Advantages of Data Warehouses in the Analysis and Study of E-Commerce Markets

Mario-Tudor CHIRIAC
Bucharest University of Economic Studies
Faculty of Cybernetics, Statistics and Economic Informatics
Bucharest, Romania
chiriacmario21@stud.ase.ro

The present paper aims to explore the advantages of using data warehouses in the analysis and study of contemporary e-commerce markets through the development of an applied case study. By integrating Python, Oracle Database, Node.js, and React.js technologies, a comprehensive Business Intelligence solution is implemented. This solution enables the centralization, transformation, and visualization of relevant data for analyzing consumer behavior and identifying market trends. The proposed application provides a user-friendly interface, dynamic report generation capabilities, and advanced filtering functionalities. The study demonstrates that the adoption of data warehouse architectures significantly enhances the efficiency of the decision-making process within organizations, particularly in the context of increasing data volumes. The paper combines a theoretical framework with practical implementation, offering a functional and extensible example for economic market analysis using modern technological tools.
Keywords: *Data Warehouses; Python; Oracle Database; Node.js; React.js; Market Analysis; E-commerce; Business Intelligence*

1 Introduction

Electronic commerce has experienced significant growth and development over the past decades, particularly as a consequence of the COVID-19 pandemic during the 2019–2021 period. This expansion was further supported by governmental policies aimed at limiting physical interaction among individuals and encouraging the adoption of electronic payment methods. Additional contributing factors include the expansion of e-banking services offered by financial institutions, the development of the cryptocurrency market, and the economic growth recorded prior to the pandemic.

In 2024, the Romanian e-commerce sector generated transactions amounting to €7.7 billion [1], compared to €2.8 billion in 2017 [2]. In nominal terms, this represents an increase of approximately 175%. In Romania, even prior to the pandemic, most merchants employed strategies designed to stimulate e-commerce within the mass consumer market, such as the development of dedicated websites, mobile applications, online discounts and promotional campaigns, free delivery policies, and similar incentives.

The most significant actors in the Romanian e-commerce market include:

1. eMAG (founded in 2001, initially as a retailer specializing exclusively in electronics and home appliances). eMAG has evolved into a major e-commerce retailer, achieving remarkable growth and reaching a turnover exceeding €2 billion in 2024 [3], thus becoming the largest retailer in Romania by revenue. Over time, the company has expanded its operations into Bulgaria and Hungary and diversified its product portfolio to include clothing, grocery products, cosmetics, and other consumer goods.
2. Altex, a major electronics retailer;

3. Dedeman, a leading retailer in the DIY, home improvement, and garden sector;

At the international level, several well-established e-commerce retailers have also gained popularity within the Romanian market, including Amazon, eBay, Alibaba, Temu, Shein, Zalando, Etsy, and JD.com. The global e-commerce market likewise experienced substantial positive impact from the pandemic. For 2023, international e-commerce transactions were estimated at approximately USD 6.5 trillion [4], representing an increase of approximately 86% compared to 2019, when global e-commerce transactions amounted to USD 3.5 trillion [5].

The leading global actors by revenue in 2024 include Amazon, with USD 638 billion; JD.com, with USD 158.8 billion; and Alibaba, with USD 130.35 billion [6],[7],[8]. These three corporations dominate the global e-commerce market, accounting for nearly 70% of the combined revenues of the largest companies operating in this sector.

2 Theoretical Frameworks on E-Commerce

The notion of e-commerce emerged in 1984 and was first used by Robert Jacobson, Principal Consultant to the California State Assembly's Utilities and Commerce Committee. The term was introduced in the title and content of the "Electronic Commerce Act," promoted by Gwen Moore and adopted in the same year [9]. In the contemporary context, driven by globalization, increased efficiency, and the automation of buying and selling processes, e-commerce has become highly specialized and differentiated, addressing an increasingly diverse range of needs from both buyers and sellers. The most significant market segments include:

- B2C (Business-to-Consumer), also referred to as online shopping, represents the most common and accessible segment. Within this model, merchants utilize databases or data warehouses, along with existing web technologies (HTML, CSS, JavaScript), to design and implement websites that function as virtual stores displaying their products. Contemporary technologies, combined with marketing strategies and business analytics, enable merchants to enhance their sales strategies through optimization, forecasting, research, operational efficiency improvements, and cost reduction. Prominent examples in this segment include Amazon, Zalando, Temu, and Shein [9].
- B2B (Business-to-Business) involves commercial transactions conducted between companies, without the direct involvement of the final consumer. This segment plays a crucial role in supply chains, distribution networks, and production processes. In the B2B context, e-commerce platforms are used for wholesale procurement, recurring orders, inventory management, customized offers, and price negotiations. Such platforms are typically optimized for efficiency, security, and integration with Enterprise Resource Planning (ERP) systems. Examples of B2B platforms include Alibaba, Amazon Business, and Mercateo. Commercial interactions in this model often involve contractual agreements and negotiable terms, in contrast to the B2C model [9].
- C2C (Consumer-to-Consumer) facilitates transactions between individuals without the direct involvement of a commercial enterprise. C2C platforms enable users to sell and purchase products or services directly, either through fixed-price listings or auctions. This model has gained popularity through platforms such as

eBay, OLX, Facebook Marketplace, and Etsy (particularly for handmade and vintage products). While the C2C model generally entails low entry costs, it also involves higher risks related to product quality, delivery reliability, seller credibility, and warranty assurance.

