Retail Sales Data Analysis and Prediction

Sai Sudheer Vishnumolakala
Sai Koushik Thatipamula
Pradeep Chand Potturi
Manogna Tummanepally
Rohith Sahini

November 15, 2024

Contents

1 Executive Summary

This report presents a comprehensive analysis of retail sales data, aiming to develop a predictive model for future sales and provide insights through data warehousing techniques. The project encompasses data collection, dimensional modeling, data preprocessing, exploratory data analysis (EDA), predictive modeling using machine learning, and the creation of analytical queries and views. By leveraging historical sales data, we aim to assist retailers in making informed inventory decisions, optimizing stock levels, and improving profitability.

2 Problem Statement

Retail businesses often struggle with inventory management due to unpredictable sales trends. Overestimating demand can lead to excess inventory costs, while underestimating can result in stockouts and lost sales. The challenge addressed in this project is:

How can retailers accurately forecast future sales to optimize inventory levels and enhance decision-making processes?

Who

Retail business owners, inventory managers, and stakeholders interested in improving inventory management and sales forecasting.

What

Develop a predictive model and analytical tools to forecast future sales based on historical data.

When and Where

The problem arises continuously in retail businesses worldwide, where inventory management is critical to operations.

Why

Accurate sales forecasting helps in reducing inventory costs, preventing stockouts, and improving customer satisfaction.

How

By analyzing historical sales data, applying data warehousing techniques, and building a predictive machine learning model.

3 Literature Review

Several resources and literature were consulted to inform this project:

- **Data Warehousing Concepts**: Understanding dimensional modeling and OLTP vs. OLAP systems from industry-standard textbooks.
- **Predictive Modeling Techniques**: Insights from machine learning literature, focusing on regression models suitable for time series data.
- **Time Series Forecasting**: Techniques from authoritative texts on forecasting principles and practices.
- **Kaggle Tutorials and Notebooks**: Examined various notebooks on retail sales forecasting to understand common practices and methodologies.
- **Oracle Database Documentation**: For implementing data warehousing features and analytical queries.

These resources provided a foundation for the data modeling, preprocessing strategies, and the machine learning techniques applied in this project.

4 Data Collection and Preparation

4.1 Dataset Description

The data used in this project is sourced from Kaggle's English Converted Datasets, which includes:

It consists of four interconnected datasets, each providing essential information for analyzing and predicting retail sales trends. These datasets cover various aspects of retail operations, including item details, shop locations, and transactional sales records. Below is a description of each dataset and its role in this analysis:

- **sales_train.csv**: This dataset captures the daily sales transactions across multiple shops and items over a specified period. Each record includes:
  - *date*: The transaction date in `DD.MM.YYYY` format.
  - *date_block_num*: A unique numeric identifier for each month in the dataset, useful for time-series analysis.
  - *shop_id*: The unique identifier for the shop where the sale occurred.
  - *item_id*: The identifier for the sold item.
  - *item_price*: The price at which the item was sold.
  - *item_cnt_day*: The number of units sold for the item on that date. (Values may include returns, indicated by negative values.)
- **items.csv**: This dataset provides descriptive information about each item, facilitating categorization and further insights into product performance.
  - *item_id*: A unique identifier for each item, linking to the sales data.
  - *item_name*: A textual description of the item.
  - *category_id*: Links each item to a category, allowing analysis by product type.
- **item_categories.csv**: This dataset categorizes items, enabling analysis of sales by broader product categories.
  - *item_category_id*: The identifier for each category, linking to the items dataset.
  - *item_category_name*: A description of the item category, which supports trend analysis at the category level.
- **shops.csv**: This dataset provides information about each retail location where sales were recorded.
  - *shop_id*: The unique identifier for each shop, linking to the sales transactions.
  - *shop_name*: The name and location of the shop, which aids in regional sales analysis and helps capture geographic trends.

These datasets collectively support a comprehensive analysis, enabling the construction of a dimensional data warehouse that aligns with the objectives of predictive modeling and analytical reporting.

4.2 Data Loading and Merging

The datasets were loaded using Python's `pandas` library and merged on relevant keys.

```
import pandas as pd

# Load datasets
sales_train = pd.read_csv('sales_train.csv')
items = pd.read_csv('items.csv')
item_categories = pd.read_csv('item_categories.csv')
shops = pd.read_csv('shops.csv')

# Merge datasets
merged_data = sales_train.merge(items, on='item_id', how='left') \
                         .merge(shops, on='shop_id', how='left') \
                         .merge(item_categories, on='category_id', how='left')
```

Listing 1: Data Loading and Merging

4.3 Data Cleaning

- Removed outliers in `item_cnt_day` and `item_price`.
- Removed negative values in sales counts and prices.
- Preprocessed shop names to standardize the data.

```
# Remove outliers
sales_train = sales_train[(sales_train['item_cnt_day'] >= 0) & (sales_train['item_cnt_day'] <= 1000)]
sales_train = sales_train[(sales_train['item_price'] >= 0) & (sales_train['item_price'] <= 100000)]
```

