**Technical Appendix**

**Data Set –**

I explored a detailed dataset from a fictional smartphone retail outlet that provides a comprehensive view of sales across an entire fiscal year. It includes product codes, product types (mobiles and accessories), quantities, prices, total amounts, and payment methods. [ [Dataset link](https://www.kaggle.com/datasets/shubham2703/smartphone-retail-outlet-sales-data). ]

| **Column Name** | **Description** |
| --- | --- |
| Date | The date of the transaction (format: dd-mm-yyyy) |
| F.Y | Fiscal year during which the transaction occurred |
| QUARTER | The quarter of the fiscal year (e.g., 1, 2, 3, or 4) |
| P\_NO | Product number or product code |
| PAYMENT TYPE | Method used for payment (e.g., CASH, CARD, ONLINE) |
| TYPE OF PRODUCT | General category of the product (e.g., ACCESSORY, MOBILE) |
| Quantity | Number of units purchased |
| Price | Price per unit of the product |
| Amount | Total amount of the transaction (Quantity × Price) |
| TYPE OF ACCESSORY/MOBILE | Specific details of the product (e.g., COVER, WIRELESS HEADSET, SMARTPHONE) |

**Data Cleaning Process**

Before normalization, the raw dataset was carefully cleaned to ensure high data quality. The following steps were performed programmatically using Python (pandas), replicating and expanding on the manual Excel cleaning you initially did.

1. **Date Standardization**  
   The Date column had inconsistent formats (some with hyphens, some with slashes). We replaced hyphens (-) with slashes (/) and converted the column to a consistent date format using pd.to\_datetime(). This ensured all date values were recognized properly as datetime objects.
2. **Blank Values**  
   We checked for missing values using df.isnull().sum(). The only blank cell was found in the Product No (P\_NO) column. We filled this blank with 'UNKNOWN' as a placeholder, or you can optionally fill it using domain knowledge if available.
3. **Numeric Cleanup**  
   Columns Price, Quantity, and Amount were forced to numeric types using pd.to\_numeric(errors='coerce') to handle any accidental text entries or corrupt data.
4. **Amount Recalculation**  
   To ensure consistency, we recalculated the Amount column as Price \* Quantity and replaced the original values.
5. **Text Column Standardization**  
   We stripped extra spaces and converted values in PAYMENT TYPE, TYPE OF PRODUCT, and TYPE OF ACCESSORY/MOBILE to uppercase for consistent grouping and reporting.
6. **Final Check**  
   A final missing values check confirmed no remaining nulls, ensuring the dataset was clean and ready for normalization.

**Data Normalization Process**

We started with the **Smartphone Retail Outlet Sales** dataset, which originally included the following columns:  
Date, Financial Year (F.Y), Quarter, Product No (P\_NO), Payment Type, Type of Product, Type of Accessory/Mobile, Quantity, Price, Amount

To achieve better structure and eliminate redundancy, we normalized the dataset into four tables, targeting **Third Normal Form (3NF)**.

**Splitting the tables**

| **Table** | **Description** | **Primary Key (PK)** | **Foreign Keys (FKs)** |
| --- | --- | --- | --- |
| DimDate | Contains unique dates, financial years, and quarters. | date\_id | — |
| DimProduct | Contains unique product numbers, product types, and accessory/mobile types. | product\_id | — |
| DimPayment | Contains unique payment types (Cash, Credit, Debit). | payment\_id | — |
| FactSales | Stores all transactions: sales ID, quantity, price, amount, and foreign keys. | sales\_id | date\_id, product\_id, payment\_id |

**Ensuring 3NF (Third Normal Form)**

* **1NF**: All tables have atomic columns with no repeating groups.
* **2NF**: All non-key columns depend fully on the primary key (for example, product details depend only on product\_id).
* **3NF**: No transitive dependencies; for example, payment\_id determines only payment type, and product details are separated from sales facts.

