Data Warehouse Implementation Report

1. Assignment Overview

This report presents the **design and implementation of a data warehouse** using **Microsoft Azure for data storage and Databricks for ETL and analytics**. The project involves **sourcing raw data, transforming it, and structuring it into a star schema** to facilitate analytical querying and business intelligence insights.

2. Data Sources

This dataset was sourced from the **Kaggle E-Commerce (Target) Sales Dataset**, which provides a detailed overview of Target's Brazilian operations and customer data. It includes information on **100,000 orders placed between 2016 and 2018**, covering aspects such as order status, pricing, payment, shipping performance, customer locations, product attributes, and customer reviews.

Dataset Overview

Target is a globally recognized brand and a leading retailer in the United States, known for offering exceptional value, innovation, and an unparalleled shopping experience. The dataset focuses on Target's Brazilian operations, providing valuable insights into business strategies and customer behavior.

Potential Use Cases:

- Understanding order processing and pricing strategies
- Evaluating payment and shipping efficiency
- Analyzing customer demographics and preferences
- Studying product characteristics and customer satisfaction

Dataset Files

The following datasets were used:

Dataset	**File Name**	**Description**
Customers	*customers.csv*	Contains customer details such as ID, location, and state.
Geolocation	*geolocation.csv*	Provides zip-code-based latitude and longitude information.
Orders	*orders.csv*	Includes order-related details such as status, timestamps, and estimated delivery dates.
Order Items	*order_items.csv*	Contains individual order item details such as product ID, seller ID, price, and freight cost.
Payments	*payments.csv*	Stores payment-related data such as payment type and amount paid.
Products	*products.csv*	Contains product information including category, weight, and dimensions.
Sellers	*sellers.csv*	Includes seller details such as seller ID, location, and state.

The full data dictionary, describing all fields in the dataset, is provided separately in the **README file** accompanying this report.

3. ETL Process Implementation

The ETL process follows the below steps:

3.1 Data Extraction

- The raw datasets stored in **Azure Blob Storage** are mounted to **Databricks**.
- Data is extracted using PySpark and stored in **staging tables** within **Databricks Delta Lake**.

3.2 Data Transformation

- **Data Cleaning:** Handling missing values, standardizing column names, and normalizing data types.
- **Handling Surrogate Keys:** Assigning unique identifiers (surrogate keys) for dimension tables.
- **Implementing Slowly Changing Dimensions (SCD Type 2):** Maintaining historical records for dimensions like customers, products and sellers.

3.3 Data Loading

- Transformed data is loaded into the **data warehouse using a star schema**.

4. Data Warehouse Design: Star Schema

4.1 Dimension Tables:

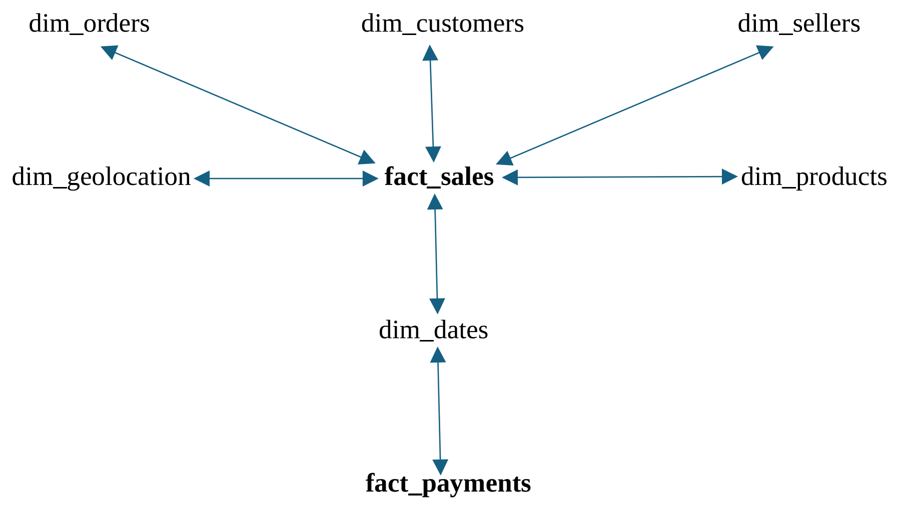
Table Name	Primary Key	Description
dim_customers	customer_key	Stores customer details with SCD Type 2 implementation.
dim_geolocation	geolocation_key	Contains geographic information for customers and sellers.
dim_products	product_key	Includes product details such as category, weight, and dimensions.
dim_sellers	seller_key	Stores seller information including location.
dim_dates	date_key	Maintains order dates, delivery dates, and other timestamps.
dim_orders	order_key	Stores order-level information such as order ID and status.

