Tugas Mata Kuliah Teknik Pengembangan Model

Retail Demand Forecasting: Pendekatan Pipeline Machine Learning dan Deployment dengan Docker

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 **5**

1.1 Latar Belakang Masalah 5

1.2 Tujuan Proyek 6

1.3 Ruang Lingkup Proyek 6

BAB II Pemahaman Data Awal (Initial Data Understanding) **8**

2.1 Tentang Dataset 8

2.1.1 Retail Store Inventory and Demand Forecasting (Data Transaksional Inti) 8

2.1.2 Retail Sales Data with Seasonal Trends & Marketing (Data Konteks Pemasaran) 10

2.1.3. Strategic Supply Chain Demand Forecasting Dataset (Data Ekonomi & Kompetitif Eksternal) 11

2.2. Import Libraries 13

2.3. Load Datasets 14

2.3.1. Data Information 15

2.3.2. Data Summary Statistics 17

2.2.3. Unique Values and Counts for Object Columns 18

2.2.4. Missing Values Analysis 20

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 Metodologi: Pipeline Persiapan Data 10 Langkah (Methodology: 10-Step Data Preparation Pipeline) **26**

Step 1: Data Cleaning and Type Conversion (Pembersihan Data dan Konversi Tipe) 26

Step 2: Column Consolidation and Selection (Konsolidasi dan Pemilihan Kolom) 33

Step 3: Exploratory Data Analysis (EDA) 35

3.3.1. Time Series Analysis: Plot daily average Demand 36

3.3.2. Target Variable Distribution Analysis 36

3.3.3. Relationship between Key Features and Demand 37

3.3.4. Time Series Analysis by Category (Example) 38

3.3.5. Distribution Analysis Using Histograms for numerical features 39

3.3.6. Count Plots for Categorical Features 40

3.8.7. Correlation Analysis for Numerical Features 43

Step 4: Outlier Detection and Treatment (Deteksi dan Perlakuan Outlier) 46
Step 5: Feature Engineering I - Time-Based Features (Rekayasa Fitur I - Fitur Berbasis Waktu)..... 50
Step 6: Feature Engineering II - Lag Features (Rekayasa Fitur II - Fitur Lag 52
Step 7: Feature Engineering III - Rolling Window Features (Rekayasa Fitur III - Fitur Jendela Bergulir) 53
Step 8: Categorical Feature Encoding (Pengkodean Fitur Kategorikal) 57
Step 9: Feature Scaling (Penskalaan Fitur) 59
Step 10: Time-Based Data Splitting (Pembagian Data Berbasis Waktu)..... 62
BAB V Pembangunan dan Evaluasi Model Prediktif 65
5.1. State of the Art Research Table (Tabel Riset Terkini) 65
5.2. Analisis Literatur dan Model yang Dipilih 69
5.2.1. Linear Regression: Pijakan Dasar (Baseline) 69
5.2.2. Decision Tree Regressor: Model Intuitif Berbasis Aturan..... 70
5.2.3. Random Forest Regressor: Kekuatan Kolektif (Ensemble)..... 70
5.2.4. XGBoost Regressor: Kuda Pacu Gradient Boosting..... 70
5.2.5. LightGBM Regressor: Efisiensi Gradient Boosting 71
5.3. Import Modeling Libraries (Impor Library Pemodelan) 71
5.4. Model Training (Pelatihan Model) 73
5.4.1. Persiapan Data Akhir untuk Pemodelan..... 73
5.4.2. Pembagian Data Berbasis Waktu (Time-Based Split)..... 74
5.5. Proses Pelatihan Model..... 76
5.5.1. Linear Regression 76
5.5.2. Decision Tree Regressor..... 76
5.5.3. Random Forest Regressor 77
5.5.4. XGBoost Regressor 77
5.4.5. LightGBM Regressor..... 77
5.6. Model Evaluation and Comparison (Evaluasi dan Perbandingan Model) 79
5.6.1. Membuat Prediksi pada Data Uji 79
5.6.2. Perbandingan Metrik Evaluasi..... 80
5.6.3. Analisis Mendalam Metrik Evaluasi Model..... 82
5.7. Visualisasi Perbandingan Performa 85
5.7.1. Analisis Mendalam Visualisasi Perbandingan Performa Model 86
5.8. Deep Dive on the Best Model (Analisis Mendalam pada Model Terbaik) 88
5.8.1. Analisis Kontribusi Fitur (Feature Importance)..... 88
5.8.2. Interpretasi Hasil Feature Importance 90

BAB VI Deployment Model dalam Kontainer Docker 91
6.1. Pengantar Deployment dan Teknologi Kontainerisasi 91
6.1.1. Mengapa Deployment Penting? Dari Model ke Produk Nyata 91
6.1.2. Paradigma Modern: Kontainerisasi dengan Docker 91
6.1.3. Arsitektur Solusi Deployment Kita 92
6.2. Persiapan Artefak untuk Deployment 93
6.2.1. Folder Assets 94
6.2.2. Folder App 104
6.2.3. Dockerfile 114
6.2.4. input.json 118
6.2.5. requirements.txt 121
6.3. Membangun dan Menjalankan Kontainer Docker (Implementasi di Lingkungan Lokal) 125
6.3.1. Langkah 1: Membangun Image Docker (docker build) 125
6.3.2. Langkah 2: Menjalankan Kontainer (docker run) 126
6.3.3. Langkah 3: Verifikasi dan Debugging 126
6.4. Langkah-langkah Akses Hasil Docker pada Device Lain 127
6.4.1. Persiapan Lingkungan Lokal 127
6.4.2. Pengambilan Image Docker dan Peluncuran Kontainer API 127
6.4.3. Pengujian Endpoint dan Verifikasi Log 128
6.5. Distribusi dan Aplikasi di Lingkungan Lokal dan Device Lain (Bukti Portabilitas) 130
6.5.1. Laptop 1 130
6.5.2. Laptop 2 131
6.5.3. Laptop 3 132
BAB VI Kesimpulan 133

Bab I Pendahuluan

1.1 Latar Belakang Masalah

Industri ritel merupakan salah satu sektor ekonomi yang sangat dinamis, di mana kemampuan untuk memahami perilaku konsumen dan mengelola persediaan barang secara efektif menjadi kunci utama keberhasilan. Dalam persaingan pasar yang semakin ketat, kemampuan perusahaan untuk meramalkan permintaan pelanggan (demand forecasting) secara akurat telah menjadi faktor penentu dalam menjaga keseimbangan vital antara ketersediaan produk dan efisiensi biaya operasional.

Kesalahan dalam peramalan permintaan dapat memicu dua masalah bisnis yang merugikan: kelebihan stok (overstock) dan kekurangan stok (stockout). *Overstock* menyebabkan membengkaknya biaya penyimpanan, meningkatkan risiko kerusakan atau kedaluwarsa produk, dan pada akhirnya menggerus profitabilitas. Sebaliknya, *stockout* mengakibatkan hilangnya peluang penjualan, menurunnya loyalitas pelanggan, dan memberikan dampak negatif jangka panjang pada reputasi merek. Kedua skenario ekstrem ini menegaskan urgensi adanya sistem peramalan yang tidak hanya akurat, tetapi juga adaptif.