- Social commerce – an emerging form of e-commerce in which the processes of product discovery, evaluation, and purchase occur directly within social media platforms (e.g., Instagram Shop, Facebook Marketplace, TikTok Shop). This model integrates digital marketing strategies with commercial functionalities, thereby reducing the distance between product promotion and acquisition.
- B2G (Business-to-Government) – this model involves transactions between private companies and governmental institutions. Typically, it encompasses participation in public procurement procedures, acquisition of equipment, software solutions, consultancy services, or infrastructure projects. Dedicated platforms are generally subject to strict regulatory frameworks and impose rigorous legal and technical requirements (for example, SEAP in Romania – the Electronic Public Procurement System).
- C2B (Consumer-to-Business) – a less common but increasingly relevant model within the digital economy. In this framework, consumers provide products, services, or data to companies. Examples include freelancers offering services through platforms such as Fiverr or Upwork, as well as influencers promoting products in exchange for remuneration. Paid reviews, survey participation, and the sale of photographs or digital content to companies also fall within this category.

2.1 Fundamental Aspects of Business Intelligence Technologies

OLAP (Online Analytical Processing) represents a form of analytical processing designed for the rapid examination of large volumes of structured data through multidimensional visualizations, reports, and complex aggregations. OLAP enables the simultaneous analysis of multiple dimensions such as time, region, product, or category, offering significantly higher efficiency compared to traditional SQL query-based analysis. It relies on multidimensional structures, commonly referred to as data cubes, which facilitate instantaneous aggregations, comparisons, drill-down and roll-up operations, as well as advanced analytical computations.

A Data Warehouse is a specialized information system used for the storage, organization, and analysis of large volumes of data originating from multiple heterogeneous sources. The importance of data warehouses lies in their role in supporting organizational decision-making processes. The data stored within a data warehouse typically originate from various sources, including application-generated log files and transactional systems [10].

The principal characteristics of a data warehouse are integration, subject orientation, time variance, and non-volatility.

Integration refers to the origin and harmonization of source data. Data within a warehouse may originate from multiple heterogeneous systems, including operational databases (SQL and NoSQL), Excel files, Customer Relationship Management (CRM) systems, log files, and other transactional or analytical sources. These disparate datasets are transformed and standardized to ensure consistency in format, structure, and semantics.

The subject-oriented characteristic relates to the functional purpose of the data ware-

house. Given the large data volumes involved, warehouses are typically organized around specific domains or thematic areas rather than around operational processes. This domain-focused structuring enables the generation of highly relevant and precise analytical outputs and performance indicators. Within the e-commerce domain, such subject areas commonly include sales, customers, products, inventory, and related business entities.

The time-variant characteristic refers to the persistence of transactional data over extended periods. Data warehouses are typically designed to store historical data for long durations in order to ensure the relevance and depth of analytical processes. Transactions are retained over months or even years, enabling longitudinal analysis, trend identification, and comparative evaluations across different time intervals.

The non-volatility characteristic indicates that data within a warehouse are not subject to frequent modification. In most cases, once data are loaded into the data warehouse, they are not deleted or altered but remain stored for extended periods. This stability ensures consistency in reporting and analytical outputs.

The classical architecture of a data warehouse includes an ETL (Extract, Transform, Load) process, which extracts data from operational sources, transforms them into a consistent format, and loads them into the central repository. Subsequently, the data may be structured into data marts dedicated to specific departments or functional areas, often employing star or snowflake schemas to optimize query performance.

Business Intelligence (BI) represents a set of technologies, processes, and tools that enable the collection, transformation, analysis, and visualization of organizational data for informed decision-making. In essence, Business Intelligence transforms raw data into meaningful information for managers, analysts, and decision-makers, facilitating a deeper understanding of organizational performance and supporting process optimization.

The primary components of a Business Intelligence (BI) system include:

1. Data Collection, which involves gathering information from multiple sources, such as databases, Excel files, enterprise applications, and CRM systems.
2. Data Integration and Storage, referring to the centralization of data, often within a Data Warehouse environment.
3. Data Analysis, performed through queries, filtering operations, calculations, and the definition of Key Performance Indicators (KPIs).
4. Data Visualization, achieved through dashboards, charts, and interactive reports that facilitate intuitive interpretation of results.
5. Decision-Making, whereby organizational leadership leverages analytical outputs to optimize operations and support strategic planning.

Examples of widely used BI tools include Power BI (Microsoft), Tableau (Salesforce), Qlik Sense, Oracle BI / Oracle Analytics Cloud, and SAP BusinessObjects.

