BlinkIt Sales Report

1. Data Sources

Raw Data Files in Original Format

The raw data is stored in **Azure Blob Storage** and mounted to **Databricks** under /mnt/blinkit_storage/. The sources include:

- blinkit_customer_feedback.csv
- blinkit_customers.csv
- blinkit_delivery_performance.csv
- blinkit_inventory.csv
- blinkit_marketing_performance.csv
- blinkit_products.csv
- blinkit_orders.csv
- blinkit_order_items.csv

Documentation Describing Sources & Relationships

Each dataset contributes to different business aspects:

- **Customers**: Contains customer profiles and segments.
- **Orders**: Tracks all customer orders.
- **Order Items**: Itemized breakdown of each order.
- **Products**: Product catalog with pricing and categories.
- **Delivery Performance**: Records actual vs. promised delivery times.
- **Marketing Performance**: Details promotional campaigns.
- **Customer Feedback**: Captures ratings and sentiments.
- **Inventory**: Logs stock movements and damages.

Data Dictionary

- customer_id: Unique identifier for customers.
- order_id: Unique identifier for orders.
- product_id: Unique identifier for products.
- order_total: Total revenue from an order.
- delivery_status: Indicates whether an order was delivered on time.
- feedback_category: Category of customer feedback.
- campaign_name: Name of marketing campaign.

2. Staging Delta Tables

Staging Tables in Delta Format

The data from **Azure Blob Storage** was loaded into **staging Delta tables** in Databricks:

```
df_staging_customers = spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load("dbfs:/mnt/blinkit_storage/blinkit_customers.csv")
df_staging_customers.write.format("delta").mode("overwrite").saveAsTable("stg_customers")
```

Each dataset was stored in **staging Delta tables** for further transformation.

3. ETL Implementation

Complete ETL Code/Workflows

The ETL pipeline includes:

- Reading raw data from **Azure Blob Storage**.
- Storing data in **staging Delta tables**.
- Transforming data for analytical queries.