Di era digital saat ini, tantangan tersebut dijawab dengan ketersediaan data dalam volume masif. Namun, data mentah yang berasal dari berbagai sumber seperti transaksi penjualan, data promosi, dan faktor eksternal seringkali tidak lengkap, tidak konsisten, dan tidak terstruktur. Untuk mengubah data ini menjadi wawasan yang dapat ditindaklanjuti, pendekatan berbasis Machine Learning (ML) menawarkan solusi modern yang mampu mengenali pola-pola non-linear yang kompleks. Namun, penerapan ML di dunia nyata menghadapi dua tantangan utama:

1. **Kualitas dan Persiapan Data:** Diperlukan sebuah pipeline Machine Learning yang sistematis untuk menggabungkan, membersihkan, dan merekayasa data agar model yang dibangun memiliki fondasi yang kuat.
2. **Jurang antara Pengembangan dan Produksi (Deployment Gap):** Sebuah model yang berkinerja tinggi di lingkungan pengembangan (seperti notebook Jupyter) seringkali gagal saat dipindahkan ke lingkungan produksi karena perbedaan dependensi dan konfigurasi. Masalah klasik "works on my machine" ini menjadi penghalang utama implementasi ML.

Menyadari kedua tantangan tersebut, proyek ini tidak hanya berfokus pada pembangunan model, tetapi mengadopsi pendekatan holistik dari hulu ke hilir. Proyek ini akan mendemonstrasikan proses lengkap, mulai dari membangun pipeline data yang tangguh, melatih dan mengevaluasi berbagai model ML berdasarkan studi literatur, hingga men-deploy model terbaik menggunakan teknologi kontainerisasi Docker untuk menciptakan sebuah layanan prediksi yang portabel, andal, dan siap untuk aplikasi dunia nyata.

1.2 Tujuan Proyek

Tujuan utama dari proyek **"Retail Demand Forecasting: Pendekatan Pipeline Machine Learning dan Deploy dengan Docker"** ini adalah untuk merancang, membangun, dan mengaplikasikan sebuah sistem peramalan permintaan end-to-end yang fungsional dan dapat direproduksi.

Secara lebih rinci, tujuan proyek ini meliputi:

1. **Membangun Pipeline Data 10 Tahap yang Komprehensif:** Mengintegrasikan tiga sumber data yang berbeda, melakukan pembersihan data secara sistematis, dan menerapkan rekayasa fitur (feature engineering) tingkat lanjut untuk menghasilkan dataset akhir yang bersih, terstruktur, dan kaya informasi.
2. **Melakukan Pemilihan, Pelatihan, dan Evaluasi Model Berbasis Literatur:** Melakukan tinjauan *state-of-the-art* untuk memilih serangkaian model machine learning yang relevan, melatihnya pada data yang telah disiapkan, dan mengevaluasi performa masing-masing model secara kuantitatif untuk mengidentifikasi model dengan akurasi prediksi tertinggi.
3. **Mengimplementasikan Deployment Model sebagai Layanan API:** Membungkus (mengenkapsulasi) pipeline preprocessing dan model terbaik ke dalam sebuah aplikasi web (API) yang fungsional menggunakan framework Flask.
4. **Mengaplikasikan Kontainerisasi untuk Portabilitas:** Menggunakan Docker untuk mengemas aplikasi API beserta seluruh dependensinya ke dalam sebuah *image* yang portabel, memastikan konsistensi dan kemudahan distribusi, serta membuktikan bahwa layanan prediksi dapat dijalankan di lingkungan yang berbeda.

1.3 Ruang Lingkup Proyek

Untuk mencapai tujuan-tujuan di atas, proyek ini mencakup empat fase utama yang saling berhubungan, merefleksikan siklus hidup penuh dari sebuah proyek machine learning.

Cakupan pekerjaan proyek ini meliputi:

- **Fase 1: Pipeline Persiapan Data**
    - Integrasi tiga dataset utama: data transaksional, data pemasaran, dan data ekonomi.
    - Implementasi pipeline 10 langkah yang mencakup: pembersihan data, penanganan nilai hilang, analisis data eksploratif (EDA), rekayasa fitur (fitur berbasis waktu, lag, dan *rolling window*), *outlier capping*, pengkodean fitur kategorikal, dan penskalaan fitur numerik.
- **Fase 2: Pemodelan dan Evaluasi**
    - Tinjauan literatur (*state-of-the-art*) untuk menjustifikasi pemilihan model.
    - Pelatihan lima model regresi yang berbeda: Linear Regression, Decision Tree, Random Forest, XGBoost, dan LightGBM.
    - Evaluasi komparatif menggunakan metrik standar (MAE, MSE, RMSE, R^2) untuk menentukan model dengan performa terbaik.
    - Analisis mendalam pada model terbaik, termasuk analisis kontribusi fitur (*feature importance*).
- **Fase 3: Deployment dengan Docker**
    - Pengembangan aplikasi API menggunakan Flask dan Gunicorn dengan struktur kode yang modular dan profesional.
    - Serialisasi artefak model dan preprocessing menggunakan joblib.
    - Pembuatan Dockerfile untuk mengemas aplikasi dan seluruh dependensinya.
    - Membangun *Docker image* dan menjalankan *container* di lingkungan lokal.
- **Fase 4: Verifikasi dan Distribusi**
    - Pengujian fungsionalitas *endpoint* API secara lokal menggunakan curl untuk mengirim data dan menerima prediksi.
    - Mendistribusikan *Docker image* ke registri publik (Docker Hub).
    - Mendemonstrasikan dan mendokumentasikan bukti portabilitas dengan berhasil menjalankan kontainer dan melakukan prediksi pada beberapa perangkat komputer yang berbeda.

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:

2.1.1 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.

2.1.2 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.

2.1.3. 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.

Kode Program untuk Melihat Isi Dataset dan Penjelasan

2.2. Import Libraries

[1]
3s

```
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.

2.2.3. 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.

2.2.4. 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`.

```
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 IV Metodologi: Pipeline Persiapan Data 10 Langkah (Methodology: 10-Step Data Preparation Pipeline)

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.

Step 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	...	sales_revenue	region_Europe	region_North America	store_type_Retail	store_type_Wholesale	category_Cabinets	category_Chairs	category_Sofas	category_Tables	future_demand
0	2022-01-01	S001	P0001	Electronics	North	195	102	252	72.72	5	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2022-01-01	S001	P0002	Clothing	North	117	117	249	80.16	15	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2022-01-01	S001	P0003	Clothing	North	247	114	612	62.94	10	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2022-01-01	S001	P0004	Electronics	North	139	45	102	87.63	10	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	2022-01-01	S001	P0005	Groceries	North	152	65	271	54.41	0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

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.

Step 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.

Step 3: Exploratory Data Analysis (EDA)

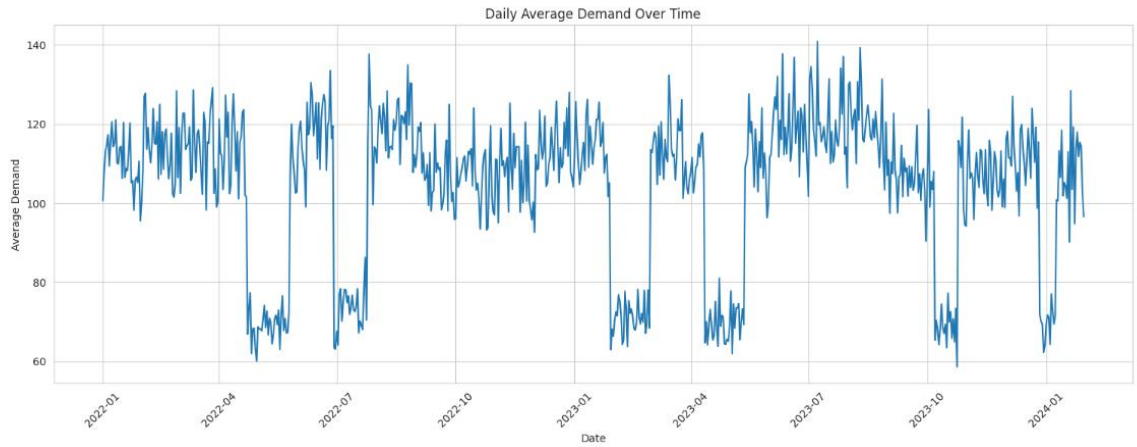
Langkah ketiga dalam pipeline persiapan data ini adalah Exploratory Data Analysis (EDA). Tahap ini sangat krusial untuk mendapatkan pemahaman mendalam tentang karakteristik data, mengidentifikasi pola, tren, anomali, dan hubungan antar variabel sebelum proses pemodelan. Visualisasi data memainkan peran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