3. Methodology

Within this scientific article, Oracle Database will be used as the practical foundation for data storage, including the creation of dimension and fact tables. The Python programming language will be employed to implement ETL (Extract, Transform, Load) processes and to integrate Machine Learning functionalities useful for predictive analysis and Business Intelligence strategies. React.js will be utilized for developing the graphical user interface (frontend) and for generating interactive reports, while Node.js will be responsible for implementing the backend layer of the application.

Python represents a highly popular programming language and an appropriate choice in this context. It constitutes a practical and cost-effective alternative due to its relatively simple syntax and extensive ecosystem of libraries applicable across diverse domains, ranging from socket programming in networking to the implementation and training of Machine Learning models. An additional advantage of Python lies in its continuous maintenance and active development cycle, which ensures the regular release of improved and updated versions. The latest stable version of Python is 3.13, and version 3.14 is scheduled to be released in prerelease form in October 2025 [11].

React is a JavaScript library developed and maintained by Meta (the company behind Facebook), specifically designed for building modern Single Page Applications (SPAs). It provides an efficient approach to constructing reactive user interfaces based on component architecture and dynamic content updates, making it particularly suitable for projects that involve intensive and frequent interaction with data. React has been used in the development of numerous web applications, including Facebook, Instagram, Airbnb, eBay, Walmart, and Alibaba.

Node.js is a JavaScript runtime environment built on Google's V8 engine [12], enabling the execution of JavaScript code on the server side. It is well suited for the development of modern, high-performance, and scalable web applications, particularly within RESTful or microservices-based architectures. In the present project, Node.js is employed to implement the backend layer of the application, being responsible for exposing API endpoints, managing communication with the database, and handling data resulting from the ETL processes implemented in Python. One of the key advantages of Node.js is its asynchronous, event-driven architecture, which enables efficient handling of concurrent requests without blocking server execution. Owing to the extensive ecosystem of packages available through npm (Node Package Manager), application development with Node.js is both rapid and flexible. The technology is widely adopted by companies such as Netflix, Uber, PayPal, and LinkedIn, demonstrating its scalability and performance in complex production environments.

For the implementation of the dimension and fact tables, the star schema model will be employed. This model is particularly appropriate due to its conceptual simplicity, ease of interpretation, and high query performance. Since it involves a limited number of join operations, it ensures efficient data retrieval, making it highly suitable for integration with frontend technologies such as React, where rapid response times are essential for dynamic reporting and visualization. The tables implemented within the star schema will include:

- A fact table, named *fapte_vanzari* (sales_facts), which will store quantitative and measurable data related to transactions.
- A dimension table, *dim_clienti* (customer_dimension), which will enable the analysis of sales according to customer characteristics.
- A dimension table, *dim_produse* (product_dimension), which will facilitate analysis based on product types, brands, and categories.
- A dimension table, *dim_magazin* (store_dimension), which will support geographical and organizational analysis of sales performance.
- A dimension table, *dim_timp* (time_dimension), which will enable temporal analysis by month, year, and day.

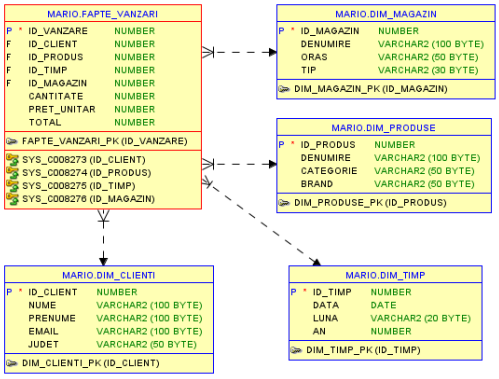


Figure 1 – Star Schema Model

For the implementation of the ETL processes, a Python-based server will be developed and connected to the Oracle Database in which the star schema tables are defined. The population of these tables will be performed using CSV files containing relevant information regarding products, customers, stores, and simulated transactions. These files emulate data generated by an e-commerce platform and will serve as input for the extraction, cleansing, transformation, and loading processes into the dimension and fact tables. Below is the implementation of the raw data extraction process from the CSV files using the *pandas* library in Python.

```python
@app.post("/upload-csv")
async def upload_csv(
    clienti : UploadFile = File(None),
    produse : UploadFile = File(None),
    timp    : UploadFile = File(None),
    magazin : UploadFile = File(None),
    vanzari : UploadFile = File(None),
):
    mapping = {
        "clienti.csv": clienti, "produse.csv": produse, "timp.csv": timp,
        "magazin.csv": magazin, "vanzari.csv": vanzari,
    }
    saved: List[str] = []
    for fname, f in mapping.items():
        if f:
            with open(CSV_DIR / fname, "wb") as dst:
                shutil.copyfileobj(f.file, dst)
            saved.append(fname)
    if not saved:
        raise HTTPException(400, "Niciun fisier trimis")
    return {"message":"CSV salvate", "saved": saved}


def _norm(df: pd.DataFrame)->pd.DataFrame:
    df.columns = (
        df.columns
          .str.replace("\ufeff|\u00a0","",regex=True)
          .str.strip().str.lower()
          .str.replace(r"\s+","_",regex=True)
    )
    return df

def _read(key:str)->pd.DataFrame:
    path = CSV_DIR / FILES[key]
    if not path.exists():
        raise HTTPException(400, f"{path.name} lipseste - incarca-l prin /upload-csv")
    return _norm(pd.read_csv(path, sep=None, engine="python", encoding="utf-8-sig"))


@app.post("/etl/extract")
def etl_extract():
    app.state.df.clear()
    for k in FILES:
        app.state.df[k] = _read(k)
    return {"message":"Extract OK", "tables": list(app.state.df)}
```

Figure 2 – Python Code Snippet Highlighting the Extract Process

Data extraction is an essential step, as it constitutes the foundation for the subsequent stages of the ETL pipeline, namely transformation and loading, which will be addressed in the following sections.

