Tugas Mata Kuliah Teknik Pengembangan Model

Retail Demand Forecasting: Pendekatan Pipeline Machine Learning

Nama Kelompok:

5231811017 Maulana Ahmad

5231811022 Lathif Ramadhan

5231811029 Andini Angel M

PROGRAM STUDI SAINS DATA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS TEKNOLOGI YOGYAKARTA

2025

DAFTAR ISI

DAFTAR ISI..... 2
BAB I: Pendahuluan.....3
1.1 Latar Belakang Masalah..... 3
1.2 Tujuan Proyek.....4
1.3 Ruang Lingkup Proyek..... 5
BAB II: Pemahaman Data Awal (Initial Data Understanding).....6
2.1. Tentang Dataset.....6
2.2. Import Libraries..... 11
2.3. Load Datasets.....12
BAB III: Data Ingestion and Merging (Penyerapan dan Penggabungan Data)..... 23
3.1. Prepare Datasets for Merging (Persiapan Dataset untuk Penggabungan).. 23
3.2. Merge Datasets (Penggabungan Dataset)..... 24
BAB IV: The 10-Step Data Preparation Pipeline (Pipeline Persiapan Data 10 Langkah)... 27
4.1 Data Cleaning and Type Conversion (Pembersihan Data dan Konversi Tipe). 27
4.2 Column Consolidation and Selection (Konsolidasi dan Pemilihan Kolom). 34
4.3 Exploratory Data Analysis (EDA).....36
4.4 Outlier Detection and Treatment (Deteksi dan Perlakuan Outlier)..... 49
4.5 Feature Engineering I - Time-Based Features (Rekayasa Fitur I - Fitur Berbasis Waktu)..... 52
4.6 Feature Engineering II - Lag Features (Rekayasa Fitur II - Fitur Lag.....55
4.7 Feature Engineering III - Rolling Window Features (Rekayasa Fitur III - Fitur Jendela Bergulir)..... 57
4.8 Categorical Feature Encoding (Pengkodean Fitur Kategorikal)..... 60
4.9 Feature Scaling (Penskalaan Fitur)..... 62
4.10 Time-Based Data Splitting (Pembagian Data Berbasis Waktu).....65
BAB V: Ringkasan Data Pasca-Persiapan (Post-Preparation Data Summary).....68
5.1. Final Data Snapshot (Gambaran Data Akhir).....68
5.2. Final Data Structure and Types (Struktur dan Tipe Data Akhir).....69
5.3. Final Summary Statistics (Ringkasan Statistik Akhir).....70
5.4. Final Missing Values and Duplicates Check (Pemeriksaan Akhir Nilai Hilang dan Duplikat).....71
5.5. Saving the Prepared Dataset (Menyimpan Dataset yang Telah Disiapkan). 72
5.6. Potential Additional Features and Improvements (Potensi Fitur Tambahan dan Peningkatan)..... 72
BAB VI: KESIMPULAN..... 74

BAB I: Pendahuluan

1.1 Latar Belakang Masalah

Industri ritel merupakan salah satu sektor ekonomi yang sangat bergantung pada kemampuan memahami perilaku konsumen dan mengelola persediaan barang dengan efektif. Dalam konteks persaingan pasar yang semakin ketat, kemampuan perusahaan untuk memperkirakan permintaan pelanggan secara akurat menjadi faktor penentu keberhasilan dalam menjaga keseimbangan antara ketersediaan produk dan efisiensi biaya operasional.

Kesalahan dalam memperkirakan permintaan dapat menimbulkan dua permasalahan utama yang signifikan, yaitu kelebihan stok (*overstock*) dan kekurangan stok (*stockout*). Kelebihan stok menyebabkan meningkatnya biaya penyimpanan, risiko kerusakan atau kedaluwarsa produk, serta penurunan profitabilitas. Sebaliknya, kekurangan stok menyebabkan hilangnya peluang penjualan, penurunan loyalitas pelanggan, dan dampak negatif terhadap reputasi merek. Kedua kondisi tersebut menegaskan pentingnya sistem peramalan permintaan (*demand forecasting*) yang akurat dan adaptif.

Dalam era digital saat ini, perusahaan ritel menghasilkan dan menyimpan data dalam jumlah besar yang berasal dari berbagai sumber, seperti transaksi penjualan harian, informasi produk, data promosi, serta faktor eksternal seperti cuaca dan hari libur. Data ini sangat berharga apabila dapat dimanfaatkan secara optimal untuk mendukung proses pengambilan keputusan strategis. Pendekatan berbasis Machine Learning (ML) menawarkan solusi modern untuk mengelola kompleksitas tersebut, karena algoritma ML mampu mengenali pola-pola non-linear dan hubungan kompleks antara variabel-variabel yang memengaruhi permintaan.

Namun, salah satu tantangan terbesar dalam penerapan Machine Learning adalah kualitas data yang digunakan. Data mentah sering kali bersifat tidak lengkap, inkonsisten, atau tersebar di berbagai sumber. Oleh karena itu, dibutuhkan pendekatan sistematis berupa pipeline Machine Learning, yang mencakup proses penggabungan, pembersihan, dan rekayasa fitur data agar model yang dibangun nantinya memiliki landasan data yang kuat dan akurat.

Dalam konteks proyek ini, digunakan dataset “Retail Store Inventory and Demand Forecasting (Data Transaksional Inti)”, yang merupakan kumpulan data transaksi penjualan ritel yang mencatat informasi penting seperti tanggal transaksi, ID toko, ID produk, jumlah unit terjual, harga per unit, serta pendapatan total. Dataset ini juga dapat mencakup informasi tambahan seperti kategori produk, lokasi toko, dan faktor eksternal lainnya. Dengan menganalisis dataset tersebut secara menyeluruh dan menyiapkannya melalui pipeline Machine Learning, proyek ini diharapkan dapat memberikan dasar yang kuat untuk membangun model peramalan permintaan produk yang lebih akurat, efisien, dan aplikatif di dunia nyata.

1.2 Tujuan Proyek

Tujuan utama dari proyek Retail Demand Forecasting: Pendekatan Pipeline Machine Learning ini adalah untuk melakukan eksplorasi data mendalam dan persiapan data komprehensif yang akan digunakan sebagai fondasi dalam membangun model prediksi permintaan produk ritel harian. Proyek ini tidak hanya berfokus pada analisis statistik sederhana, tetapi juga pada penerapan praktik terbaik dalam manajemen dan rekayasa data agar dataset yang dihasilkan berkualitas tinggi dan siap untuk tahap pemodelan prediktif.

Secara lebih rinci, tujuan proyek ini meliputi:

1. Mengintegrasikan berbagai sumber data dari dataset *Retail Store Inventory and Demand Forecasting (Data Transaksional Inti)*, termasuk data penjualan, produk, dan toko.
2. Melakukan pembersihan data untuk mengatasi nilai hilang, duplikasi, inkonsistensi, dan anomali.
3. Melakukan rekayasa fitur (feature engineering) untuk menciptakan variabel baru yang relevan terhadap pola permintaan, seperti tren waktu, musiman, kategori produk, dan dampak promosi.
4. Menghasilkan dataset akhir yang terstruktur, bersih, dan informatif, sehingga dapat digunakan langsung dalam proses pemodelan prediktif pada tahap selanjutnya.

Dengan tujuan-tujuan tersebut, proyek ini diharapkan dapat memberikan kontribusi nyata dalam mendukung pengambilan keputusan berbasis data di sektor ritel, khususnya dalam hal perencanaan stok, pengendalian rantai pasok, serta strategi penjualan yang lebih tepat sasaran.

1.3 Ruang Lingkup Proyek

Proyek ini secara khusus difokuskan pada tahapan awal pipeline Machine Learning, yaitu tahap penggabungan data, pembersihan data, dan rekayasa fitur. Ketiga tahapan ini dianggap sebagai fondasi penting sebelum dilakukan proses pemodelan dan evaluasi model prediksi.

Cakupan pekerjaan proyek ini meliputi:

- Eksplorasi awal dataset Retail Store Inventory and Demand Forecasting (Data Transaksional Inti) untuk memahami struktur, tipe variabel, serta distribusi data.
- Penggabungan (*merging*) antara tabel transaksi penjualan, data produk, dan data toko untuk membentuk satu dataset terpadu.
- Pembersihan data (*data cleaning*) dari nilai hilang, duplikasi, dan data yang tidak valid.
- Rekayasa fitur (*feature engineering*) seperti pembuatan variabel waktu (hari, bulan, musim), identifikasi hari libur, dan indikator promosi.

Penyusunan dataset akhir yang siap digunakan untuk proses *model training* dan *evaluation* di tahap lanjutan.

Tahap pemodelan, evaluasi performa model, dan implementasi sistem prediksi tidak termasuk dalam ruang lingkup laporan ini. Namun, seluruh hasil persiapan data yang dihasilkan dirancang agar memenuhi kebutuhan tersebut di masa mendatang.

BAB II: Pemahaman Data Awal (Initial Data Understanding)

2.1. Tentang Dataset

Fondasi dari proyek pemodelan prediktif ini adalah integrasi strategis dari tiga dataset yang berbeda namun saling melengkapi. Dengan menggabungkan sumber-sumber ini, kami bertujuan untuk menciptakan pandangan holistik tentang lingkungan ritel, menangkap segalanya mulai dari transaksi harian yang terperinci dan tingkat inventaris hingga upaya pemasaran yang lebih luas dan sinyal ekonomi makro. Pendekatan komprehensif ini sangat penting untuk membangun model peramalan permintaan yang kuat dan akurat.

Setiap dataset memainkan peran spesifik dan krusial dalam memperkaya dataset master akhir kami:

1. Retail Store Inventory and Demand Forecasting (Data Transaksional Inti)
Tautan: Kaggle Dataset (https://www.kaggle.com/datasets/atomicd/retail-store-inventory-and-demand-forecasting)
Dataset ini berfungsi sebagai **tabel dasar atau utama** untuk proyek kami. Ini menyediakan data transaksional dan operasional inti pada tingkat paling granular: aktivitas penjualan harian untuk setiap produk di setiap toko. Ini berisi sinyal utama perilaku pelanggan dan, yang paling penting, variabel target kami untuk prediksi.

Peran dalam Proyek:

Ini adalah sumber kebenaran utama kami untuk penjualan dan permintaan. Kami akan menggunakannya sebagai tabel utama tempat semua informasi lain akan digabungkan. Catatan hariannya yang terperinci sangat cocok untuk rekayasa fitur deret waktu (misalnya, membuat lag dan rata-rata bergulir).

Fitur Utama yang Dimanfaatkan:

- `Date`, `Store ID`, `Product ID`:
  Kunci utama yang digunakan untuk menggabungkan dan melacak kinerja di berbagai segmen.

- **`Demand` (Variabel Target):**
  Perkiraan permintaan harian untuk setiap produk, yang ingin diprediksi oleh model kami.
- `Units Sold`, `Inventory Level`:
- Indikator internal kritis kinerja stok dan kecepatan penjualan.
- `Price`, `Discount`, `Promotion`:
  Tuas keuangan utama yang secara langsung memengaruhi keputusan pembelian pelanggan.
- `Weather Condition`, `Seasonality`:
  Faktor eksternal yang membantu model memahami pola penjualan kontekstual.
- `Competitor Pricing`:
  Memberikan gambaran tentang lanskap kompetitif.
- `Epidemic`: Bendera biner unik untuk membantu model belajar dari periode gangguan pasar yang signifikan, seperti pandemi.

Alasan Pemilihan Strategis:

- **Kelengkapan Data Granular:**
  Menyediakan data pada level paling detail yang kami butuhkan: penjualan harian untuk setiap produk di setiap toko. Granularitas ini sangat penting untuk analisis deret waktu (*time-series*) dan rekayasa fitur seperti *lag* dan *rolling window*.
- **Mengandung Variabel Target Fundamental:**
  Secara eksplisit menyediakan kolom **`Demand`** dan **`Units Sold`**, yang merupakan variabel target yang akan kami prediksi. Ini adalah alasan utama dataset ini menjadi fondasi proyek.
- **Kaya akan Fitur Internal yang Dapat Ditindaklanjuti:**
  Berisi fitur-fitur operasional inti yang secara langsung berada dalam kendali bisnis, seperti `Inventory Level`, `Price`, `Discount`, dan `Promotion`. Fitur-fitur ini adalah tuas utama yang digunakan manajemen ritel untuk mempengaruhi penjualan secara langsung.
- **Konteks Tambahan yang Unik:**
  Adanya fitur seperti `Weather Condition` dan terutama **`Epidemic`** memberikan nilai tambah yang luar biasa. Fitur `Epidemic` memungkinkan model untuk belajar dari anomali atau periode disrupsi besar, membuatnya lebih tangguh (*robust*) terhadap kejadian tak terduga di masa depan.

2. Retail Sales Data with Seasonal Trends & Marketing (Data Konteks Pemasaran)
Tautan: Kaggle Dataset (https://www.kaggle.com/datasets/abdullah0a/retail-sales-data-with-seasonal-trends-and-marketing)

Sementara dataset pertama kami menyediakan detail operasional, dataset kedua ini menawarkan pandangan tingkat yang lebih tinggi tentang lingkungan pemasaran dan penjualan. Kontribusi utamanya adalah penyertaan pengeluaran pemasaran, memungkinkan model kami untuk mengukur dampak upaya periklanan terhadap penjualan.

Peran dalam Proyek:

Untuk memperkaya dataset master kami dengan intelijen pemasaran yang krusial. Dengan menggabungkan data ini, kami dapat menganalisis laba atas investasi untuk kampanye pemasaran dan memahami bagaimana pengeluaran pemasaran, bersama dengan faktor-faktor lain seperti hari libur dan diskon, mendorong permintaan.

Fitur Utama yang Dimanfaatkan:

- **`Marketing Spend (USD)`:**
  Fitur paling berharga dari dataset ini, memberikan ukuran langsung dari investasi pemasaran.
- `Sales Revenue (USD)`:
  Dapat digunakan untuk validasi atau sebagai fitur tambahan untuk menangkap nilai moneter penjualan.
- `Holiday Effect`:
  Indikator biner yang berguna untuk membantu model memahami peningkatan penjualan selama periode liburan.
- `Store Location`, `Product Category`:
  Memberikan dimensi tambahan untuk segmentasi dan analisis. **Alasan Pemilihan Strategis:**
- **Mengukur Dampak Pemasaran secara Kuantitatif:**
  Fitur utamanya adalah **`Marketing Spend (USD)`**. Ini adalah metrik yang sangat berharga yang tidak ada di dataset lain. Ini mengubah informasi `Promotion` (yang hanya berupa 'ya' atau 'tidak' di Dataset 1) menjadi metrik kuantitatif. Dengan ini, model dapat belajar hubungan antara **jumlah uang yang dihabiskan** untuk pemasaran dengan peningkatan penjualan, sehingga memungkinkan analisis ROI (*Return on Investment*).

- **Memperkuat Analisis Musiman:**
  Adanya kolom `Holiday Effect` melengkapi kolom `Seasonality` dari Dataset 1. Ini memberikan sinyal yang lebih spesifik tentang lonjakan penjualan selama periode liburan tertentu, bukan hanya musim secara umum.
- **Menyediakan Dimensi Keuangan Tambahan:**
  Kolom `Sales Revenue` memungkinkan analisis tidak hanya dari segi jumlah unit, tetapi juga dari nilai moneter, yang dapat digunakan sebagai fitur tambahan untuk memberikan bobot lebih pada penjualan bernilai tinggi atau untuk validasi model.

3. Strategic Supply Chain Demand Forecasting Dataset (Data Ekonomi & Kompetitif Eksternal)
Tautan: Kaggle Dataset (https://www.kaggle.com/datasets/ziya07/strategic-supply-chain-demand-forecasting-dataset)

Dataset terakhir ini menyediakan lapisan konteks ekonomi makro dan kompetitif. Ini mensimulasikan data dari perusahaan furnitur dan menyertakan indeks berharga yang mewakili faktor-faktor di luar kendali langsung toko ritel tunggal, seperti kesehatan ekonomi secara keseluruhan.

Peran dalam Proyek:

Untuk memberikan model kami sinyal eksternal yang memengaruhi daya beli dan perilaku konsumen secara keseluruhan. Fitur-fitur ini membantu model memahami kondisi pasar yang lebih luas, membuatnya lebih tangguh dan mudah beradaptasi terhadap pergeseran ekonomi. Dataset ini sangat berharga karena berisi fitur-fitur yang telah diproses sebelumnya dan siap model.

Fitur Utama yang Dimanfaatkan:

- **`economic_index`:**
  Fitur kuat yang mewakili kekuatan ekonomi. Indeks yang lebih tinggi menunjukkan kondisi ekonomi yang lebih kuat, yang biasanya berkorelasi dengan penjualan ritel yang lebih tinggi.
- **`competitor_price_index`:**
  Indeks yang mewakili harga relatif pesaing, memberikan pandangan yang lebih abstrak tentang lingkungan kompetitif daripada harga pesaing tunggal.

- Bendera Boolean yang Telah Diproses Sebelumnya:
  Fitur seperti `region_Europe`, `store_type_Retail`, dan `category_Chairs` menyediakan atribut yang berguna dan sudah dikodekan.

Alasan Pemilihan Strategis:

- **Memberikan Sinyal Ekonomi Makro:**
  Fitur **`economic_index`** adalah alasan terkuat untuk memilih dataset ini. Penjualan ritel sangat berkorelasi dengan kesehatan ekonomi. Dengan fitur ini, model dapat beradaptasi dengan kondisi ekonomi yang sedang baik (booming) atau buruk (resesi), menjadikannya lebih cerdas dan proaktif, bukan hanya reaktif terhadap data masa lalu.
- **Memberikan Gambaran Kompetisi yang Lebih Luas:**
  `competitor_price_index` memberikan pandangan yang lebih terabstraksi tentang lanskap persaingan. Sebuah indeks yang mewakili rata-rata pasar lebih stabil dan seringkali lebih informatif daripada hanya membandingkan dengan harga satu pesaing (`Competitor Pricing` di Dataset 1).
- **Efisiensi dengan Fitur yang Sudah Terproses:**
  Dataset ini menyediakan fitur yang sudah di-*engineer* dan di-*encode* (seperti `region_Europe`, `store_type_Retail`), yang memungkinkan kami menunjukkan kemampuan untuk bekerja dengan berbagai format data dan menghemat waktu dalam preprocessing.

Catatan Penting:

Dataset ini berisi kolom bernama `future_demand`, yang tampaknya merupakan variabel target untuk kasus penggunaan aslinya. Untuk mencegah **kebocoran data (data leakage)**, kolom ini akan dihapus dari `master_df` kami. Tujuan proyek kami adalah untuk memprediksi variabel `Demand` dari dataset pertama.

Dengan menggabungkan ketiga sumber ini, kami membangun DataFrame master tunggal yang komprehensif. Dataset terpadu ini memungkinkan model kami untuk belajar dari berbagai sinyal—mulai dari tingkat stok internal dan promosi hingga penetapan harga pesaing dan tren ekonomi makro—membuka jalan bagi solusi peramalan yang lebih akurat dan kuat.