4.2 Fact Tables:

Table Name	Primary Key	Description
fact_sales	order_id	Stores sales transactions including price, freight, and shipping details.
fact_payments	order_id	Contains payment-related data such as amount and payment method.

4.3 Star Schema Design

Visual Representation of the Star Schema:



1. Fact Table: fact_sales

Primary Fact Table - Stores transaction-level sales data.
Connected to:

- **dim_customers** → To track which customer placed the order (customer_key).
- **dim_products** → To identify which products were sold (product_key).
- **dim_sellers** → To determine which seller sold the item (seller_key).
- **dim_dates** → To track order timestamps (order_date_key).
- **dim_orders** → To link order-specific details (order_key).
- **dim_geolocation** → To track location-based insights (geolocation_key).

2. Fact Table: fact_payments

Stores payment transactions for each order.
Connected to:

- **fact_sales** → Payments belong to orders (order_id).
- **dim_dates** → Payment dates (date_key).

3. Dimension Tables:

Dimension	Linked to Fact Table(s)	Purpose
dim_customers	fact_sales	Tracks customer details using customer_key.
dim_products	fact_sales	Provides product-specific attributes.
dim_sellers	fact_sales	Identify the seller for each sale.
dim_dates	fact_sales, fact_payments	Used for date-based filtering and trends.
dim_orders	fact_sales	Stores order status and metadata.
dim_geolocation	fact_sales	Enables location-based analytics.

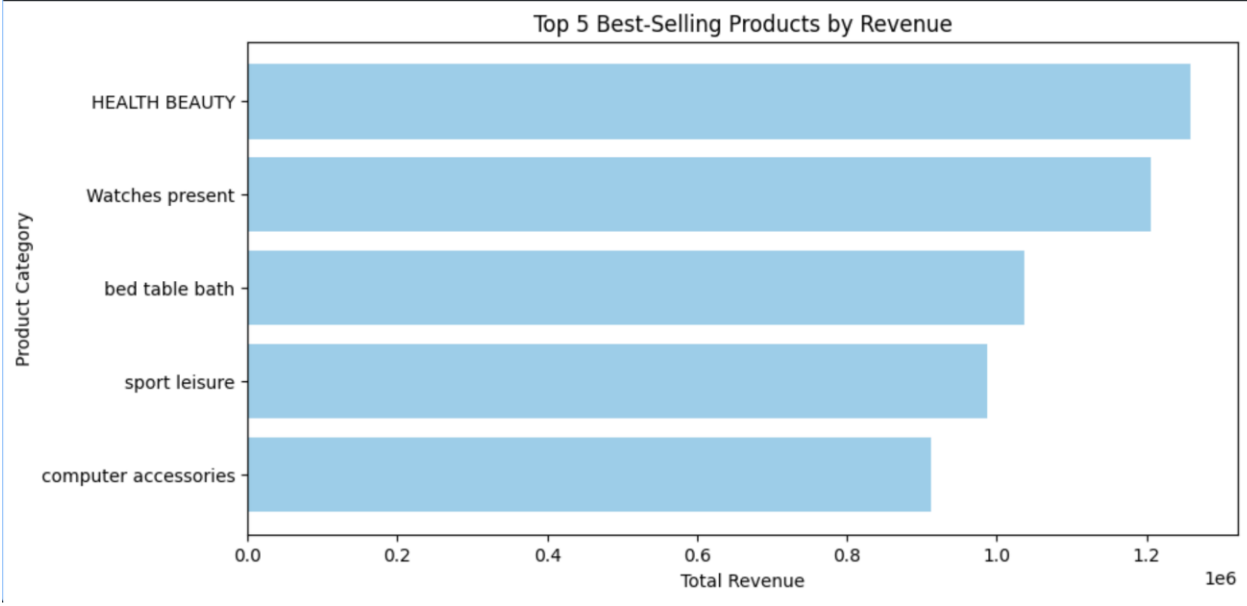
This structure ensures **optimized analytical querying** while maintaining **data integrity and reducing redundancy**.

