using neural networks: A review and application study," *Computer Science & IT Research Journal*, vol. 5, pp. 651–670, 2024.

[34] E. M. Haryono, I. Gunawan, A. N. Hidayanto, U. Rahardja *et al.*, "Comparison of the e-lt vs etl method in data warehouse implementation: A qualitative study," in *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. IEEE, 2020, pp. 115–120.

[35] A. Katari and A. Rodwal, "Next-generation etl in fintech: Leveraging ai and ml for intelligent data transformation," pp. 3491–3500, 2023.

[36] M. Patel and D. B. Patel, "Progressive growth of etl tools: A literature review of past to equip future," *Rising Threats in Expert Applications and Solutions: Proceedings of FICR-TEAS 2020*, pp. 389–398, 2020.

[37] N. Biswas, A. Sarkar, and K. C. Mondal, "Efficient incremental loading in etl processing for real-time data integration," *Innovations in Systems and Software Engineering*, vol. 16, pp. 53–61, 2020.

[38] S. M. F. Ali, "Next-generation etl framework to address the challenges posed by big data." in *DOLAP*, 2018.

[39] K. X. L. X. Xuekui Zhang, Yuying Huang, "Novel modelling strategies for high-frequency stock trading data," *Financial Innovation*, vol. 9, no. 39, pp. 1–25, 2023.

[40] J. Wu, D. Bein, J. Huang, and S. Kurwadkar, "Etl and ml forecasting modeling process automation system," *Applied Human Factors and Ergonomics International*.

[41] O. College, "AlgorithmicTradingPublic [python]," 2025, accessed: 2025-11-24. [Online]. Available: https://github.com/youry/AlgorithmicTradingPublic.git

[42] D. R. A. of Canada, "SSH Tunnelling (drac)," 2025, accessed: 2025-11-26. [Online]. Available: https://docs.alliancecan.ca/wiki/SSH_tunnelling