3.3.1. 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()
```

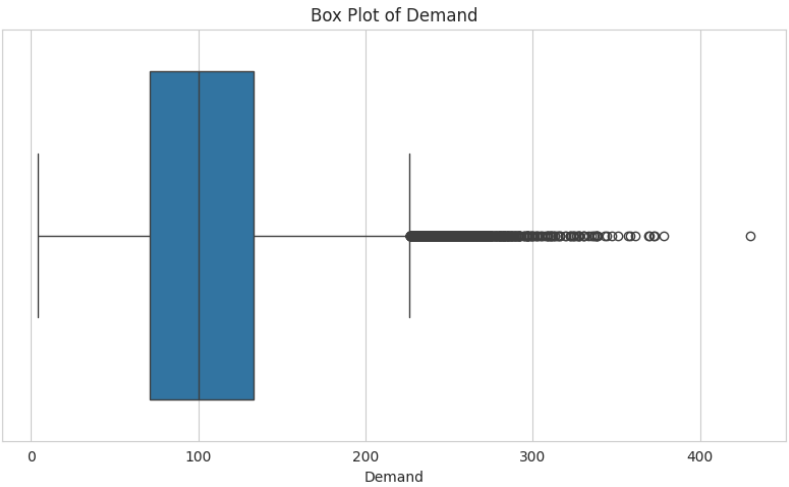


3.3.2. Target Variable Distribution Analysis

```
# 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:
```

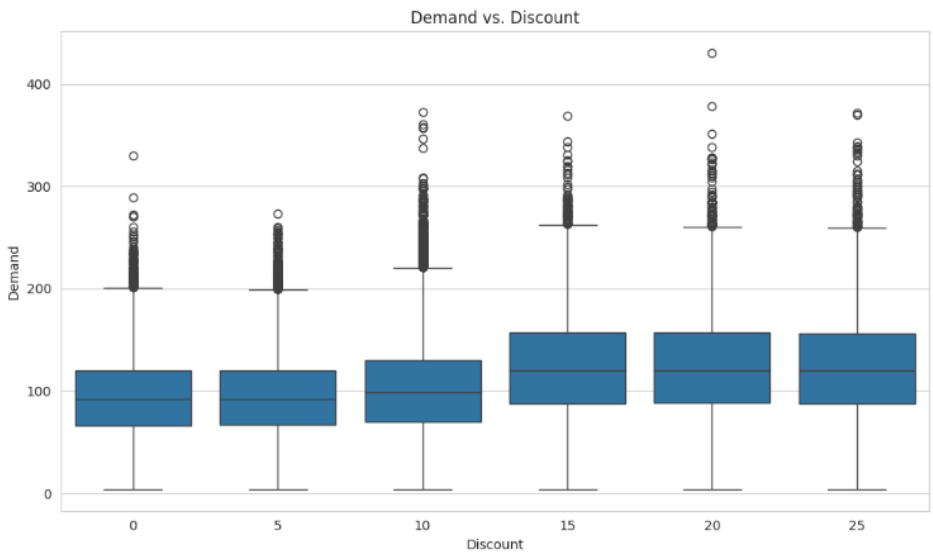
	Demand
count	76000.000000
mean	104.317158
std	46.964801
min	4.000000
25%	71.000000
50%	100.000000
75%	133.000000
max	430.000000



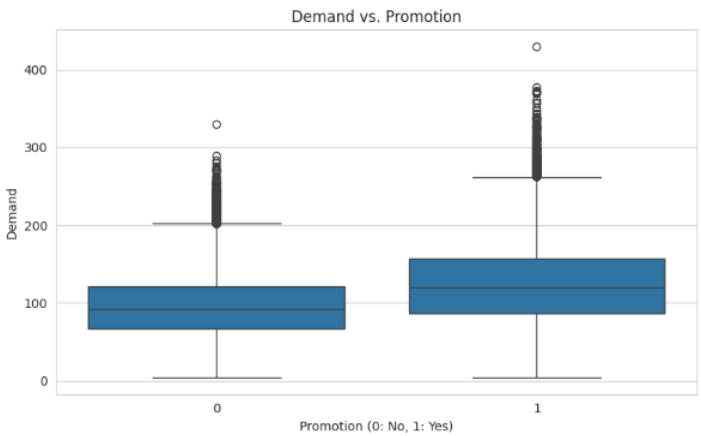
3.3.3. Relationship between Key Features and Demand

```
# 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()
```

```python
# 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()
```



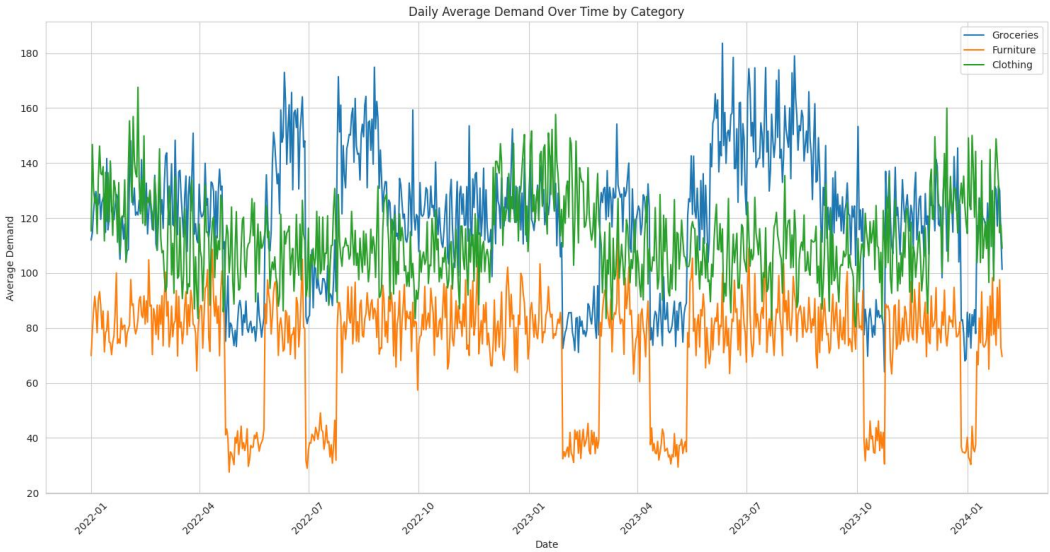

```python
# Box plot: Promotion vs. Demand (assuming Promotion is binary or has few categories)
plt.figure(figsize=(8, 5))
sns.boxplot(data=master_df, x='Promotion', y='Demand')
plt.title('Demand vs. Promotion')
plt.xlabel('Promotion (0: No, 1: Yes)')
plt.ylabel('Demand')
plt.tight_layout()
plt.show()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




```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()
```