5. Analytical Querying and Business Insights

5.1 Query 1: Top 5 Best-Selling Product Categories

Objective: Identify product categories that generate the most revenue.

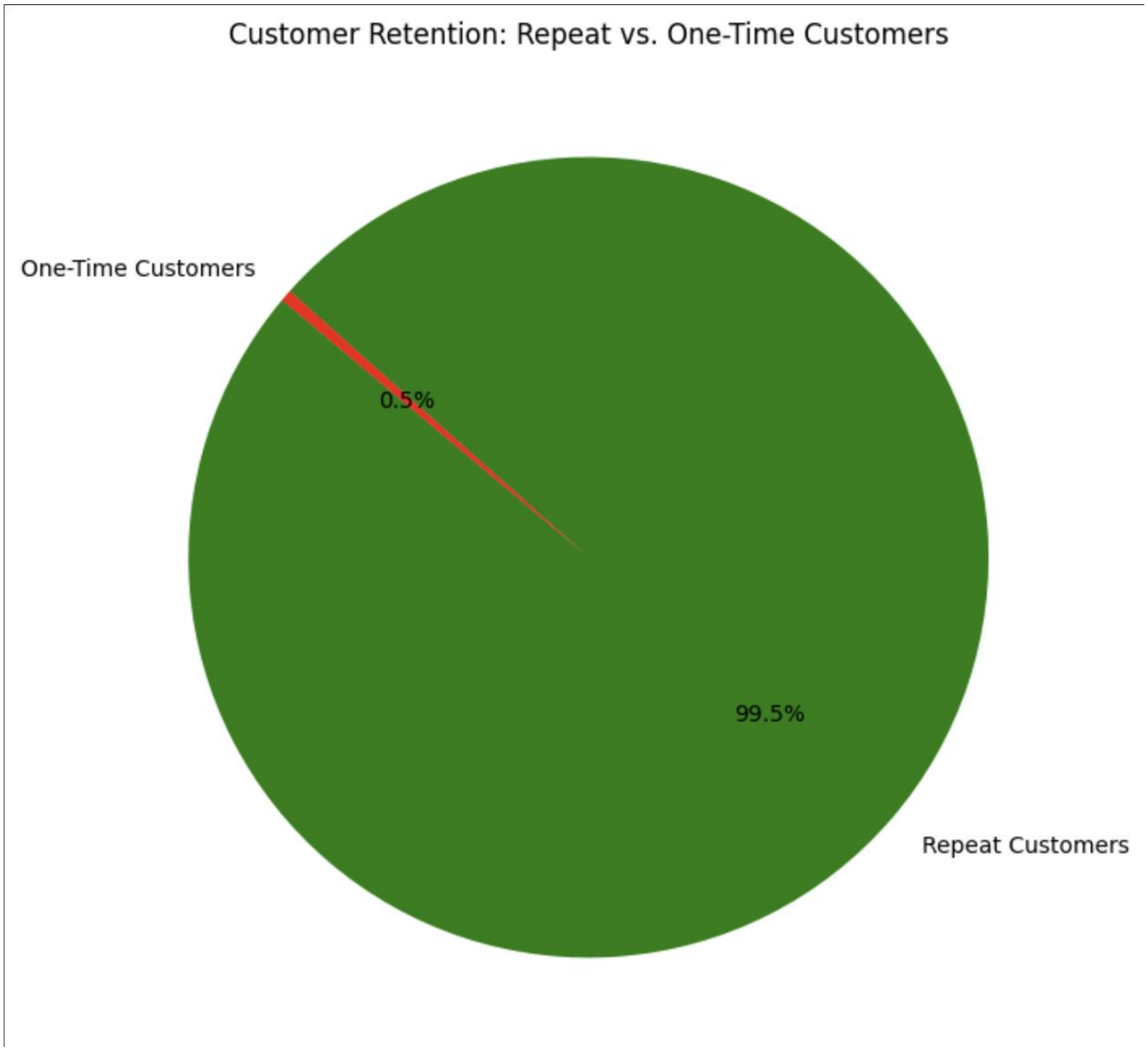
```
import matplotlib.pyplot as plt

df_best_products = spark.sql("""
SELECT p.product_category,
      ROUND(SUM(s.price)) AS total_revenue
FROM fact_sales s
JOIN dim_products p ON s.product_key = p.product_key
GROUP BY p.product_category
ORDER BY total_revenue DESC
LIMIT 5
""").toPandas()

plt.figure(figsize=(10, 5))
plt.barh(df_best_products["product_category"], df_best_products["total_revenue"], color="skyblue")
plt.xlabel("Total Revenue")
plt.ylabel("Product Category")
plt.title("Top 5 Best-Selling Products by Revenue")
plt.gca().invert_yaxis()
plt.show()
```