Listing 2: Outlier Removal

4.4 Feature Engineering

Created lag features and rolling means to capture temporal dependencies.

```
def create_lag_features(df, lags, group_cols, target_col):
    for lag in lags:
        df[f'{target_col}_lag_{lag}'] = df.groupby(group_cols)[target_col].shift(lag)
    return df

lags = [1, 2, 3, 6]
monthly_sales = create_lag_features(monthly_sales, lags, ['shop_id', 'item_id'], 'item_cnt_month')
monthly_sales.fillna(0, inplace=True)
```

Listing 3: Feature Engineering

4.5 Data Loading into Oracle Database

Used `cx_Oracle` to load data into an Oracle database.

```
import cx_Oracle

# Define connection
dsn_tns = cx_Oracle.makedsn('reade.forest.usf.edu', 1521, sid='cdb9')
conn = cx_Oracle.connect(user='username', password='password', dsn=dsn_tns)
cursor = conn.cursor()

# Function to load DataFrame to Oracle
def load_to_oracle(df, table_name):
    cols = ','.join(list(df.columns))
    placeholders = ','.join([':' + str(i+1) for i in range(len(df.columns))])
    sql = f'INSERT INTO {table_name} ({cols}) VALUES ({placeholders})'

    data = [tuple(x) for x in df.to_numpy()]
    cursor.executemany(sql, data)
    conn.commit()

# Rename the column to match the Oracle table schema
item_categories.rename(columns={'category_id': 'item_category_id'}, inplace=True)

# Load data into Oracle tables in the correct order
load_to_oracle(item_categories, 'item_categories_data')  # Parent table
load_to_oracle(shops, 'shops_data')                      # Independent table
load_to_oracle(items, 'items_data')
load_to_oracle(sales_train, 'sales_transactions')
```

Listing 4: Loading Data into Oracle

5 Database Design

A robust database design is crucial for efficient data retrieval and analysis. In this project, we utilized dimensional modeling to design a data warehouse that supports analytical queries effectively.

5.1 Dimensional Modeling

Dimensional modeling involves designing a data warehouse using fact and dimension tables, optimizing for query performance and understandability. The design follows a star schema, where a central fact table is connected to multiple dimension tables.

5.1.1 Fact Table

The **sales_fact** table is the central fact table in our schema, capturing the quantitative data about sales transactions. Each record in the fact table represents a single transaction at the most granular level (transaction-level data).

- **Granularity**: One record per item sold per transaction.
- **Measures (Facts)**:
  - `item_cnt_day`: Number of items sold.
  - `item_price`: Price of the item.
  - `total_sales`: Computed as `item_cnt_day` × `item_price`.
- **Additive Facts**: All measures are additive across all dimensions.

5.1.2 Dimension Tables

The dimension tables provide descriptive context to the facts.

- **date_dim**: Contains detailed information about dates.
- **shops_dim**: Contains details about each shop.
- **item_dim**: Contains details about each item, including category information.