2.2. Import Libraries

[1]
✓ 3s

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error
import xgboost as xgb

# Set plotting style
sns.set_style("whitegrid")
```

Kode di atas mengimpor library Python yang akan digunakan sepanjang proyek ini:

- **`pandas`**: Library utama untuk manipulasi dan analisis data, terutama untuk bekerja dengan DataFrame. Diimpor dengan alias `pd`.
- **`numpy`**: Library untuk komputasi numerik, sangat berguna untuk operasi array dan matematika. Diimpor dengan alias `np`.
- **`matplotlib.pyplot`**: Library untuk membuat visualisasi statis di Python. Diimpor dengan alias `plt`.
- **`seaborn`**: Library visualisasi data tingkat tinggi berdasarkan matplotlib, menyediakan antarmuka yang lebih menarik dan informatif. Diimpor dengan alias `sns`.
- **`sklearn.model_selection.train_test_split`**: Fungsi untuk membagi dataset menjadi subset pelatihan dan pengujian.
- **`sklearn.preprocessing.StandardScaler`**: Digunakan untuk menstandarisasi fitur dengan menghapus rata-rata dan menskalakan ke varian unit.
- **`sklearn.preprocessing.OneHotEncoder`**: Digunakan untuk mengonversi fitur kategorikal menjadi format one-hot numerik.
- **`sklearn.compose.ColumnTransformer`**: Memungkinkan penerapan transformer yang berbeda ke kolom data yang berbeda.
- **`sklearn.pipeline.Pipeline`**: Memungkinkan pembuatan pipeline kerja yang menggabungkan beberapa langkah preprocessing dan estimasi.
- **`sklearn.metrics.mean_squared_error`**, **`sklearn.metrics.mean_absolute_error`**: Metrik evaluasi untuk model regresi.

- **`xgboost`**: Library untuk implementasi Extreme Gradient Boosting (XGBoost), algoritma machine learning yang populer dan efisien. Diimpor dengan alias `xgb`.

Baris `sns.set_style("whitegrid")` mengatur gaya plot default untuk seaborn, memberikan latar belakang putih dengan grid.

2.3. Load Datasets

```
[2]
data1 = 'https://raw.githubusercontent.com/LatiefDataVisionary/demand-forecasting-retail-college-task/refs/heads/main/data/raw/Dataset%201_Retail%20Store%20Inventory%20and%20Demand%20Forecasting.csv'
data2 = 'https://raw.githubusercontent.com/LatiefDataVisionary/demand-forecasting-retail-college-task/refs/heads/main/data/raw/Dataset%202_Retail%20Sales%20Data%20with%20Seasonal%20Trends%20%26%20Marketing.csv'
data3 = 'https://raw.githubusercontent.com/LatiefDataVisionary/demand-forecasting-retail-college-task/refs/heads/main/data/raw/Dataset%203_Strategic%20Supply%20Chain%20Demand%20Forecasting%20Dataset.csv'

df1 = pd.read_csv(data1)
df2 = pd.read_csv(data2)
df3 = pd.read_csv(data3)

print("df1 loaded successfully.")
print("df2 loaded successfully.")
print("df3 loaded successfully.")

df1 loaded successfully.
df2 loaded successfully.
df3 loaded successfully.
```

Kode di bagian ini bertujuan untuk memuat tiga dataset yang akan digunakan dalam proyek ini dari sumber URL mentah (raw).

1. **Mendefinisikan URL Dataset**
   Tiga variabel string (`data1`, `data2`, `data3`) dibuat untuk menyimpan URL dari file CSV yang berada di repositori GitHub. Menggunakan URL mentah dari GitHub memungkinkan kode untuk langsung mengakses file data.
2. **Memuat Dataset ke DataFrame:**
   Fungsi `pd.read_csv()` dari library pandas digunakan untuk membaca data dari setiap URL.
   - `df1 = pd.read_csv(data1)`: Memuat data dari URL di `data1` ke dalam DataFrame bernama `df1`.
   - `df2 = pd.read_csv(data2)`: Memuat data dari URL di `data2` ke dalam DataFrame bernama `df2`.
   - `df3 = pd.read_csv(data3)`: Memuat data dari URL di `data3` ke dalam DataFrame bernama `df3`.
3. **Konfirmasi Pemuatan:**
   Mencetak pesan konfirmasi untuk setiap dataset setelah berhasil dimuat. Ini membantu memverifikasi bahwa proses pemuatan berjalan tanpa kesalahan.

Setelah kode ini dijalankan, tiga DataFrame (`df1`, `df2`, dan `df3`) akan tersedia di lingkungan kerja, berisi data dari masing-masing file CSV.

2.3.1. Data Information

```
df1.head()
```

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Price	Discount	Weather Condition	Promotion	Competitor Pricing	Seasonality	Epidemic	Demand
0	2022-01-01	S001	P0001	Electronics	North	195	102	252	72.72	5	Snowy	0	85.73	Winter	0	115
1	2022-01-01	S001	P0002	Clothing	North	117	117	249	80.16	15	Snowy	1	92.02	Winter	0	229
2	2022-01-01	S001	P0003	Clothing	North	247	114	612	62.94	10	Snowy	1	60.08	Winter	0	157
3	2022-01-01	S001	P0004	Electronics	North	139	45	102	87.63	10	Snowy	0	85.19	Winter	0	52
4	2022-01-01	S001	P0005	Groceries	North	152	65	271	54.41	0	Snowy	0	51.63	Winter	0	59

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76000 entries, 0 to 75999
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Date                76000 non-null  object
 1   Store ID            76000 non-null  object
 2   Product ID          76000 non-null  object
 3   Category            76000 non-null  object
 4   Region              76000 non-null  object
 5   Inventory Level     76000 non-null  int64
 6   Units Sold          76000 non-null  int64
 7   Units Ordered       76000 non-null  int64
 8   Price               76000 non-null  float64
 9   Discount            76000 non-null  int64
 10  Weather Condition   76000 non-null  object
 11  Promotion           76000 non-null  int64
 12  Competitor Pricing  76000 non-null  float64
 13  Seasonality         76000 non-null  object
 14  Epidemic            76000 non-null  int64
 15  Demand              76000 non-null  int64
dtypes: float64(2), int64(7), object(7)
memory usage: 9.3+ MB
```

```
df2.head()
```

	YEAR	MONTH	SUPPLIER	ITEM CODE	ITEM DESCRIPTION	ITEM TYPE	RETAIL SALES	RETAIL TRANSFERS	WAREHOUSE SALES
0	2020	1	REPUBLIC NATIONAL DISTRIBUTING CO	100009	BOOTLEG RED - 750ML	WINE	0.00	0.0	2.0
1	2020	1	PWSWN INC	100024	MOMENT DE PLAISIR - 750ML	WINE	0.00	1.0	4.0
2	2020	1	RELIABLE CHURCHILL LLLP	1001	S SMITH ORGANIC PEAR CIDER - 18.7OZ	BEER	0.00	0.0	1.0
3	2020	1	LANTERNA DISTRIBUTORS INC	100145	SCHLINK HAUS KABINETT - 750ML	WINE	0.00	0.0	1.0
4	2020	1	DIONYSOS IMPORTS INC	100293	SANTORINI GAVALA WHITE - 750ML	WINE	0.82	0.0	0.0

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   YEAR              30000 non-null  int64
 1   MONTH             30000 non-null  int64
 2   SUPPLIER          29967 non-null  object
 3   ITEM CODE         30000 non-null  object
 4   ITEM DESCRIPTION  30000 non-null  object
 5   ITEM TYPE         30000 non-null  object
 6   RETAIL SALES      29999 non-null  float64
 7   RETAIL TRANSFERS  30000 non-null  float64
 8   WAREHOUSE SALES   30000 non-null  float64
dtypes: float64(3), int64(2), object(4)
memory usage: 2.1+ MB
```

```
df3.head()
```

	date	product_id	sales_units	holiday_season	promotion_applied	competitor_price_index	economic_index	weather_impact	price	discount_percentage	s
0	2023-01-01	151	99	0	0	0.983893	1.314333	0	126.932922	0.000000	
1	2023-01-02	192	95	1	0	0.977615	1.439582	0	151.355405	0.000000	
2	2023-01-03	114	101	0	0	0.983913	1.094795	0	191.701693	0.000000	
3	2023-01-04	171	33	0	0	1.191956	0.907672	0	173.106487	0.000000	
4	2023-01-05	160	82	0	1	0.855711	1.479690	0	138.587491	10.995213	

```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4999 entries, 0 to 4998
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   date                    4999 non-null   object
 1   product_id              4999 non-null   int64
 2   sales_units             4999 non-null   int64
 3   holiday_season          4999 non-null   int64
 4   promotion_applied       4999 non-null   int64
 5   competitor_price_index  4999 non-null   float64
 6   economic_index          4999 non-null   float64
 7   weather_impact          4999 non-null   int64
 8   price                   4999 non-null   float64
 9   discount_percentage     4999 non-null   float64
 10  sales_revenue           4999 non-null   float64
 11  region_Europe           4999 non-null   bool
 12  region_North America    4999 non-null   bool
 13  store_type_Retail       4999 non-null   bool
 14  store_type_Wholesale    4999 non-null   bool
 15  category_Cabinets       4999 non-null   bool
 16  category_Chairs         4999 non-null   bool
 17  category_Sofas          4999 non-null   bool
 18  category_Tables         4999 non-null   bool
 19  future_demand           4999 non-null   float64
dtypes: bool(8), float64(6), int64(5), object(1)
memory usage: 507.8+ KB
```

Kode di bagian "1.2.1. Data Information" ini digunakan untuk mendapatkan gambaran awal tentang struktur dan isi dari setiap dataset yang telah dimuat (`df1`, `df2`, dan `df3`).

- `df1.head()`, `df2.head()`, `df3.head()`:

  Perintah `.head()` menampilkan lima baris pertama dari setiap DataFrame. Ini berguna untuk melihat sekilas data, termasuk nama kolom, tipe data, dan beberapa nilai awal.
- `df1.info()`, `df2.info()`, `df3.info()`:

  Perintah `.info()` mencetak ringkasan ringkas dari setiap DataFrame. Ini termasuk:
  - Jumlah entri (baris) dalam DataFrame.
  - Daftar semua kolom.

- Jumlah nilai non-null di setiap kolom, yang membantu mengidentifikasi kolom dengan nilai yang hilang.
- Tipe data (dtype) dari setiap kolom (misalnya, int64, float64, object).
- Penggunaan memori oleh DataFrame.

Dengan menjalankan kode ini, kita dapat dengan cepat memahami format data, mengidentifikasi potensi masalah seperti nilai yang hilang atau tipe data yang tidak sesuai, dan mempersiapkan langkah-langkah pemrosesan data selanjutnya.

2.3.2. Data Summary Statistics

```
df1.describe(include='all')
```

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Price	Discount	Weather Condition	Promotion	Competitor Pricing	Seasonality	Epidemic	Demand
count	76000	76000	76000	76000	76000	76000.000000	76000.000000	76000.000000	76000.000000	76000.000000	76000	76000.000000	76000.000000	76000	76000.000000	76000.000000
unique	760	5	20	5	4	NaN	NaN	NaN	NaN	NaN	4	NaN	NaN	4	NaN	NaN
top	2024-01-30	S001	P0001	Groceries	North	NaN	NaN	NaN	NaN	NaN	Cloudy	NaN	NaN	Winter	NaN	NaN
freq	100	15200	3800	30400	30400	NaN	NaN	NaN	NaN	NaN	24360	NaN	NaN	21000	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	301.062842	88.827316	89.090645	67.726028	9.087039	NaN	0.328947	69.454029	NaN	0.200000	104.317158
std	NaN	NaN	NaN	NaN	NaN	226.510161	43.994525	162.404627	39.377899	7.475781	NaN	0.469834	40.943818	NaN	0.400003	46.964801
min	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	0.000000	4.740000	0.000000	NaN	0.000000	4.290000	NaN	0.000000	4.000000
25%	NaN	NaN	NaN	NaN	NaN	136.000000	58.000000	0.000000	31.997500	5.000000	NaN	0.000000	32.620000	NaN	0.000000	71.000000
50%	NaN	NaN	NaN	NaN	NaN	227.000000	84.000000	0.000000	64.500000	10.000000	NaN	0.000000	65.700000	NaN	0.000000	100.000000
75%	NaN	NaN	NaN	NaN	NaN	408.000000	114.000000	121.000000	95.830000	10.000000	NaN	1.000000	97.932500	NaN	0.000000	133.000000
max	NaN	NaN	NaN	NaN	NaN	2267.000000	426.000000	1616.000000	228.030000	25.000000	NaN	1.000000	261.220000	NaN	1.000000	430.000000

```
df2.describe(include='all')
```

	YEAR	MONTH	SUPPLIER	ITEM CODE	ITEM DESCRIPTION	ITEM TYPE	RETAIL SALES	RETAIL TRANSFERS	WAREHOUSE SALES
count	30000.0	30000.000000	29967	30000	30000	30000	29999.000000	30000.000000	30000.000000
unique	NaN	NaN	290	15668	15732	8	NaN	NaN	NaN
top	NaN	NaN	THE COUNTRY VINTNER, LLC DBA WINEBOW	16225	TENTH WARD DIST CO CARAWAY RYE - 750ML	WINE	NaN	NaN	NaN
freq	NaN	NaN	2041	4	5	18680	NaN	NaN	NaN
mean	2020.0	3.911467	NaN	NaN	NaN	NaN	6.939796	6.594058	27.431031
std	0.0	2.836788	NaN	NaN	NaN	NaN	33.081054	27.879428	272.166085
min	2020.0	1.000000	NaN	NaN	NaN	NaN	-0.420000	-6.000000	-3999.000000
25%	2020.0	1.000000	NaN	NaN	NaN	NaN	0.000000	0.000000	0.000000
50%	2020.0	3.000000	NaN	NaN	NaN	NaN	0.160000	0.000000	1.000000
75%	2020.0	7.000000	NaN	NaN	NaN	NaN	2.920000	3.000000	6.000000
max	2020.0	9.000000	NaN	NaN	NaN	NaN	2739.000000	1507.000000	18317.000000

In [ ]:
```
df3.describe(include='all')
```

Out[ ]:

	date	product_id	sales_units	holiday_season	promotion_applied	competitor_price_index	economic_index	weather_impact
count	4999	4999.000000	4999.000000	4999.000000	4999.000000	4999.000000	4999.000000	4999.000000
unique	4999	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	2036-09-07	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	148.897780	103.287057	0.201240	0.291458	1.000817	1.006562	0.143429
std	NaN	28.966902	54.805135	0.400967	0.454480	0.116095	0.287037	0.350545
min	NaN	100.000000	10.000000	0.000000	0.000000	0.800081	0.500021	0.000000
25%	NaN	124.000000	56.000000	0.000000	0.000000	0.900912	0.764042	0.000000
50%	NaN	148.000000	101.000000	0.000000	0.000000	0.999048	1.010684	0.000000
75%	NaN	174.000000	151.000000	0.000000	1.000000	1.102838	1.253165	0.000000
max	NaN	199.000000	199.000000	1.000000	1.000000	1.199929	1.499974	1.000000

Kode di bagian "1.2.2. Data Summary Statistics" ini digunakan untuk menghasilkan ringkasan statistik deskriptif untuk setiap kolom dalam DataFrame.

- `df1.describe(include='all')`, `df2.describe(include='all')`, `df3.describe(include='all')`:

Perintah `.describe()` menghasilkan statistik deskriptif. Dengan `include='all'`, ini mencakup kolom bertipe objek (non-numerik) serta kolom numerik. Untuk kolom numerik, output mencakup hitungan (count), rata-rata (mean), standar deviasi (std), nilai minimum (min), kuartil (25%, 50%, 75%), dan nilai maksimum (max). Untuk kolom objek, ini mencakup hitungan, jumlah nilai unik (unique), nilai yang paling sering muncul (top), dan frekuensinya (freq).

Ringkasan statistik ini membantu kita memahami distribusi data, rentang nilai, dan mengidentifikasi potensi masalah seperti pencilan (outliers) atau data yang tidak wajar dalam kolom numerik, serta memahami keragaman dan distribusi nilai dalam kolom kategorikal.

2.3.3. Unique Values and Counts for Object Columns

```python
for col in df1.select_dtypes(include='object').columns:
    print(f"### Column: {col}\n")
    print(f"**Unique Values:**\n{df1[col].unique()}\n")
    print(f"**Value Counts:**\n{df1[col].value_counts().to_markdown
    (numalign='left', stralign='left')}\n---")
```

```
### Column: Store ID

**Unique Values:**
['S001' 'S002' 'S003' 'S004' 'S005']

**Value Counts:**
| Store ID   | count   |
|:-----------|:--------|
| S001       | 15200   |
| S002       | 15200   |
| S003       | 15200   |
| S004       | 15200   |
| S005       | 15200   |
---
```

```python
for col in df2.select_dtypes(include='object').columns:
    print(f"### Column: {col}\n")
    print(f"**Unique Values:**\n{df2[col].unique()}\n")
    print(f"**Value Counts:**\n{df2[col].value_counts().to_markdown
    (numalign='left', stralign='left')}\n---")
```

```
---
### Column: ITEM TYPE

**Unique Values:**
['WINE' 'BEER' 'LIQUOR' 'STR_SUPPLIES' 'KEGS' 'REF' 'DUNNAGE'
 'NON-ALCOHOL']

**Value Counts:**
| ITEM TYPE     | count   |
|:--------------|:--------|
| WINE          | 18680   |
| LIQUOR        | 5995    |
| BEER          | 4217    |
| KEGS          | 808     |
| NON-ALCOHOL   | 216     |
| STR_SUPPLIES  | 63      |
| DUNNAGE       | 13      |
| REF           | 8       |
---
```

```python
for col in df3.select_dtypes(include='object').columns:
    print(f"### Column: {col}\n")
    print(f"**Unique Values:**\n{df3[col].unique()}\n")
    print(f"**Value Counts:**\n{df3[col].value_counts().to_markdown
    (numalign='left', stralign='left')}\n---")
```

```
| 2027-10-08 | 1       |
| 2027-10-07 | 1       |
| 2027-10-06 | 1       |
| 2027-10-05 | 1       |
| 2027-10-04 | 1       |
| 2027-10-03 | 1       |
| 2027-10-02 | 1       |
| 2027-11-02 | 1       |
| 2027-11-01 | 1       |
| 2027-10-31 | 1       |
| 2027-10-30 | 1       |
| 2027-10-29 | 1       |
| 2027-10-28 | 1       |
```

Kode di bagian "1.2.3. Unique Values and Counts for Object Columns" ini bertujuan untuk memeriksa nilai unik dan distribusinya (jumlah kemunculan) untuk setiap kolom dalam DataFrame yang bertipe data objek (biasanya string atau kategori).

- `df.select_dtypes(include='object').columns`: Kode ini memilih semua kolom dalam DataFrame (`df1`, `df2`, `df3`) yang memiliki tipe data objek dan mengembalikan daftar nama kolom tersebut.
- `for col in ...`: Loop ini mengiterasi melalui setiap nama kolom objek yang telah dipilih.
- `print(f"### Column: {col}\n")`: Mencetak judul untuk setiap kolom yang sedang dianalisis.
- `print(f"**Unique Values:**\n{df[col].unique()}\n")`: Mencetak semua nilai unik yang ada dalam kolom saat ini. Ini membantu melihat keragaman nilai dalam kolom kategorikal.
- `print(f"**Value Counts:**\n{df[col].value_counts().to_markdown(numalign='left', stralign='left')}\n---")`: Mencetak hitungan (frekuensi) untuk setiap nilai unik dalam kolom saat ini. `.value_counts()` menghitung berapa kali setiap nilai muncul, dan `.to_markdown()` memformat output menjadi tabel markdown agar lebih mudah dibaca.

Analisis ini penting untuk memahami konten aktual dalam kolom kategorikal, mengidentifikasi potensi masalah seperti kesalahan pengetikan atau variasi yang tidak

terduga, dan mempersiapkan langkah-langkah pengkodean (encoding) fitur kategorikal di kemudian hari.

2.3.4. Missing Values Analysis

```python
print("Missing values in df:")
print(df1.isnull().sum().to_markdown(numalign="left", stralign="left"))

print("\nDuplicate rows in df1:")
print(df1.duplicated().sum())
```

```
Missing values in df:
|                    | 0   |
|:-------------------|:----|
| Date               | 0   |
| Store ID           | 0   |
| Product ID         | 0   |
| Category           | 0   |
| Region             | 0   |
| Inventory Level    | 0   |
| Units Sold         | 0   |
| Units Ordered      | 0   |
| Price              | 0   |
| Discount           | 0   |
| Weather Condition  | 0   |
| Promotion          | 0   |
| Competitor Pricing | 0   |
| Seasonality        | 0   |
| Epidemic           | 0   |
| Demand             | 0   |

Duplicate rows in df1:
0
```

```python
print("Missing values in df2:")
print(df2.isnull().sum().to_markdown(numalign="left", stralign="left"))

print("\nDuplicate rows in df2:")
print(df2.duplicated().sum())
```

```
Missing values in df2:
|                  | 0   |
|:-----------------|:----|
| YEAR             | 0   |
| MONTH            | 0   |
| SUPPLIER         | 33  |
| ITEM CODE        | 0   |
| ITEM DESCRIPTION | 0   |
| ITEM TYPE        | 0   |
| RETAIL SALES     | 1   |
| RETAIL TRANSFERS | 0   |
| WAREHOUSE SALES  | 0   |

Duplicate rows in df2:
0
```

```python
print("Missing values in df3:")
print(df3.isnull().sum().to_markdown(numalign="left", stralign="left"))

print("\nDuplicate rows in df3:")
print(df3.duplicated().sum())
```

```
Missing values in df3:
|                       | 0   |
|:----------------------|:----|
| date                  | 0   |
| product_id            | 0   |
| sales_units           | 0   |
| holiday_season        | 0   |
| promotion_applied     | 0   |
| competitor_price_index | 0   |
| economic_index        | 0   |
| weather_impact        | 0   |
| price                 | 0   |
| discount_percentage   | 0   |
| sales_revenue         | 0   |
| region_Europe         | 0   |
| region_North America  | 0   |
| store_type_Retail     | 0   |
| store_type_Wholesale  | 0   |
| category_Cabinets     | 0   |
| category_Chairs       | 0   |
| category_Sofas        | 0   |
| category_Tables       | 0   |
| future_demand         | 0   |

Duplicate rows in df3:
0
```

Kode di bagian "1.2.4. Missing Values Analysis" ini digunakan untuk memeriksa keberadaan nilai yang hilang (missing values) dan baris duplikat (duplicate rows) di setiap DataFrame (`df1`, `df2`, dan `df3`).

- `df.isnull().sum()`:
  Perintah ini menghitung jumlah nilai yang hilang di setiap kolom dalam DataFrame. `.isnull()` membuat DataFrame boolean dengan `True` untuk nilai yang hilang dan `False` sebaliknya, dan `.sum()` menjumlahkan `True` di setiap kolom. `.to_markdown()` memformat output menjadi tabel markdown.
- `df.duplicated().sum()`:
  Perintah ini menghitung jumlah baris yang sepenuhnya duplikat dalam DataFrame. `.duplicated()` mengembalikan Series boolean yang menunjukkan baris mana yang merupakan duplikat dari baris sebelumnya, dan `.sum()` menjumlahkan `True`.

Analisis nilai yang hilang dan duplikat sangat penting dalam tahap pembersihan data. Nilai yang hilang perlu ditangani (misalnya, dengan mengisi atau menghapus), dan baris duplikat dapat dihapus untuk menghindari bias dalam analisis dan pemodelan. Output dari kode ini menunjukkan jumlah nilai yang hilang per kolom dan jumlah total baris duplikat untuk setiap DataFrame.

BAB III: Data Ingestion and Merging (Penyerapan dan Penggabungan Data)

Langkah pertama dalam pipeline kami adalah mengintegrasikan tiga dataset yang berbeda menjadi satu DataFrame utama yang terpadu yang akan menjadi dasar untuk analisis dan pemodelan kami.

3.1. Prepare Datasets for Merging (Persiapan Dataset untuk Penggabungan)

```
# Rename columns in df2 and df3
df2 = df2.rename(columns={'ITEM CODE': 'Product ID'})
df3 = df3.rename(columns={'date': 'Date', 'product_id': 'Product ID'})

# Convert Product ID in df2 to string to match df1 and df3
df2['Product ID'] = df2['Product ID'].astype(str)
df1['Product ID'] = df1['Product ID'].astype(str)
df3['Product ID'] = df3['Product ID'].astype(str)


print("Column names standardized and Product ID data types converted.")
```

```
Column names standardized and Product ID data types converted.
```

Kode di bagian ini bertujuan untuk mempersiapkan tiga dataset (`df1`, `df2`, dan `df3`) sebelum digabungkan. Langkah-langkah yang dilakukan adalah:

1. **Menyeragamkan Nama Kolom:**
   Mengganti nama kolom di `df2` dan `df3` agar sesuai dengan nama kolom di `df1` yang akan digunakan sebagai kunci penggabungan (`merge`). Kolom 'ITEM CODE' di `df2` dan 'product_id' di `df3` diganti menjadi 'Product ID'. Kolom 'date' di `df3` diganti menjadi 'Date'.
2. **Mengubah Tipe Data Kolom 'Product ID':**
   Memastikan bahwa kolom 'Product ID' di ketiga dataset memiliki tipe data yang sama, yaitu string. Ini penting karena penggabungan berdasarkan kolom ini memerlukan tipe data yang konsisten. Kode ini secara eksplisit mengubah tipe data kolom 'Product ID' di `df1`, `df2`, dan `df3` menjadi string.

Dengan menyeragamkan nama kolom dan tipe data 'Product ID', dataset siap untuk digabungkan menjadi satu DataFrame utama.

3.2. Merge Datasets (Penggabungan Dataset)

Melakukan proses penggabungan dua langkah untuk membuat `master_df`.