Data transformation represents the second stage of the ETL process. During this phase, the data obtained through extraction are standardized in accordance with the organization's requirements and internal standards. Regardless of specific business objectives, it is crucial that the data do not contain null values in critical fields, inconsistent or unreadable formats, duplicate entries, missing records, or special and ambiguous characters that may hinder processing or compromise data integrity.

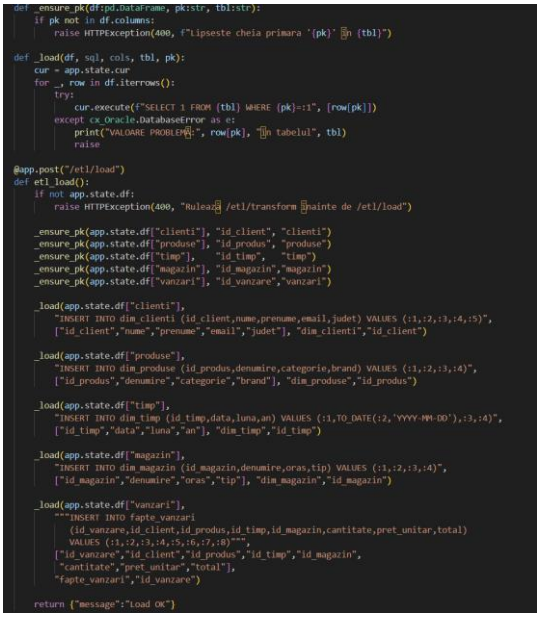
```python
def _need_extract():
    if not app.state.df:
        raise HTTPException(400, "Ruleaza /etl/extract inainte de /etl/transform")

def _clean_dim(df: pd.DataFrame) -> pd.DataFrame:
    df = (df.replace(r"^\s*$", pd.NA, regex=True)
            .dropna().drop_duplicates())
    df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)


    id_cols = [c for c in df.columns if c.startswith("id_")]
    for c in id_cols:
        df[c] = pd.to_numeric(df[c], errors="coerce")
    df.dropna(subset=id_cols, inplace=True)
    df[id_cols] = df[id_cols].astype(int)

    return df


def _clean_timp(df: pd.DataFrame)->pd.DataFrame:
    if "data" not in df.columns:
        raise HTTPException(400, f"Coloana 'data' lipseste in timp.csv; gasite: {list(df.columns)}")
    df["data"] = pd.to_datetime(df["data"], errors="coerce")
    df = df.dropna(subset=["data"]).drop_duplicates()
    return df

def _clean_vanzari(df: pd.DataFrame)->pd.DataFrame:
    df[["cantitate","pret_unitar"]] = df[["cantitate","pret_unitar"]].apply(pd.to_numeric, errors="coerce")
    df = df.dropna(subset=["cantitate","pret_unitar"])
    df = df[(df["cantitate"]>0) & (df["pret_unitar"]>0)]
    df["total"] = df["cantitate"]*df["pret_unitar"]
    return df.drop_duplicates()

@app.post("/etl/transform")
def etl_transform():
    _need_extract()
    for k,df in app.state.df.items():
        if k in {"clienti","produse","magazin"}:  app.state.df[k] = _clean_dim(df)
        elif k=="timp":     app.state.df[k] = _clean_timp(df)
        elif k=="vanzari":  app.state.df[k] = _clean_vanzari(df)

    for k,df in app.state.df.items():
        df.to_parquet(STAGE_DIR / f"{k}.pq", index=False)
    return {"message":"Transform OK"}
```

Figure 3 - Python Code Snippet Highlighting the Transform Process

Data loading represents the final stage of the ETL process. This phase involves inserting the previously extracted and transformed data into the designated data warehouse structures.

Given that the schema of the tables does not change frequently and updates typically occur at extended intervals, it is essential that the loaded data be accurate, thoroughly processed, and consistent. The reliability of