5.1.3 Entity-Relationship Diagram

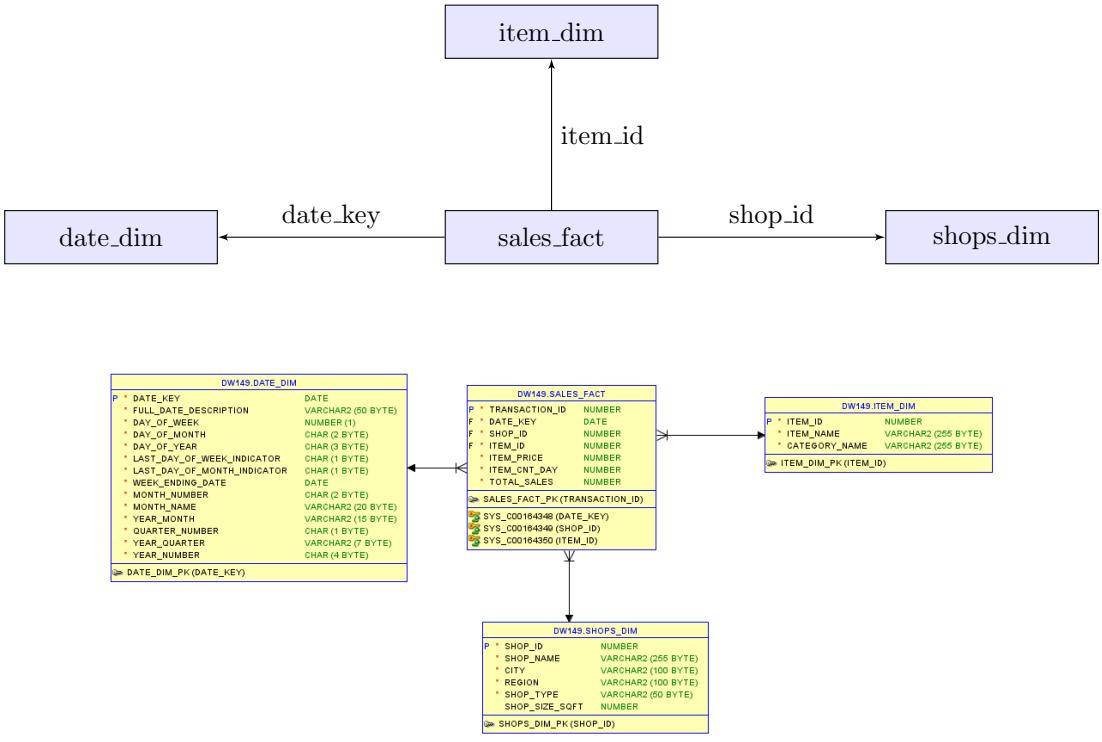

Figure 1: Entity-Relationship Diagram of the Dimensional Model

5.2 Handling Slowly Changing Dimensions (SCD) - Type 1

In this project, **Slowly Changing Dimension (SCD) Type 1** is applied to manage changes in certain dimension attributes, particularly in the `SHOPS_DIM` table. SCD Type 1 is a method used in data warehousing to handle changes in dimension data by simply *overwriting* the existing values with new information. This approach is used when retaining historical data is not necessary and only the latest values are relevant for analysis.

Example of SCD Type 1 in `SHOPS_DIM` In the `SHOPS_DIM` table, attributes like `shop_name`, `region`, and `shop_type` describe various properties of each shop. If a shop's region or type changes over time, using SCD Type 1 allows us to update the affected row with the new value, thereby reflecting the current state of each shop without keeping historical records of prior values.

For example:

- If a shop relocates to a new region, the `region` attribute in the `SHOPS_DIM` table is updated to the new region, and the old region information is overwritten.

This method provides simplicity and reduces storage requirements, as only the latest data is stored, making it suitable for cases where historical tracking of changes in dimension data is not necessary.

Rationale for Using SCD Type 1 SCD Type 1 is used in this project because the analysis primarily focuses on current data rather than historical changes in dimension attributes. This approach ensures that the data warehouse remains efficient and easy to maintain, as it eliminates the need for additional columns or rows to track historical data. It is ideal for attributes that undergo infrequent changes or where only the latest information is required for decision-making.

In summary, by implementing SCD Type 1 in the `SHOPS_DIM` table, this project prioritizes simplicity and relevance, ensuring that the data warehouse always reflects the current state of each shop's attributes.

5.3 Applying Kimball's Four-Step Dimensional Design

1. **Understand the Business Process**:
   - The business process focuses on **retail sales management and forecasting**. This involves tracking sales transactions, items, shops, and dates to support decision-making on inventory management, revenue analysis, and demand forecasting. The data warehouse is designed to consolidate this information, enabling insights into sales trends, seasonal patterns, and performance of individual items and shops.
2. **Choose the Grain**:
   - The grain of this data warehouse is defined at the **individual transaction level**. Each row in the `SALES_FACT` table represents a single transaction of an item sold at a specific shop on a particular day. This level of granularity allows for detailed analysis, such as sales by date, item, and shop, which supports both daily and aggregated views for reporting and forecasting purposes.
3. **Identify the Dimensions**:
   - The primary dimensions identified in the schema are:
     - **Date Dimension** (`DATE_DIM`): Provides various time-based attributes, including day, week, month, and year information, enabling temporal analysis like seasonal sales trends.
     - **Shop Dimension** (`SHOPS_DIM`): Contains shop-specific information, such as shop name, city, region, and shop type, supporting geographic and location-based analysis.
     - **Item Dimension** (`ITEM_DIM`): Describes each item with details like item name and category name, allowing for product-based analysis and performance comparison within categories.
   - These dimensions provide the contextual data necessary to "slice and dice" the fact data for comprehensive analysis.
4. **Identify the Facts**:

- The key facts, or measurable data points, in this schema are stored in the `SALES_FACT` table:
  - **Item Price**: The selling price of each item in a transaction.
  - **Item Count (`item_cnt_day`)**: The quantity of each item sold in a transaction.
  - **Total Sales**: A derived measure, calculated as `Item Count * Item Price`, representing the revenue from each transaction.
- These facts are essential for calculating sales revenue, understanding customer demand, and supporting inventory decisions.

5.4 Demonstrating Dimensional Modeling Skills

Our ERD highlights several key aspects of dimensional modeling:

5.4.1 Data Cube Design

The star schema effectively represents a data cube where the `sales_fact` table contains measures, and the connected dimension tables provide the axes (dimensions) along which data can be sliced and diced.

5.4.2 Fact and Dimension Tables

We have a clear separation between fact tables and dimension tables:

- **Fact Table**:
  - `sales_fact`
- **Dimension Tables**:
  - `date_dim`
  - `shops_dim`
  - `item_dim`

5.4.3 Shared Dimensions

Dimensions like `date_dim` and `item_dim` can be shared across multiple fact tables if we expand our model in the future (e.g., including inventory or procurement data), demonstrating conformed dimensions.

5.4.4 Handling Slowly Changing Dimensions (SCD)

We adopt strategies for managing slowly changing dimensions:

- **Type 1 SCD**: For attributes where historical changes are not tracked (e.g., correcting a data entry error).
- **Type 2 SCD**: For attributes where historical changes need to be tracked (e.g., a shop changes its region). This involves adding new records to the dimension table with versioning or effective date ranges.