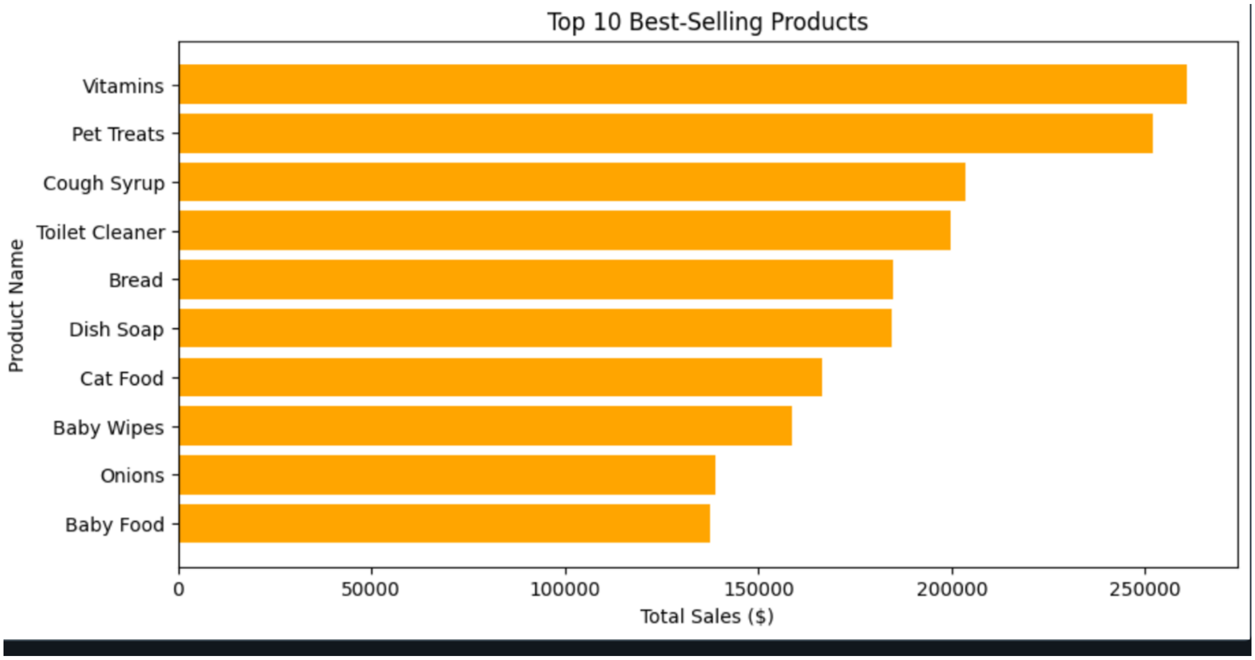
Handling ETL Challenges

- **Data Quality**: Ensured via schema inference and type validation.
- **Slowly Changing Dimensions (SCD Type 2)**: Implemented for Customers & Products.
- **Handling Missing Values**: Used `fillna()` to handle null values.

4. Analytical Queries

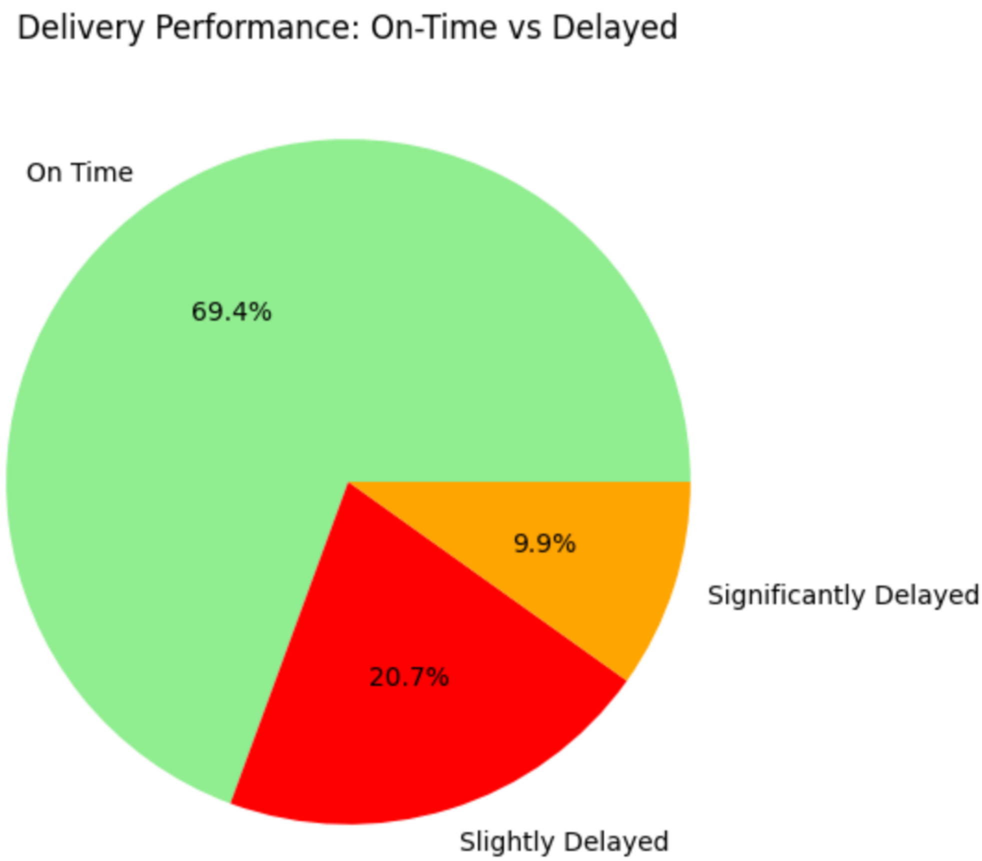
Query 1: Top 10 Best Selling Products

```
df_top_products = spark.sql("""
  SELECT
    p.product_name,
    SUM(f.quantity) AS total_units_sold,
    SUM(f.quantity * f.unit_price) AS total_sales
  FROM fact_orders f
  JOIN dim_products p ON f.product_id = p.product_id
  GROUP BY p.product_name
  ORDER BY total_sales DESC
  LIMIT 10
""")
```