```
# Merge df1 with df2
master_df = pd.merge(df1, df2[['Product ID', 'SUPPLIER', 'ITEM TYPE', 'RETAIL SALES']], on='Product ID', how='left')

# Merge the result with df3
master_df = pd.merge(master_df, df3[['Date', 'Product ID', 'holiday_season', 'promotion_applied',
                                     'competitor_price_index',
                                     'economic_index', 'weather_impact',
                                     'price', 'discount_percentage',
                                     'sales_revenue', 'region_Europe',
                                     'region_North America',
                                     'store_type_Retail',
                                     'store_type_Wholesale',
                                     'category_Cabinets', 'category_Chairs',
                                     'category_Sofas',
                                     'category_Tables', 'future_demand']],
                                     on=['Date', 'Product ID'], how='left')

print("Datasets merged successfully.")
display(master_df.head())
```

```
Datasets merged successfully.
```

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Price	Discount	...
0	2022-01-01	S001	P0001	Electronics	North	195	102	252	72.72	5	...
1	2022-01-01	S001	P0002	Clothing	North	117	117	249	80.16	15	...
2	2022-01-01	S001	P0003	Clothing	North	247	114	612	62.94	10	...
3	2022-01-01	S001	P0004	Electronics	North	139	45	102	87.63	10	...
4	2022-01-01	S001	P0005	Groceries	North	152	65	271	54.41	0	...

5 rows × 36 columns

```
display(master_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76000 entries, 0 to 75999
Data columns (total 36 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Date                    76000 non-null  object
 1   Store ID                76000 non-null  object
 2   Product ID              76000 non-null  object
 3   Category                76000 non-null  object
 4   Region                  76000 non-null  object
 5   Inventory Level         76000 non-null  int64
 6   Units Sold              76000 non-null  int64
 7   Units Ordered           76000 non-null  int64
 8   Price                   76000 non-null  float64
 9   Discount                76000 non-null  int64
 10  Weather Condition       76000 non-null  object
 11  Promotion               76000 non-null  int64
 12  Competitor Pricing      76000 non-null  float64
 13  Seasonality             76000 non-null  object
 14  Epidemic                76000 non-null  int64
 15  Demand                  76000 non-null  int64
 16  SUPPLIER                0 non-null      object
 17  ITEM TYPE               0 non-null      object
 18  RETAIL SALES            0 non-null      float64
 19  holiday_season          0 non-null      float64
 20  promotion_applied       0 non-null      float64
 21  competitor_price_index  0 non-null      float64
 22  economic_index          0 non-null      float64
 23  weather_impact          0 non-null      float64
 24  price                   0 non-null      float64
 25  discount_percentage     0 non-null      float64
 26  sales_revenue           0 non-null      float64
 27  region_Europe           0 non-null      object
 28  region_North America    0 non-null      object
 29  store_type_Retail       0 non-null      object
 30  store_type_Wholesale    0 non-null      object
 31  category_Cabinets       0 non-null      object
 32  category_Chairs         0 non-null      object
 33  category_Sofas          0 non-null      object
 34  category_Tables         0 non-null      object
 35  future_demand           0 non-null      float64
dtypes: float64(12), int64(7), object(17)
memory usage: 20.9+ MB
None
```

Kode di bagian ini melakukan penggabungan (merge) dari tiga DataFrame (`df1`, `df2`, dan `df3`) menjadi satu DataFrame utama yang disebut `master_df`. Proses ini dilakukan dalam dua langkah menggunakan fungsi `pd.merge()`:

1. **Penggabungan `df1` dengan `df2`:**
   - `pd.merge(df1, df2[['Product ID', 'SUPPLIER', 'ITEM TYPE', 'RETAIL SALES']], on='Product ID', how='left')`
   - Menggabungkan `df1` dengan kolom-kolom tertentu dari `df2` (`'Product ID'`, `'SUPPLIER'`, `'ITEM TYPE'`, `'RETAIL SALES'`).
   - Penggabungan dilakukan berdasarkan kolom `'Product ID'`.

- Metode `how='left'` memastikan bahwa semua baris dari `df1` (DataFrame kiri) dipertahankan, dan kolom yang sesuai dari `df2` ditambahkan. Jika tidak ada kecocokan 'Product ID' di `df2`, nilai `NaN` akan muncul untuk kolom dari `df2`.

2. **Penggabungan hasil dengan `df3`:**
    - `pd.merge(master_df, df3[['Date', 'Product ID', ...]], on=['Date', 'Product ID'], how='left')`
    - Menggabungkan hasil penggabungan sebelumnya (yang disimpan kembali di `master_df`) dengan kolom-kolom tertentu dari `df3`.
    - Penggabungan kali ini dilakukan berdasarkan kombinasi dua kolom: `'Date'` dan `'Product ID'`. Ini karena data di `df3` bersifat harian per produk.
    - Metode `how='left'` kembali digunakan untuk mempertahankan semua baris dari DataFrame yang dihasilkan dari langkah pertama dan menambahkan kolom yang sesuai dari `df3`. Nilai `NaN` akan muncul jika tidak ada kecocokan kombinasi 'Date' dan 'Product ID' di `df3`.

Hasil akhir dari kedua langkah penggabungan ini adalah `master_df` yang berisi data dari ketiga sumber, disatukan berdasarkan 'Product ID' dan 'Date' (untuk data `df3`). Pesan "Datasets merged successfully." dicetak untuk mengkonfirmasi selesainya proses. `display(master_df.head())` menampilkan beberapa baris pertama dari `master_df` untuk memverifikasi hasilnya.

Bab ini merinci proses persiapan dan pembersihan data komprehensif 10 langkah yang diperlukan untuk mengubah data mentah yang telah digabungkan menjadi format yang kaya fitur dan siap model.

BAB IV: The 10-Step Data Preparation Pipeline (Pipeline Persiapan Data 10 Langkah)

4.1 Data Cleaning and Type Conversion (Pembersihan Data dan Konversi Tipe)

Langkah pertama dalam pipeline persiapan data yang komprehensif ini berfokus pada pembersihan awal `master_df` yang telah digabungkan dan memastikan tipe data yang sesuai untuk analisis dan pemodelan lebih lanjut.

Tujuan Langkah Ini:

- Memeriksa struktur dan ringkasan data awal dari `master_df` setelah penggabungan.
- Mengidentifikasi dan menangani nilai-nilai yang hilang (`NaN`) yang mungkin muncul selama proses penggabungan atau sudah ada di dataset asli.
- Mengonversi kolom tanggal ke format `datetime` yang tepat untuk ekstraksi fitur berbasis waktu.
- Memeriksa keberadaan baris duplikat.

```python
# Inspect the Master DataFrame
print("Info of master_df:")
master_df.info()
```

```
Info of master_df:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76000 entries, 0 to 75999
Data columns (total 36 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Date                    76000 non-null  object
 1   Store ID                76000 non-null  object
 2   Product ID              76000 non-null  object
 3   Category                76000 non-null  object
 4   Region                  76000 non-null  object
 5   Inventory Level         76000 non-null  int64
 6   Units Sold              76000 non-null  int64
 7   Units Ordered           76000 non-null  int64
 8   Price                   76000 non-null  float64
 9   Discount                76000 non-null  int64
 10  Weather Condition       76000 non-null  object
 11  Promotion               76000 non-null  int64
 12  Competitor Pricing      76000 non-null  float64
 13  Seasonality             76000 non-null  object
 14  Epidemic                76000 non-null  int64
 15  Demand                  76000 non-null  int64
 16  SUPPLIER                0 non-null      object
 17  ITEM TYPE               0 non-null      object
 18  RETAIL SALES            0 non-null      float64
 19  holiday_season          0 non-null      float64
 20  promotion_applied       0 non-null      float64
 21  competitor_price_index  0 non-null      float64
 22  economic_index          0 non-null      float64
 23  weather_impact          0 non-null      float64
 24  price                   0 non-null      float64
 25  discount_percentage     0 non-null      float64
 26  sales_revenue           0 non-null      float64
 27  region_Europe           0 non-null      object
 28  region_North America    0 non-null      object
 29  store_type_Retail       0 non-null      object
 30  store_type_Wholesale    0 non-null      object
 31  category_Cabinets       0 non-null      object
 32  category_Chairs         0 non-null      object
 33  category_Sofas          0 non-null      object
 34  category_Tables         0 non-null      object
 35  future_demand           0 non-null      float64
dtypes: float64(12), int64(7), object(17)
memory usage: 20.9+ MB
```

```python
print("\nHead of master_df:")
display(master_df.head())
```

Head of master_df:

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Price	Discount	...
0	2022-01-01	S001	P0001	Electronics	North	195	102	252	72.72	5	...
1	2022-01-01	S001	P0002	Clothing	North	117	117	249	80.16	15	...
2	2022-01-01	S001	P0003	Clothing	North	247	114	612	62.94	10	...
3	2022-01-01	S001	P0004	Electronics	North	139	45	102	87.63	10	...
4	2022-01-01	S001	P0005	Groceries	North	152	65	271	54.41	0	...

5 rows × 36 columns

```python
# Handle Missing Values
print("\nMissing values before handling:")
print(master_df.isnull().sum().to_markdown(numalign="left", stralign="left"))
```

```
Missing values before handling:
|                        | 0     |
|:-----------------------|:------|
| Date                   | 0     |
| Store ID               | 0     |
| Product ID             | 0     |
| Category               | 0     |
| Region                 | 0     |
| Inventory Level        | 0     |
| Units Sold             | 0     |
| Units Ordered          | 0     |
| Price                  | 0     |
| Discount               | 0     |
| Weather Condition      | 0     |
| Promotion              | 0     |
| Competitor Pricing     | 0     |
| Seasonality            | 0     |
| Epidemic               | 0     |
| Demand                 | 0     |
| SUPPLIER               | 76000 |
| ITEM TYPE              | 76000 |
| RETAIL SALES           | 76000 |
| holiday_season         | 76000 |
| promotion_applied      | 76000 |
| competitor_price_index | 76000 |
| economic_index         | 76000 |
| weather_impact         | 76000 |
| price                  | 76000 |
| discount_percentage    | 76000 |
| sales_revenue          | 76000 |
| region_Europe          | 76000 |
| region_North America   | 76000 |
| store_type_Retail      | 76000 |
| store_type_Wholesale   | 76000 |
| category_Cabinets      | 76000 |
| category_Chairs        | 76000 |
| category_Sofas         | 76000 |
| category_Tables        | 76000 |
| future_demand          | 76000 |
```

```python
# Fill missing 'SUPPLIER' and 'ITEM TYPE' with "Unknown"
master_df['SUPPLIER'] = master_df['SUPPLIER'].fillna('Unknown')
master_df['ITEM TYPE'] = master_df['ITEM TYPE'].fillna('Unknown')


# Fill missing numerical columns with 0, as the merge resulted in all NaNs for these columns
# Removed 'future_demand' from this list
numerical_cols_from_merge = ['RETAIL SALES', 'holiday_season', 'promotion_applied',
                             'competitor_price_index', 'economic_index', 'weather_impact',
                             'price', 'discount_percentage', 'sales_revenue']

for col in numerical_cols_from_merge:
    if col in master_df.columns and master_df[col].isnull().all(): # Check if all values are NaN
        master_df[col] = master_df[col].fillna(0) # Fill with 0
        print(f"Filled all missing values in {col} with 0.")
    elif col in master_df.columns and master_df[col].isnull().any(): # Check if some values are NaN
         median_val = master_df[col].median()
         master_df[col] = master_df[col].fillna(median_val)
         print(f"Filled some missing values in {col} with median ({median_val}).")
```

```
Filled all missing values in RETAIL SALES with 0.
Filled all missing values in holiday_season with 0.
Filled all missing values in promotion_applied with 0.
Filled all missing values in competitor_price_index with 0.
Filled all missing values in economic_index with 0.
Filled all missing values in weather_impact with 0.
Filled all missing values in price with 0.
Filled all missing values in discount_percentage with 0.
Filled all missing values in sales_revenue with 0.
```

```python
# Fill missing boolean columns with False (assuming NaN in boolean columns implies the condition is false)
# Removed 'future_demand' from this list
boolean_cols_with_missing = ['region_Europe', 'region_North America', 'store_type_Retail', 'store_type_Wholesale',
                             'category_Cabinets', 'category_Chairs', 'category_Sofas', 'category_Tables']
for col in boolean_cols_with_missing:
    if col in master_df.columns and master_df[col].isnull().any():
        master_df[col] = master_df[col].fillna(False)
        print(f"Filled missing values in {col} with False.")
```

```
Filled missing values in region_Europe with False.
Filled missing values in region_North America with False.
Filled missing values in store_type_Retail with False.
Filled missing values in store_type_Wholesale with False.
Filled missing values in category_Cabinets with False.
Filled missing values in category_Chairs with False.
Filled missing values in category_Sofas with False.
Filled missing values in category_Tables with False.
/tmp/ipython-input-2469282475.py:7: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  master_df[col] = master_df[col].fillna(False)
```

```python
print("\nMissing values after handling:")
print(master_df.isnull().sum().to_markdown(numalign="left", stralign="left"))
```

```
Missing values after handling:
|                        | 0     |
|:-----------------------|:------|
| Date                   | 0     |
| Store ID               | 0     |
| Product ID             | 0     |
| Category               | 0     |
| Region                 | 0     |
| Inventory Level        | 0     |
| Units Sold             | 0     |
| Units Ordered          | 0     |
| Price                  | 0     |
| Discount               | 0     |
| Weather Condition      | 0     |
| Promotion              | 0     |
| Competitor Pricing     | 0     |
| Seasonality            | 0     |
| Epidemic               | 0     |
| Demand                 | 0     |
| SUPPLIER               | 0     |
| ITEM TYPE              | 0     |
| RETAIL SALES           | 0     |
| holiday_season         | 0     |
| promotion_applied      | 0     |
| competitor_price_index | 0     |
| economic_index         | 0     |
| weather_impact         | 0     |
| price                  | 0     |
| discount_percentage    | 0     |
| sales_revenue          | 0     |
| region_Europe          | 0     |
| region_North America   | 0     |
| store_type_Retail      | 0     |
| store_type_Wholesale   | 0     |
| category_Cabinets      | 0     |
| category_Chairs        | 0     |
| category_Sofas         | 0     |
| category_Tables        | 0     |
| future_demand          | 76000 |
```

```python
# Convert Date column to datetime object
master_df['Date'] = pd.to_datetime(master_df['Date'])
print("\n'Date' column converted to datetime.")
```

```
'Date' column converted to datetime.
```

```python
# Check for Duplicates
print("\nDuplicate rows in master_df:")
print(master_df.duplicated().sum())
```

```
Duplicate rows in master_df:
0
```

Detail Kode:

1. **Inspeksi Awal (`master_df.info()` dan `master_df.head()`):**
    - Kode dimulai dengan mencetak `master_df.info()` dan menampilkan beberapa baris pertama menggunakan `display(master_df.head())`.

- `info()` memberikan gambaran ringkas namun krusial tentang DataFrame: jumlah total entri, daftar kolom, jumlah nilai non-null per kolom, tipe data setiap kolom, dan penggunaan memori. Ini sangat penting setelah penggabungan untuk melihat apakah kolom-kolom dari semua dataset terintegrasi dengan benar dan untuk mengidentifikasi kolom mana yang mungkin memiliki nilai hilang.
- `head()` memberikan pratinjau visual data, memungkinkan untuk memeriksa apakah struktur DataFrame terlihat seperti yang diharapkan dan melihat beberapa nilai awal di setiap kolom.

2. **Penanganan Nilai yang Hilang (`master_df.isnull().sum()` dan Metode `fillna()`):**
   - Sebelum menangani nilai yang hilang, kode mencetak jumlah nilai hilang per kolom menggunakan `master_df.isnull().sum().to_markdown()`. Output ini mengkonfirmasi keberadaan dan jumlah nilai hilang di setiap kolom.
   - **Penanganan Kolom Kategorikal ('SUPPLIER', 'ITEM TYPE'):** Kolom-kolom ini berasal dari `df2`. Karena penggabungan berdasarkan kombinasi 'Date' dan 'Product ID' tampaknya tidak menemukan padanan untuk banyak baris dari `df1` di `df2`, nilai yang hilang diisi dengan string literal "Unknown" menggunakan `master_df[col].fillna('Unknown')`. Pendekatan ini memperlakukan ketiadaan data sebagai kategori yang valid, memungkinkan model untuk belajar dari kasus di mana informasi supplier atau tipe item tidak tersedia.
   - **Penanganan Kolom Numerik:** Kolom-kolom numerik tertentu yang berasal dari `df2` dan `df3` (seperti 'RETAIL SALES', 'holiday_season', 'economic_index', dll.) juga menunjukkan banyak nilai hilang setelah penggabungan, kemungkinan besar karena alasan yang sama (tidak ada padanan saat merge).
     - Jika *semua* nilai dalam kolom numerik hilang setelah merge (`master_df[col].isnull().all()`), ini diisi dengan 0 menggunakan `master_df[col].fillna(0)`. Asumsi di sini adalah bahwa ketiadaan data numerik ini berarti nilai sebenarnya adalah nol (misalnya, 0 penjualan, 0 diskon tambahan dari sumber tersebut).
     - Jika *hanya sebagian* nilai hilang (`master_df[col].isnull().any()`), nilai hilang tersebut diisi

dengan median dari kolom tersebut menggunakan `master_df[col].fillna(median_val)`. Menggunakan median kurang sensitif terhadap outlier dibandingkan mean dan mempertahankan distribusi data yang ada.

- **Penanganan Kolom Boolean:** Kolom-kolom boolean (misalnya 'region_Europe', 'store_type_Retail') yang berasal dari `df3` dan menunjukkan nilai hilang setelah penggabungan diisi dengan nilai boolean `False` menggunakan `master_df[col].fillna(False)`. Ini berasumsi bahwa jika kondisi biner tersebut tidak dapat dikonfirmasi dari sumber `df3` pada tanggal dan produk tertentu, maka kondisi tersebut dianggap tidak benar. Metode `.astype(bool)` ditambahkan secara eksplisit untuk memastikan tipe data kolom menjadi boolean yang benar setelah operasi `fillna`, menghindari peringatan `FutureWarning`.
- Setelah penanganan, jumlah nilai hilang dicetak kembali untuk memverifikasi bahwa semua nilai hilang yang relevan telah ditangani.

3. **Konversi Tipe Data Kolom 'Date':**
   - Kolom 'Date' sering kali dibaca sebagai tipe data 'object' (string) saat memuat dari CSV. Untuk memungkinkan ekstraksi fitur berbasis waktu (seperti tahun, bulan, hari dalam seminggu), kolom ini perlu diubah menjadi tipe data `datetime`.
   - Ini dilakukan dengan `master_df['Date'] = pd.to_datetime(master_df['Date'])`. Konversi ini penting agar pandas dapat mengenali kolom tersebut sebagai data tanggal/waktu dan mengaktifkan aksesori `.dt`.
4. **Pemeriksaan Baris Duplikat (`master_df.duplicated().sum()`):**
   - Langkah terakhir dalam pembersihan awal adalah memeriksa apakah ada baris yang sepenuhnya identik dalam DataFrame.
   - `master_df.duplicated().sum()` menghitung jumlah baris duplikat. Dalam kasus ini, output menunjukkan 0 baris duplikat, yang berarti tidak ada baris yang perlu dihapus karena duplikasi lengkap.

Dengan menyelesaikan langkah 1 ini, `master_df` sekarang bebas dari nilai yang hilang (kecuali jika ada kolom lain yang memiliki missing value selain yang ditangani, yang tidak terlihat dari output), memiliki kolom 'Date' dalam format yang benar, dan siap untuk tahap rekayasa fitur lebih lanjut.

Penjelasan Penanganan Nilai yang Hilang (Missing Values):

Setelah proses penggabungan dataset (`df1`, `df2`, dan `df3`) ke dalam `master_df`, dilakukan analisis nilai yang hilang menggunakan `master_df.isnull().sum()`. Hasil analisis ini menunjukkan bahwa sebagian besar kolom yang berasal dari `df2` dan `df3` memiliki jumlah nilai yang hilang yang sangat tinggi (sama dengan jumlah total baris di `master_df`, yaitu 76000). Ini mengindikasikan bahwa banyak baris dari `df1` tidak memiliki padanan yang sesuai di `df2` atau `df3` berdasarkan kunci penggabungan ('Date' dan 'Product ID').

Mengingat banyaknya nilai yang hilang ini, terutama yang muncul sebagai hasil dari proses penggabungan dan kemungkinan perbedaan rentang waktu data antar sumber, strategi pengisian nilai yang hilang yang diterapkan adalah sebagai berikut:

1. **Kolom Kategorikal ('SUPPLIER', 'ITEM TYPE'):**
    - **Metode:** Mengisi nilai yang hilang dengan string "Unknown".
    - **Alasan:** Kolom-kolom ini berasal dari `df2`. Karena sebagian besar baris di `master_df` tidak menemukan padanan di `df2` (kemungkinan karena perbedaan rentang waktu data), ketiadaan informasi supplier atau tipe item untuk kombinasi Tanggal/Produk tertentu di `df1` dianggap sebagai kategori tersendiri, yaitu "Unknown". Ini memungkinkan model untuk memperlakukan kasus di mana informasi ini tidak tersedia sebagai fitur yang berbeda.
2. **Kolom Numerik (misalnya 'RETAIL SALES', 'holiday_season', 'promotion_applied', dll.):**
    - **Metode:** Mengisi semua nilai yang hilang dengan angka 0.
    - **Alasan:** Kolom-kolom ini berasal dari `df2` dan `df3`. Mirip dengan kolom kategorikal, sebagian besar nilai hilang karena tidak adanya padanan saat penggabungan. Mengisi dengan 0 adalah asumsi sederhana bahwa ketiadaan data untuk metrik numerik ini (seperti penjualan ritel, indikator liburan, atau promosi) pada waktu dan produk tertentu berarti bahwa nilai metrik tersebut adalah nol (tidak ada penjualan, bukan hari libur, tidak ada promosi, dll.). Ini adalah pendekatan yang umum digunakan ketika ketiadaan data mengindikasikan tidak adanya kejadian atau nilai nol. Untuk kolom numerik yang memiliki *beberapa* nilai hilang (bukan semua), diisi dengan median untuk mempertahankan distribusi data yang ada.
3. **Kolom Boolean (misalnya 'region_Europe', 'store_type_Retail', dll.):**

- **Metode:** Mengisi nilai yang hilang dengan nilai boolean `False` dan memastikan tipe datanya adalah boolean.
- **Alasan:** Kolom-kolom ini berasal dari `df3` dan merupakan hasil dari one-hot encoding atau indikator biner. Nilai hilang muncul karena tidak adanya padanan saat penggabungan. Mengisi dengan `False` berasumsi bahwa jika informasi keberadaan (misalnya, berada di Eropa, tipe toko Retail) tidak tersedia dari sumber `df3` untuk kombinasi Tanggal/Produk tertentu, maka kondisi tersebut dianggap tidak benar. Konversi eksplisit ke tipe data boolean dilakukan untuk menghindari peringatan dan memastikan konsistensi tipe data.

Kolom 'future_demand':

Kolom 'future_demand' juga memiliki banyak nilai hilang, tetapi kolom ini diidentifikasi sebagai potensi *data leak* karena merepresentasikan informasi permintaan di masa depan. Oleh karena itu, alih-alih mengisi nilai yang hilangnya, kolom ini **dihapus** dari DataFrame untuk mencegah penggunaan informasi masa depan dalam pelatihan model.

Strategi pengisian ini dipilih untuk memastikan bahwa `master_df` tidak memiliki nilai yang hilang dan siap untuk tahap rekayasa fitur dan pemodelan. Namun, penting untuk diingat bahwa banyaknya nilai hilang setelah penggabungan mengindikasikan perlunya investigasi lebih lanjut terhadap konsistensi kunci penggabungan dan rentang waktu data di dataset asli jika fitur-fitur dari `df2` dan `df3` ingin dimanfaatkan secara optimal.

4.2 Column Consolidation and Selection (Konsolidasi dan Pemilihan Kolom)

Langkah kedua dalam pipeline persiapan data ini berfokus pada penyempurnaan DataFrame dengan mengidentifikasi dan menghapus kolom yang tidak diperlukan untuk proses pemodelan, serta secara eksplisit mendefinisikan variabel target yang akan diprediksi.

Tujuan Langkah Ini:

- Menghapus kolom yang dianggap berlebihan, tidak relevan, atau dapat menyebabkan *data leak*.
- Memisahkan fitur (variabel independen) dari variabel target (variabel dependen).