5.2 Query 2: Customer Retention Analysis

Objective: Identify the percentage of repeat customers vs. one-time buyers.

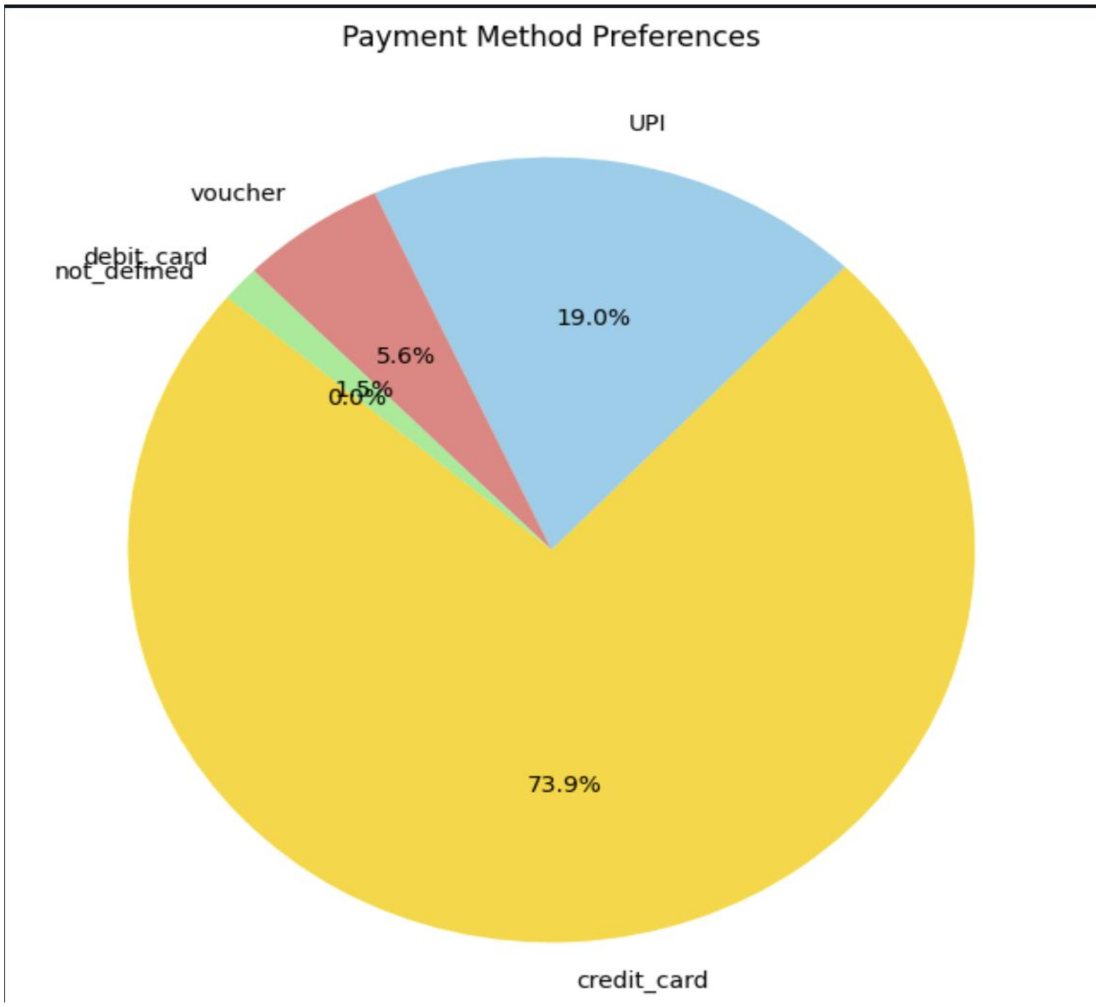
```
df_customer_retention = spark.sql("""
WITH CustomerOrders AS (
   SELECT c.customer_unique_id, COUNT(f.order_id) AS total_orders
   FROM fact_sales f
   JOIN dim_customers c ON f.customer_key = c.customer_key
   GROUP BY c.customer_unique_id
)
SELECT COUNT(CASE WHEN total_orders > 1 THEN 1 END) AS repeat_customers,
      COUNT(CASE WHEN total_orders = 1 THEN 1 END) AS one_time_customers
FROM CustomerOrders;
""").toPandas()
```