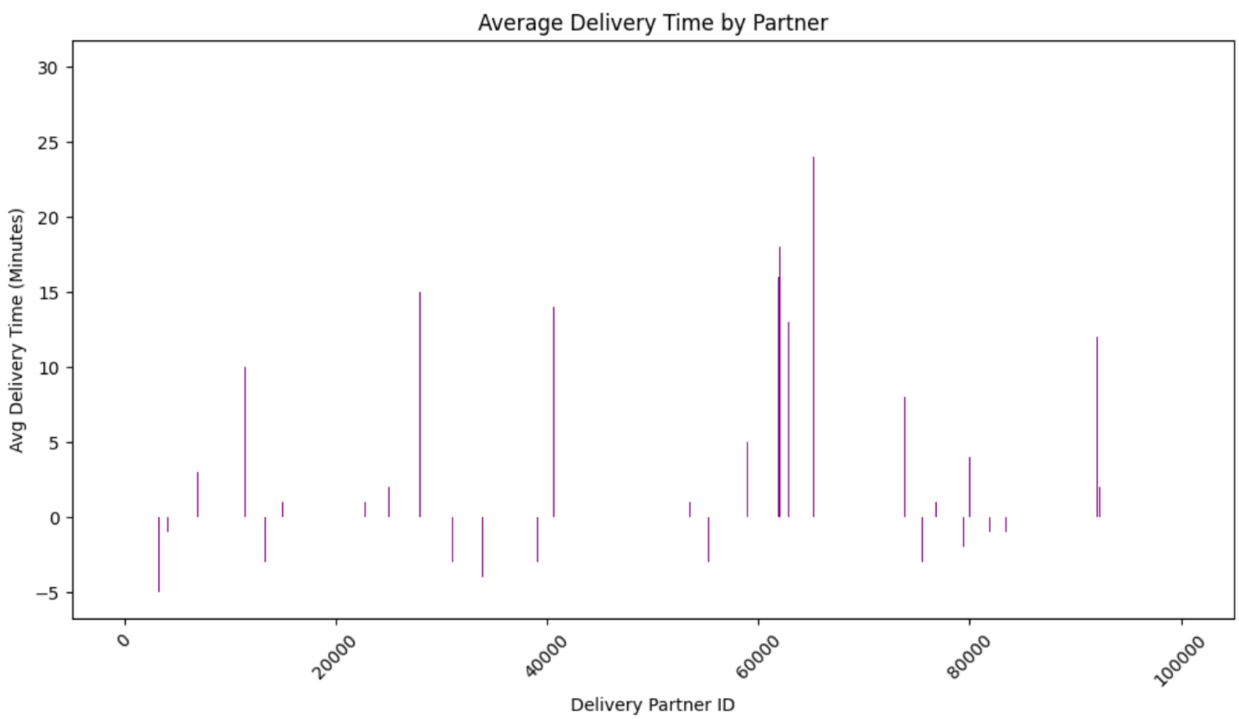
Query 2: On-Time vs Delayed Delivery

```
df_delivery_performance = spark.sql("""
  SELECT
    d.delivery_status,
    COUNT(f.order_id) AS total_orders,
    AVG(d.delivery_time_minutes) AS avg_delivery_time
  FROM fact_orders f
  JOIN dim_delivery d ON f.delivery_partner_id = d.delivery_partner_id
  GROUP BY d.delivery_status ”””)
```