5.4.5 Surrogate Keys

Each dimension table uses surrogate keys (e.g., `shop_id`, `item_id`, `date_key`) as primary keys. This practice decouples the data warehouse from the operational system and allows for better handling of SCDs.

5.4.6 Degenerate Dimensions

In our `sales_fact` table, attributes like `transaction_id` act as degenerate dimensions. They are operational identifiers that do not warrant a separate dimension table but are useful for tracking at the transaction level.

5.4.7 Granularity and Fact Table Design

By declaring the grain at the most atomic level (one row per item per transaction), we ensure maximum flexibility for analysis. This level of detail supports aggregations across various dimensions without loss of information.

5.4.8 Additive, Non-Additive, and Derived Facts

Our fact table includes:

- **Additive Facts**: `item_cnt_day`, `total_sales`.
- **Derived Facts**: Calculated fields like `total_sales` (though stored for performance).

5.4.9 Dimension Attributes and Hierarchies

Dimension tables include rich attributes:

- **Date Dimension**: Includes calendar and fiscal hierarchies, flags for holidays, weekends, etc.
- **Item Dimension**: Contains category hierarchies, brand information, etc.
- **Shop Dimension**: Includes attributes like location, region, shop type, and size.

5.4.10 Extensibility and Snowflaking

Our design is extensible; new dimensions or attributes can be added as needed. While we avoid snowflaking to maintain simplicity, if necessary, attributes can be normalized (e.g., having a separate table for item categories) but at the cost of query performance.

5.5 Transactional vs. Dimensional Models

5.5.1 OLTP Model

The OLTP model (`oltp_model.sql`) focuses on efficient transaction processing with normalized tables and foreign keys, suitable for operational databases.

5.5.2 Dimensional Model

The dimensional model (`dimensional_model.sql`) is optimized for analytical queries, denormalized for performance, and supports complex aggregations and calculations.

5.6 Handling Nulls and Data Quality

We ensure that foreign keys in the fact table always reference existing dimension records. For unknown or missing dimension data, we implement surrogate keys pointing to default "unknown" records in dimension tables.

5.7 Centipede Fact Tables and High-Dimensionality

While high-dimensionality can lead to "centipede" fact tables with many foreign keys, we carefully select relevant dimensions to balance query performance and analytical needs.

6 Exploratory Data Analysis (EDA)

Performed extensive EDA to understand data distributions and relationships.

6.1 Distribution of Item Count per Day

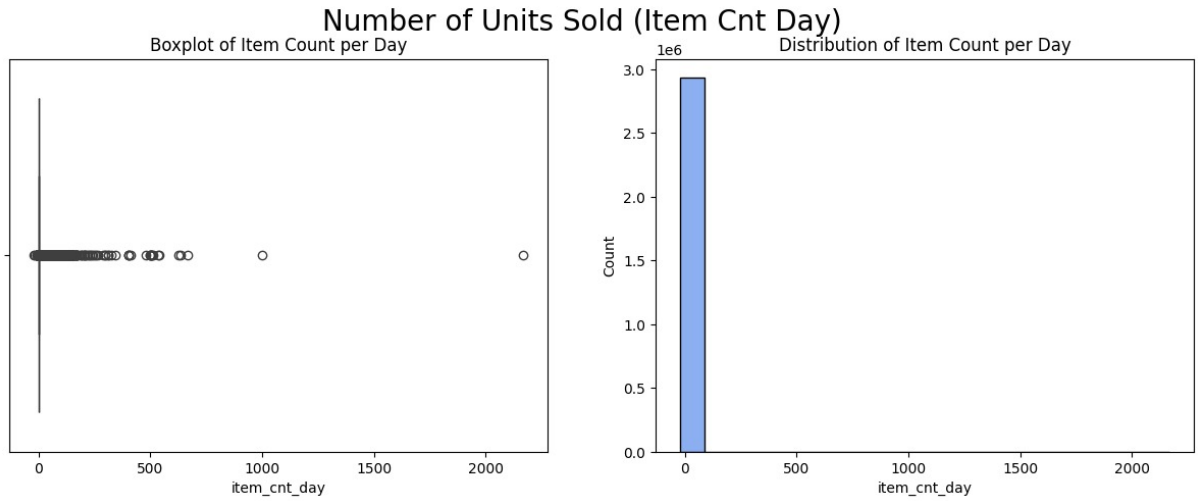

Figure 2: Distribution of Item Count per Day

6.2 Sales per Shop

Analyzed total units sold and turnover per shop.

```
# Sales per shop
df['Sales_per_item'] = df['item_cnt_day'] * df['item_price']
df_shop_sales = df.groupby('shop_id').agg({'Sales_per_item': 'sum', 'item_cnt_day': 'sum'
    }).reset_index()
```