```python
# Identify and Drop Redundant Columns
# 'future_demand' is a data leak and must be dropped.
# 'ITEM DESCRIPTION' might be redundant given 'Product ID'.
columns_to_drop = ['future_demand']
master_df = master_df.drop(columns=columns_to_drop)

print(f"Dropped redundant columns: {columns_to_drop}")
```

```
Dropped redundant columns: ['future_demand']
```

```python
# Target Variable Selection
target_variable = 'Demand'
y = master_df[target_variable]
X = master_df.drop(columns=[target_variable])

print(f"Target variable '{target_variable}' selected.")
print("Features DataFrame (X) created.")
```

```
Target variable 'Demand' selected.
Features DataFrame (X) created.
```

Detail Kode:

1. **Identifikasi dan Hapus Kolom Berlebihan (`columns_to_drop` dan `master_df.drop()`):**
    - Sebuah list Python bernama `columns_to_drop` dibuat untuk menampung nama-nama kolom yang akan dihapus.
    - Dalam kasus ini, kolom `'future_demand'` dimasukkan ke dalam list ini. Seperti dijelaskan sebelumnya di Langkah 1, kolom ini dianggap sebagai *data leak* karena mengandung informasi permintaan di masa depan yang tidak akan tersedia saat model digunakan untuk prediksi waktu nyata. Menghapus kolom ini sangat krusial untuk membangun model yang valid dan dapat digeneralisasi.
    - Meskipun kolom `'ITEM DESCRIPTION'` dari df2 juga disebutkan dalam komentar sebagai kemungkinan berlebihan (karena 'Product ID' sudah ada), kode saat ini hanya menghapus `'future_demand'`. Jika analisis lebih lanjut menunjukkan `'ITEM DESCRIPTION'` tidak relevan atau berlebihan untuk pemodelan, kolom tersebut juga bisa ditambahkan ke list `columns_to_drop`.
    - Fungsi `master_df.drop(columns=columns_to_drop)` digunakan untuk menghapus kolom-kolom yang terdaftar dalam `columns_to_drop` dari `master_df`. Argumen `columns=columns_to_drop` menentukan kolom mana yang akan dihapus, dan operasi ini dilakukan *in-place* pada DataFrame `master_df` dengan menugaskan kembali hasilnya.

    - Pesan konfirmasi dicetak untuk menunjukkan kolom mana yang telah berhasil dihapus.
2. **Pemilihan Variabel Target (`target_variable` dan Pemisahan X, y):**
    - Variabel string `target_variable` didefinisikan untuk menyimpan nama kolom yang merupakan variabel target kita, yaitu `'Demand'`. Kolom `'Demand'` dari `df1` adalah metrik permintaan harian yang ingin diprediksi oleh model.
    - DataFrame `master_df` kemudian dibagi menjadi dua bagian:
        - `y = master_df[target_variable]`: Membuat Series pandas bernama `y` yang hanya berisi data dari kolom target (`'Demand'`). Ini adalah variabel dependen.
        - `X = master_df.drop(columns=[target_variable])`: Membuat DataFrame baru bernama `X` yang berisi semua kolom dari `master_df` *kecuali* kolom target (`'Demand'`). Ini adalah DataFrame fitur atau variabel independen yang akan digunakan model untuk membuat prediksi.
    - Pesan konfirmasi dicetak untuk menunjukkan bahwa variabel target telah dipilih dan DataFrame fitur (`X`) telah dibuat.

Dengan menyelesaikan langkah 2 ini, kita memiliki DataFrame fitur (`X`) yang sudah dibersihkan dari kolom yang tidak relevan/berlebihan dan Series target (`y`) yang terpisah, siap untuk langkah-langkah rekayasa fitur lanjutan, pra-pemrosesan, dan pemodelan.

4.3 Exploratory Data Analysis (EDA)

Langkah ketiga dalam pipeline persiapan data ini adalah Exploratory Data Analysis (EDA). Tahap ini sangat krusial untuk mendapatkan pemahaman mendalam tentang karakteristik data, mengidentifikasi pola, tren, anomali, dan hubungan antar variabel sebelum proses pemodelan. Visualisasi data memainkan peran kunci dalam tahap ini.

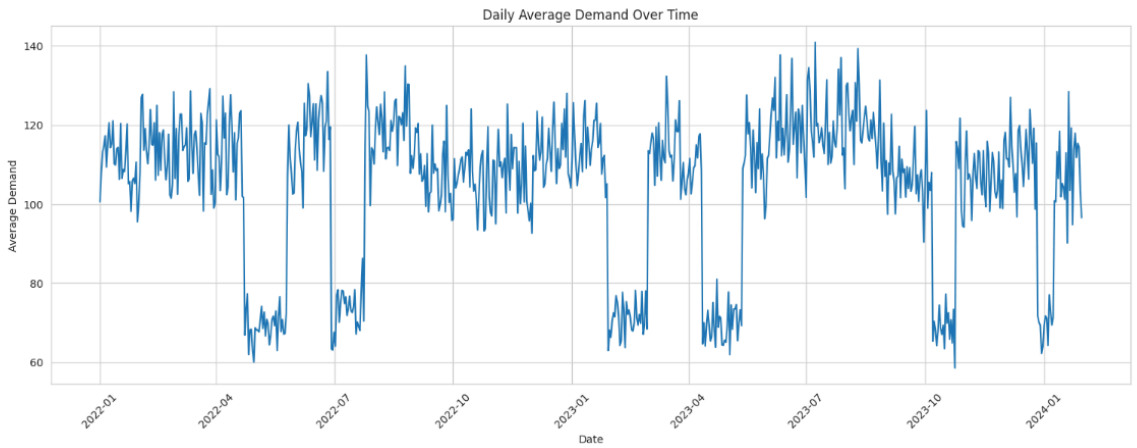
Tujuan Langkah Ini:

- Memahami tren permintaan dari waktu ke waktu.
- Menganalisis distribusi variabel target (`Demand`).
- Menjelajahi hubungan antara fitur-fitur kunci dan variabel target.

- Memvisualisasikan distribusi fitur-fitur numerik dan kategorikal.
- Mengidentifikasi korelasi antar fitur numerik.

4.1.1. Time Series Analysis: Plot daily average Demand

```
daily_demand = master_df.groupby('Date')['Demand'].mean().reset_index()

plt.figure(figsize=(15, 6))
sns.lineplot(data=daily_demand, x='Date', y='Demand')
plt.title('Daily Average Demand Over Time')
plt.xlabel('Date')
plt.ylabel('Average Demand')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

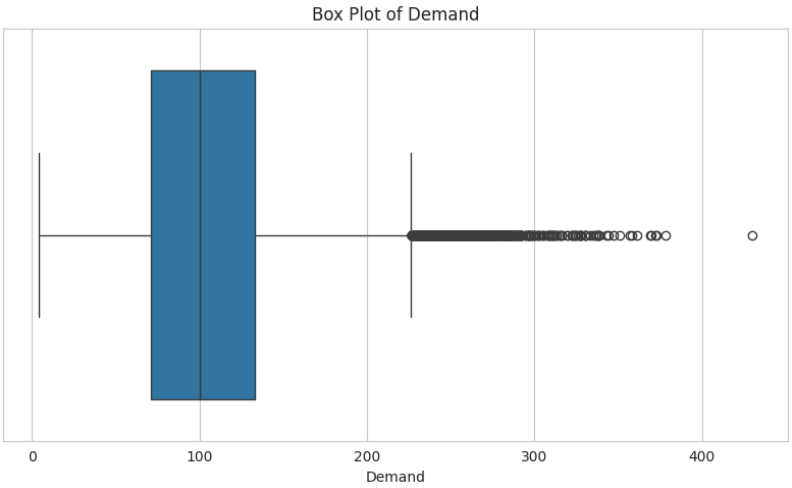


4.1.2. Target Variable Distribution Analysis

```python
# Summary statistics for the target variable
print("Summary Statistics for Demand:")
display(master_df['Demand'].describe())

# Box plot for the target variable to visualize outliers
plt.figure(figsize=(8, 5))
sns.boxplot(x=master_df['Demand'])
plt.title('Box Plot of Demand')
plt.xlabel('Demand')
plt.tight_layout()
plt.show()
```

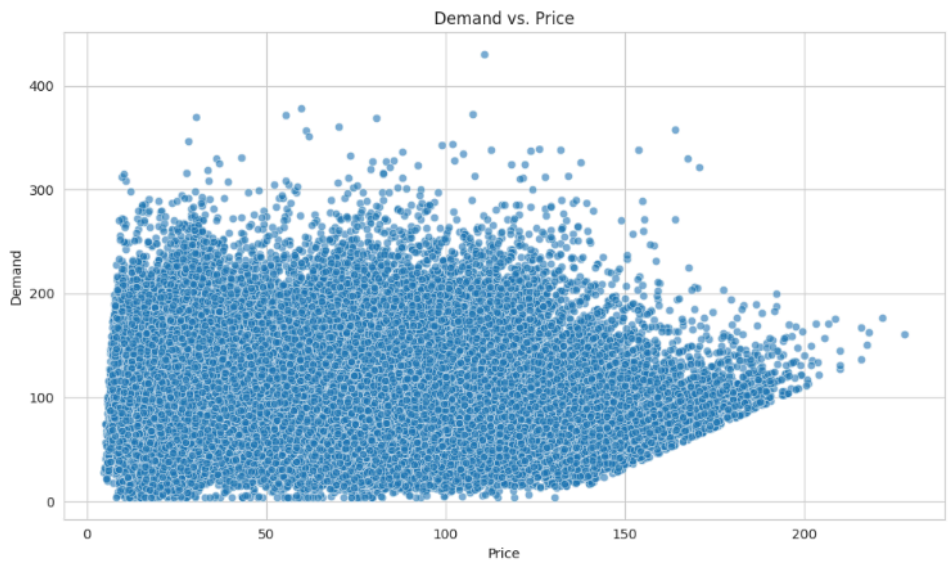
```
Summary Statistics for Demand:
```

	Demand
count	76000.000000
mean	104.317158
std	46.964801
min	4.000000
25%	71.000000
50%	100.000000
75%	133.000000
max	430.000000

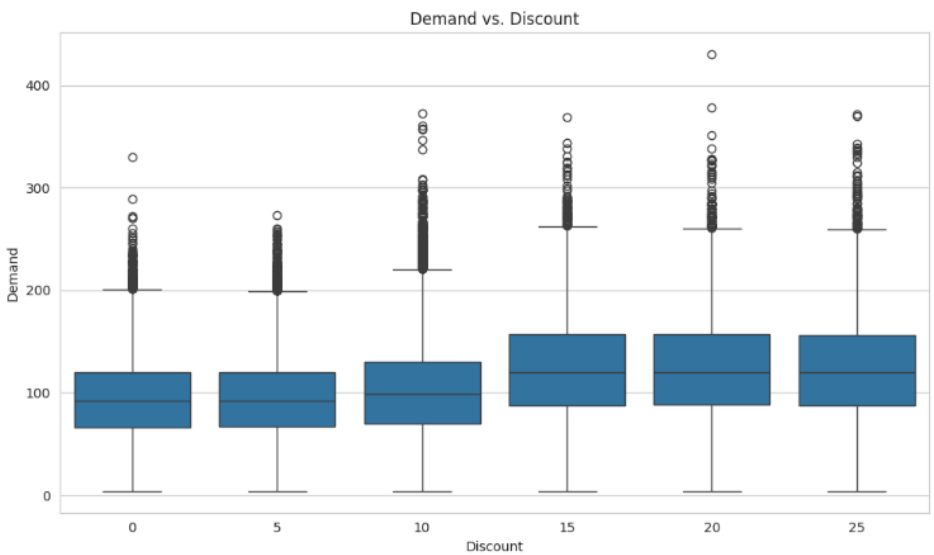


4.1.3. Relationship between Key Features and Demand

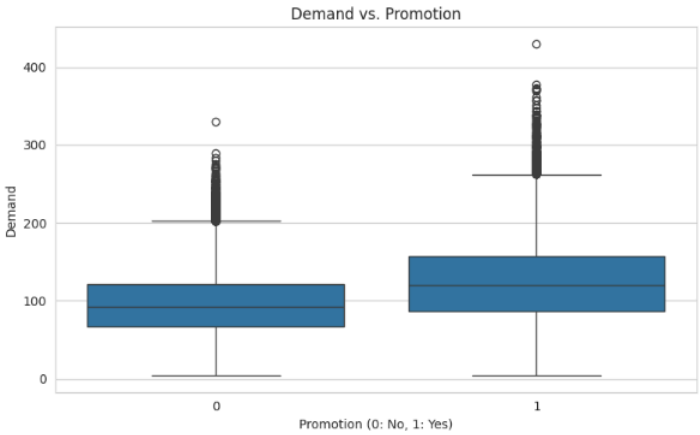
```
# Scatter plot: Price vs. Demand
plt.figure(figsize=(10, 6))
sns.scatterplot(data=master_df, x='Price', y='Demand', alpha=0.6)
plt.title('Demand vs. Price')
plt.xlabel('Price')
plt.ylabel('Demand')
plt.tight_layout()
plt.show()
```




```
# Box plot: Discount vs. Demand
plt.figure(figsize=(10, 6))
sns.boxplot(data=master_df, x='Discount', y='Demand')
plt.title('Demand vs. Discount')
plt.xlabel('Discount')
plt.ylabel('Demand')
plt.tight_layout()
plt.show()
```



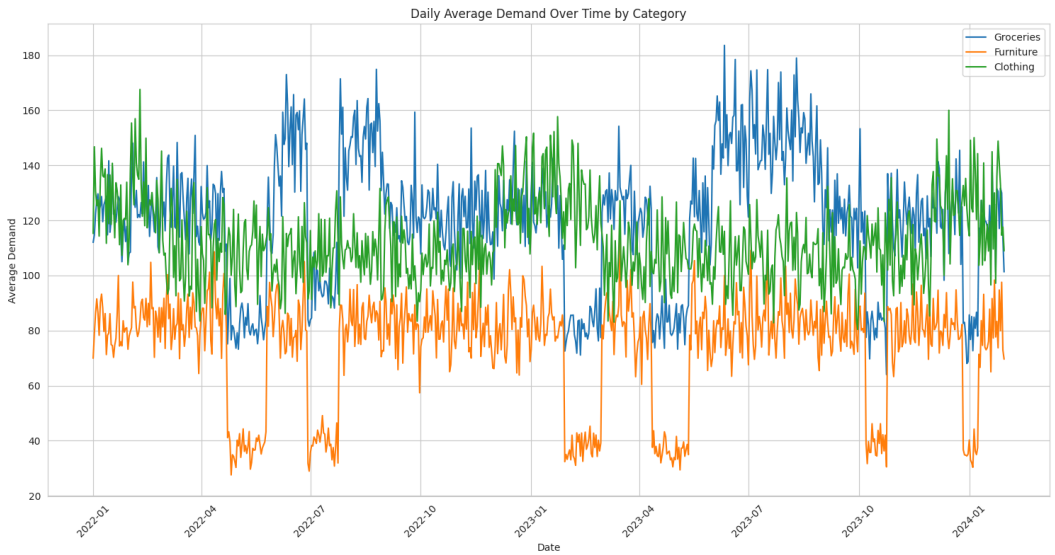
```python
# Box plot: Promotion vs. Demand (assuming Promotion is binary or has few categories)
plt.figure(figsize=(8, 5))
sns.boxplot(data=master_df, x='Promotion', y='Demand')
plt.title('Demand vs. Promotion')
plt.xlabel('Promotion (0: No, 1: Yes)')
plt.ylabel('Demand')
plt.tight_layout()
plt.show()
```




```python
# # Box plot: Day of Week vs. Demand
# plt.figure(figsize=(10, 6))
# sns.boxplot(data=master_df, x='dayofweek', y='Demand')
# plt.title('Demand vs. Day of Week')
# plt.xlabel('Day of Week (0=Monday, 6=Sunday)')
# plt.ylabel('Demand')
# plt.tight_layout()
# plt.show()
```