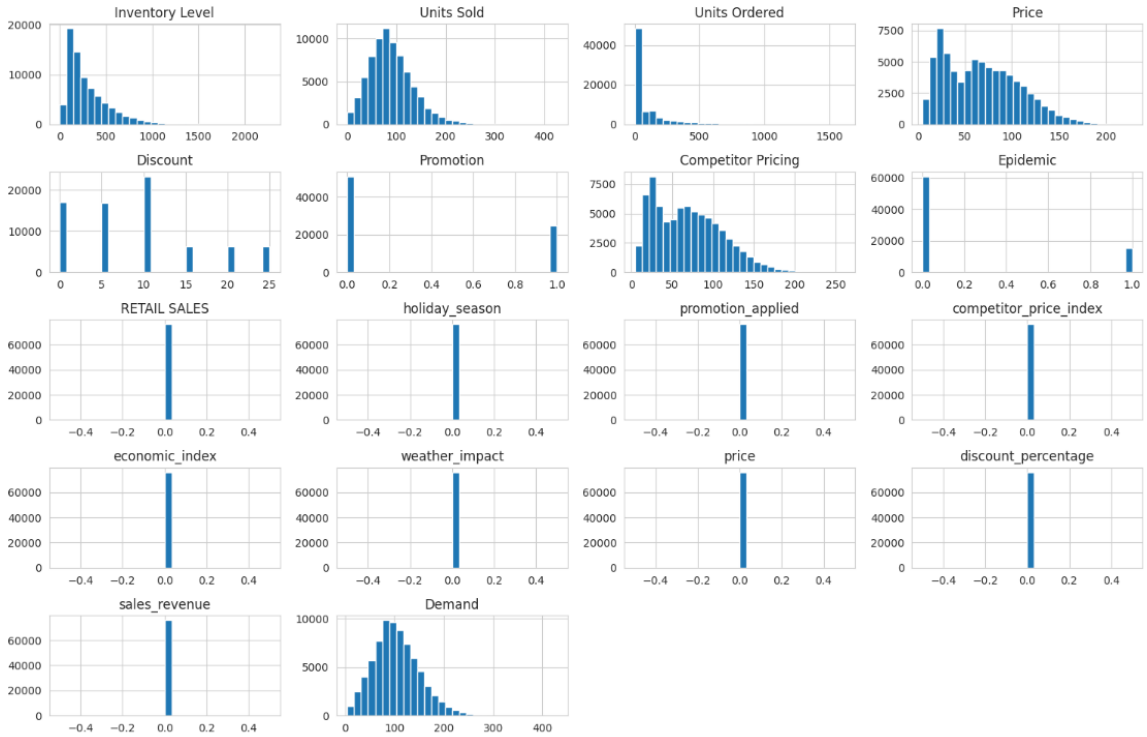
3.3.4. Time Series Analysis by Category (Example)

```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()
```



3.3.5. Distribution Analysis Using Histograms for numerical features

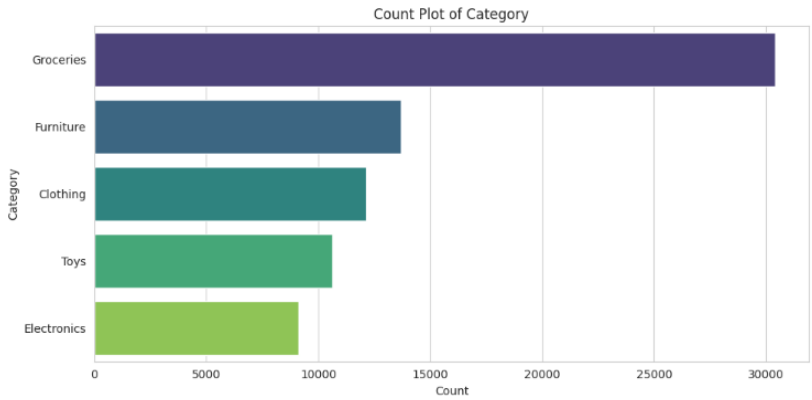
```python
# Define numerical features more explicitly
numerical_features = ['Inventory Level', 'Units Sold', 'Units Ordered', 'Price', 'Discount',
                      'Promotion', 'Competitor Pricing', 'Epidemic', 'RETAIL SALES',
                      'holiday_season', 'promotion_applied', 'competitor_price_index',
                      '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()
```



3.3.6. Count Plots for Categorical Features

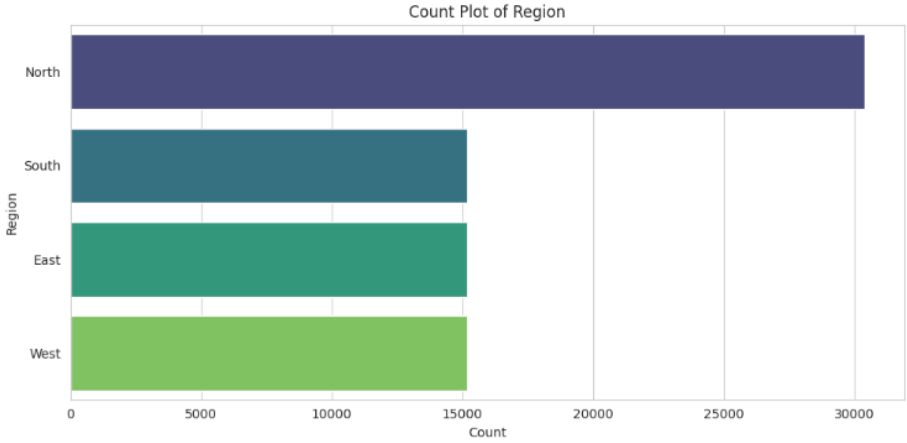
```
# Count plots for categorical features
# Define categorical features more explicitly
categorical_features = ['Category', 'Region', 'Store ID', 'Weather Condition', 'Seasonality', 'SUPPLIER', 'ITEM TYPE'] # Added SUPPLIER and ITEM TYPE

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')
```

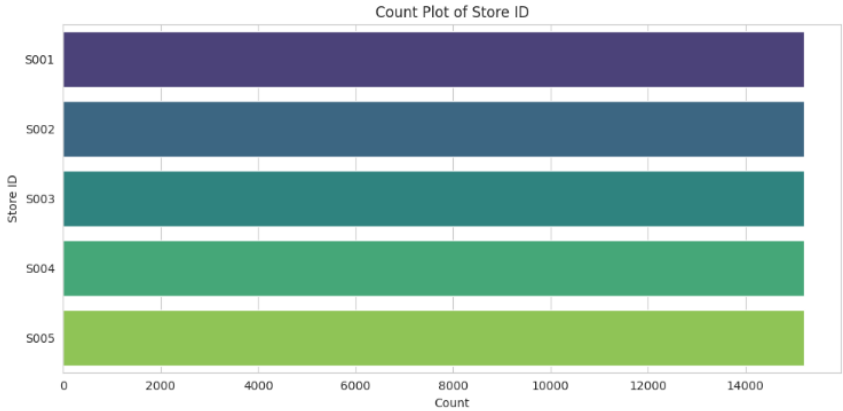



```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

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')
```

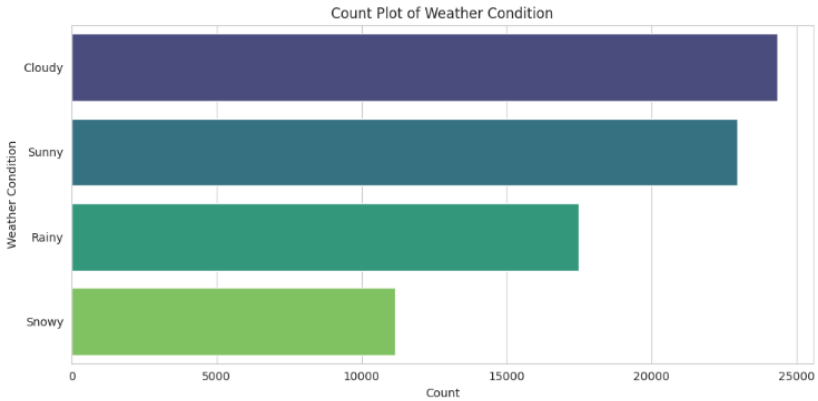
```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

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')
```



