Collage of Computing

Department of Software Engineering

Fundamentals of Big Data Analytics and BI

Prepared by Henok Ketema

ID DBUR/1754/13

Sumitted on 13/02/25

Submitted to: Derbew Felasman(MSc)

Table of Contents

Data Cleaning and Preprocessing Documentation 3
Overview 3
Dependencies 3
Step-by-Step Breakdown 4
1. Load the Dataset 4
2. Display Initial Data Insights 4
3. Check Dataset Shape 4
4. Identifying Missing Values 4
5. Visualizing Missing Values 5
6. Dropping Unnecessary Columns 5
7. Handling Missing Values 5
Final Dataset Insights 6
PostgreSQL CSV Data Import Script 7
Overview 7
Technologies Used 7
Code Breakdown 8
1. Define PostgreSQL Connection Details 8
2. Read the CSV File 8
3. Ensure the 'working_date' Column Exists 8
4. Connect to PostgreSQL Database 8
5. Create Table if Not Exists 9
6. Convert Date Columns 10
7. Insert Data into PostgreSQL Efficiently 10
8. Error Handling and Connection Closure 11

Data Cleaning and Preprocessing Documentation

Overview

This script is designed to clean and preprocess the "Pakistan Largest Ecommerce Dataset" by performing the following steps:

- Loading the dataset
- Displaying basic information and statistics
- Identifying and handling missing values
- Dropping unnecessary columns
- Handling duplicate and missing data
- Converting date-related columns to datetime format
- Saving the cleaned dataset

Dependencies

The script requires the following Python libraries:

- pandas for data manipulation
- numpy for numerical operations
- matplotlib.pyplot for data visualization
- seaborn for statistical plotting
- scipy for scientific computing
- Install Required Libraries

Ensure all dependencies are installed using the following command:

pip install pandas numpy matplotlib seaborn scipy

Step-by-Step Breakdown

1. Load the Dataset

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import scipy as sp


df = pd.read_csv(r"C:\Users\Post Lab\Desktop\New folder\Pakistan Largest Ecommerce Dataset.csv")
```

This reads the dataset from a CSV file into a Pandas DataFrame.

2. Display Initial Data Insights

```
print(df.head())    # Display first 5 rows

print(df.tail(6))    # Display last 6 rows

print(df.info())      # Show column names, data types, and non-null values

print(df.describe())    # Summary statistics for numerical columns
```

This provides an overview of the dataset, helping us understand its structure and content**.**

3. Check Dataset Shape

```
print(f"In this dataset numbers of rows {df.shape[0]} and numbers of columns {df.shape[1]}")
```

This prints the number of rows and columns in the dataset before cleaning.

4. Identifying Missing Values

```
print(df.isnull().sum())   # Count of missing values in each column
```

This helps identify columns with missing data**.**

5. Visualizing Missing Values

```
missing_percentage = df.isnull().sum() / len(df) * 100

missing_percentage.plot(kind='bar')

plt.xlabel('columns')

plt.ylabel('percentage')

plt.title("Percentage of Missing Values in Each Column")

plt.show()
```

A bar plot is generated to visualize missing data percentages across columns.

6. Dropping Unnecessary Columns

```
columns_to_drop = ["Unnamed: 21", "Unnamed: 22", "Unnamed: 23", "Unnamed: 24",
"Unnamed: 25", "sales_commission_code"]

df.drop(columns=columns_to_drop, inplace=True)

print(f"After removing columns, dataset has {df.shape[0]} rows and {df.shape[1]} columns")
```

This removes irrelevant columns to streamline the dataset.

7. Handling Missing Values

Dropping Rows with Missing Values

```
drop_Nan_rows = df.dropna()

print(drop_Nan_rows.shape)   # Shape after dropping rows
```

This removes rows containing missing values.

Handling Duplicates