subsequent analytical outputs and Business Intelligence strategies depends directly on the quality and integrity of the stored data.
The loading process applies to both fact tables and dimension tables, ensuring that the star schema is fully populated and ready for analytical querying and reporting.



```
def _ensure_pk(df:pd.DataFrame, pk:str, tbl:str):
    if pk not in df.columns:
        raise HTTPException(400, f"Lipseste cheia primara '{pk}' în {tbl}")

def _load(df, sql, cols, tbl, pk):
    cur = app.state.cur
    for _, row in df.iterrows():
        try:
            cur.execute(f"SELECT 1 FROM {tbl} WHERE {pk}=:1", [row[pk]])
        except cx_Oracle.DatabaseError as e:
            print("VALOARE PROBLEMĂ:", row[pk], "în tabelul", tbl)
            raise

@app.post("/etl/load")
def etl_load():
    if not app.state.df:
        raise HTTPException(400, "Rulează /etl/transform înainte de /etl/load")

    _ensure_pk(app.state.df["clienti"], "id_client", "clienti")
    _ensure_pk(app.state.df["produse"], "id_produs", "produse")
    _ensure_pk(app.state.df["timp"],    "id_timp",   "timp")
    _ensure_pk(app.state.df["magazin"], "id_magazin","magazin")
    _ensure_pk(app.state.df["vanzari"], "id_vanzare","vanzari")

    _load(app.state.df["clienti"],
        "INSERT INTO dim_clienti (id_client,nume,prenume,email,judet) VALUES (:1,:2,:3,:4,:5)",
        ["id_client","nume","prenume","email","judet"], "dim_clienti","id_client")

    _load(app.state.df["produse"],
        "INSERT INTO dim_produse (id_produs,denumire,categorie,brand) VALUES (:1,:2,:3,:4)",
        ["id_produs","denumire","categorie","brand"], "dim_produse","id_produs")

    _load(app.state.df["timp"],
        "INSERT INTO dim_timp (id_timp,data,luna,an) VALUES (:1,TO_DATE(:2,'YYYY-MM-DD'),:3,:4)",
        ["id_timp","data","luna","an"], "dim_timp","id_timp")

    _load(app.state.df["magazin"],
        "INSERT INTO dim_magazin (id_magazin,denumire,oras,tip) VALUES (:1,:2,:3,:4)",
        ["id_magazin","denumire","oras","tip"], "dim_magazin","id_magazin")

    _load(app.state.df["vanzari"],
        """INSERT INTO fapte_vanzari
           (id_vanzare,id_client,id_produs,id_timp,id_magazin,cantitate,pret_unitar,total)
           VALUES (:1,:2,:3,:4,:5,:6,:7,:8)""",
        ["id_vanzare","id_client","id_produs","id_timp","id_magazin",
         "cantitate","pret_unitar","total"],
        "fapte_vanzari","id_vanzare")

    return {"message":"Load OK"}
```

Figure 4 – Python Code Snippet Highlighting the Load Process

The first part of the code implements the extraction phase, during which the application receives, validates, and stores the CSV files provided by the user. The dedicated upload endpoint accepts five datasets corresponding to the core entities of the model: customers, products, time, stores, and sales. Each file is mapped to a predefined identifier and stored locally, while an auxiliary CSV-reading function converts each file into a Pandas DataFrame, handling potential encoding variations or header-related inconsistencies. The extraction endpoint aggregates all these DataFrames into a unified in-memory structure, enabling their subsequent use in later processing stages. This section marks the completion of the "Extract" component and ensures the availability of raw data for further cleansing operations.
The second part of the code represents the transformation phase, during which the raw data undergo preprocessing operations designed to ensure analytical model quality and consistency. The transformation functions perform tasks such as removing empty rows, standardizing textual values by trimming unnecessary whitespace, and converting identifier columns to numeric types. In the case of the time dimension, the code explicitly verifies the presence of the "date" column, converts its contents into a valid calendar format, and derives the corresponding year values to be used as a temporal dimension within the OLAP model. For the sales table, quantitative fields such as quantity, unit price, and total value are converted to numeric types, and duplicate records are removed to ensure that the fact table remains coherent and suitable for aggregation. The transformation endpoint applies these functions to all DataFrames loaded in the previous stage and updates the in-memory data structures accordingly, thereby completing the "Transform" phase.
The final section of the code corresponds to the loading phase and is responsible for transferring the processed data into the Oracle database schema. Prior to insertion, the code performs strict validation of primary keys through a dedicated function that verifies the existence of the corresponding column in each DataFrame. In the absence of this column, the process is halted in order to prevent compromising database integrity. The actual loading operation is carried out through a function that iterates over each row and executes an INSERT statement. Before inserting each record, a verification query is performed to prevent duplicate entries. The final endpoint orchestrates the entire procedure: it verifies that the transformation stage has been completed, validates the primary keys for all dimension tables and the fact table, and subsequently

inserts the corresponding records into the Oracle tables. The completion of the process is indicated by a confirmation message signaling the successful loading of the data.
Thus, Python proves to be highly flexible, enabling seamless connectivity with the Oracle Database, while the loading process is implemented using the cx_Oracle library. The pandas library facilitates both CSV file reading and content preprocessing. In the subsequent phase, Python will be employed to implement Machine Learning (ML) algorithms using the scikit-learn library, as well as the FastAPI framework to integrate the Python backend with the Node.js server and the web interface. Furthermore, the following chapter will address the implementation of the graphical user interface, which will display predictive outputs and generate the reports necessary for informed decision-making.