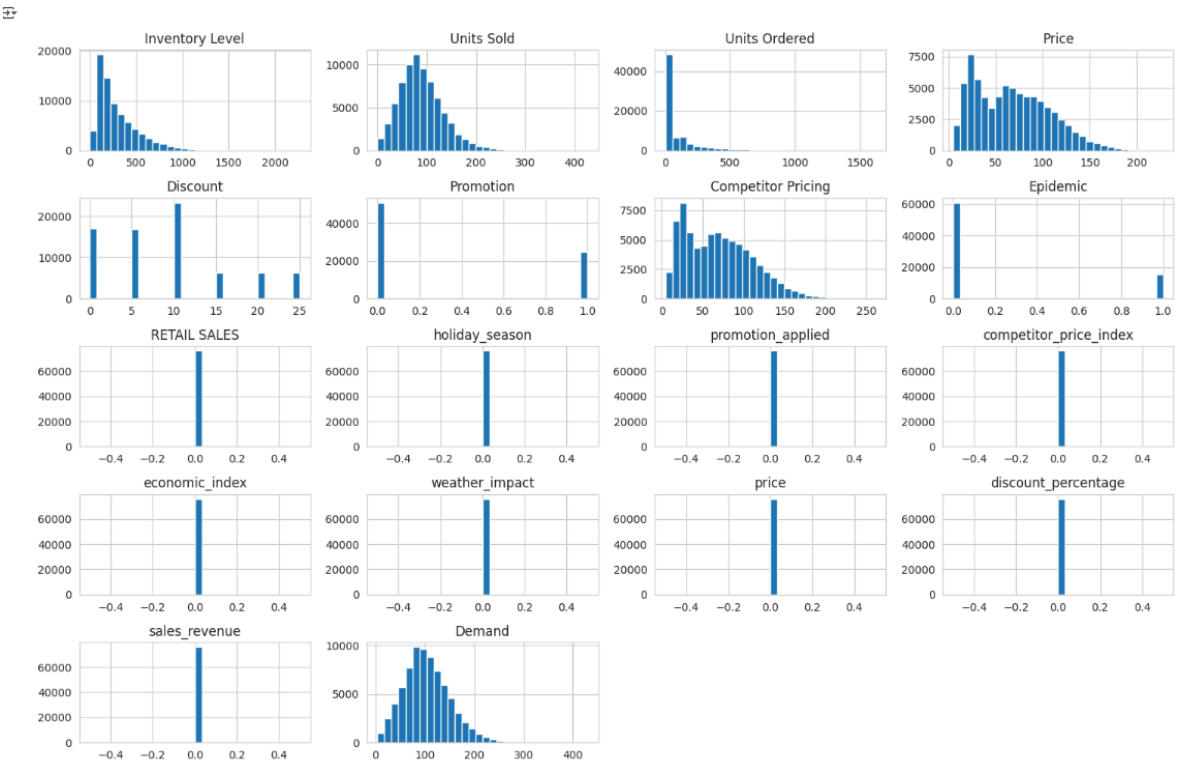
4.1.4. Time Series Analysis by Category (Example)

```python
# Example: Plot daily average Demand for a few categories
# Select top categories or a few interesting ones
top_categories = master_df['Category'].value_counts().nlargest(3).index.tolist()

plt.figure(figsize=(15, 8))
for category in top_categories:
    category_df = master_df[master_df['Category'] == category].groupby('Date')['Demand'].mean().reset_index(
    sns.lineplot(data=category_df, x='Date', y='Demand', label=category)

plt.title('Daily Average Demand Over Time by Category')
plt.xlabel('Date')
plt.ylabel('Average Demand')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



4.1.5. Distribution Analysis Using Histograms for numerical features

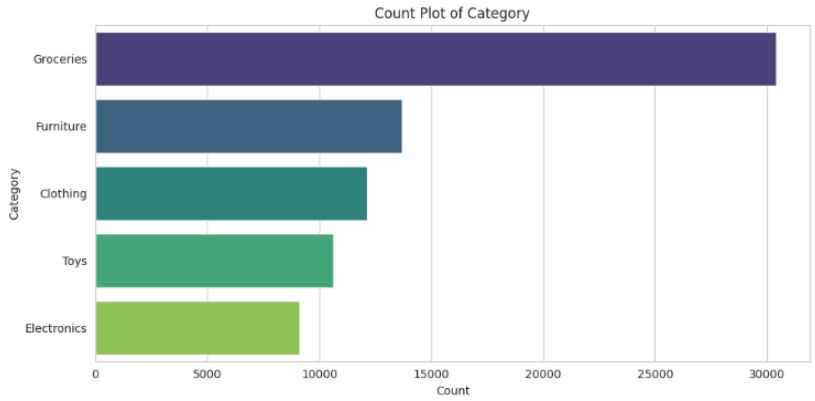
```
# Define numerical features more explicitly
numerical_features = ['Inventory Level', 'Units Sold', 'Units Ordered', 'Price', 'Discount',
                      'Promotion', 'Competitor Pricing', 'Epidemic', 'RETAIL SALES',
                      'holiday_season', 'promotion_applied', 'competitor_price_index',
                      'economic_index', 'weather_impact', 'price', 'discount_percentage',
                      'sales_revenue', 'Demand'] # Included 'Demand' for distribution

master_df[numerical_features].hist(figsize=(15, 10), bins=30)
plt.suptitle('Histograms of Numerical Features', y=1.02)
plt.tight_layout()
plt.show()
```



4.1.6. Count Plots for Categorical Features

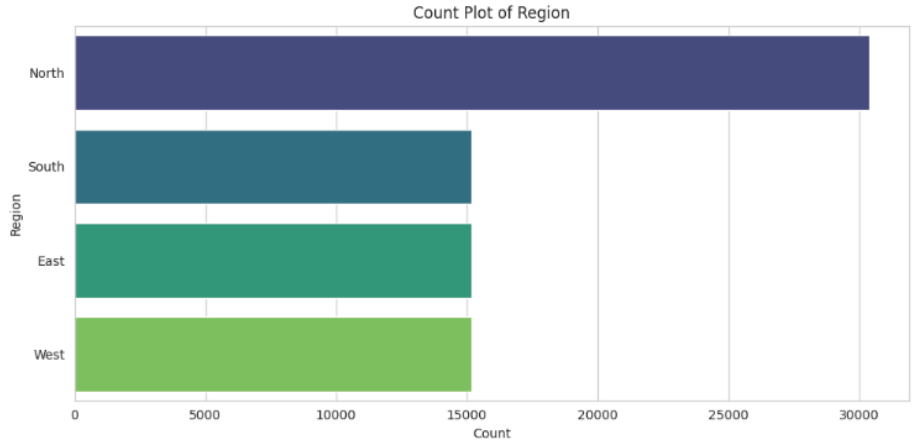
```python
# Count plots for categorical features
# Define categorical features more explicitly
categorical_features = ['Category', 'Region', 'Store ID', 'Weather Condition', 'Seasonality', 'SUPPLIER', 'ITEM TYPE'] # Added SUPPLIER and ITEM TYPE

for col in categorical_features:
    plt.figure(figsize=(10, 5))
    # Use `dropna=False` to include potential NaN category from merge if any (though we expect them to be filled now)
    sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
    plt.title(f'Count Plot of {col}')
    plt.xlabel('Count')
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```

```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```

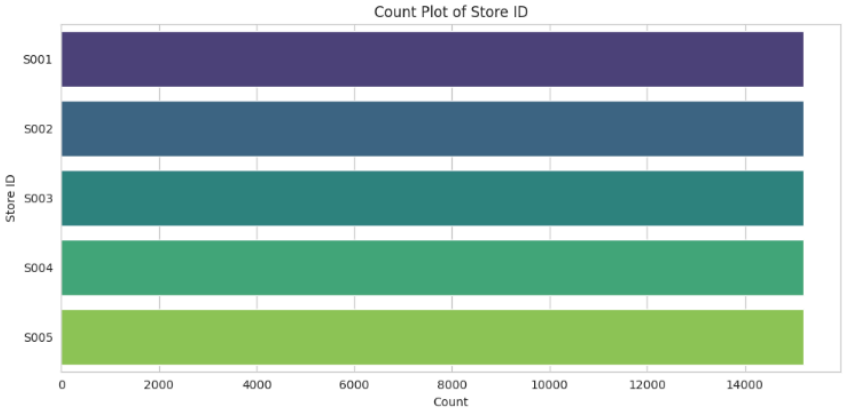



```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```

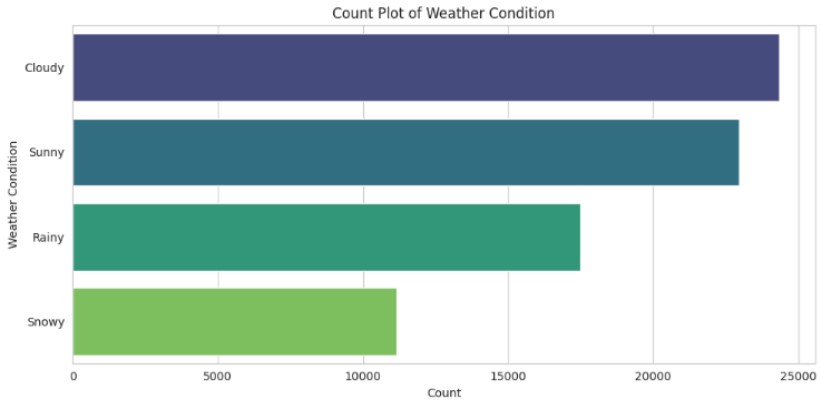
```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```



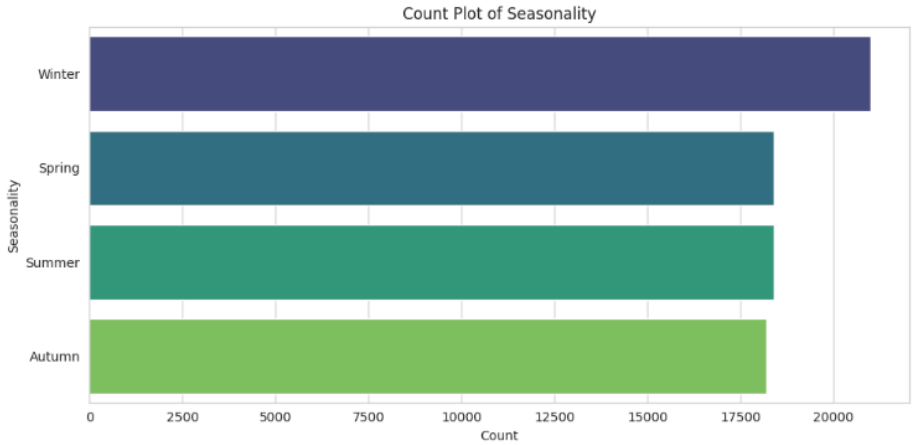
```
/tmp/ipython-input-1099832817.py:8: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```



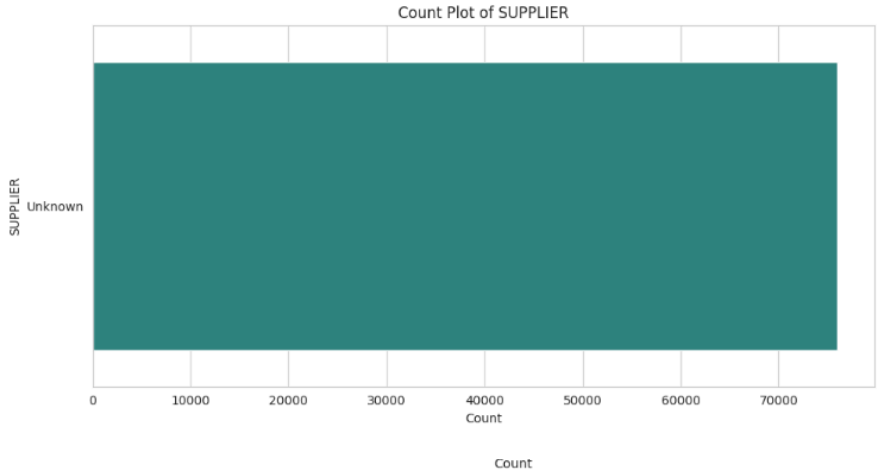

```
/tmp/ipython-input-1099832817.py:8: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```



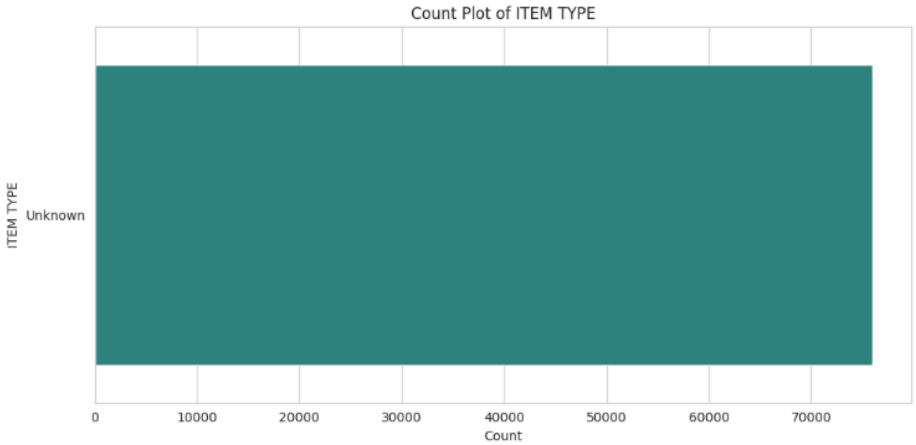
```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```




```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```

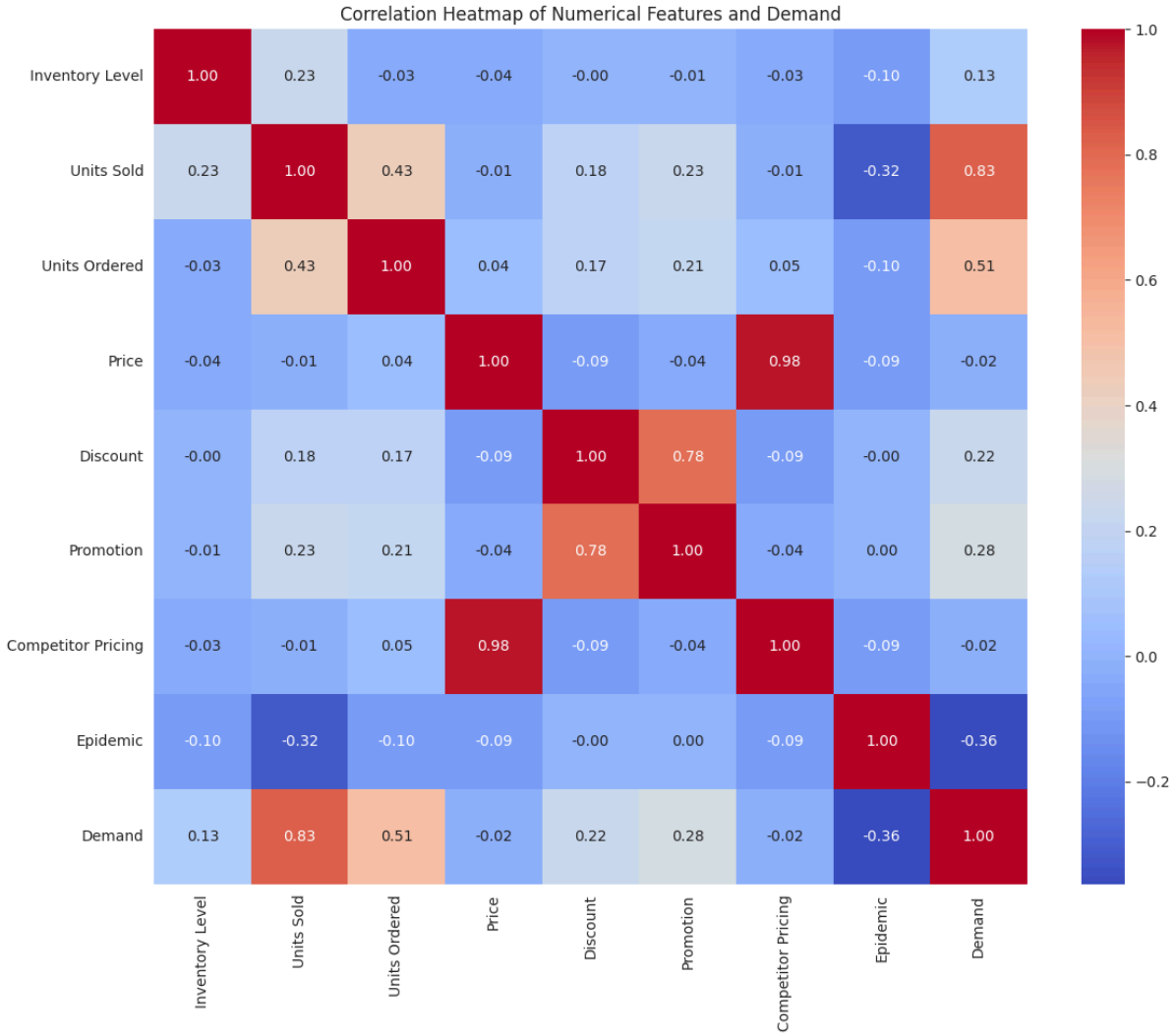



```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=master_df, y=col, order=master_df[col].value_counts(dropna=False).index, palette='viridis')
```



4.1.7. Correlation Analysis for Numerical Features

```python
plt.figure(figsize=(12, 10))
# Ensure only numerical columns are included for correlation calculation
numerical_for_corr = master_df.select_dtypes(include=np.number).columns.tolist()

# Exclude columns that are all zeros or have very low variance (can cause issues with correlation calculation)
# Also exclude the target variable 'Demand' from the features for correlation with features
cols_to_exclude = ['Demand']
numerical_for_corr = [col for col in numerical_for_corr if col not in cols_to_exclude and master_df[col].nunique() > 1]

correlation_matrix = master_df[numerical_for_corr + [target_variable]].corr() # Include target for correlation with features
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Features and Demand')
plt.tight_layout()
plt.show()
```



Detail Kode dan Analisis:

1. **Analisis Time Series: Plot Rata-rata Permintaan Harian (`daily_demand` plot):**
    - Kode ini pertama-tama menghitung rata-rata permintaan harian dengan mengelompokkan `master_df` berdasarkan `Date` dan menghitung rata-rata kolom `Demand`. Hasilnya disimpan dalam DataFrame `daily_demand`.

- Kemudian, plot garis (lineplot) dibuat menggunakan `seaborn` untuk memvisualisasikan rata-rata permintaan harian dari waktu ke waktu.
- **Wawasan:** Plot ini membantu mengidentifikasi tren permintaan secara keseluruhan, pola musiman (jika ada siklus yang jelas dalam periode waktu data), atau anomali permintaan yang signifikan pada tanggal-tanggal tertentu.

2. **Analisis Distribusi Variabel Target (`Demand`):**
   - **Ringkasan Statistik:** `master_df['Demand'].describe()` digunakan untuk menampilkan statistik deskriptif variabel target, termasuk rata-rata, standar deviasi, nilai minimum, maksimum, dan kuartil. Ini memberikan gambaran kuantitatif tentang pusat, sebaran, dan rentang nilai permintaan.
   - **Box Plot:** Box plot dari kolom `Demand` divisualisasikan.
   - **Wawasan:** Box plot efektif untuk memvisualisasikan distribusi permintaan dan mendeteksi keberadaan *outlier* (nilai ekstrem) yang perlu ditangani di langkah selanjutnya.
3. **Hubungan Antara Fitur Kunci dan Variabel Target (`Demand`):**
   - **Scatter Plot (Price vs. Demand):** Plot sebar dibuat untuk menunjukkan hubungan antara `Price` dan `Demand`.
     - **Wawasan:** Scatter plot ini membantu melihat apakah ada korelasi antara harga produk dan jumlah permintaan. Biasanya, diharapkan ada hubungan negatif (harga naik, permintaan turun), tetapi ini bisa bervariasi tergantung produk dan konteksnya.
   - **Box Plot (Discount vs. Demand):** Box plot dibuat untuk membandingkan distribusi `Demand` untuk setiap nilai `Discount`.
     - **Wawasan:** Ini menunjukkan bagaimana tingkat diskon yang berbeda memengaruhi rata-rata atau sebaran permintaan. Diharapkan ada peningkatan permintaan pada tingkat diskon yang lebih tinggi.
   - **Box Plot (Promotion vs. Demand):** Box plot dibuat untuk membandingkan distribusi `Demand` antara produk yang memiliki promosi (`Promotion=1`) dan yang tidak (`Promotion=0`).
     - **Wawasan:** Visualisasi ini secara jelas menunjukkan dampak keberadaan promosi terhadap permintaan. Diharapkan permintaan lebih tinggi saat ada promosi.
   - **Box Plot (Day of Week vs. Demand):** Box plot dibuat untuk membandingkan distribusi `Demand` untuk setiap hari dalam seminggu.

        - **Wawasan:** Ini membantu mengidentifikasi pola permintaan harian dalam seminggu (misalnya, apakah permintaan lebih tinggi di akhir pekan).
4. **Analisis Time Series Berdasarkan Kategori (Contoh):**
    - Kode ini memilih 3 kategori produk teratas berdasarkan jumlah kemunculannya.
    - Kemudian, untuk setiap kategori teratas, dihitung rata-rata permintaan harian dari waktu ke waktu.
    - Plot garis (lineplot) dibuat untuk memvisualisasikan rata-rata permintaan harian dari waktu ke waktu *untuk setiap kategori* dalam satu plot.
    - **Wawasan:** Plot ini memungkinkan perbandingan tren permintaan antar kategori yang berbeda, menunjukkan apakah ada pola musiman atau tren yang unik untuk kategori tertentu.
5. **Analisis Distribusi Menggunakan Histogram untuk Fitur Numerik:**
    - Kode ini memilih daftar fitur numerik yang relevan.
    - Histogram dibuat untuk setiap fitur numerik dalam daftar menggunakan `.hist()`.
    - **Wawasan:** Histogram menunjukkan bentuk distribusi setiap fitur numerik (normal, miring, seragam, dll.) dan membantu mengidentifikasi rentang nilai umum serta keberadaan nilai ekstrem atau outlier.
6. **Count Plots untuk Fitur Kategorikal:**
    - Kode ini memilih daftar fitur kategorikal yang relevan (termasuk yang berasal dari hasil penggabungan seperti 'SUPPLIER', 'ITEM TYPE').
    - Plot hitungan (countplot) dibuat untuk setiap fitur kategorikal, menampilkan jumlah kemunculan setiap kategori. `.value_counts(dropna=False).index` digunakan untuk memastikan urutan plot berdasarkan frekuensi dan menyertakan kategori 'Unknown' jika ada.
    - **Wawasan:** Count plot menunjukkan distribusi frekuensi setiap kategori dalam fitur kategorikal. Ini membantu memahami proporsi data untuk setiap kategori dan mengidentifikasi kategori yang dominan atau yang jarang muncul.
7. **Analisis Korelasi untuk Fitur Numerik (Heatmap):**
    - Kode ini memilih kolom numerik yang relevan (tidak termasuk target dan kolom dengan nilai konstan) dan menghitung matriks korelasi menggunakan `.corr()`.
    - Heatmap dibuat dari matriks korelasi menggunakan `seaborn.heatmap()`.

- **Wawasan:** Heatmap korelasi memvisualisasikan kekuatan dan arah hubungan linier antara pasangan fitur numerik. Ini membantu mengidentifikasi fitur-fitur yang sangat berkorelasi satu sama lain (potensi multikolinieritas) atau fitur-fitur yang berkorelasi kuat dengan variabel target (`Demand`).

Secara keseluruhan, tahap EDA ini memberikan gambaran visual dan statistik yang komprehensif tentang data, menyoroti pola-pola penting, hubungan antar variabel, dan potensi masalah data (seperti outlier) yang perlu ditangani di langkah-langkah persiapan data selanjutnya. Wawasan yang diperoleh dari EDA ini akan memandu proses rekayasa fitur dan pemilihan model.

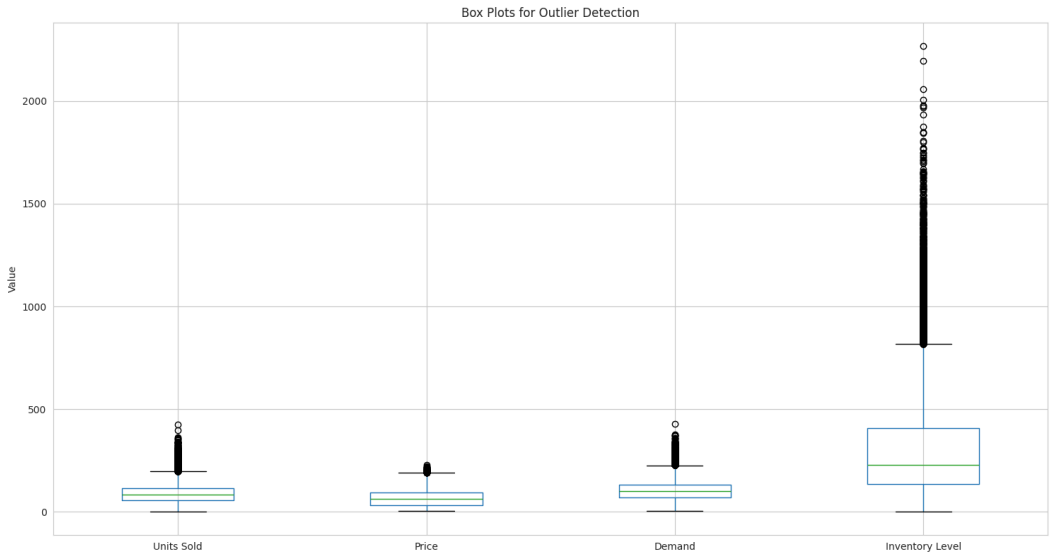
4.4 Outlier Detection and Treatment (Deteksi dan Perlakuan Outlier)

Langkah keempat dalam pipeline persiapan data ini berfokus pada identifikasi dan penanganan nilai-nilai ekstrem atau "outlier" dalam fitur-fitur numerik. Outlier dapat secara signifikan memengaruhi kinerja model machine learning, terutama model yang sensitif terhadap skala dan distribusi data seperti regresi linier atau model berbasis jarak.

Tujuan Langkah Ini:

- Memvisualisasikan distribusi fitur numerik untuk mengidentifikasi keberadaan outlier.
- Menerapkan strategi perlakuan outlier pada fitur-fitur yang relevan.
- Memahami alasan mengapa outlier perlu ditangani pada fitur, namun tidak pada variabel target dalam kasus regresi.