```
print(drop_Nan_rows.duplicated().sum())   # Count of duplicate rows

print(drop_Nan_rows.info())   # Dataset info after dropping NaN rows
```

This ensures there are no duplicate rows.

Filling Missing Values Instead of Dropping

df.fillna(method='ffill', inplace=True) # Forward fill missing values

print(df.shape)

print(df.isnull().sum())

Instead of dropping rows, missing values are filled with the previous row's values using forward fill (ffill).

8. Converting Date Columns to Datetime Format

df['Working Date'] = pd.to_datetime(df['Working Date'])

df['Customer Since'] = pd.to_datetime(df['Customer Since'])

df['created_at'] = pd.to_datetime(df['created_at'])

This ensures that date-related columns are in the proper datetime format.

9. Saving the Cleaned Dataset

cleaned_df = df.to_csv(r"C:\Users\Post Lab\Desktop\New folder\E-commerce

dataset1.csv")

This exports the cleaned dataset to a new CSV file.

Summary of Changes

- Removed unnecessary columns (Unnamed: 21-25, sales_commission_code)
- Identified and handled missing values
- Filled missing data using forward fill instead of dropping rows
- Removed duplicate rows
- Converted date columns to proper datetime format
- Saved the cleaned dataset

Final Dataset Insights

- The dataset now has 1M+ rows with no missing values.
- It is structured and ready for further analysis or modeling.

PostgreSQL CSV Data Import Script

Overview

This script reads an E-commerce dataset from a CSV file and imports it into a PostgreSQL database. It ensures that necessary data transformations and database operations are handled efficiently.

Technologies Used

Python for scripting

Pandas for CSV handling and data transformation

Psycopg2 for connecting to PostgreSQL and executing SQL queries

Key Functionalities

Read CSV Data: The script reads the CSV file using `pandas.read_csv()`, ensuring proper data loading.

Database Connection: It establishes a connection to a PostgreSQL database using `psycopg2`.

Table Creation: If the table `eco5` does not exist, it creates it with the necessary schema.

Data Transformation: It ensures date columns are properly formatted.

Efficient Data Insertion: It converts the DataFrame into a list of tuples and inserts them into the database efficiently using `execute_values()`.

Error Handling: Implements exception handling to catch and display errors if they occur during execution.

Connection Closure: Ensures proper closure of the database connection to avoid resource leaks.

Code Breakdown

1. Define PostgreSQL Connection Details

```
# PostgreSQL Connection Details

db_params = {

     "dbname": "Ecommerce",

    "user": "postgres",

    "password": "henok15",

    "host": "localhost",

    "port": "5432"

}
```

This dictionary stores PostgreSQL credentials required for authentication.

2. Read the CSV File

```
df = pd.read_csv(csv_file, dtype=str, low_memory=False)

df.columns = df.columns.str.strip()
```

Reads the CSV file into a pandas DataFrame.

Ensures column names do not contain leading/trailing spaces.

Uses `dtype=str` to avoid type inconsistencies.

3. Ensure the 'working_date' Column Exists

```
if "working_date" not in df.columns:

    print("Missing column: 'working_date' in CSV. Creating an empty
column.")

    df["working_date"] = None
```

Checks if `working_date` exists; if not, adds it as an empty column to prevent errors.

4. Connect to PostgreSQL Database

```
conn = psycopg2.connect(**db_params)

cursor = conn.cursor()
```

Establishes a connection to PostgreSQL and initializes a cursor to execute SQL queries.

5. Create Table if Not Exists

```
create_table_query = """
CREATE TABLE IF NOT EXISTS eco5 (
    id SERIAL PRIMARY KEY,
    item_id FLOAT,
    status TEXT,
    created_at TIMESTAMP,
    sku TEXT,
    price FLOAT,
    qty_ordered FLOAT,
    grand_total FLOAT,
    increment_id TEXT,
    category_name_1 TEXT,
    sales_commission_code TEXT,
    discount_amount FLOAT,
    payment_method TEXT,
    working_date TIMESTAMP NULL,
    bi_status TEXT,
    mv TEXT,
    year INT,
    month INT,
    customer_since TIMESTAMP,
    my TEXT,
    fy TEXT,
    customer_id FLOAT
);
"""
cursor.execute(create_table_query)
conn.commit()
```

Ensures the `eco5` table exists with the appropriate schema.

Uses `SERIAL` for auto-incrementing `id`.

Commits the changes to the database.

6. Convert Date Columns

```
date_columns = ["created_at", "working_date", "customer_since"]

for col in date_columns:

    if col in df.columns:

        df[col] = pd.to_datetime(df[col], errors='coerce')
```

Converts date columns into proper timestamp format.

Uses `errors='coerce'` to handle invalid date formats.

7. Insert Data into PostgreSQL Efficiently

```
insert_columns = [

    "item_id", "status", "created_at", "sku", "price", "qty_ordered",
"grand_total",

    "increment_id", "category_name_1", "sales_commission_code",
"discount_amount",

    "payment_method", "working_date", "bi_status", "mv", "year",
"month",

    "customer_since", "my", "fy", "customer_id"

]


data_tuples = [tuple(x) for x in df[insert_columns].to_numpy()]
```

Defines the exact columns for insertion (excluding `id`).

Converts DataFrame into a list of tuples for batch insertion.