5.3 Query 3: Payment Method Preferences

Objective: Analyze customer payment preferences.

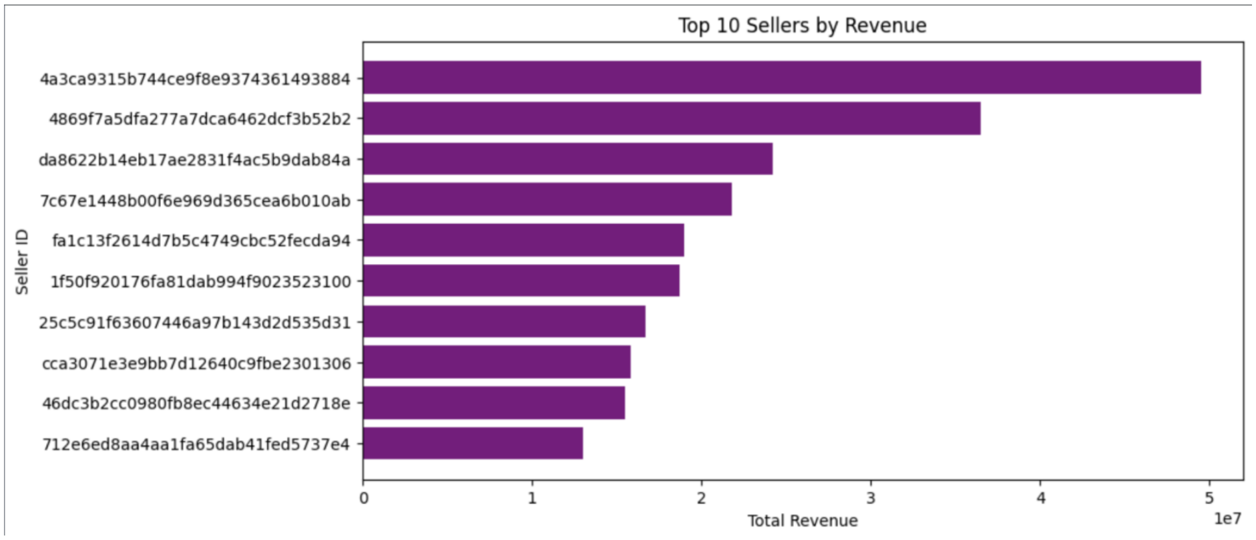
```
df_payment_methods = spark.sql("""
SELECT p.payment_type,
    ROUND(COUNT(p.order_id)) AS total_transactions
FROM fact_payments p
GROUP BY p.payment_type
ORDER BY total_transactions DESC
""").toPandas()

plt.figure(figsize=(8, 8))
plt.pie(df_payment_methods["total_transactions"], labels=df_payment_methods["payment_type"],
autopct="%1.1f%%", startangle=140, colors=["gold", "skyblue", "lightcoral", "lightgreen"])
plt.title("Payment Method Preferences")
plt.show()
```



5.4 Query 4: Top 10 Sellers by Revenue

Objective: Identify the top revenue-generating sellers. Since there is no Name attribute in the dataset, SellerID is used for representation.