4 Case Study: Analysis of the Electronics Market

The electronics market represents a significant segment within the broader e-commerce ecosystem. According to source [13], in Romania, online commerce in the "Electronics" category accounts for approximately 16.7% of the total e-commerce turnover. This sector plays a critical role in digital retail activity, as an increasing number of consumers prefer purchasing electronic products through online platforms.
Given the advanced development of this niche, appropriate solutions are required to optimize Business Intelligence processes. Such solutions can be implemented through Machine Learning (ML) algorithms, which are capable of analyzing the data produced through the ETL framework and identifying relevant patterns, correlations, and predictive models. In this chapter, we will implement the graphical user interface that will display the results of ML analyses alongside visual charts designed to facilitate interpretation.

The user interface will be developed using React.js, in combination with the MUI (Material UI) library, which provides modern web design components such as icons, buttons, typography systems, and layout utilities. To ensure an intuitive and user-friendly experience, the application will include:

- A dedicated section for uploading data in CSV format;
- A section for managing and visualizing ETL processes;
- A module for training the ML model based on the processed and loaded data;
- A dashboard area where specific performance indicators and the corresponding analytical charts will be generated and displayed.

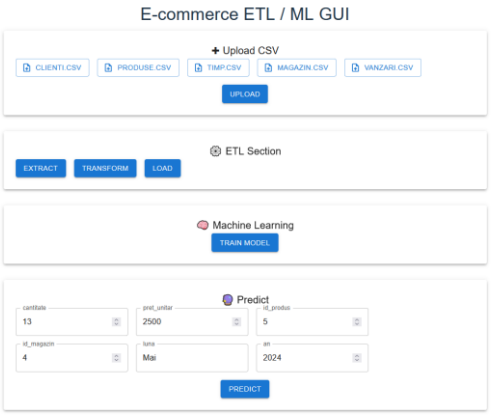


Figure 5 – Overview of the Application

The web interface communicates bidirectionally with the Node.js server. All client-side requests are first processed and structured by Node.js, after which they are forwarded to the Python server implemented using FastAPI. Conversely, responses generated by the Python backend pass through the Node.js server, which then delivers the final response to the client. One of the key advantages of Node.js lies in its architecture, which is specifically oriented toward web-based applications, thereby facilitating rapid integration and

communication between the client and the Python backend. Although Python offers robust web frameworks such as Django, Flask, and FastAPI, many projects prefer integrating Node.js as an intermediary layer or API gateway. This preference is largely due to Node.js's modern ecosystem, optimized for real-time web communication, WebSocket support, streaming capabilities, and efficient resource management. Additionally, Node.js provides seamless integration with Single Page Applications (SPAs) developed using React, Angular, or Vue.

As a result, a modern and efficient workflow is achieved: Node.js handles all incoming client requests, manages sessions, file uploads, and streaming operations, and delegates advanced processing tasks—such as data analysis or Machine Learning computations—to Python-based APIs [14], [15], [16].

Within the Machine Learning component implemented in Python, the Random Forest algorithm was selected in order to predict the sales volume of a specific product, based on the variables sales and time. To generate the decision trees, the model must first receive the processed dataset. The endpoint @app.post("/ml/train") utilizes the RandomForestRegressor class from the scikit-learn library to train the model on the processed data (sales and time-related features). The trained model is subsequently saved as a .joblib file, enabling later reuse for prediction purposes. Up to 120 decision trees are generated through the parameter n_estimators=120. The random_state parameter represents the seed used by scikit-learn's internal random number generator, ensuring reproducibility of results. The model is trained using the model.fit(x, y) method, where the previously generated trees are fitted to the input data. Following training, several performance metrics are computed to evaluate the model:

- The coefficient of determination (R^2) on the training data (r2_train);
- The Root Mean Squared Error (RMSE) calculated on the training dataset (rmse_train);
- The mean and standard deviation of the R^2 score obtained through cross-validation (r2_cv_mean, r2_cv_std);

Random Forest is an ensemble learning algorithm that constructs multiple decision trees on random subsets of the data. The final prediction is obtained by averaging the outputs of the individual trees in regression tasks, or by majority voting in classification tasks. This algorithm is widely recognized for its high predictive accuracy, robustness to noisy data, and ability to handle a large number of features without requiring manual feature selection. Furthermore, Random Forest is less prone to overfitting compared to a single decision tree and can provide insights into feature importance within the model. Due to these advantages, Random Forest is extensively applied in business analytics, healthcare, finance, and numerous other domains [17], [18].

```
@app.post("/ml/train")
def ml_train():
    if not (STAGE_DIR/"vanzari.pq").exists() or not (STAGE_DIR/"timp.pq").exists():
        raise HTTPException(400,"Use ETL functions first!")
    df = _train_df()
    X = pd.get_dummies(df[["cantitate","pret_unitar","id_produs","id_magazin","luna","an"]])
    y = df["total"]

    model = RandomForestRegressor(n_estimators=120, random_state=42)
    cv = cross_val_score(model, X, y, cv=5, scoring="r2")
    model.fit(X,y)
    dump(model, MODEL_F)
    X.columns.to_series().to_csv(XCOLS_F, index=False, header=False)

    yhat = model.predict(X)
    return {
        "r2_train": round(r2_score(y,yhat),4),
        "rmse_train": round(np.sqrt(mean_squared_error(y, yhat)), 2),
        "r2_cv_mean": round(cv.mean(),4),
        "r2_cv_std":  round(cv.std(),4),
    }
```