```
insert_query = f"""

INSERT INTO eco5 ({', '.join(insert_columns)})

VALUES %s

"""

execute_values(cursor, insert_query, data_tuples)

conn.commit()
```

Uses `execute_values()` to efficiently insert all rows in one query, optimizing performance.

8. Error Handling and Connection Closure

```
except Exception as e:
    print(f"Error importing CSV: {e}")


finally:
    if cursor:
        cursor.close()
    if conn:
        conn.close()
    print("Connection closed.")
```

Handles any errors that occur during execution.

Closes the cursor and database connection properly.

Key Benefits of This Approach

Ensures Data Integrity: Proper handling of missing columns and data types.

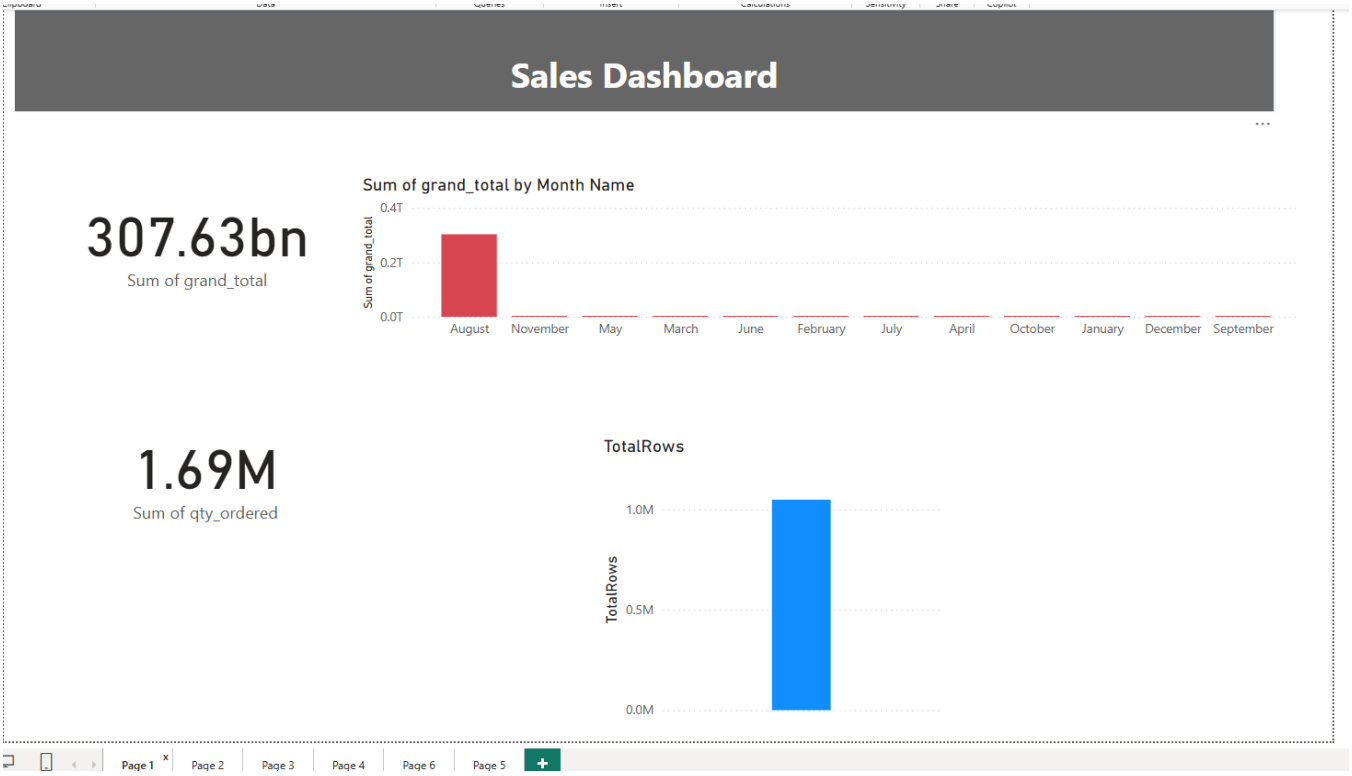
Automates Table Creation: Prevents issues if the table doesn't exist.

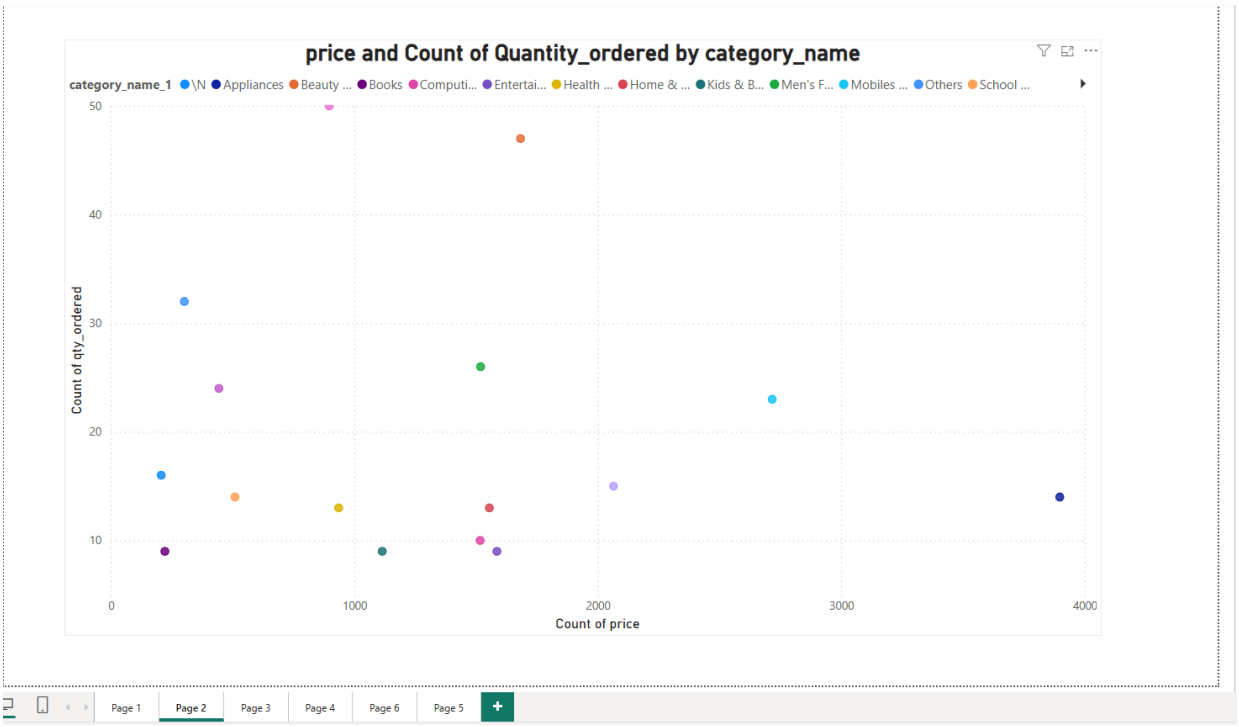
Optimized Performance: Batch insertion using `execute_values()` improves speed.

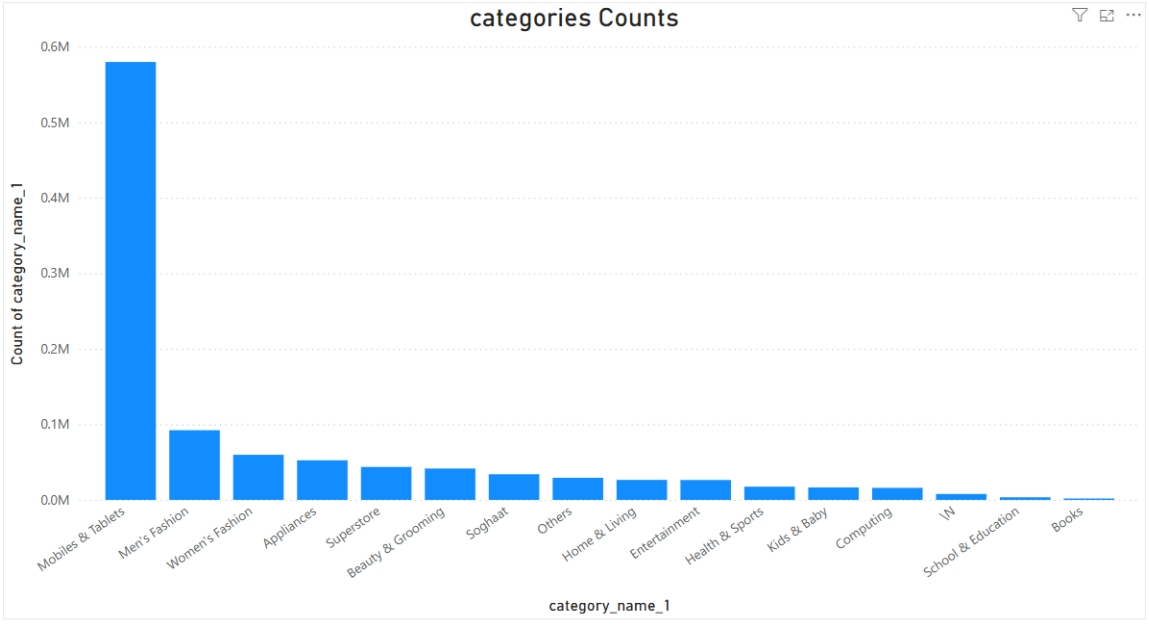