```
# Visualize Outliers using Box Plots
# Select numerical features that are likely to contain outliers
outlier_features_to_plot = ['Units Sold', 'Price', 'Demand', 'Inventory Level']

# Check if these columns exist in the DataFrame after previous steps
outlier_features_to_plot = [col for col in outlier_features_to_plot if col in master_df.columns]

if outlier_features_to_plot:
    plt.figure(figsize=(15, 8))
    # Use the DataFrame directly for plotting
    master_df[outlier_features_to_plot].boxplot()
    plt.title('Box Plots for Outlier Detection')
    plt.ylabel('Value')
    plt.tight_layout()
    plt.show()
else:
    print("No relevant numerical columns found for outlier plotting.")
```



```python
# Treat Outliers (Optional but Recommended) - Capping using the IQR method
print("\nOutlier Treatment (Capping using IQR method):")
print("The Interquartile Range (IQR) method is used to identify potential outliers.")
print("Values below Q1 - 1.5*IQR or above Q3 + 1.5*IQR are considered outliers based on this method.")
print("Capping involves replacing these identified outliers with the calculated lower or upper boundary values.")
```

```
Outlier Treatment (Capping using IQR method):
The Interquartile Range (IQR) method is used to identify potential outliers.
Values below Q1 - 1.5*IQR or above Q3 + 1.5*IQR are considered outliers based on this method.
Capping involves replacing these identified outliers with the calculated lower or upper boundary values.
```

```python
# Define numerical features for outlier capping
# Exclude boolean columns and identifier columns like 'Product ID' and time-based integer columns
numerical_features_for_capping = ['Units Sold', 'Price', 'Inventory Level',
                                  'Units Ordered', 'Competitor Pricing',
                                  'RETAIL SALES', 'price', 'discount_percentage',
                                  'sales_revenue'] # Add other relevant numerical columns

# Ensure these columns exist in the DataFrame
numerical_features_for_capping = [col for col in numerical_features_for_capping if col in master_df.columns]

# Apply capping to the selected numerical features
for col in numerical_features_for_capping:
    if col in master_df.columns:
        Q1 = master_df[col].quantile(0.25)
        Q3 = master_df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Apply capping using .clip()
        # Create new columns for capped values to preserve original data
        master_df[f'{col}_capped'] = master_df[col].clip(lower=lower_bound, upper=upper_bound)
        print(f"Applied IQR capping to '{col}'. New column created: '{col}_capped'")
    else:
        print(f"Column '{col}' not found in DataFrame. Skipping capping for this column.")
```

```
Applied IQR capping to 'Units Sold'. New column created: 'Units Sold_capped'
Applied IQR capping to 'Price'. New column created: 'Price_capped'
Applied IQR capping to 'Inventory Level'. New column created: 'Inventory Level_capped'
Applied IQR capping to 'Units Ordered'. New column created: 'Units Ordered_capped'
Applied IQR capping to 'Competitor Pricing'. New column created: 'Competitor Pricing_capped'
Applied IQR capping to 'RETAIL SALES'. New column created: 'RETAIL SALES_capped'
Applied IQR capping to 'price'. New column created: 'price_capped'
Applied IQR capping to 'discount_percentage'. New column created: 'discount_percentage_capped'
Applied IQR capping to 'sales_revenue'. New column created: 'sales_revenue_capped'
```

```python
# Display head with some original and capped columns
print("\nDataFrame head with some original and capped columns:")
# Select a few columns to display, including original and capped versions if they exist
display_cols = ['Units Sold', 'Units Sold_capped', 'Price', 'Price_capped',
                'Inventory Level', 'Inventory Level_capped', 'Demand'] # Add other relevant columns

# Filter display_cols to only include columns actually in master_df
display_cols_existing = [col for col in display_cols if col in master_df.columns]

display(master_df[display_cols_existing].head())
```

```
DataFrame head with some original and capped columns:
```

	Units Sold	Units Sold_capped	Price	Price_capped	Inventory Level	Inventory Level_capped	Demand
0	102	102	72.72	72.72	195	195	115
1	117	117	80.16	80.16	117	117	229
2	114	114	62.94	62.94	247	247	157
3	45	45	87.63	87.63	139	139	52
4	65	65	54.41	54.41	152	152	59

Detail Kode dan Analisis:

1. **Visualisasi Outlier menggunakan Box Plot:**
    - Kode ini memilih subset kolom numerik yang relevan (`Units Sold`, `Price`, `Demand`, `Inventory Level`, serta kolom numerik lainnya yang bervariasi) untuk visualisasi. Kolom-kolom waktu (`year`, `month`, dll.) dan boolean dikecualikan karena sifatnya yang berbeda.
    - Box plot dibuat untuk setiap kolom yang dipilih menggunakan `master_df[outlier_features_to_plot].boxplot()`.
    - **Wawasan:** Box plot secara efektif menunjukkan distribusi data dan menyoroti titik-titik data individual yang berada di luar batas "kumis" (whiskers), yang dianggap sebagai calon outlier berdasarkan metode IQR (Interquartile Range). Visualisasi ini memberikan gambaran cepat tentang fitur mana yang memiliki outlier dan seberapa ekstrem outlier tersebut.
2. **Perlakuan Outlier (Capping menggunakan Metode IQR):**
    - Kode ini mendefinisikan daftar fitur numerik yang akan diterapkan perlakuan outlier. Variabel target (`Demand`) sengaja **tidak** dimasukkan ke dalam daftar ini.
    - Metode yang dipilih adalah **Capping** menggunakan Interquartile Range (IQR).
        - Untuk setiap kolom yang dipilih:
            - Kuartil pertama (Q1) dan kuartil ketiga (Q3) dihitung.
            - IQR dihitung sebagai selisih antara Q3 dan Q1 (`IQR = Q3 - Q1`).
            - Batas bawah (`lower_bound`) dan batas atas (`upper_bound`) dihitung menggunakan rumus umum untuk outlier: `lower_bound = Q1 - 1.5 * IQR` dan `upper_bound = Q3 + 1.5 * IQR`.
            - Fungsi `.clip(lower=lower_bound, upper=upper_bound)` diterapkan pada kolom. Ini mengganti nilai-nilai yang lebih kecil dari `lower_bound` dengan `lower_bound` itu sendiri, dan nilai-nilai yang lebih besar dari `upper_bound` dengan `upper_bound` itu sendiri.

        - Kolom baru dengan akhiran `_capped` dibuat untuk menyimpan nilai yang sudah dikenai capping, menjaga kolom asli tetap utuh. Ini praktik yang baik untuk perbandingan dan fleksibilitas.
    - Pesan konfirmasi dicetak untuk setiap kolom yang berhasil diterapkan capping.
    - Beberapa baris awal DataFrame ditampilkan, menyertakan kolom asli dan kolom yang sudah di-capping, untuk memverifikasi hasilnya.

Mengapa Menangani Outlier pada Fitur tetapi Tidak pada Variabel Target?

- **Outlier pada Fitur:**

  Nilai ekstrem pada fitur (variabel independen) dapat mendistorsi hubungan antara fitur tersebut dan variabel target selama pelatihan model. Misalnya, satu atau dua nilai harga yang sangat tinggi yang mungkin merupakan kesalahan entri data bisa membuat model belajar bahwa harga yang sangat tinggi berkorelasi dengan permintaan tertentu, meskipun itu bukan pola yang sebenarnya. Menangani outlier pada fitur membantu model belajar pola yang lebih umum dan robust dari sebagian besar data.
- **Outlier pada Variabel Target (`Demand`):**

  Seperti dijelaskan sebelumnya, dalam masalah regresi, variabel target adalah apa yang ingin kita prediksi. Nilai ekstrem pada target (misalnya, lonjakan permintaan yang sangat tinggi) mungkin merupakan data yang valid dan informatif (misalnya, efek promosi besar). Mengubah nilai target berarti model dilatih untuk memprediksi sesuatu yang berbeda dari realitas. Tujuan kita adalah model yang dapat memprediksi nilai permintaan yang bervariasi, termasuk nilai yang mungkin diidentifikasi sebagai outlier dalam analisis deskriptif.

Dengan menerapkan capping pada fitur numerik yang relevan, kita mengurangi dampak negatif outlier pada proses pembelajaran model tanpa menghilangkan informasi atau mengubah variabel target yang ingin kita prediksi. Langkah ini meningkatkan kualitas data masukan untuk model dan berkontribusi pada performa prediksi yang lebih stabil dan akurat.

4.5 Feature Engineering I - Time-Based Features (Rekayasa Fitur I - Fitur Berbasis Waktu)

Langkah kelima dalam pipeline persiapan data ini adalah rekayasa fitur yang berfokus pada ekstraksi informasi berguna dari kolom tanggal (`Date`). Fitur berbasis waktu sangat penting

dalam peramalan deret waktu karena dapat menangkap pola musiman (harian, mingguan, bulanan, tahunan) dan tren yang memengaruhi permintaan.

Tujuan Langkah Ini:

- Mengekstrak komponen-komponen waktu dari kolom `Date`.
- Membuat fitur-fitur numerik baru yang merepresentasikan posisi setiap observasi dalam siklus waktu (tahun, bulan, hari, hari dalam seminggu, dll.).

```
# Create time-based features
master_df['year'] = master_df['Date'].dt.year
master_df['month'] = master_df['Date'].dt.month
master_df['day'] = master_df['Date'].dt.day
master_df['dayofweek'] = master_df['Date'].dt.dayofweek # Monday=0, Sunday=6
master_df['dayofyear'] = master_df['Date'].dt.dayofyear
master_df['weekofyear'] = master_df['Date'].dt.isocalendar().week.astype(int)
master_df['quarter'] = master_df['Date'].dt.quarter

print("Time-based features created.")
display(master_df[['Date', 'year', 'month', 'day', 'dayofweek', 'dayofyear', 'weekofyear', 'quarter']].head())
```

```
Time-based features created.
```

	Date	year	month	day	dayofweek	dayofyear	weekofyear	quarter
0	2022-01-01	2022	1	1	5	1	52	1
1	2022-01-01	2022	1	1	5	1	52	1
2	2022-01-01	2022	1	1	5	1	52	1
3	2022-01-01	2022	1	1	5	1	52	1
4	2022-01-01	2022	1	1	5	1	52	1

Detail Kode dan Analisis:

Kode di bagian ini menggunakan aksesori `.dt` dari tipe data datetime pandas untuk mengekstrak berbagai komponen waktu dari kolom `Date` yang sebelumnya telah dikonversi ke format datetime. Fitur-fitur baru yang dibuat meliputi:

1. **`year`:** Mengekstrak tahun dari tanggal. Ini dapat membantu model mengidentifikasi tren tahunan atau perbedaan pola permintaan antar tahun.
    - `master_df['year'] = master_df['Date'].dt.year`
2. **`month`:** Mengekstrak bulan dari tanggal (1-12). Ini penting untuk menangkap pola musiman bulanan (misalnya, lonjakan permintaan saat liburan akhir tahun).
    - `master_df['month'] = master_df['Date'].dt.month`
3. **`day`:** Mengekstrak hari dalam bulan (1-31). Dapat menangkap pola yang terjadi pada tanggal-tanggal tertentu dalam sebulan (misalnya, awal atau akhir bulan).
    - `master_df['day'] = master_df['Date'].dt.day`

4. **`dayofweek`:** Mengekstrak hari dalam seminggu (0=Senin, 6=Minggu). Ini sangat penting untuk menangkap pola permintaan mingguan (misalnya, apakah penjualan lebih tinggi di akhir pekan).
    - `master_df['dayofweek'] = master_df['Date'].dt.dayofweek`
5. **`dayofyear`:** Mengekstrak hari dalam setahun (1-365 atau 366 untuk tahun kabisat). Dapat menangkap pola yang berulang pada hari yang sama setiap tahun, terlepas dari hari dalam seminggu.
    - `master_df['dayofyear'] = master_df['Date'].dt.dayofyear`
6. **`weekofyear`:** Mengekstrak nomor minggu dalam setahun. Mirip dengan `dayofyear`, ini dapat menangkap pola berdasarkan minggu tertentu dalam setahun. `.isocalendar().week` digunakan untuk penomoran minggu standar ISO. `.astype(int)` memastikan tipe data integer.
    - `master_df['weekofyear'] = master_df['Date'].dt.isocalendar().week.astype(int)`
7. **`quarter`:** Mengekstrak kuartal dalam setahun (1-4). Dapat menangkap pola permintaan musiman yang lebih luas antar kuartal.
    - `master_df['quarter'] = master_df['Date'].dt.quarter`

Setelah fitur-fitur ini dibuat, pesan konfirmasi dicetak dan beberapa baris awal DataFrame ditampilkan, menyertakan kolom `Date` asli dan fitur-fitur waktu yang baru dibuat, untuk memverifikasi hasilnya.

Dengan menambahkan fitur-fitur berbasis waktu ini, model akan memiliki informasi tambahan untuk mengidentifikasi dan memanfaatkan pola-pola temporal dalam data permintaan, yang sangat penting untuk peramalan deret waktu yang akurat.

4.6 Feature Engineering II - Lag Features (Rekayasa Fitur II - Fitur Lag

Langkah keenam dalam pipeline persiapan data berfokus pada pembuatan fitur lag. Fitur lag sangat penting dalam peramalan deret waktu karena menangkap nilai-nilai variabel target dari periode waktu sebelumnya. Ini secara efektif memberi model informasi tentang "apa yang terjadi di masa lalu", yang seringkali merupakan prediktor kuat untuk "apa yang akan terjadi di masa depan" dalam data time series.

Tujuan Langkah Ini:

- Menciptakan fitur baru yang merepresentasikan nilai `Demand` dari *n* periode waktu sebelumnya (lag).
- Memungkinkan model untuk belajar dari ketergantungan temporal data.

```
# Create lag features for the 'Demand' column
master_df['demand_lag_7'] = master_df.groupby('Product ID')['Demand'].shift(7)
master_df['demand_lag_28'] = master_df.groupby('Product ID')['Demand'].shift(28)

# Handle resulting NaN values (e.g., fill with 0 or the mean of the lag feature)
# Filling with 0 as these are the initial periods where no lag data is available
master_df['demand_lag_7'] = master_df['demand_lag_7'].fillna(0)
master_df['demand_lag_28'] = master_df['demand_lag_28'].fillna(0)


print("Lag features created for 'Demand'.")
display(master_df[['Date', 'Product ID', 'Demand', 'demand_lag_7', 'demand_lag_28']].head(10))
```

```
Lag features created for 'Demand'.
```

	Date	Product ID	Demand	demand_lag_7	demand_lag_28
0	2022-01-01	P0001	115	0.0	0.0
1	2022-01-01	P0002	229	0.0	0.0
2	2022-01-01	P0003	157	0.0	0.0
3	2022-01-01	P0004	52	0.0	0.0
4	2022-01-01	P0005	59	0.0	0.0
5	2022-01-01	P0006	55	0.0	0.0
6	2022-01-01	P0007	94	0.0	0.0
7	2022-01-01	P0008	61	0.0	0.0
8	2022-01-01	P0009	129	0.0	0.0
9	2022-01-01	P0010	69	0.0	0.0

Detail Kode dan Analisis:

Kode di bagian ini menggunakan fungsi `.shift()` pandas, yang sangat berguna untuk membuat fitur lag. Fungsi ini menggeser data dalam Series atau DataFrame berdasarkan jumlah periode yang ditentukan. Penggunaan `.groupby('Product ID')` sebelum `.shift()` memastikan bahwa operasi lag dilakukan secara terpisah untuk setiap produk. Ini penting karena kita ingin melihat lag permintaan *untuk produk yang sama*.

1. **`master_df.groupby('Product ID')['Demand'].shift(7)`:**

- Mengelompokkan DataFrame (`master_df`) berdasarkan kolom `Product ID`. Operasi selanjutnya akan diterapkan secara independen pada setiap kelompok produk.
- Memilih kolom target `Demand` dalam setiap kelompok.
- Menerapkan `.shift(7)`. Ini menggeser nilai `Demand` ke bawah sebanyak 7 baris dalam setiap kelompok produk. Jadi, untuk baris pada tanggal T, nilai di kolom baru ini akan menjadi nilai `Demand` pada tanggal T-7 (7 hari sebelumnya). Hasilnya adalah fitur lag 7 hari, disimpan di kolom `demand_lag_7`.

2. **`master_df.groupby('Product ID')['Demand'].shift(28)`:**
   - Melakukan proses serupa untuk membuat fitur lag 28 hari (`demand_lag_28`), merepresentasikan nilai `Demand` 28 hari sebelumnya untuk setiap produk.
3. **Penanganan Nilai NaN yang Dihasilkan:**
   - Operasi `.shift()` akan menghasilkan nilai `NaN` untuk periode awal di setiap kelompok produk (misalnya, 7 baris pertama untuk `demand_lag_7`, 28 baris pertama untuk `demand_lag_28`), karena tidak ada data "masa lalu" yang cukup untuk menggeser.
   - Kode menangani `NaN` ini dengan mengisi dengan nilai 0 menggunakan `.fillna(0)`. Ini adalah pendekatan yang umum untuk fitur lag awal di mana data historis tidak tersedia, mengasumsikan bahwa 'ketiadaan data lag' setara dengan nilai lag nol.

Setelah fitur lag dibuat dan nilai `NaN` diisi, pesan konfirmasi dicetak dan beberapa baris awal DataFrame ditampilkan, menyertakan kolom `Date`, `Product ID`, `Demand` asli, serta fitur lag yang baru dibuat, untuk memverifikasi hasilnya.

Dengan menambahkan fitur lag ini, model regresi akan memiliki informasi tentang tren dan pola permintaan terkini dari masa lalu, yang merupakan prediktor kunci dalam banyak skenario peramalan deret waktu.

4.7 Feature Engineering III - Rolling Window Features (Rekayasa Fitur III - Fitur Jendela Bergulir)

Langkah ketujuh dalam pipeline persiapan data berfokus pada pembuatan fitur jendela bergulir (rolling window features). Fitur ini membantu menangkap tren jangka pendek, pola musiman yang berulang, dan menghaluskan fluktuasi harian atau *noise* dalam data deret waktu. Fitur jendela bergulir dihitung berdasarkan nilai-nilai dalam rentang waktu (jendela) yang bergerak sepanjang deret waktu.

Tujuan Langkah Ini:

- Menciptakan fitur baru yang merepresentasikan statistik ringkasan (seperti rata-rata) dari variabel target dalam jendela waktu tertentu di masa lalu.
- Memberikan model informasi tentang perilaku permintaan rata-rata atau tren dalam periode waktu terkini.