Figure 6 – Implementation of the /ml/train Endpoint

The next stage in the logical workflow involves generating predictions based on the independent variables. For this purpose, a separate endpoint must be implemented to load the trained model and return prediction

results. After the training phase, the model is saved to disk within the /ml/train endpoint as a .joblib file. Subsequently, this model is loaded by the /ml/predict route in order to perform inference. The prediction process includes the following components:

- A DataFrame is created using the input data received from the user (p);
- One-hot encoding is applied to categorical variables (for example, the month). This technique transforms categorical variables (text-based features such as month, city, or category) into numerical columns, allowing machine learning models to process them appropriately;
- The columns are reordered to match exactly the structure used during training (using xcols), with any missing columns being automatically filled with zero values to maintain dimensional consistency.

```
@app.post("/ml/predict")
def ml_predict(p: PredictInput):
    if not MODEL_F.exists() or not XCOLS_F.exists():
        raise HTTPException(400,"Model is not trained - run /ml/train")
    model = load(MODEL_F)
    xcols = pd.read_csv(XCOLS_F, header=None).iloc[:,0].tolist()
    X_new = pd.get_dummies(pd.DataFrame([p.dict()])).reindex(columns=xcols, fill_value=0)
    pred = model.predict(X_new)[0]
    return {"prediction": round(pred,2)}
```

Figure 7 – Implementation of the */ml/predict* Endpoint

The backend implemented in Node.js functions as a gateway between the Python web server and the React.js frontend. This architectural strategy offers several advantages:

1. **Technological decoupling**, which ensures a clear separation between business logic (ML/ETL processes implemented in Python) and the presentation layer (user interface developed in React.js). Each component operates within the technology stack best suited to its purpose, enhancing maintainability and scalability.
2. **API route uniformity and centralization**, meaning that the Node.js gateway can expose a single consolidated API to the frontend, regardless of the number of backend microservices operating within the application. This simplifies integration and reduces frontend complexity.
3. **Efficient handling of files and sessions**, as Node.js is particularly well optimized for file uploads, streaming operations, WebSocket communication, and file system interactions. This makes it highly suitable for mediating data exchange between the client and the backend services.

```
app.post('/upload-csv',
  upload.fields([
    {name:'clienti'},{name:'produse'},{name:'timp'},
    {name:'magazin'},{name:'vanzari'}
  ]), async (req,res)=>{
  try {
    const form = new FormData()
    Object.entries(req.files).forEach(([k,[f]])=>{
      form.append(k, fs.createReadStream(f.path), f.originalname)
    })
    const r = await axios.post(`${FASTAPI}/upload-csv`, form,
      { headers: form.getHeaders() })


    Object.values(req.files).forEach(([f])=>fs.unlinkSync(f.path))
    res.json(r.data)
  } catch (err) {
    console.error(err.message); res.status(500).json({error:err.message})
  }
})


;['/etl/extract','/etl/transform','/etl/load',
  '/ml/train','/ml/predict'].forEach(path=>{
  app.post(path, async (req, res) => {
  try {
    const r = await axios.post(`${FASTAPI}${path}`, req.body);
    res.status(r.status).json(r.data);
  } catch (err) {
    const status = err.response?.status || 500;
    const data   = err.response?.data   || { error: err.message };
    console.error(data);
    res.status(status).json(data);
  }
});
})
```

Figure 8 – Node.js Code Snippet

Thus, the integration of React.js, Node.js, and Python within an e-commerce environment proves to be highly practical and efficient. When combined within the same project architecture, these technologies provide enhanced performance, scalability, and advanced functionalities, making them particularly suitable for Business Intelligence strategies related to pricing optimization, marketing campaigns, and

customer behavior analysis. This technological synergy enables real-time data processing, dynamic visualization, and predictive modeling, thereby supporting informed strategic decisions in highly competitive digital markets.

5 Results and Benefits

Data interpretation constitutes a fundamental component of the Business Intelligence process. While the fact table plays a crucial role in storing metric values, a comprehensive and in-depth analysis requires the examination of additional, more advanced indicators. These indicators are typically derived either from the data stored within the fact table or inferred through correlations with dimension tables. The evaluation of multiple performance metrics is essential in order to obtain a clear and accurate understanding of the organization's actions and strategic decisions. Within the analytical framework, the mere presentation of numerical indicators is insufficient. It is imperative to complement quantitative results with graphical representations, as visualizations enhance interpretability, facilitate pattern recognition, and support more effective decision-making.


```
{
  "r2_train": 0.991,
 "rmse_train": 215.09,
 "r2_cv_mean": 0.9219,
  "r2_cv_std": 0.038
}
```


Figure 9 – Output Regarding Model Performance Indicators After ETL Processing and Model Training

The resulting Random Forest model demonstrates a very high level of accuracy on the training dataset (r2_train = 0.991), as well as strong performance under cross-validation (r2_cv_mean = 0.92), indicating good generalization capability. The Root Mean Squared Error (rmse_train = 215.09) is situated at a reasonable level relative to the scale of the analyzed sales data. Furthermore, the variation across cross-validation folds is minimal (r2_cv_std = 0.038), suggesting model stability and consistent predictive performance across different data partitions.