Handles Date Parsing Gracefully: Prevents errors from incorrect date formats.

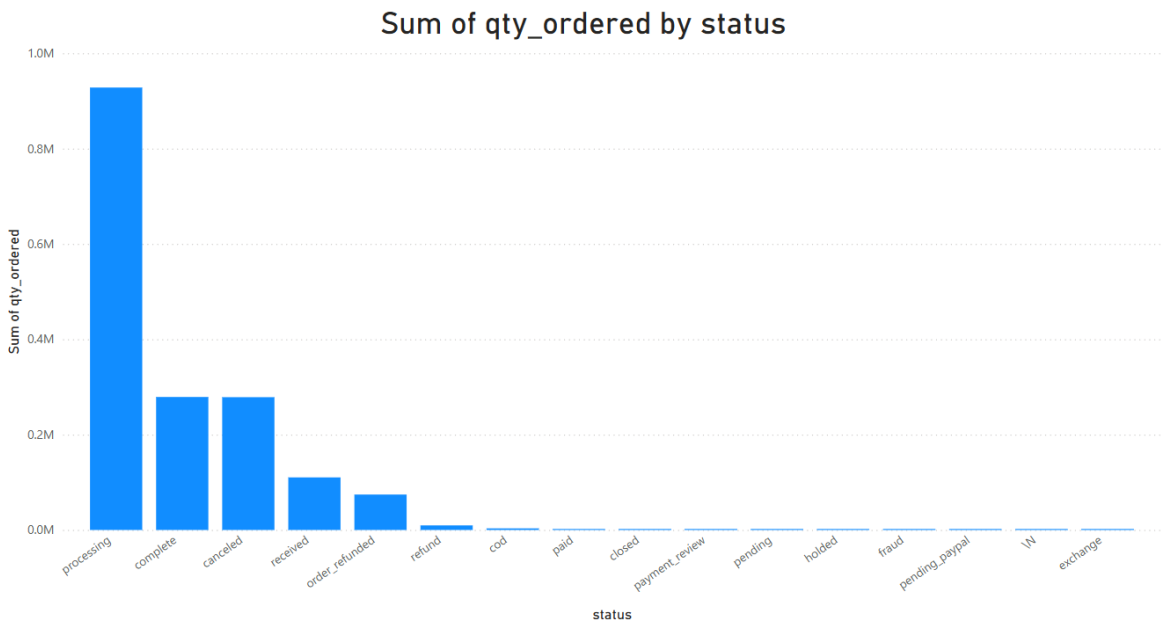
Error Handling: Catches and reports issues during execution.

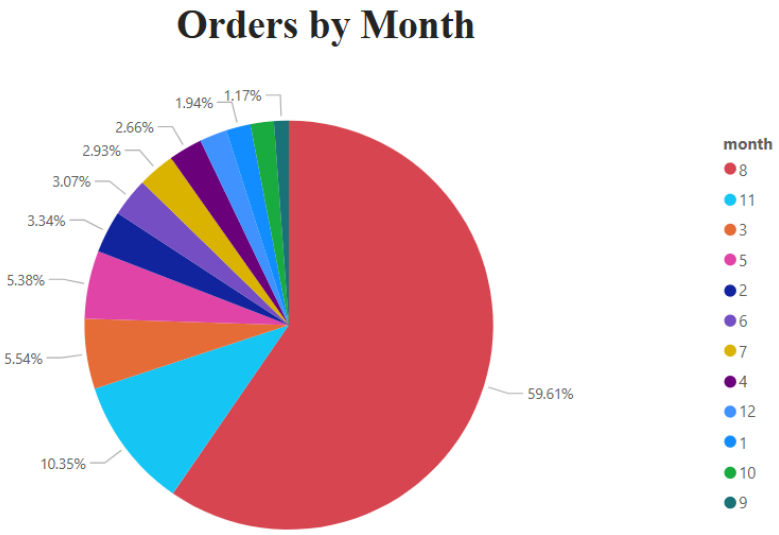
Power BI Visualizations

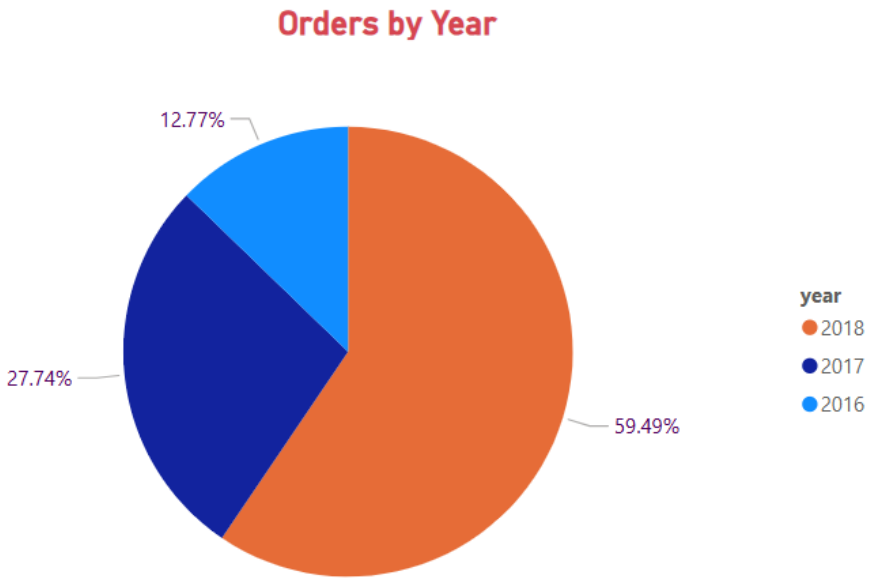



price and Count of Quantity_ordered by category_name
category_name_1 \N Appliances Beauty ... Books Computi... Entertai... Health ... Home & ... Kids & B... Men's F... Mobiles ... Others School ...
Count of qty_ordered
50
40
30
20
10
0
1000
2000
3000
4000
Count of price
Page 1
Page 2
Page 3
Page 4
Page 6
Page 5


categories Counts
Count of category_name_1
0.6M
0.5M
0.4M
0.3M
0.2M
0.1M
0.0M
Mobiles & Tablets
Men's Fashion
Women's Fashion
Appliances
Superstore
Beauty & Grooming
Soghaat
Others
Home & Living
Entertainment
Health & Sports
Kids & Baby
Computing
\N
School & Education
Books
category_name_1


Sum of qty_ordered by status
1.0M
0.8M
0.6M
0.4M
0.2M
0.0M