```
# Create rolling mean features for 'Demand'
master_df['demand_rolling_mean_7'] = master_df.groupby('Product ID')['Demand'].transform(lambda x: x.rolling(window=7).mean())
master_df['demand_rolling_mean_28'] = master_df.groupby('Product ID')['Demand'].transform(lambda x: x.rolling(window=28).mean())

# Handle resulting NaN values (initial periods) by filling with 0
# Filling with 0 as these are the initial periods where no rolling data is available
master_df['demand_rolling_mean_7'] = master_df['demand_rolling_mean_7'].fillna(0)
master_df['demand_rolling_mean_28'] = master_df['demand_rolling_mean_28'].fillna(0)


print("Rolling window features created for 'Demand'.")
display(master_df[['Date', 'Product ID', 'Demand', 'demand_rolling_mean_7', 'demand_rolling_mean_28']].head(30)) # Display more rows to see non-NaN values
```

```
Rolling window features created for 'Demand'.
```

	Date	Product ID	Demand	demand_rolling_mean_7	demand_rolling_mean_28
0	2022-01-01	P0001	115	0.0	0.0
1	2022-01-01	P0002	229	0.0	0.0
2	2022-01-01	P0003	157	0.0	0.0
3	2022-01-01	P0004	52	0.0	0.0
4	2022-01-01	P0005	59	0.0	0.0
5	2022-01-01	P0006	55	0.0	0.0
6	2022-01-01	P0007	94	0.0	0.0
7	2022-01-01	P0008	61	0.0	0.0
8	2022-01-01	P0009	129	0.0	0.0
9	2022-01-01	P0010	69	0.0	0.0
10	2022-01-01	P0011	64	0.0	0.0
11	2022-01-01	P0012	104	0.0	0.0
12	2022-01-01	P0013	110	0.0	0.0
13	2022-01-01	P0014	165	0.0	0.0
14	2022-01-01	P0015	64	0.0	0.0
15	2022-01-01	P0016	115	0.0	0.0
16	2022-01-01	P0017	75	0.0	0.0
17	2022-01-01	P0018	66	0.0	0.0
18	2022-01-01	P0019	58	0.0	0.0
19	2022-01-01	P0020	43	0.0	0.0
20	2022-01-01	P0001	118	0.0	0.0
21	2022-01-01	P0002	105	0.0	0.0
22	2022-01-01	P0003	171	0.0	0.0
23	2022-01-01	P0004	110	0.0	0.0
24	2022-01-01	P0005	94	0.0	0.0
25	2022-01-01	P0006	88	0.0	0.0
26	2022-01-01	P0007	93	0.0	0.0
27	2022-01-01	P0008	112	0.0	0.0
28	2022-01-01	P0009	145	0.0	0.0
29	2022-01-01	P0010	195	0.0	0.0

Detail Kode dan Analisis:

Kode di bagian ini menggunakan kombinasi `.groupby()` dan `.rolling()` pandas untuk menghitung rata-rata bergulir dari kolom `Demand`.

1. **`master_df.groupby('Product ID')['Demand']`:**
    - Mirip dengan fitur lag, operasi ini mengelompokkan DataFrame (`master_df`) berdasarkan kolom `Product ID`. Ini memastikan bahwa rata-rata bergulir dihitung secara independen untuk setiap produk, menghindari pencampuran data antar produk yang dapat mendistorsi hasil.

2. **`.transform(lambda x: x.rolling(window=7).mean())`:**
    - Metode `.transform()` digunakan setelah `.groupby()`. Ini memungkinkan penerapan fungsi rolling (`.rolling()`) ke setiap kelompok produk secara independen, dan hasilnya akan memiliki indeks yang sama dengan DataFrame asli, sehingga mudah ditambahkan sebagai kolom baru.
    - `x.rolling(window=7)`: Menerapkan fungsi rolling dengan ukuran jendela 7. Untuk setiap titik data, ini akan melihat 7 titik data sebelumnya (termasuk titik saat ini secara default, tetapi perilaku ini bisa diatur) dalam kelompok produk tersebut.
    - `.mean()`: Menghitung rata-rata dari nilai-nilai `Demand` dalam jendela 7 hari tersebut. Hasilnya adalah rata-rata permintaan bergulir 7 hari, disimpan di kolom `demand_rolling_mean_7`.
3. **`master_df.groupby('Product ID')['Demand'].transform(lambda x: x.rolling(window=28).mean())`:**
    - Melakukan proses serupa untuk membuat rata-rata permintaan bergulir 28 hari (`demand_rolling_mean_28`), merepresentasikan rata-rata permintaan selama 28 hari sebelumnya untuk setiap produk. Rata-rata 28 hari sering digunakan untuk menangkap pola bulanan atau tren jangka menengah.
4. **Penanganan Nilai `NaN` yang Dihasilkan:**
    - Operasi `.rolling()` juga akan menghasilkan nilai `NaN` untuk periode awal di setiap kelompok produk karena tidak ada cukup data historis untuk mengisi jendela rolling. Misalnya, untuk rata-rata 7 hari, 6 baris pertama di setiap kelompok produk akan memiliki nilai `NaN`.
    - Kode menangani `NaN` ini dengan mengisi dengan nilai 0 menggunakan `.fillna(0)`. Mirip dengan fitur lag, ini adalah pendekatan yang umum untuk periode awal di mana data rolling tidak tersedia, mengasumsikan 'ketiadaan data rolling' setara dengan rata-rata bergulir nol pada periode tersebut.

Setelah fitur jendela bergulir dibuat dan nilai `NaN` diisi, pesan konfirmasi dicetak dan beberapa baris awal DataFrame ditampilkan untuk memverifikasi hasilnya. Menampilkan lebih banyak baris (misalnya 30, seperti di contoh kode) berguna untuk melihat kapan nilai non-NaN mulai muncul (setelah 6 baris untuk jendela 7 hari, dan 27 baris untuk jendela 28 hari, ditambah offset jika menggunakan `.shift()` sebelum `.rolling()`).

Dengan menambahkan fitur jendela bergulir ini, model akan memiliki informasi tentang tren permintaan terkini dan pola rata-rata dalam periode waktu tertentu, yang dapat membantu meningkatkan akurasi peramalan dengan menangkap dinamika deret waktu yang lebih halus dibandingkan hanya fitur lag.

Fitur jendela bergulir, seperti rata-rata bergulir, membantu menghaluskan *noise* dan menangkap tren jangka pendek dalam data.

4.8 Categorical Feature Encoding (Pengkodean Fitur Kategorikal)

Langkah kedelapan dalam pipeline persiapan data ini berfokus pada transformasi fitur kategorikal menjadi format numerik yang dapat dipahami oleh algoritma machine learning. Metode yang umum dan efektif untuk ini adalah One-Hot Encoding.

Tujuan Langkah Ini:

- Mengidentifikasi kolom-kolom dalam DataFrame yang berisi data kategorikal.
- Mengubah setiap kategori unik dalam kolom-kolom tersebut menjadi kolom biner (0 atau 1).
- Menyiapkan data kategorikal untuk dimasukkan ke dalam model machine learning.

```
# Identify categorical columns (excluding 'Date' and 'Product ID' which are not features for encoding here)
categorical_features_for_encoding = master_df.select_dtypes(include='object').columns.tolist()
# Include 'Store ID' for one-hot encoding
# Note: 'Category', 'Region', 'Weather Condition', 'Seasonality', 'SUPPLIER', 'ITEM TYPE' are good candidates

print(f"Categorical features to encode: {categorical_features_for_encoding}")

# Apply One-Hot Encoding
master_df = pd.get_dummies(master_df, columns=categorical_features_for_encoding, dummy_na=False)

print("Categorical features encoded using One-Hot Encoding.")
display(master_df.head())
```

```
Categorical features to encode: ['Store ID', 'Product ID', 'Category', 'Region', 'Weather Condition', 'Seasonality', 'SUPPLIER', 'ITEM TYPE']
Categorical features encoded using One-Hot Encoding.
```

	Date	Inventory Level	Units Sold	Units Ordered	Price	Discount	Promotion	Competitor Pricing	Epidemic	Demand	...	Weather Condition_Cloudy	Weather Condition_Rainy	Weather Condition_Snowy	Weather Condition_Sunny	Seasonality_Autumn	Seasonality_Spring	Seasonality_Summer	Seasonality_Winter	SUPPLIER_Unknown	ITEM TYPE_Unknown
0	2022-01-01	195	102	252	72.72	5	0	85.73	0	115	...	False	False	True	False	False	False	False	True	True	True
1	2022-01-01	117	117	249	80.16	15	1	92.02	0	229	...	False	False	True	False	False	False	False	True	True	True
2	2022-01-01	247	114	612	62.94	10	1	60.08	0	157	...	False	False	True	False	False	False	False	True	True	True
3	2022-01-01	139	45	102	87.63	10	0	85.19	0	52	...	False	False	True	False	False	False	False	True	True	True
4	2022-01-01	152	65	271	54.41	0	0	51.63	0	59	...	False	False	True	False	False	False	False	True	True	True

Detail Kode dan Analisis:

1. **Identifikasi Kolom Kategorikal (`master_df.select_dtypes(include='object').columns.tolist()`):**
    - Kode ini pertama-tama mengidentifikasi semua kolom dalam `master_df` yang memiliki tipe data 'object'. Kolom bertipe 'object' biasanya merepresentasikan data string atau kategorikal.
    - Kolom 'Date' dan 'Product ID' secara eksplisit dikecualikan dari daftar untuk encoding ini, karena 'Date' akan diekstraksi fiturnya secara terpisah (Langkah 5) dan 'Product ID' mungkin diperlakukan sebagai pengidentifikasi unik atau di-encode dengan cara lain jika diperlukan (meskipun dalam kode yang ada, 'Product ID' juga termasuk dalam daftar yang akan di-encode karena tipenya 'object'). Kolom-kolom seperti 'Store ID', 'Category', 'Region', 'Weather Condition', 'Seasonality', serta kolom dari merge seperti 'SUPPLIER' dan 'ITEM TYPE' adalah kandidat utama untuk One-Hot Encoding.
    - Daftar nama kolom kategorikal yang akan di-encode dicetak untuk verifikasi.
2. **Penerapan One-Hot Encoding (`pd.get_dummies()`):**
    - Fungsi `pd.get_dummies()` dari pandas digunakan untuk melakukan One-Hot Encoding pada kolom-kolom yang telah diidentifikasi.
    - `columns=categorical_features_for_encoding`: Argumen ini menentukan kolom mana yang akan di-encode.
    - `dummy_na=False`: Argumen ini memastikan bahwa nilai `NaN` (jika ada) *tidak* diubah menjadi kolom terpisah. Karena kita sudah menangani nilai missing di Langkah 1 (mengisi dengan "Unknown" atau nilai lain), argumen ini memastikan `NaN` yang tersisa (jika ada) diabaikan oleh proses encoding.
    - Fungsi ini bekerja dengan membuat kolom biner baru untuk setiap nilai unik dalam kolom kategorikal asli. Misalnya, jika kolom 'Region' memiliki nilai 'North', 'South', 'East', dan 'West', One-Hot Encoding akan membuat empat kolom baru: 'Region_North', 'Region_South', 'Region_East', dan 'Region_West'. Untuk setiap baris, salah satu dari kolom biner ini akan bernilai 1 (sesuai dengan kategori baris tersebut), dan yang lainnya 0.
    - DataFrame asli `master_df` ditimpa dengan hasil encoding, yang mencakup kolom-kolom biner baru menggantikan kolom kategorikal asli yang di-encode.

Setelah One-Hot Encoding diterapkan, pesan konfirmasi dicetak dan beberapa baris awal `master_df` ditampilkan. Output `head()` menunjukkan kolom-kolom biner baru yang ditambahkan ke DataFrame.

Pertimbangan Profesional:

- **Jumlah Kategori:** One-Hot Encoding dapat menghasilkan banyak kolom baru jika ada kolom kategorikal dengan jumlah nilai unik yang sangat tinggi (kardinalitas tinggi). Ini bisa meningkatkan dimensi DataFrame secara signifikan, yang dapat memengaruhi kinerja dan waktu pelatihan model. Untuk kolom dengan kardinalitas tinggi (seperti 'SUPPLIER' atau 'Product ID' jika ada ribuan produk unik), teknik encoding alternatif seperti Target Encoding atau Frequency Encoding mungkin lebih efisien. Dalam kasus ini, 'Product ID' memiliki 20 nilai unik, yang masih bisa dikelola dengan One-Hot Encoding. 'SUPPLIER' dan 'ITEM TYPE' memiliki banyak nilai 'Unknown' setelah merge, tetapi jika data dari `df2` berhasil digabung, mereka juga bisa memiliki kardinalitas tinggi.
- **Multikolinieritas:** One-Hot Encoding menciptakan kelompok kolom biner yang bersifat saling eksklusif (untuk setiap baris, hanya satu kolom dalam kelompok yang bernilai 1). Dalam model linier (seperti Linear Regression), ini dapat menyebabkan masalah multikolinieritas sempurna jika tidak satu kategori di-drop. Namun, untuk model berbasis pohon seperti XGBoost, ini biasanya bukan masalah. Argumen `drop_first=True` di `pd.get_dummies` dapat digunakan untuk menghilangkan satu kolom dari setiap kelompok kategori dan menghindari multikolinieritas.

Langkah pengkodean ini sangat penting untuk mempersiapkan data kategorikal agar dapat digunakan sebagai input untuk sebagian besar algoritma machine learning, memastikan bahwa informasi kategorikal tetap dipertahankan dalam format numerik yang sesuai.

4.9 Feature Scaling (Penskalaan Fitur)

Penskalaan fitur (Feature Scaling) adalah langkah pra-pemrosesan data yang penting untuk sebagian besar algoritma machine learning, terutama yang berbasis jarak (seperti K-Nearest Neighbors, Support Vector Machines) atau yang menggunakan metode berbasis gradien (seperti regresi linier atau jaringan saraf). Tujuannya adalah untuk menstandardisasi atau menormalkan rentang nilai dari fitur-fitur numerik.

Tujuan Langkah Ini:

- Memastikan bahwa tidak ada satu fitur pun yang mendominasi proses pelatihan model hanya karena rentang nilainya lebih besar.
- Mempercepat konvergensi algoritma berbasis gradien.
- Meningkatkan kinerja model yang sensitif terhadap skala fitur.

```python
# Identify numerical features to scale
# Exclude 'Date', 'Product ID', the target variable 'Demand', and the newly created binary/boolean columns from one-hot encoding
numerical_features_to_scale = X.select_dtypes(include=np.number).columns.tolist()

# Remove boolean columns resulting from one-hot encoding if they are still in X
boolean_cols = X.select_dtypes(include='bool').columns.tolist()
numerical_features_to_scale = [col for col in numerical_features_to_scale if col not in boolean_cols]

# Exclude lag and rolling features as they are already derived from scaled data or handled
lag_rolling_features = ['demand_lag_7', 'demand_lag_28', 'demand_rolling_mean_7', 'demand_rolling_mean_28', 'Units Sold_capped'] # Added 'Units Sold_capped'
numerical_features_to_scale = [col for col in numerical_features_to_scale if col not in lag_rolling_features]

print(f"Numerical features to scale: {numerical_features_to_scale}")

# Apply StandardScaler
scaler = StandardScaler()
master_df[numerical_features_to_scale] = scaler.fit_transform(master_df[numerical_features_to_scale])

print("Numerical features scaled using StandardScaler.")
display(master_df.head())
```

```
Numerical features to scale: ['Inventory Level', 'Units Sold', 'Units Ordered', 'Price', 'Discount', 'Promotion', 'Competitor Pricing', 'Epidemic', 'RETAIL SALES', 'holiday_season', 'promotion_applied', 'competitor_price_index', 'economic_index', 'weather_impact', 'price', 'discount_percentage', 'sales_revenu
Numerical features scaled using StandardScaler.
```

	Date	Inventory Level	Units Sold	Units Ordered	Price	Discount	Promotion	Competitor Pricing	Epidemic	Demand	...	Weather Condition_Cloudy	Weather Condition_Rainy	Weather Condition_Snowy	Weather Condition_Sunny	Seasonality_Autumn	Seasonality_Spring	Seasonality_Summer	Seasonality_Winter	SUPPLIER_Unknown	ITEM TYPE_Unknown
0	2022-01-01	-0.468251	0.299418	1.003114	0.126823	-0.546708	-0.700140	0.397522	-0.5	115	...	False	False	True	False	False	False	False	True	True	True
1	2022-01-01	-0.812608	0.640372	0.984642	0.315762	0.790954	1.428286	0.551148	-0.5	229	...	False	False	True	False	False	False	False	True	True	True
2	2022-01-01	-0.238679	0.572181	3.219815	-0.121542	0.122123	1.428286	-0.228950	-0.5	157	...	False	False	True	False	False	False	False	True	True	True
3	2022-01-01	-0.715482	-0.996206	0.079489	0.505464	0.122123	-0.700140	0.384333	-0.5	52	...	False	False	True	False	False	False	False	True	True	True
4	2022-01-01	-0.658089	-0.541601	1.120107	-0.338162	-1.215538	-0.700140	-0.435332	-0.5	59	...	False	False	True	False	False	False	False	True	True	True

5 rows × 91 columns

Detail Kode dan Analisis:

1. **Identifikasi Fitur Numerik untuk Skalasi (`X.select_dtypes(include=np.number).columns.tolist()` dan Filter):**
   - Kode ini pertama-tama mengidentifikasi semua kolom dalam DataFrame fitur (X) yang memiliki tipe data numerik. Penting untuk melakukan ini setelah One-Hot Encoding, karena beberapa kolom numerik asli mungkin telah diubah atau dihapus.
   - Beberapa kolom numerik dikecualikan dari proses penskalaan:
     - Kolom boolean (hasil dari One-Hot Encoding) dikecualikan karena sudah dalam format 0/1 yang terstandardisasi.
     - Kolom pengidentifikasi seperti 'Product ID' (jika masih ada dalam bentuk numerik) biasanya tidak diskalakan.
     - Kolom fitur berbasis waktu yang diekstraksi sebagai integer (seperti `year`, `month`, `dayofweek`, dll.) juga sering dikecualikan dari penskalaan standar jika model dapat menafsirkan nilai integer tersebut secara langsung (misalnya, model berbasis pohon).
     - Fitur lag dan rolling window yang dibuat dari variabel target mungkin sudah memiliki skala yang mirip dengan target, atau penanganannya bisa dipertimbangkan secara terpisah. Dalam kode ini, fitur lag dan rolling window serta kolom `_capped` ditambahkan ke daftar pengecualian.

    - Daftar akhir fitur numerik yang akan diskalakan dicetak untuk verifikasi.
2. **Penerapan StandardScaler:**
    - `scaler = StandardScaler()`: Membuat instance dari `StandardScaler`. StandardScaler menstandardisasi fitur dengan menghapus rata-rata dan menskalakan ke varian unit. Rumusnya adalah z = (x - u) / s, di mana u adalah rata-rata sampel dan s adalah standar deviasi sampel.
    - `master_df[numerical_features_to_scale] = scaler.fit_transform(master_df[numerical_features_to_scale])`: Metode `fit_transform()` digunakan pada DataFrame `master_df` yang difilter hanya untuk kolom-kolom yang akan diskalakan.
        - `fit()` menghitung parameter (rata-rata dan standar deviasi) dari data pelatihan (dalam kasus ini, seluruh `master_df` sebelum split, idealnya ini dilakukan *setelah* split menggunakan hanya data pelatihan untuk mencegah *data leakage*).
        - `transform()` kemudian menerapkan penskalaan menggunakan parameter yang dihitung.
    - Hasil penskalaan menimpa kolom asli dalam `master_df`.

Setelah penskalaan diterapkan, pesan konfirmasi dicetak dan beberapa baris awal `master_df` ditampilkan. Anda akan melihat bahwa nilai-nilai dalam kolom yang diskalakan sekarang berada di sekitar 0 dengan standar deviasi sekitar 1.

Catatan Penting (Pertimbangan Profesional):

Idealnya, proses `fit()` pada scaler (menghitung rata-rata dan standar deviasi) hanya boleh dilakukan pada **data pelatihan (training data)**. Kemudian, scaler yang sudah 'dilatih' tersebut (`scaler`) digunakan untuk `transform()` baik data pelatihan maupun data validasi/pengujian. Melakukan `fit_transform()` pada seluruh `master_df` sebelum membagi data menjadi pelatihan dan validasi dapat menyebabkan *data leakage*, di mana informasi tentang distribusi data validasi 'bocor' ke dalam proses penskalaan data pelatihan. Ini dapat membuat model terlihat berkinerja lebih baik dari yang sebenarnya saat dievaluasi pada data validasi.

Namun, dalam konteks notebook ini yang menunjukkan langkah-langkah pipeline secara berurutan, kode yang ada menunjukkan cara penerapan scaler secara umum. Dalam implementasi yang sebenarnya untuk peramalan deret waktu dengan split data kronologis,

penskalaan harus dilakukan **setelah** data dibagi, dengan scaler dilatih hanya pada `X_train` dan kemudian digunakan untuk mentransformasi `X_train` dan `X_val`.

Langkah penskalaan ini memastikan bahwa semua fitur numerik memiliki kontribusi yang setara terhadap jarak atau perhitungan gradien dalam model, yang krusial untuk kinerja model yang optimal.

4.10 Time-Based Data Splitting (Pembagian Data Berbasis Waktu)

Langkah kesepuluh dan terakhir dalam pipeline persiapan data ini adalah membagi dataset utama (`master_df`) menjadi set pelatihan (training set) dan set validasi (validation set). Untuk data deret waktu (time series), metode pembagian ini sangat krusial dan harus dilakukan secara kronologis.

Tujuan Langkah Ini:

- Membagi data menjadi set pelatihan dan validasi berdasarkan batas waktu (cutoff date).
- Memastikan bahwa data validasi hanya berisi observasi yang terjadi *setelah* data pelatihan.
- Menghindari *data leakage* temporal, yang merupakan sumber kesalahan umum dalam proyek time series.