Listing 5: Sales per Shop

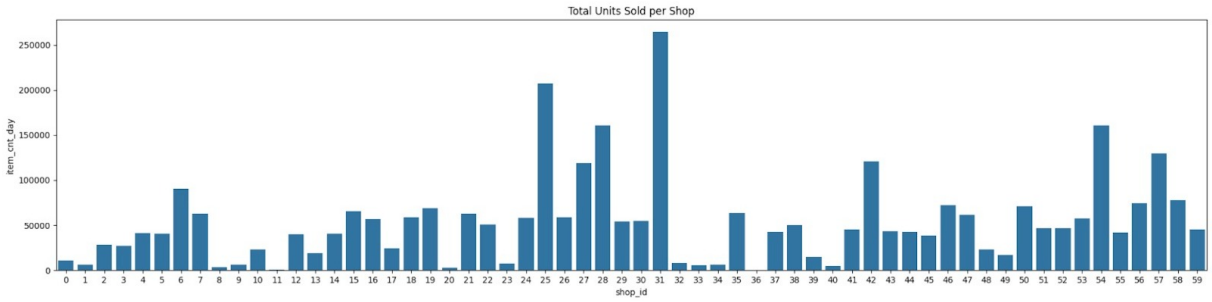

Figure 3: Total Sales per Shop

6.3 Time Series Analysis

Analyzed sales trends over time and identified seasonal patterns.

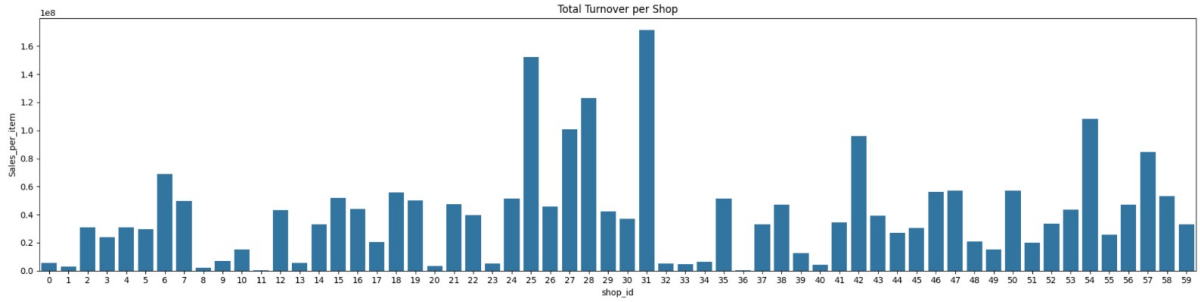

Figure 4: Sales Over Time

7 Reporting, Modeling, and Storytelling

7.1 Analytical SQL Queries and Insights

This section provides key analytical SQL queries developed to extract actionable insights from the data warehouse. Each query is accompanied by an explanation and sample output.

7.1.1 Monthly Sales and Units Sold Trend

Purpose: Analyze the trend of total sales and units sold each month to understand seasonal or temporal variations in performance.

```sql
SELECT
    d.YEAR_MONTH AS Year_Month,
    SUM(sf.total_sales) AS Total_Sales,
    SUM(sf.item_cnt_day) AS Total_Units_Sold
FROM
    sales_fact sf
JOIN
    date_dim d ON sf.date_key = d.DATE_KEY
GROUP BY
    d.YEAR_MONTH
ORDER BY
    Year_Month;
```

Listing 6: Monthly Sales and Units Sold Trend

Sample Output:

Year_Month	Total Sales	Total Units Sold
2023-01	1,234,567	45,678
2023-02	1,456,789	56,789
2023-03	1,345,678	50,123

Table 1: Monthly Sales and Units Sold Trend

7.1.2 Day of the Week Sales Analysis

Purpose: Identify sales patterns based on the day of the week to optimize operations and promotions.

```sql
SELECT
    d.DAY_OF_WEEK,
    CASE
        WHEN d.DAY_OF_WEEK = 0 THEN 'Sunday'
        WHEN d.DAY_OF_WEEK = 1 THEN 'Monday'
        ...
    END AS Day_Name,
    SUM(sf.total_sales) AS Total_Sales,
    SUM(sf.item_cnt_day) AS Total_Units_Sold
FROM
    sales_fact sf
JOIN
    date_dim d ON sf.date_key = d.DATE_KEY
GROUP BY
    d.DAY_OF_WEEK
ORDER BY
    d.DAY_OF_WEEK;
```

Listing 7: Day of the Week Sales Analysis

Sample Output:

Day Name	Total Sales	Total Units Sold
Sunday	345,678	12,345
Monday	567,890	20,567
Tuesday	678,901	22,789

Table 2: Sales by Day of the Week

7.1.3 Cumulative Sales Over Time

Purpose: Calculate monthly sales and cumulative sales over time to observe the growth pattern and overall trends.

```
SELECT
    d.YEAR_MONTH,                                    -- Year and month
    SUM(sf.total_sales) AS Monthly_Sales,            -- Total monthly sales
    SUM(SUM(sf.total_sales)) OVER (ORDER BY d.YEAR_MONTH) AS Cumulative_Sales --
    Cumulative monthly total
FROM
    sales_fact sf
JOIN
    date_dim d ON sf.date_key = d.DATE_KEY
GROUP BY
    d.YEAR_MONTH                                     -- Group by year and month
ORDER BY
    d.YEAR_MONTH;                                    -- Order by year and month
```

Listing 8: Cumulative Sales Over Time

Sample Output:

Year-Month	Monthly Sales	Cumulative Sales
2023-01	1,234,567	1,234,567
2023-02	1,456,789	2,691,356
2023-03	1,345,678	4,037,034