Sum of qty_ordered
processing
complete
canceled
received
order_refunded
refund
cod
paid
closed
payment_review
pending
holded
fraud
pending_paypal
\N
exchange
status


Orders by Month
1.94%
1.17%
2.66%
2.93%
3.07%
3.34%
5.38%
5.54%
59.61%
10.35%
month
8
11
3
5
2
6
7
4
12
1
10
9



Clarification for the Pie Charts in Power BI

Orders by Month Pie Chart

Overview

This pie chart represents the distribution of orders across different months. The data has been grouped by month, and the proportion of total orders for each month is visualized using a pie chart.

Key Insights

The highest number of sales occurred in November, making up 34.8% of total orders.

December and October also had significant order volumes at 6.5% and 6.8%, respectively.

The lowest number of orders was recorded in January at 3.5%.

This trend follows typical holiday shopping patterns, where demand peaks in the last quarter of the year.

Interpretation

The seasonal variations in order volume suggest that consumer behavior is influenced by external factors such as holidays, sales events, and industry-specific demand fluctuations.

Orders by Year Pie Chart

Overview

This pie chart illustrates the variation in total orders over different years. The data has been grouped by year, and the proportion of orders for each year is displayed.

Key Insights

2017 had the highest order volume, making up 65% of total orders.

2016 accounted for 29.9% of total orders.

2018 had the lowest order volume, representing 5.0% of the total.

Interpretation

Order volumes varied significantly year-over-year, with a peak in 2017.

The high volume in 2017 suggests strong industry demand or a favorable economic climate.

The decline in 2018 could indicate a shift in market trends, economic downturns, or changing consumer behavior.

These trends highlight how external market conditions and internal business strategies can impact order volumes over time.