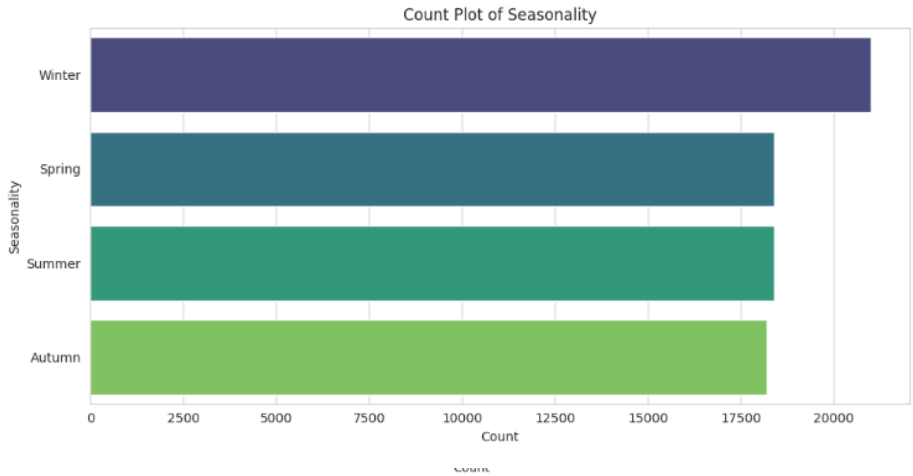

```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

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')
```



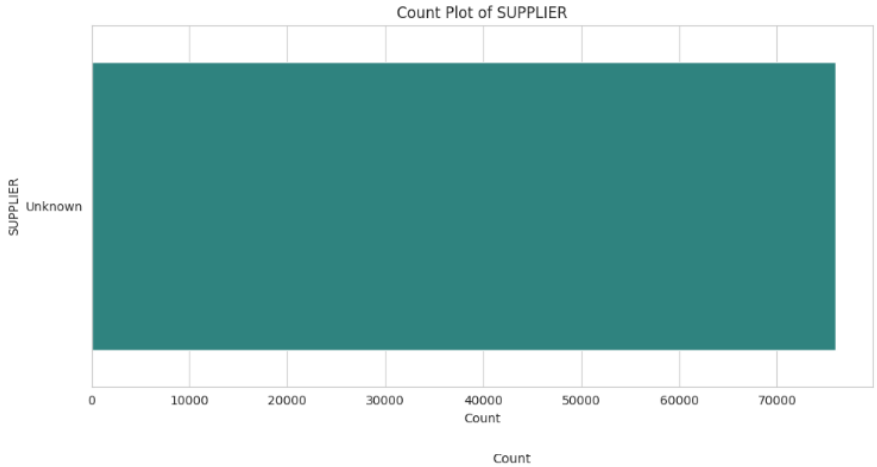

```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

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')
```



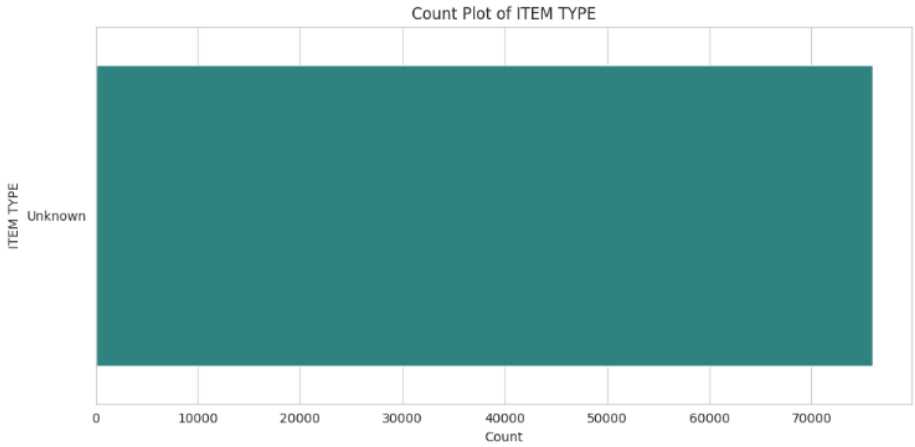
```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

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')
```

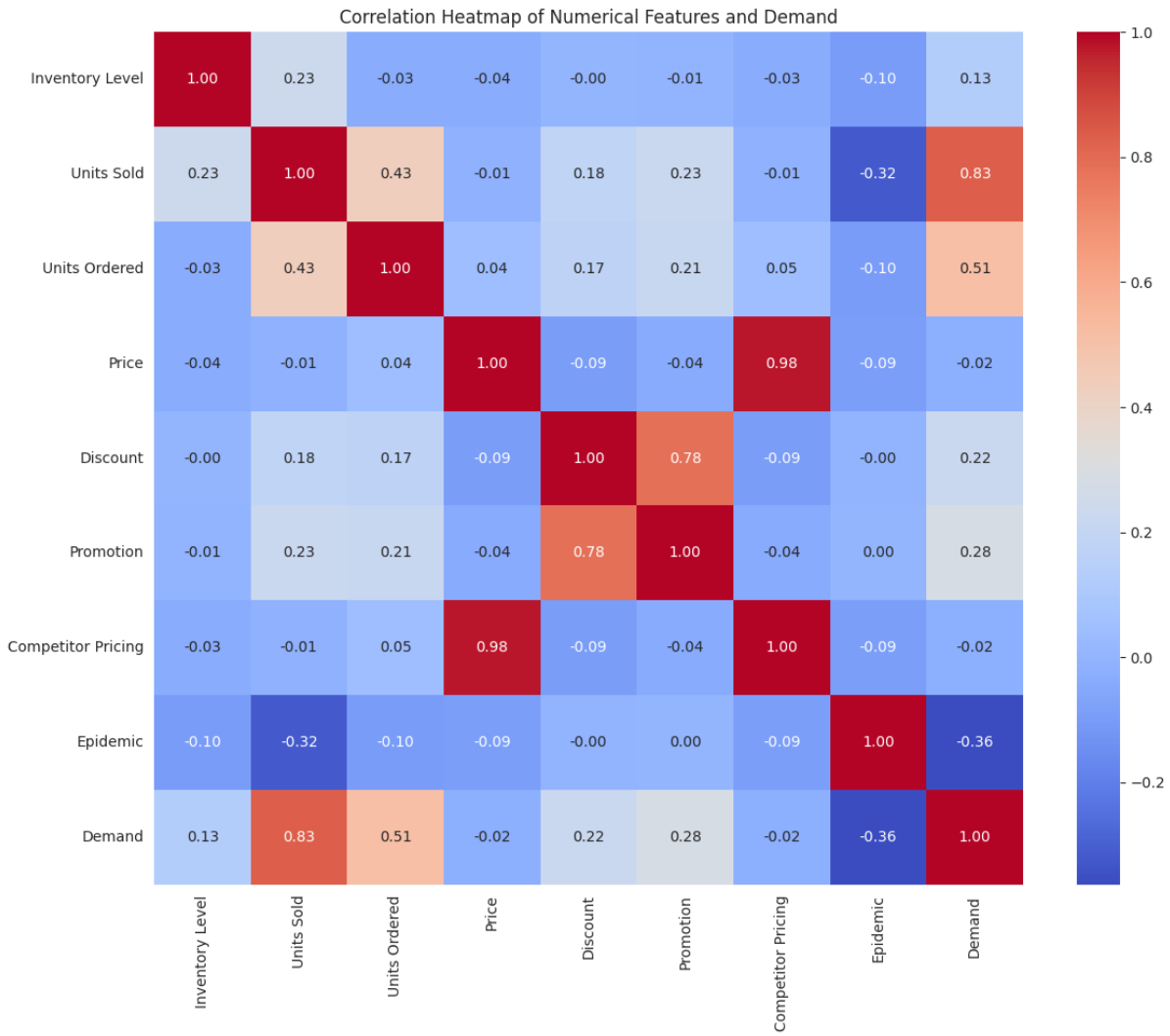



```
/tmp/ipython-input-1099832817.py:8: FutureWarning:

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')
```



3.8.7. 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.