Table 3: Cumulative Sales Over Time

7.1.4 Top 10 Shops by Sales Quartile

Purpose: Rank top-performing shops and categorize them into quartiles based on their sales performance.

```
SELECT
    s.shop_name,
    SUM(sf.total_sales) AS Total_Sales,
    NTILE(4) OVER (ORDER BY SUM(sf.total_sales) DESC) AS sales_quartile
FROM
    sales_fact sf
JOIN
    shops_dim s ON sf.shop_id = s.shop_id
GROUP BY
    s.shop_name
ORDER BY
    Total_Sales DESC
FETCH FIRST 10 ROWS ONLY;
```

Listing 9: Top 10 Shops by Sales Quartile

Sample Output:

Shop Name	Total Sales	Sales Quartile
Shop A	1,234,567	1
Shop B	1,123,456	1
Shop C	1,045,678	2

Table 4: Top 10 Shops by Sales Quartile

7.1.5 Correlation Between Item Price and Units Sold

Purpose: Calculate the correlation between item price and units sold to understand price elasticity.

```
SELECT
    CORR(sf.item_price, sf.item_cnt_day) AS price_units_correlation
FROM
    sales_fact sf
WHERE
```

```
    sf.item_price > 0 AND sf.item_cnt_day > 0;
```

Listing 10: Correlation Between Item Price and Units Sold

Sample Output:

Price-Units Correlation
0.75

Table 5: Correlation Between Item Price and Units Sold

7.1.6 Rolling 3-Month Sales Total per Shop

Purpose: Calculate a 3-month rolling total of sales for each shop to observe trends and fluctuations.

```
WITH monthly_sales AS (
    SELECT
        d.YEAR_MONTH,
        s.shop_name,
        SUM(sf.total_sales) AS total_sales
    FROM
        sales_fact sf
    JOIN
        date_dim d ON sf.date_key = d.DATE_KEY
    JOIN
        shops_dim s ON sf.shop_id = s.shop_id
    GROUP BY
        d.YEAR_MONTH, s.shop_name
)
SELECT
    ms.YEAR_MONTH,
    ms.shop_name,
    SUM(ms.total_sales) OVER (
        PARTITION BY ms.shop_name
        ORDER BY ms.YEAR_MONTH
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS rolling_3_month_sales
FROM
    monthly_sales ms
ORDER BY
    ms.shop_name, ms.YEAR_MONTH;
```

Listing 11: Rolling 3-Month Sales Total per Shop

Sample Output:

Shop Name	Year-Month	Rolling 3-Month Sales
Shop A	2023-01	1,234,567
Shop A	2023-02	1,678,901
Shop A	2023-03	1,890,123

Table 6: Rolling 3-Month Sales Total per Shop

7.1.7 Quarterly Sales Growth

Purpose: Calculate the quarterly sales growth and percentage using the LAG function

```
SELECT
    q.current_quarter AS Year_Quarter,
    q.total_sales,
    q.total_sales - NVL(q.previous_quarter_sales, 0) AS sales_growth,
    ROUND(((q.total_sales - NVL(q.previous_quarter_sales, 0)) /
           NULLIF(q.previous_quarter_sales, 0)) * 100, 2) AS growth_percentage
FROM (
    SELECT
        d.YEAR_QUARTER AS current_quarter,
        SUM(sf.total_sales) AS total_sales,
        LAG(SUM(sf.total_sales)) OVER (ORDER BY d.YEAR_QUARTER) AS previous_quarter_sales
    FROM
        sales_fact sf
```

```
    JOIN
        date_dim d ON sf.date_key = d.DATE_KEY
    GROUP BY
        d.YEAR_QUARTER
) q
ORDER BY
    q.current_quarter;
```

Listing 12: Quarterly Sales Growth

Sample Output:

Year-Quarter	Total Sales	Sales Growth	Growth Percentage
2023-Q1	1,234,567	-	-
2023-Q2	1,567,890	333,323	27.0%
2023-Q3	1,678,901	111,011	7.1%

Table 7: Quarterly Sales Growth

7.1.8 Year-Over-Year Growth by Month

Purpose: Compare sales from the same month across different years to calculate Year-over-Year (YOY) growth.

```
SELECT
    d.YEAR_MONTH AS Year_Month,
    SUM(sf.total_sales) AS Total_Sales,
    LAG(SUM(sf.total_sales)) OVER (PARTITION BY d.MONTH_NUMBER ORDER BY d.YEAR_NUMBER) AS
     Previous_Year_Sales,
    SUM(sf.total_sales) - LAG(SUM(sf.total_sales)) OVER (PARTITION BY d.MONTH_NUMBER
    ORDER BY d.YEAR_NUMBER) AS Year_Over_Year_Growth,
    ROUND(((SUM(sf.total_sales) - LAG(SUM(sf.total_sales)) OVER (PARTITION BY d.
    MONTH_NUMBER ORDER BY d.YEAR_NUMBER)) /
           NULLIF(LAG(SUM(sf.total_sales)) OVER (PARTITION BY d.MONTH_NUMBER ORDER BY d.
    YEAR_NUMBER), 0)) * 100, 2) AS Growth_Percentage
FROM
    sales_fact sf
JOIN
    date_dim d ON sf.date_key = d.DATE_KEY
GROUP BY
    d.YEAR_MONTH, d.MONTH_NUMBER, d.YEAR_NUMBER
ORDER BY
    d.YEAR_MONTH;
```