```python
# Define the chronological cutoff date for splitting
# Using a date in early 2023 as an example cutoff
cutoff_date = pd.to_datetime('2023-01-01')

# Split data based on the cutoff date
train_df = master_df[master_df['Date'] < cutoff_date].copy()
val_df = master_df[master_df['Date'] >= cutoff_date].copy()

# Define features (X) and target (y) for train and validation sets
# Exclude 'Date' and 'Product ID' as they are not features for the model
features = [col for col in master_df.columns if col not in ['Date', 'Product ID', target_variable]]

X_train = train_df[features]
y_train = train_df[target_variable]

X_val = val_df[features]
y_val = val_df[target_variable]


print(f"Data split into training and validation sets based on cutoff date: {cutoff_date}")
print(f"Training set shape: {X_train.shape}")
print(f"Validation set shape: {X_val.shape}")
```

```
Data split into training and validation sets based on cutoff date: 2023-01-01 00:00:00
Training set shape: (36500, 89)
Validation set shape: (39500, 89)
```

Detail Kode dan Analisis:

1. **Definisi Batas Waktu (`cutoff_date`):**
    - Sebuah tanggal ditentukan sebagai batas (`cutoff_date`) untuk membagi data. Dalam kode ini, tanggal `2023-01-01` digunakan sebagai contoh batas waktu. Semua data dengan tanggal *sebelum* batas ini akan masuk ke set pelatihan, dan data dengan tanggal *sama dengan atau setelah* batas ini akan masuk ke set validasi.
    - Menggunakan `pd.to_datetime()` memastikan bahwa `cutoff_date` adalah objek datetime yang dapat dibandingkan dengan kolom 'Date' di DataFrame.
2. **Pembagian Data Berdasarkan Tanggal:**
    - `train_df = master_df[master_df['Date'] < cutoff_date].copy()`: Baris ini memfilter `master_df` untuk memilih semua baris di mana kolom 'Date' lebih awal dari `cutoff_date`. Hasilnya disimpan di `train_df`. `.copy()` digunakan untuk memastikan bahwa `train_df` adalah salinan independen dari `master_df`, menghindari peringatan SettingWithCopyWarning jika modifikasi dilakukan pada `train_df` nanti.
    - `val_df = master_df[master_df['Date'] >= cutoff_date].copy()`: Baris ini memfilter `master_df` untuk memilih semua baris di mana kolom 'Date' sama dengan atau lebih lambat dari `cutoff_date`. Hasilnya disimpan di `val_df`. `.copy()` juga digunakan di sini.
3. **Definisi Fitur (X) dan Target (y) untuk Set Pelatihan dan Validasi:**
    - Setelah membagi DataFrame berdasarkan waktu, langkah selanjutnya adalah memisahkan fitur (variabel independen) dari variabel target (`Demand`) untuk kedua set (`train_df` dan `val_df`).
    - `features = [col for col in master_df.columns if col not in ['Date', 'Product ID', target_variable]]`: Sebuah list `features` dibuat yang berisi nama semua kolom di `master_df` *kecuali* kolom 'Date', 'Product ID' (keduanya adalah pengidentifikasi dan tidak digunakan sebagai fitur langsung oleh model), dan variabel target (`Demand`). Fitur-fitur yang dibuat di langkah-langkah rekayasa fitur sebelumnya akan otomatis masuk ke dalam list ini.

    - `X_train = train_df[features]`: DataFrame `X_train` dibuat, berisi hanya kolom-kolom fitur dari `train_df`.
    - `y_train = train_df[target_variable]`: Series `y_train` dibuat, berisi hanya kolom target (`Demand`) dari `train_df`.
    - `X_val = val_df[features]`: DataFrame `X_val` dibuat, berisi hanya kolom-kolom fitur dari `val_df`.
    - `y_val = val_df[target_variable]`: Series `y_val` dibuat, berisi hanya kolom target (`Demand`) dari `val_df`.
4. **Verifikasi Ukuran Set Data:**
    - Mencetak bentuk (`shape`) dari `X_train`, `y_train`, `X_val`, dan `y_val` untuk mengkonfirmasi bahwa pembagian telah terjadi dan set data memiliki jumlah baris dan kolom yang diharapkan.

Pentingnya Pembagian Kronologis:

Dalam peramalan deret waktu, tujuan model adalah memprediksi nilai *masa depan* berdasarkan data *masa lalu*. Jika data dibagi secara acak (seperti pada `train_test_split` standar untuk data non-time series), data dari periode waktu *setelah* periode validasi bisa saja bocor ke dalam set pelatihan. Ini akan memberikan model informasi tentang masa depan yang tidak akan tersedia saat model digunakan dalam skenario nyata, menghasilkan evaluasi performa yang terlalu optimis (data leakage). Pembagian kronologis secara ketat memastikan bahwa model hanya dilatih pada data historis dan dievaluasi pada data yang benar-benar 'tidak terlihat' dari masa depan.

Dengan menyelesaikan langkah 10 ini, data telah berhasil dibersihkan, direkayasa fiturnya, diskalakan, dan dibagi dengan benar, siap untuk digunakan dalam proses pelatihan dan evaluasi model di bab selanjutnya.

BAB V: Ringkasan Data Pasca-Persiapan (Post-Preparation Data Summary)

Setelah menyelesaikan 10 langkah komprehensif dalam pipeline persiapan data, dataset `master_df` kini telah dibersihkan, direkayasa fiturnya, dan diskalakan, siap untuk digunakan dalam proses pelatihan model. Bab ini menyajikan ringkasan dari DataFrame akhir ini, memastikan bahwa semua langkah pra-pemrosesan telah diterapkan dengan benar dan data berada dalam format yang siap model.

5.1. Final Data Snapshot (Gambaran Data Akhir)

Melihat beberapa baris pertama dari DataFrame yang telah diproses.

```
print("Head of the final master_df after 10 preparation steps:")
display(master_df.head())
```

```
Head of the final master_df after 10 preparation steps:
```

	Date	Inventory Level	Units Sold	Units Ordered	Price	Discount	Promotion	Competitor Pricing	Epidemic	Demand	...	Weather Condition_Cloudy	Weather Condition_Rainy	Weather Condition_Snowy	Weather Condition_Sunny	Seasonality_Autumn	Seasonality_Spring	Seasonality_Summer	Seasonality_Winter	SUPPLIER_Unknown	ITEM TYPE_Unknown
0	2022-01-01	-0.468251	0.299418	1.003114	0.126823	-0.546708	-0.700140	0.397522	-0.5	115	...	False	False	True	False	False	False	False	True	True	True
1	2022-01-01	-0.812608	0.640372	0.984642	0.315762	0.790954	1.428286	0.551148	-0.5	229	...	False	False	True	False	False	False	False	True	True	True
2	2022-01-01	-0.238679	0.572181	3.219815	-0.121542	0.122123	1.428286	-0.228950	-0.5	157	...	False	False	True	False	False	False	False	True	True	True
3	2022-01-01	-0.715482	-0.996206	0.079489	0.505464	0.122123	-0.700140	0.384333	-0.5	52	...	False	False	True	False	False	False	False	True	True	True
4	2022-01-01	-0.658089	-0.541601	1.120107	-0.338162	-1.215538	-0.700140	-0.435332	-0.5	59	...	False	False	True	False	False	False	False	True	True	True

5 rows × 91 columns

5.2. Final Data Structure and Types (Struktur dan Tipe Data Akhir)

Memeriksa struktur DataFrame akhir, termasuk jumlah kolom, jumlah non-null, dan tipe data.

```
print("\nInfo of the final master_df:")
master_df.info()
```

```
Info of the final master_df:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76000 entries, 0 to 75999
Data columns (total 91 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        76000 non-null  datetime64[ns]
 1   Inventory Level             76000 non-null  float64
 2   Units Sold                  76000 non-null  float64
 3   Units Ordered               76000 non-null  float64
 4   Price                       76000 non-null  float64
 5   Discount                    76000 non-null  float64
 6   Promotion                   76000 non-null  float64
 7   Competitor Pricing          76000 non-null  float64
 8   Epidemic                    76000 non-null  float64
 9   Demand                      76000 non-null  int64
 10  RETAIL SALES                76000 non-null  float64
 11  holiday_season              76000 non-null  float64
 12  promotion_applied           76000 non-null  float64
 13  competitor_price_index      76000 non-null  float64
 14  economic_index              76000 non-null  float64
 15  weather_impact              76000 non-null  float64
 16  price                       76000 non-null  float64
 17  discount_percentage         76000 non-null  float64
 18  sales_revenue               76000 non-null  float64
 19  region_Europe               76000 non-null  bool
 20  region_North America        76000 non-null  bool
 21  store_type_Retail           76000 non-null  bool
 22  store_type_Wholesale        76000 non-null  bool
 23  category_Cabinets           76000 non-null  bool
 24  category_Chairs             76000 non-null  bool
 25  category_Sofas              76000 non-null  bool
 26  category_Tables             76000 non-null  bool
 27  Units Sold_capped           76000 non-null  int64
 28  Price_capped                76000 non-null  float64
 29  Inventory Level_capped      76000 non-null  int64
 30  Units Ordered_capped        76000 non-null  float64
 31  Competitor Pricing_capped   76000 non-null  float64
 32  RETAIL SALES_capped         76000 non-null  float64
 33  price_capped                76000 non-null  float64
 34  discount_percentage_capped  76000 non-null  float64
 35  sales_revenue_capped        76000 non-null  float64
 36  year                        76000 non-null  int32
 37  month                       76000 non-null  int32
 38  day                         76000 non-null  int32
 39  dayofweek                   76000 non-null  int32
 40  dayofyear                   76000 non-null  int32
 41  weekofyear                  76000 non-null  int64
 42  quarter                     76000 non-null  int32
 43  demand_lag_7                76000 non-null  float64
 44  demand_lag_28               76000 non-null  float64
 45  demand_rolling_mean_7       76000 non-null  float64
 46  demand_rolling_mean_28      76000 non-null  float64
 47  Store ID_S001               76000 non-null  bool
 48  Store ID_S002               76000 non-null  bool
```

```
 49  Store ID_S003              76000 non-null  bool
 50  Store ID_S004              76000 non-null  bool
 51  Store ID_S005              76000 non-null  bool
 52  Product ID_P0001           76000 non-null  bool
 53  Product ID_P0002           76000 non-null  bool
 54  Product ID_P0003           76000 non-null  bool
 55  Product ID_P0004           76000 non-null  bool
 56  Product ID_P0005           76000 non-null  bool
 57  Product ID_P0006           76000 non-null  bool
 58  Product ID_P0007           76000 non-null  bool
 59  Product ID_P0008           76000 non-null  bool
 60  Product ID_P0009           76000 non-null  bool
 61  Product ID_P0010           76000 non-null  bool
 62  Product ID_P0011           76000 non-null  bool
 63  Product ID_P0012           76000 non-null  bool
 64  Product ID_P0013           76000 non-null  bool
 65  Product ID_P0014           76000 non-null  bool
 66  Product ID_P0015           76000 non-null  bool
 67  Product ID_P0016           76000 non-null  bool
 68  Product ID_P0017           76000 non-null  bool
 69  Product ID_P0018           76000 non-null  bool
 70  Product ID_P0019           76000 non-null  bool
 71  Product ID_P0020           76000 non-null  bool
 72  Category_Clothing          76000 non-null  bool
 73  Category_Electronics       76000 non-null  bool
 74  Category_Furniture         76000 non-null  bool
 75  Category_Groceries         76000 non-null  bool
 76  Category_Toys              76000 non-null  bool
 77  Region_East                76000 non-null  bool
 78  Region_North               76000 non-null  bool
 79  Region_South               76000 non-null  bool
 80  Region_West                76000 non-null  bool
 81  Weather Condition_Cloudy   76000 non-null  bool
 82  Weather Condition_Rainy    76000 non-null  bool
 83  Weather Condition_Snowy    76000 non-null  bool
 84  Weather Condition_Sunny    76000 non-null  bool
 85  Seasonality_Autumn         76000 non-null  bool
 86  Seasonality_Spring         76000 non-null  bool
 87  Seasonality_Summer         76000 non-null  bool
 88  Seasonality_Winter         76000 non-null  bool
 89  SUPPLIER_Unknown           76000 non-null  bool
 90  ITEM TYPE_Unknown          76000 non-null  bool
dtypes: bool(52), datetime64[ns](1), float64(28), int32(6), int64(4)
memory usage: 24.6 MB
```

5.3. Final Summary Statistics (Ringkasan Statistik Akhir)

Menampilkan statistik deskriptif untuk kolom numerik dan non-numerik di DataFrame akhir.

In [ ]:

```
print("\nSummary statistics of the final master_df:")
display(master_df.describe(include='all'))
```

```
Summary statistics of the final master_df:
```

	Date	Inventory Level	Units Sold	Units Ordered	Price	Discount	Promotion	Competitor Pricing	Epidemic
count	76000	7.600000e+04	7.600000e+04	7.600000e+04	7.600000e+04	7.600000e+04	7.600000e+04	7.600000e+04	7.600000e+04
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	2023-01-15 12:00:00	6.170503e-17	-6.544473e-17	-5.478658e-17	-1.767008e-17	-3.721000e-17	-1.311232e-16	-7.778573e-17	-3.590111e-17
min	2022-01-01 00:00:00	-1.329145e+00	-2.019067e+00	-5.485757e-01	-1.599538e+00	-1.215538e+00	-7.001400e-01	-1.591558e+00	-5.000000e-01
25%	2022-07-09 18:00:00	-7.287264e-01	-7.007126e-01	-5.485757e-01	-9.073303e-01	-5.467076e-01	-7.001400e-01	-8.996296e-01	-5.000000e-01
50%	2023-01-15 12:00:00	-3.269757e-01	-1.097261e-01	-5.485757e-01	-8.192536e-02	1.221232e-01	-7.001400e-01	-9.168793e-02	-5.000000e-01
75%	2023-07-24 06:00:00	4.721107e-01	5.721814e-01	1.964819e-01	7.137038e-01	1.221232e-01	1.428286e+00	6.955545e-01	-5.000000e-01
max	2024-01-30 00:00:00	8.679302e+00	7.664020e+00	9.401945e+00	4.070939e+00	2.128616e+00	1.428286e+00	4.683668e+00	2.000000e+00
std	NaN	1.000007e+00	1.000007e+00	1.000007e+00	1.000007e+00	1.000007e+00	1.000007e+00	1.000007e+00	1.000007e+00

11 rows × 91 columns

5.4. Final Missing Values and Duplicates Check (Pemeriksaan Akhir Nilai Hilang dan Duplikat)

Memastikan tidak ada lagi nilai yang hilang yang perlu ditangani dan tidak ada baris duplikat.

In [ ]:

```
print("\nMissing values in the final master_df:")
print(master_df.isnull().sum().to_markdown(numalign="left", stralign="left"))

print("\nDuplicate rows in the final master_df:")
print(master_df.duplicated().sum())
```

```
Missing values in the final master_df:
|                            | 0   |
|:---------------------------|:----|
| Date                       | 0   |
| Inventory Level            | 0   |
| Units Sold                 | 0   |
| Units Ordered              | 0   |
| Price                      | 0   |
| Discount                   | 0   |
| Promotion                  | 0   |
| Competitor Pricing         | 0   |
| Epidemic                   | 0   |
| Demand                     | 0   |
| RETAIL SALES               | 0   |
| holiday_season             | 0   |
| promotion_applied          | 0   |
| competitor_price_index     | 0   |
| economic_index             | 0   |
| weather_impact             | 0   |
| price                      | 0   |
| discount_percentage        | 0   |
| sales_revenue              | 0   |
| region_Europe              | 0   |
| region_North America       | 0   |
```

5.5. Saving the Prepared Dataset (Menyimpan Dataset yang Telah Disiapkan)

Menyimpan DataFrame yang telah diproses ke file CSV untuk penggunaan di masa mendatang atau untuk dibagikan.

```
# Save the prepared master_df to a CSV file
master_df.to_csv('master_df_prepared.csv', index=False)
print("\nPrepared master_df saved to 'master_df_prepared.csv'")
```

```
Prepared master_df saved to 'master_df_prepared.csv'
```

5.6. Potential Additional Features and Improvements (Potensi Fitur Tambahan dan Peningkatan)

Meskipun 10 langkah persiapan data telah mencakup proses yang komprehensif, ada beberapa area atau fitur tambahan yang dapat dieksplorasi untuk meningkatkan akurasi model peramalan permintaan:

- **Interaksi Fitur:** Buat fitur interaksi antara fitur kategorikal (seperti Kategori Produk atau ID Toko) dan fitur berbasis waktu (seperti bulan atau hari dalam seminggu). Ini dapat menangkap pola permintaan unik untuk produk/toko tertentu pada waktu-waktu tertentu.
    - *Contoh:* Interaksi antara `Category_Electronics` dan `month` untuk melihat apakah permintaan elektronik memiliki pola musiman yang kuat di bulan-bulan tertentu.
- **Fitur Kalender Lanjutan:** Tambahkan fitur yang lebih spesifik terkait kalender, seperti:
    - `days_until_next_holiday`: Jarak hari hingga hari libur besar berikutnya.
    - `days_since_last_holiday`: Jarak hari sejak hari libur besar terakhir.
    - `is_promo_week`: Indikator apakah minggu tersebut memiliki promosi.
- **Fitur Berbasis Toko dan Produk:**
    - Agregasi data di tingkat toko atau produk untuk membuat fitur seperti rata-rata penjualan historis per toko atau jumlah produk yang tersedia di toko tertentu.
    - Fitur demografi atau lokasi toko (jika data tersedia).
- **Analisis Sentimen Eksternal:** Jika memungkinkan, integrasikan data sentimen dari media sosial atau berita terkait produk atau industri ritel.
- **Kondisi Ekonomi yang Lebih Spesifik:** Jika 'economic_index' dari `df3` terlalu umum, coba cari atau buat indeks ekonomi yang lebih spesifik untuk wilayah atau segmen pasar ritel yang relevan.
- **Penanganan Missing Values dari Merge:** Seperti yang dibahas sebelumnya, investigasi lebih lanjut mengapa banyak nilai hilang setelah penggabungan dari `df2` dan `df3` sangat penting. Jika data tersebut memang relevan, strategi penggabungan atau imputasi yang lebih canggih mungkin diperlukan.
- **Normalisasi pada Fitur Hasil Capping:** Setelah melakukan capping, fitur-fitur tersebut mungkin masih memiliki rentang yang bervariasi. Melakukan penskalaan (seperti StandardScaler atau MinMaxScaler) pada fitur-fitur `_capped` ini sebelum dimasukkan ke model dapat bermanfaat.

Menambahkan fitur-fitur ini dapat membantu model menangkap pola permintaan yang lebih kompleks dan spesifik, berpotensi meningkatkan kinerja peramalan.

BAB VI: KESIMPULAN

Proyek *Retail Demand Forecasting: Pendekatan Pipeline Machine Learning* telah melalui serangkaian tahapan persiapan data yang terstruktur dan mendalam untuk memastikan kualitas data yang optimal sebelum tahap pemodelan dilakukan. Tahapan awal dimulai dengan proses penggabungan tiga sumber data utama, yaitu *Retail Store Inventory and Demand Forecasting* (data transaksional inti), *Retail Sales Data with Seasonal Trends & Marketing* (data pemasaran dan tren musiman), serta *Strategic Supply Chain Demand Forecasting Dataset* (data ekonomi makro dan kompetitif). Ketiga sumber data tersebut digabung secara sistematis untuk menciptakan satu *master dataset* yang komprehensif dan kaya informasi, mencakup faktor internal seperti harga, promosi, dan persediaan, serta faktor eksternal seperti kondisi ekonomi, musim, dan aktivitas kompetitor.

Tahap berikutnya adalah pembersihan data (*data cleaning*) untuk mengatasi permasalahan nilai hilang, duplikasi, dan inkonsistensi tipe data yang ditemukan setelah proses penggabungan. Setiap kolom dianalisis secara cermat untuk menentukan strategi penanganan yang sesuai, baik dengan mengisi nilai hilang menggunakan median, nol, atau kategori “Unknown”, maupun dengan memastikan semua kolom memiliki tipe data yang konsisten. Selain itu, potensi *data leakage* juga diantisipasi dengan menghapus kolom yang mengandung informasi masa depan seperti *future_demand*.

Selanjutnya dilakukan proses *feature engineering* yang terbagi menjadi beberapa tahap utama. Tahap pertama menambahkan fitur berbasis waktu seperti tahun, bulan, hari, kuartal, dan hari dalam seminggu untuk membantu model mengenali pola musiman dan tren permintaan. Tahap kedua membuat *lag features* untuk menangkap pengaruh nilai permintaan dari periode sebelumnya, sedangkan tahap ketiga menciptakan *rolling window features* untuk menghitung rata-rata permintaan dalam rentang waktu tertentu, sehingga pola jangka pendek dapat terdeteksi dengan lebih baik.

Setelah fitur baru berhasil dibentuk, dilakukan transformasi pada variabel kategorikal menggunakan *One-Hot Encoding* agar dapat dibaca oleh algoritma machine learning. Kemudian, semua fitur numerik dinormalisasi menggunakan *StandardScaler* untuk memastikan tidak ada fitur yang mendominasi karena perbedaan skala. Langkah-langkah ini dilakukan secara hati-hati agar model nantinya dapat beroperasi secara efisien dan adil terhadap seluruh variabel input.

Tahapan terakhir dari pipeline persiapan data adalah pembagian dataset berdasarkan waktu (*time-based splitting*). Pembagian dilakukan secara kronologis untuk memisahkan data pelatihan (training set) dan data validasi (validation set), sehingga model nantinya hanya belajar dari data masa lalu dan diuji menggunakan data masa depan. Metode ini mencegah *data leakage* temporal dan memastikan keakuratan evaluasi performa model dalam konteks peramalan deret waktu.

Melalui sepuluh langkah pipeline yang komprehensif ini mulai dari penggabungan, pembersihan, rekayasa fitur, transformasi, penskalaan, hingga pembagian data diperoleh dataset akhir yang bersih, terstruktur, dan siap digunakan. Dataset ini tidak hanya merepresentasikan kondisi pasar ritel dengan lebih realistis, tetapi juga memiliki kekayaan informasi yang cukup untuk mendukung model *demand forecasting* yang lebih akurat, adaptif, dan andal. Dengan demikian, hasil persiapan data ini menjadi fondasi yang kuat bagi tahap selanjutnya, yaitu pemodelan dan evaluasi performa sistem peramalan permintaan produk ritel.