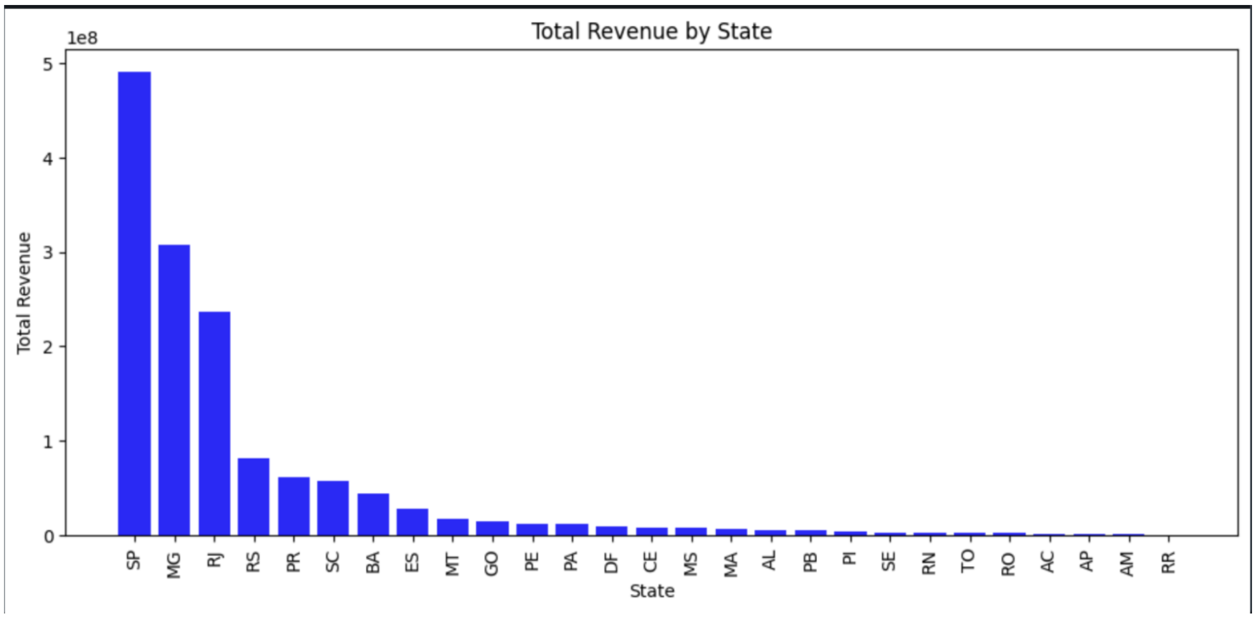
```
df_top_sellers = spark.sql("""
SELECT s.seller_id,
    g.city,
    g.state,
    ROUND(SUM(f.price)) AS total_revenue
FROM fact_sales f
JOIN dim_sellers s ON f.seller_key = s.seller_key
JOIN dim_geolocation g ON s.seller_zip_code_prefix = g.zip_code_prefix
GROUP BY s.seller_id, g.city, g.state
ORDER BY total_revenue DESC
LIMIT 10
""").toPandas()

plt.figure(figsize=(10, 5))
plt.barh(df_top_sellers["seller_id"], df_top_sellers["total_revenue"], color="purple")
plt.xlabel("Total Revenue")
plt.ylabel("Seller ID")
plt.title("Top 10 Sellers by Revenue")
plt.gca().invert_yaxis()
plt.show()
```



5.5 Query 5: Total Revenue by State

Objective: Identify the states generating the highest revenue and order volume.

```
df_revenue_by_state = spark.sql("""
SELECT g.state,
    COUNT(f.order_id) AS total_orders,
    ROUND(SUM(f.price)) AS total_revenue
FROM fact_sales f
JOIN dim_customers c ON f.customer_key = c.customer_key
JOIN dim_geolocation g ON c.customer_zip_code_prefix = g.zip_code_prefix
GROUP BY g.state
ORDER BY total_revenue DESC
""").toPandas()

plt.figure(figsize=(12, 5))
plt.bar(df_revenue_by_state["state"], df_revenue_by_state["total_revenue"], color="blue")
plt.xlabel("State")
plt.ylabel("Total Revenue")
plt.title("Total Revenue by State")
plt.xticks(rotation=90)
plt.show()
```



6. Conclusion and Future Improvements

This data warehouse implementation successfully provides **scalable, efficient, and insightful analytics**. Future enhancements could include:

- **Integrating machine learning models** for demand forecasting.
- **Optimizing query performance** using indexing and materialized views.
- **Automating ETL workflows** with Apache Airflow.