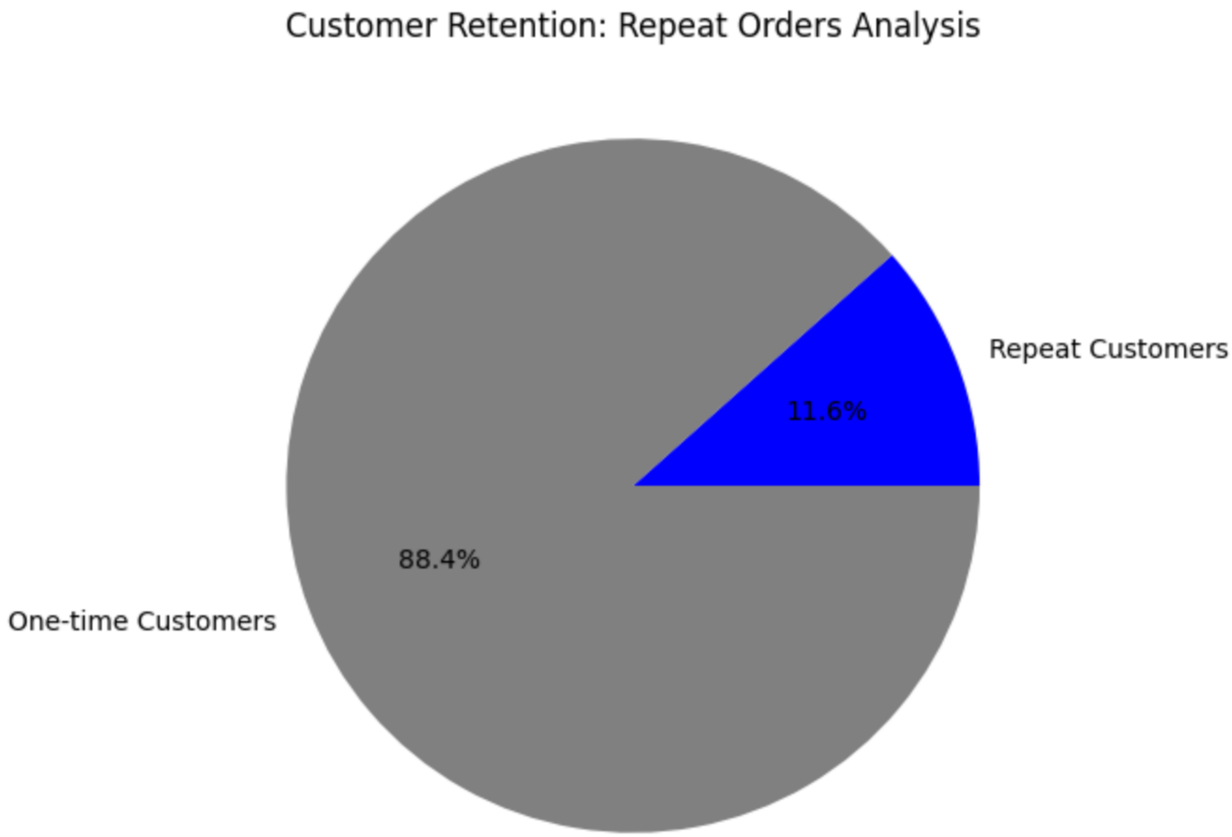
Query 3: Average Delivery Time by Partner

```
df_delivery_time = spark.sql("""
  SELECT
     d.delivery_partner_id,
     AVG(d.delivery_time_minutes) AS avg_delivery_time,
     COUNT(f.order_id) AS total_deliveries
  FROM fact_orders f
  JOIN dim_delivery d ON f.delivery_partner_id = d.delivery_partner_id
  GROUP BY d.delivery_partner_id
  ORDER BY avg_delivery_time
""")
```



Query 4: Customer Retention

```
df_repeat_customers = spark.sql("""
   SELECT
       COUNT(DISTINCT CASE WHEN c.total_orders > 1 THEN c.customer_id END) AS repeat_customers,
       COUNT(DISTINCT c.customer_id) AS total_customers
   FROM dim_customers c
""")
```



5. Summary of Project Approach & Implementation

This project implemented a **data warehouse** to analyze e-commerce transactions. The pipeline:

1. **Extracted** raw data from **Azure Blob Storage** into Databricks.
2. **Stored** data in **staging Delta tables**.
3. **Transformed** data for analytical queries.
4. **Executed analytical queries** to generate insights.

Challenges & Solutions

- **Handling SCD (Slowly Changing Dimensions)**: Implemented SCD Type 2.
- **Ensuring Data Integrity**: Used constraints and data validation.
- **Optimizing Query Performance**: Partitioned `fact_orders` on `time_key`.

Effectiveness of the ETL Pipeline

- **Simplified Data Management**: Delta tables provide incremental updates.
- **Performance Optimization**: Queries run efficiently on Databricks.

- **Scalability**: Can accommodate future data growth.

Recommendations for Future Improvements

- **Real-time data streaming** for instant analytics.
- **Machine learning-based predictions** for customer behavior.
- **Improved indexing** for faster query performance.