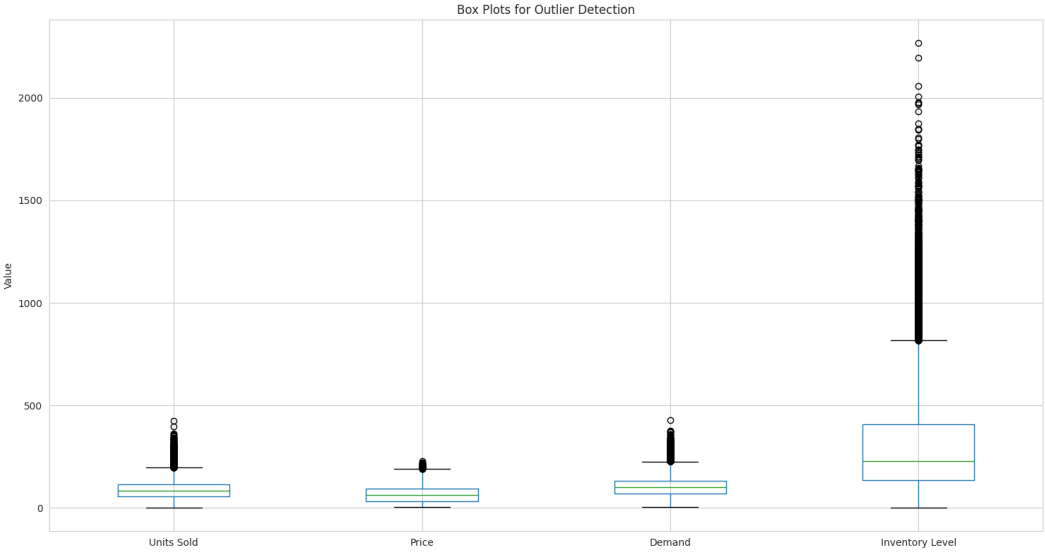
Step 4: Outlier Detection and Treatment (Deteksi dan Perlakuan Outlier)

Langkah keempat dalam pipeline persiapan data ini berokus 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.

Step 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.

Step 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.

Step 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.

Step 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.

Step 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	-1.215536	-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.