Listing 13: Year-Over-Year Growth by Month

Sample Output:

Year-Month	Total Sales	Previous Year Sales	YOY Growth	Growth Percentage
2023-01	1,234,567	1,045,678	188,889	18.1%
2023-02	1,456,789	1,234,567	222,222	18.0%
2023-03	1,345,678	1,567,890	-222,212	-14.2%

Table 8: Year-Over-Year Growth by Month

7.1.9 Monthly Sales by Category

Purpose: Track monthly sales performance by category to identify trends and opportunities for growth.

```
SELECT
    d.YEAR_MONTH AS Year_Month,
    i.category_name AS Category,
    SUM(sf.total_sales) AS Total_Sales
FROM
    sales_fact sf
JOIN
    date_dim d ON sf.date_key = d.DATE_KEY
JOIN
    item_dim i ON sf.item_id = i.item_id
```

```
GROUP BY
    d.YEAR_MONTH, i.category_name
ORDER BY
    Year_Month, Total_Sales DESC;
```

Listing 14: Monthly Sales by Category

Sample Output:

Year-Month	Category	Total Sales
2023-01	Electronics	678,901
2023-01	Furniture	345,678
2023-01	Apparel	210,456

Table 9: Monthly Sales by Category

7.1.10 Top 5 Items by Units Sold per Shop

Purpose: Identify the top 5 products by units sold for each shop to improve inventory planning.

```
WITH RankedProducts AS (
    SELECT
        s.shop_name AS Shop,
        i.item_name AS Product,
        SUM(sf.item_cnt_day) AS Units_Sold,
        RANK() OVER (PARTITION BY s.shop_name ORDER BY SUM(sf.item_cnt_day) DESC) AS
    Sales_Rank
    FROM
        sales_fact sf
    JOIN
        shops_dim s ON sf.shop_id = s.shop_id
    JOIN
        item_dim i ON sf.item_id = i.item_id
    GROUP BY
        s.shop_name, i.item_name
)
SELECT
    Shop,
    Product,
    Units_Sold,
    Sales_Rank
FROM
    RankedProducts
WHERE
    Sales_Rank <= 5
ORDER BY
    Shop, Sales_Rank;
```

Listing 15: Top 5 Items by Units Sold per Shop

Sample Output:

Shop	Product	Units Sold	Rank
Shop A	Product X	1,234	1
Shop A	Product Y	1,123	2
Shop A	Product Z	1,045	3

Table 10: Top 5 Items by Units Sold per Shop

7.2 Feature Selection

Selected features based on correlation analysis and domain knowledge.

- `date_block_num`
- `shop_id`
- `item_id`

- item_price
- Lag features and rolling means

7.3 Modeling

Built a Random Forest Regressor model to predict future sales.

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Split data
X_train, X_test, y_train, y_test  = train_test_split(X, y, train_size=0.1, random_state
    =42)