This design prevents:

* Redundant storage of product and payment details
* Update anomalies
* Inconsistent reporting across time and product lines

**SQL Queries Explanation**

1. **Total sales amount by product type**

SELECT dp."TYPE OF PRODUCT", SUM(fs."Amount") AS total\_sales

FROM FactSales fs

JOIN DimProduct dp ON fs.product\_id = dp.product\_id

GROUP BY dp."TYPE OF PRODUCT"

ORDER BY total\_sales DESC;

Groups sales by product type and calculates total revenue.

1. **Payment type with highest revenue**

SELECT dpay."PAYMENT TYPE", SUM(fs."Amount") AS total\_sales

FROM FactSales fs

JOIN DimPayment dpay ON fs.payment\_id = dpay.payment\_id

GROUP BY dpay."PAYMENT TYPE"

ORDER BY total\_sales DESC;  
Shows revenue contribution of each payment method.

1. **Monthly sales trends across financial years**

SELECT strftime('%Y-%m', dd.Date) AS month, dd."F.Y", SUM(fs."Amount") AS total\_sales

FROM FactSales fs

JOIN DimDate dd ON fs.date\_id = dd.date\_id

GROUP BY month, dd."F.Y"

ORDER BY month;

Aggregates sales amounts by month and year for trend analysis.

1. **Accessory/mobile type with highest sales quantity**

SELECT dp."TYPE OF ACCESSORY/MOBILE", SUM(fs.Quantity) AS total\_quantity

FROM FactSales fs

JOIN DimProduct dp ON fs.product\_id = dp.product\_id

GROUP BY dp."TYPE OF ACCESSORY/MOBILE"

ORDER BY total\_quantity DESC;

Identifies the most sold accessory or mobile type by quantity.

1. **Top 5 product codes by revenue**

SELECT dp."P\_NO", SUM(fs."Amount") AS total\_sales

FROM FactSales fs

JOIN DimProduct dp ON fs.product\_id = dp.product\_id

GROUP BY dp."P\_NO"

ORDER BY total\_sales DESC

LIMIT 5;

Ranks the top five products by total sales revenue.

**Pandas Queries Explanation**

1. **Total sales amount by product type**

fact\_sales.merge(dim\_product, on='product\_id') \

.groupby('TYPE OF PRODUCT')['Amount'].sum() \

.sort\_values(ascending=False)

Merges fact\_sales with dim\_product, groups by product type, and sums amounts.

1. **Payment type with highest revenue**

fact\_sales.merge(dim\_payment, on='payment\_id') \

.groupby('PAYMENT TYPE')['Amount'].sum() \

.sort\_values(ascending=False)

Merges fact\_sales with dim\_payment, groups by payment type, and sums amounts.

1. **Monthly sales trends across financial years**

fact\_date = fact\_sales.merge(dim\_date, on='date\_id')

fact\_date['month'] = fact\_date['Date'].dt.to\_period('M').astype(str)

fact\_date.groupby(['month', 'F.Y'])['Amount'].sum()

Merges fact\_sales with dim\_date, extracts the month, groups by month and financial year, and sums amounts.

1. **Accessory/mobile type with highest sales quantity**

fact\_sales.merge(dim\_product, on='product\_id') \

.groupby('TYPE OF ACCESSORY/MOBILE')['Quantity'].sum() \

.sort\_values(ascending=False)

Merges fact\_sales with dim\_product, groups by accessory/mobile type, and sums quantities.

1. **Top 5 product codes by revenue**

fact\_sales.merge(dim\_product, on='product\_id') \

.groupby('P\_NO')['Amount'].sum() \

.sort\_values(ascending=False).head(5)

Merges fact\_sales with dim\_product, groups by product number, sums amounts, and selects the top five.

**Summary**

This technical appendix explains:

* How the dataset was normalized into dimension and fact tables
* How we ensured the design fits Third Normal Form (3NF)
* What each SQL and pandas query does to deliver meaningful business insights

With this foundation, the stakeholder report and public-facing story are built on clean data, a robust schema, and reproducible analysis.