Step 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.

```
# 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 Pembangunan dan Evaluasi Model Prediktif

Di bab ini, kami akan mendasarkan pemilihan model kami pada tinjauan penelitian terkini di bidang peramalan permintaan menggunakan machine learning. Pendekatan ini memastikan bahwa metode yang kita pilih relevan, teruji, dan memiliki dasar ilmiah yang kuat.

5.1. State of the Art Research Table (Tabel Riset Terkini)

Tabel berikut merangkum berbagai penelitian relevan yang telah dianalisis untuk memandu pemilihan model dalam proyek ini. Berikut adalah tabelnya:

NO	Judul	Penulis	Sumber	Metode	Hasil
1	Demand Forecasting Based on Machine Learning to Determine Order Quantity: A Case Study of Bahagia Kopi Bandung	Farel Nanda Rosya	COSTING: Journal of Economic, Business and Accounting (2024)	**Model:** MLR, Decision Tree (DT), SVR, Neural Network (NN). **Data:** Internal (POS) dan Eksternal (cuaca, hari libur).	Model **Decision Tree (DT)** dengan semua variabel memberikan hasil terbaik berdasarkan error terkecil dan pola prediksi yang paling mendekati data aktual.

2	Machine Learning in Demand Forecasting	Trang T. M. Ho, et al.	International Research Journal of Advanced Engineering and Science (2022)	**Model:** Hybrid Support Vector Machine (SVM) yang dikombinasikan dengan Holt-Winters dan ARIMA.	Model **Hybrid SVM-ARIMA** menunjukkan performa paling unggul, membuktikan bahwa metode *machine learning* (khususnya model hybrid) memberikan akurasi yang lebih baik.
3	Demand Forecasting with Machine Learning	Chi Sheng Tseng, Turgay Turkmen	Tesis Master, Massachusetts Institute of Technology (2024)	**Model:** Prophet, XGBoost dibandingkan dengan model tradisional (MA, Holt-Winter, ARIMA).	Model **Prophet** secara signifikan mengungguli semua model lain. Setelah *hyperparameter tuning*, model ini mencapai MAPE sebesar **7.05%**, terbukti mampu menangkap tren dan musiman.
4	Demand Forecasting for a Large Grocery Chain in Ecuador	Varsha Prabhakar, et al.	Makalah Proyek, Purdue University	**Model:** Regresi Linier, Gradient Boosting, Neural Network (NN), Support Vector Machine (SVM).	Model **Gradient Boosting** menunjukkan performa terbaik berdasarkan nilai RMSE. Model ini dinilai memiliki keseimbangan yang baik antara bias dan varians.

5	A Machine Learning-Based Approach for Retail Demand Forecasting	Ni Putu Erica Puspita Adriani, et al.	Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI) (2025)	**Model:** Random Forest (RF), Decision Tree (DT), SVM. **Inovasi:** Menggunakan fitur rekayasa 'Spending Score' dan *hyperparameter tuning*.	Setelah optimisasi, **Random Forest** menjadi model terbaik. Penggunaan *Spending Score* terbukti menjadi fitur paling berpengaruh yang signifikan meningkatkan akurasi.
6	Demand forecast models for online supermarkets	J.M. Evers, et al.	Prosiding Konferensi, Molde University (2018)	**Model:** Decision Regression Trees (DT) dan Random Forest Regression (RF) untuk peramalan per pelanggan.	*Tree-based models* memiliki akurasi sangat tinggi (99.9%). **Random Forest** lebih disukai karena kecenderunganny a untuk *overestimate*, yang lebih aman untuk menghindari kekurangan stok.
7	Application of Machine Learning in Supply Chain Management	Erfan Babaee Tirkolaee, et al.	Mathematical Problems in Engineering (2021)	**Jenis Penelitian:** Ulasan Literatur Komprehensif (*Review Paper*) yang membahas berbagai model ML.	Menyimpulkan bahwa teknik ML jauh lebih unggul daripada metode tradisional. Untuk peramalan permintaan, ML terbukti

					meningkatkan akurasi, yang berujung pada optimisasi inventaris.
8	A Multi-Model Approach to Supermarket Sales Forecasting System	Hammaad Rizwan, et al.	Makalah Penelitian (Preprint)	**Model:** Pendekatan *stacking ensemble* yang menggabungkan ARIMA, Random Forest, dan XGBoost.	Pendekatan multi-model yang mengintegrasikan data eksternal (cuaca, ekonomi) memberikan akurasi yang lebih tinggi. **XGBoost** digunakan sebagai model penjualan utama.
9	Sales Prediction using Machine Learning	Jain Harshit, et al.	Preprint SSRN (Social Science Research Network)	**Jenis Penelitian:** Ulasan Literatur yang membahas model Hybrid (ARIMA-ANN) dan Deep Learning (LSTM).	Ulasan ini menyimpulkan bahwa model ML, terutama *hybrid* dan *deep learning*, secara konsisten memberikan prediksi yang lebih akurat dibandingkan metode statistik tradisional.

10	Demand Forecasting in Retail Business Using the Ensemble Machine Learning Framework	Chukwuka Izuchukwu C. Obi	American Academic Scientific Research Journal (2024)	**Model:** Stacking Ensemble (RF + LightGBM + Extra Trees) dibandingkan dengan Multilayer Perceptron (MLP).	Model **Stacking Ensemble** secara signifikan mengungguli MLP dengan akurasi **99.98%**, menunjukkan efektivitas menggabungkan beberapa model ML.

5.2. Analisis Literatur dan Model yang Dipilih

Berdasarkan tinjauan mendalam terhadap tabel *State of the Art* di atas, terlihat jelas bahwa beberapa keluarga model machine learning secara konsisten menunjukkan performa unggul untuk kasus peramalan permintaan. Keputusan untuk memilih serangkaian model spesifik dalam proyek ini bukanlah tanpa dasar; setiap model dipilih untuk mewakili pendekatan yang berbeda, dari yang paling sederhana hingga yang paling canggih, yang semuanya didukung oleh temuan-temuan dalam penelitian sebelumnya. Pendekatan komparatif ini memungkinkan kami untuk tidak hanya menemukan model terbaik untuk dataset kami, tetapi juga untuk memahami spektrum solusi yang ada untuk masalah ini.

Berikut adalah justifikasi rinci untuk pemilihan masing-masing dari lima model yang akan diuji:

5.2.1. Linear Regression: Pijakan Dasar (Baseline)

- **Peran & Karakteristik:** Linear Regression adalah model statistik paling fundamental untuk tugas regresi. Meskipun seringkali paling sederhana, perannya dalam proyek ini sangat krusial, yaitu sebagai **pijakan dasar (baseline)**. Performa dari setiap model yang lebih kompleks akan diukur dari kemampuannya untuk mengungguli baseline ini secara signifikan. Jika sebuah model canggih gagal mengalahkan Linear Regression, maka kompleksitasnya tidak memberikan nilai tambah.
- **Dukungan Literatur:** Penggunaan Regresi Linier sebagai titik awal divalidasi oleh penelitian **Prabhaka, et al.** dalam studi mereka tentang peramalan permintaan di rantai grosir besar. Meskipun pada akhirnya Gradient Boosting menjadi model

terbaik, inklusi Regresi Linier menunjukkan bahwa ini adalah praktik standar untuk memulai analisis komparatif dalam penelitian peramalan.

5.2.2. Decision Tree Regressor: Model Intuitif Berbasis Aturan

- **Peran & Karakteristik:** Decision Tree adalah model non-parametrik yang memprediksi nilai target dengan mempelajari serangkaian aturan keputusan sederhana (jika-maka) yang disimpulkan dari fitur data. Keunggulan utamanya adalah **interpretabilitas**; hasilnya mudah dipahami dan dijelaskan dari sudut pandang bisnis.
- **Dukungan Literatur:** Pemilihan model ini didukung kuat oleh penelitian **Rosya (2024)** pada kasus "Bahagia Kopi Bandung", di mana **Decision Tree (DT) terbukti memberikan hasil peramalan terbaik** di antara beberapa model lain yang diuji. Selain itu, **Adriani, et al. (2025)** dan **Evers, et al. (2018)** juga menyertakan Decision Tree dalam analisis mereka, menegaskan relevansinya sebagai model tunggal yang kuat dan populer untuk tugas-tugas berbasis pohon keputusan.

5.2.3. Random Forest Regressor: Kekuatan Kolektif (Ensemble)

- **Peran & Karakteristik:** Random Forest adalah langkah evolusioner dari Decision Tree tunggal. Sebagai model *ensemble* (khususnya *bagging*), ia membangun banyak Decision Tree secara independen dan mengambil rata-rata dari prediksi mereka. Ini secara efektif mengurangi *overfitting* yang sering menjadi kelemahan Decision Tree tunggal dan umumnya menghasilkan akurasi yang lebih tinggi dan stabil.
- **Dukungan Literatur:** Beberapa penelitian menyoroti keandalan Random Forest. **Adriani, et al. (2025)** menemukan bahwa setelah melalui proses optimisasi dan rekayasa fitur, **Random Forest menjadi model terbaik** dalam studi kasus ritel mereka. Lebih menarik lagi, **Evers, et al. (2018)** secara spesifik mencatat bahwa meskipun memiliki akurasi sangat tinggi, Random Forest lebih disukai karena kecenderungannya untuk sedikit *overestimate*, yang dari perspektif manajemen inventaris, **lebih aman karena dapat membantu menghindari kekurangan stok (stockouts)**.

5.2.4. XGBoost Regressor: Kuda Pacu Gradient Boosting

- **Peran & Karakteristik:** XGBoost (Extreme Gradient Boosting) adalah implementasi yang sangat optimal dari algoritma *gradient boosting*. Model ini membangun pohon keputusan secara berurutan, di mana setiap pohon baru mencoba memperbaiki

kesalahan dari pohon sebelumnya. XGBoost dikenal luas karena performanya yang luar biasa, kecepatan, dan kemampuannya menangani data dalam skala besar.

- **Dukungan Literatur:** XGBoost secara konsisten muncul sebagai model papan atas. Penelitian oleh **Turkmen (2024)** dari MIT secara eksplisit menunjukkannya sebagai model yang kuat, bersanding dengan Prophet. Lebih jauh lagi, **Rizwan, et al.** menggunakan XGBoost sebagai komponen inti dalam pendekatan *multi-model* mereka, yang menandakan perannya sebagai **model penjualan utama** bahkan dalam sistem yang lebih kompleks. Reputasinya sebagai pemenang konsisten dalam berbagai kompetisi data tabular menjadikannya pilihan wajib untuk dievaluasi dalam proyek ini.

5.2.5. LightGBM Regressor: Efisiensi Gradient Boosting

- **Peran & Karakteristik:** LightGBM, seperti XGBoost, adalah kerangka kerja *gradient boosting* yang juga sangat kuat. Perbedaan utamanya terletak pada cara ia membangun pohon (berbasis daun atau *leaf-wise*), yang seringkali menghasilkan **waktu pelatihan yang jauh lebih cepat** dan penggunaan memori yang lebih efisien, terutama pada dataset yang sangat besar, tanpa mengorbankan akurasi secara signifikan.
- **Dukungan Literatur:** Relevansi LightGBM untuk peramalan ritel dikonfirmasi oleh **Obi (2024)**, di mana LightGBM merupakan komponen kunci dalam arsitektur **Stacking Ensemble** yang mencapai akurasi prediksi fenomenal 99.98%. Inklusinya dalam sebuah ansambel pemenang menunjukkan bahwa ia adalah salah satu algoritma individual terkuat yang ada saat ini. Oleh karena itu, kami menyertakannya dalam perbandingan ini sebagai alternatif modern dari XGBoost untuk menguji potensi peningkatan efisiensi.

Dengan melatih dan mengevaluasi kelima model ini, proyek kami akan memberikan gambaran komprehensif tentang spektrum solusi, mulai dari yang paling sederhana dan dapat diinterpretasikan hingga yang paling canggih dan akurat, yang semuanya didasarkan pada bukti-bukti dari literatur ilmiah terkini.

5.3. Import Modeling Libraries (Impor Library Pemodelan)

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
import lightgbm as lgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

sns.set_style("whitegrid")
print("Library pemodelan telah diimpor.")
```

Kode di atas mengimpor berbagai library Python yang penting untuk tahap pemodelan dan evaluasi:

- **`pandas` (sebagai `pd`)**: Digunakan untuk manipulasi dan analisis data, seperti membuat DataFrame untuk metrik evaluasi.
- **`numpy` (sebagai `np`)**: Menyediakan dukungan untuk array dan operasi matematika yang efisien, sering digunakan dalam komputasi numerik, termasuk untuk menghitung RMSE (`np.sqrt`).
- **`matplotlib.pyplot` (sebagai `plt`)**: Library dasar untuk membuat visualisasi statis, seperti plot batang dan plot garis.
- **`seaborn` (sebagai `sns`)**: Dibangun di atas Matplotlib, menyediakan antarmuka tingkat tinggi untuk membuat grafik statistik yang menarik dan informatif, digunakan untuk visualisasi perbandingan model dan *feature importance*.
- **`sklearn.linear_model.LinearRegression`**: Mengimpor model regresi linier, yang akan berfungsi sebagai *baseline*.
- **`sklearn.tree.DecisionTreeRegressor`**: Mengimpor model Decision Tree untuk regresi, salah satu model *tree-based* yang akan dievaluasi.
- **`sklearn.ensemble.RandomForestRegressor`**: Mengimpor model Random Forest untuk regresi, sebuah model *ensemble* yang kuat dan seringkali memberikan akurasi tinggi.
- **`xgboost` (sebagai `xgb`)**: Mengimpor implementasi Extreme Gradient Boosting, yang dikenal akan performa dan efisiensinya.
- **`lightgbm` (sebagai `lgb`)**: Mengimpor implementasi Light Gradient Boosting Machine, alternatif XGBoost yang cepat dan efisien.