# Train model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Evaluate model
y_pred = rf.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f'Random Forest MAE: {mae}')
```

Listing 16: Model Training

7.4 Feature Importance

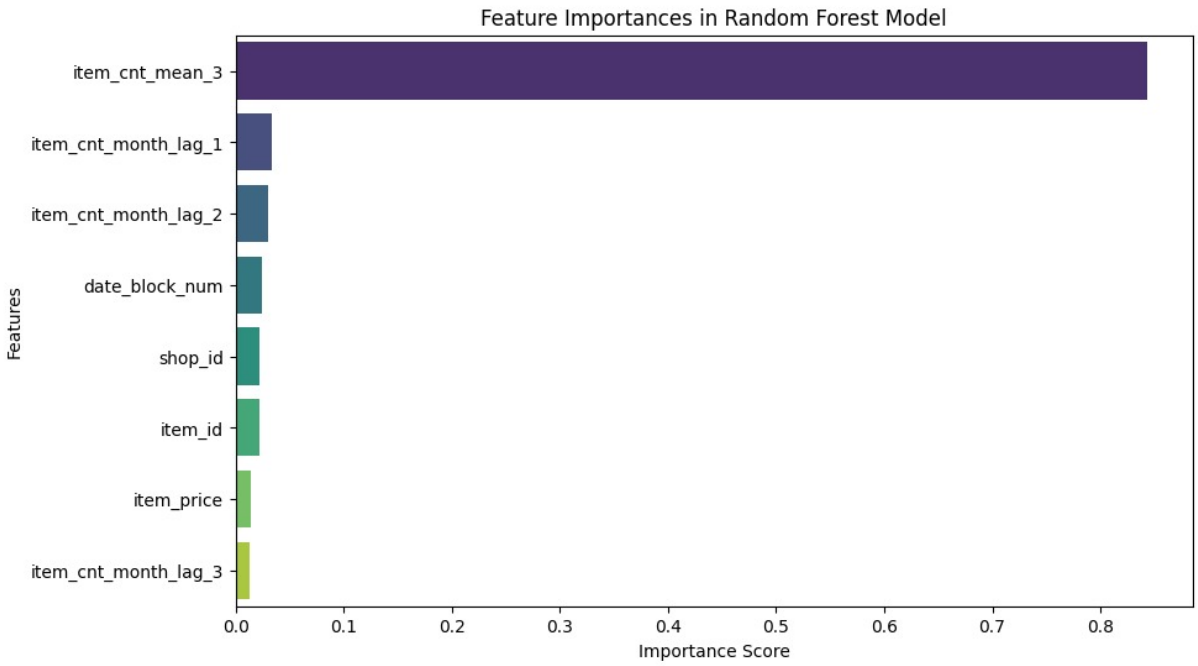

Figure 5: Feature Importances in Random Forest Model

7.5 Discussion

The Random Forest model achieved a Mean Absolute Error (MAE) of **0.1042**. The most significant features were the lag features and rolling means, indicating that past sales data is a strong predictor of future sales. Seasonal patterns were also evident in the data, reinforcing the importance of temporal features.

8 Conclusions

The project successfully:

- Demonstrated the application of data warehousing concepts and dimensional modeling.
- Showcased an ERD that highlights data cube design, fact and dimension tables, shared dimensions, and strategies for handling slowly changing dimensions.
- Performed extensive data preprocessing and exploratory data analysis.
- Built a predictive model to forecast future sales with reasonable accuracy.
- Created analytical queries and views to extract insights from the data warehouse.

Future work could involve:

- Incorporating additional features such as promotions or external economic indicators.
- Exploring other machine learning algorithms like Gradient Boosting or Neural Networks.
- Deploying the model in a production environment for real-time forecasting.

A Appendix

A.1 Power BI Dashboard

The Power BI dashboard developed for this project provides an interactive visualization of the data insights. You can access the dashboard using the following link:

Power BI Dashboard Link

A.2 Project Dashboard Image

The following figure showcases an example screenshot of the Power BI dashboard developed as part of the project:

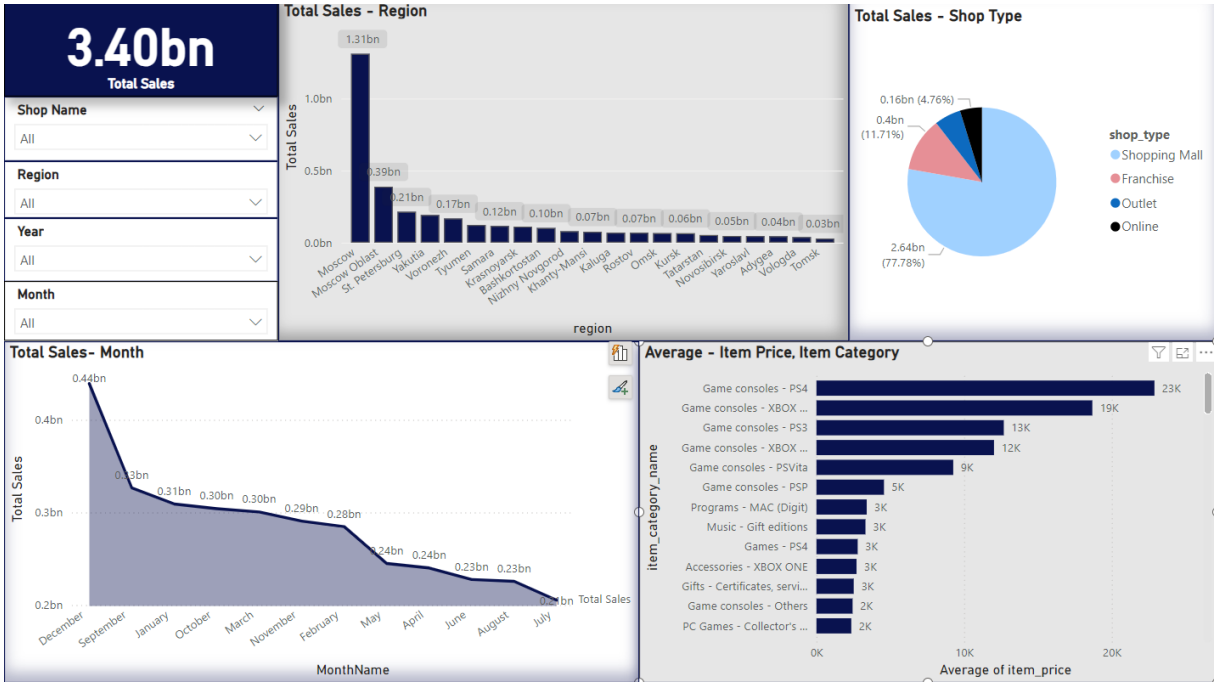

Figure 6: Power BI Dashboard Overview

B References

1. Kaggle Dataset: English Converted Datasets
2. Inmon, W. H. (1995). *Building the Data Warehouse.* John Wiley & Sons.
3. Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit.* John Wiley & Sons.
4. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media.
5. Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice.* OTexts.