Figure 10 – Estimation of Total Revenue over a Three-Month Period

The monthly revenue chart indicates a strong financial performance in May 2024, followed by a moderate decline in the subsequent months. This pattern suggests the presence of a seasonal peak or the impact of successful promotional campaigns implemented at the beginning of the analysed period. Although the general trend shows a decrease after the May peak, overall revenue levels remain relatively high. This indicates that the company experienced a peak performance phase, followed by a gradual adjustment in sales volume rather than a sharp contraction.

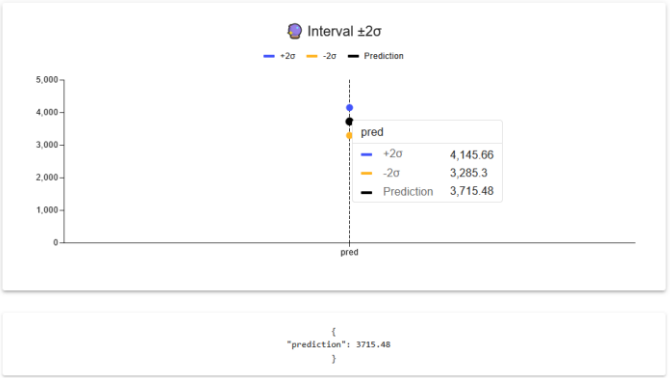


Figure 11 – Generation of the Uncertainty Interval Chart

The predicted value of 3715.48 RON, corresponding to the input dataset (quantity: 2, unit price: 1799 RON, product_id: 5, store_id: 1, month: May, year: 2024), represents the estimated revenue for a transaction with the specified characteristics. The ±2σ interval provides a confidence range around the prediction, meaning that in approximately 95% of cases, the actual sales value is expected to fall between 3285.30 RON and 4145.66 RON. The narrower the interval, the greater the model's confidence in its prediction. In this case, the interval is relatively tight compared to the central predicted value, indicating good model precision and stable predictive performance.

6 Conclusions and Future Directions

Electronic commerce is currently undergoing a phase of market maturation, in which individual actors can no longer radically transform the industry through traditional strategies alone. In this context, the integration of Artificial Intelligence (AI) and Machine Learning (ML) into Business Intelligence processes can significantly enhance sales optimization and operational efficiency. Moreover, AI technologies can substantially improve product search and recommendation processes for customers. A highly personalized AI model is capable of identifying products or offers that are more aligned with individual user preferences than conventional search methods. Traditional ranking algorithms typically rely on generalized indicators such as popularity, rating, average price, or product availability. However, these indicators often become overly generic and may fail to accurately reflect the nuanced expectations and behavioral patterns of individual customers.

References

[1] https://agerpres.ro/economic/2025/03/12/piata-ecommerce-din-romania-va-depasi-valoarea-de-8-8-miliarde-de-euro-in-acest-an-estimari--1430016
[2] https://www.economica.net/romanii-au-cumparat-online-de-2-8-miliarde-de-euro-in-2017_148597.html
[3] https://economedia.ro/afacerile-grupului-emag-au-trecut-de-pragul-de-2-miliarde-euro-iar-romania-este-cea-mai-mare-piata-planuri-de-investitii-de-900-milioane-lei-in-anul-urmator.html
[4] https://en.wikipedia.org/wiki/E-commerce#E-commerce_during_COVID-19
[5] https://en.wikipedia.org/wiki/Economic_impact_of_the_COVID-19_pandemic#E-commerce
[6] https://www.shopify.com/blog/ecommerce-companies
[7] https://en.wikipedia.org/wiki/JD.com
[8] https://en.wikipedia.org/wiki/List_of_largest_technology_companies_by_revenue
[9] https://en.wikipedia.org/wiki/E-commerce#Defining_e-commerce
[10] https://www.oracle.com/database/what-is-a-data-warehouse/
[11] https://devguide.python.org/versions/
[12] https://en.wikipedia.org/wiki/Node.js
[13] https://ecommercedb.com/markets/ro/consumer-electronics
[14] https://mobilunity.com/blog/node-js-vs-python/
[15] https://syndelltech.com/nodejs-vs-python/
[16] https://www.peerbits.com/blog/nodejs-vs-python.html
[17] https://www.ibm.com/think/topics/random-forest
[18] https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

CHIRIAC Mario-Tudor Second year student, in the Master's Programme named: "Databases - business support", within the Faculty of Cybernetics, Statistics and Economic Informatics - the Bucharest University of Economic Studies. In 2024, I graduated from the Bachelor's programme - Economic Informatics in Romanian, within the Faculty of Cybernetics, Statistics and Economic Informatics at the Bucharest University of Economic Studies. Areas of interest: study of SQL and No-SQL databases, programming web applications using Vue.js, React.js.