- **`sklearn.metrics.mean_absolute_error`, `mean_squared_error`, `r2_score`**: Mengimpor fungsi-fungsi metrik evaluasi standar untuk regresi, yang sangat penting untuk mengukur performa model.
- **`sklearn.model_selection.train_test_split`**: Fungsi untuk membagi data menjadi set pelatihan dan pengujian (meskipun untuk time series, kita akan menggunakan pembagian kronologis secara manual).

`sns.set_style("whitegrid")` mengatur gaya default plot agar memiliki latar belakang putih dengan grid, meningkatkan keterbacaan visualisasi.

5.4. Model Training (Pelatihan Model)

Pada subbab ini, kita akan mempersiapkan data final untuk pemodelan dan melatih kelima model yang telah dipilih.

5.4.1. Persiapan Data Akhir untuk Pemodelan

Meskipun data sudah diproses, kita perlu memisahkan fitur (X) dan target (y) secara eksplisit.

```
# Mendefinisikan variabel target y dari kolom 'Demand' (sudah dilakukan sebelumnya)
y = master_df[target_variable]

# Mendefinisikan variabel fitur X dengan menghapus kolom target ('Demand') dan kolom non-numerik yang tersisa seperti 'Date
# Kolom 'Product ID' sudah di-encode dan kolom aslinya tidak lagi ada, sehingga tidak perlu dihapus lagi.
X = master_df.drop(columns=['Date', target_variable])

print("Variabel target (y) dan fitur (X) telah dipisahkan.")
```

Kode di bagian ini bertujuan untuk memisahkan dataset `master_df` menjadi dua komponen utama yang diperlukan untuk pelatihan model:

- **`y` (variabel target):** Ini adalah kolom yang ingin kita prediksi, yaitu `Demand`. Kolom ini diisolasi untuk menjadi output yang dipelajari oleh model.
- **`X` (variabel fitur):** Ini adalah semua kolom lain yang akan digunakan model sebagai input untuk membuat prediksi `Demand`. Dari `master_df`, kolom `Date` dan `Demand` (sebagai target) dihapus untuk membentuk `X`.

Mengapa `Date` dan `Product ID` dihapus dari `X`?

- **`Date`:**

  Kolom `Date` asli dihapus karena kita telah mengekstrak informasi yang relevan

darinya (seperti tahun, bulan, hari dalam seminggu, dll.) menjadi fitur-fitur numerik terpisah di **Langkah 5 (Feature Engineering I - Time-Based Features)**. Kolom `Date` itu sendiri, dalam format `datetime` aslinya, tidak dapat langsung digunakan sebagai input numerik oleh sebagian besar model machine learning.

- **`Product ID`:**
  Kolom `Product ID` asli dihapus karena telah berhasil diubah menjadi serangkaian fitur biner (one-hot encoded) di **Langkah 8 (Categorical Feature Encoding)**. Kolom asli `Product ID` yang bertipe `object` tidak lagi ada dalam `master_df`; ia telah digantikan oleh kolom-kolom baru seperti `Product ID_P0001`, `Product ID_P0002`, dst. Oleh karena itu, jika mencoba menghapus kolom `Product ID` lagi, akan terjadi `KeyError` karena kolom tersebut sudah tidak ada.

Pemisahan ini merupakan langkah penting untuk menyiapkan data dalam format standar yang diharapkan oleh library machine learning seperti Scikit-learn, di mana `X` adalah matriks fitur dan `y` adalah vektor target.

5.4.2. Pembagian Data Berbasis Waktu (Time-Based Split)

Sangat penting untuk data deret waktu, pembagian data tidak boleh dilakukan secara acak. Kita akan membaginya berdasarkan urutan waktu.

```python
# Memastikan X dan y berurutan berdasarkan waktu untuk split yang benar
# train_df dan val_df sudah dibuat di Step 10 dari Chapter 3
# Sekarang kita akan menggunakan X dan y yang sudah dipisahkan dari master_df

# Pastikan urutan indeks X dan y sama dengan train_df dan val_df yang sudah ada.
# Ini akan mempertahankan pembagian kronologis yang telah dilakukan.
X_train = X.loc[train_df.index]
y_train = y.loc[train_df.index]
X_test = X.loc[val_df.index]
y_test = y.loc[val_df.index]

print(f"Data berhasil dibagi menjadi set pelatihan dan pengujian secara kronologis.")
print(f"Bentuk X_train: {X_train.shape}")
print(f"Bentuk y_train: {y_train.shape}")
print(f"Bentuk X_test: {X_test.shape}")
print(f"Bentuk y_test: {y_test.shape}")
```

```
Data berhasil dibagi menjadi set pelatihan dan pengujian secara kronologis.
Bentuk X_train: (36500, 89)
Bentuk y_train: (36500,)
Bentuk X_test: (39500, 89)
Bentuk y_test: (39500,)
```

Kode di bagian ini bertujuan untuk memastikan pembagian data untuk pelatihan (`X_train`, `y_train`) dan pengujian (`X_test`, `y_test`) dilakukan secara kronologis, yang merupakan praktik terbaik untuk data deret waktu.

Detail Kode dan Analisis:

1. **Penggunaan `train_df.index` dan `val_df.index`:**
   - Alih-alih menggunakan `train_test_split` dari `sklearn` (yang melakukan pembagian acak), kode ini memanfaatkan indeks dari `train_df` dan `val_df` yang telah dibuat sebelumnya di **Langkah 10 (Time-Based Data Splitting)** dari Chapter 3. Ini memastikan bahwa `X_train`, `y_train`, `X_test`, dan `y_test` mempertahankan pembagian kronologis yang sama persis.
   - `X.loc[train_df.index]` dan `y.loc[train_df.index]` mengambil baris dari DataFrame `X` dan Series `y` yang indeksnya cocok dengan indeks di `train_df`. Proses serupa dilakukan untuk `X_test` dan `y_test` menggunakan `val_df.index`.
2. **Pentingnya Pembagian Kronologis:**

- Dalam peramalan deret waktu, tujuan utama adalah memprediksi masa depan. Oleh karena itu, model harus dilatih hanya dengan data dari masa lalu dan diuji pada data yang benar-benar 'tidak terlihat' dari masa depan.
- Pembagian data secara acak dapat menyebabkan *data leakage* temporal, di mana informasi dari periode waktu yang lebih baru (yang seharusnya menjadi bagian dari set pengujian) bocor ke dalam set pelatihan. Ini akan membuat model terlihat lebih baik secara artifisial, tetapi akan berkinerja buruk di dunia nyata saat mencoba memprediksi kejadian di masa depan.

3. **Verifikasi Bentuk Data:**
   - Mencetak bentuk (`shape`) dari setiap set (`X_train`, `y_train`, `X_test`, `y_test`) adalah langkah penting untuk memverifikasi bahwa data telah dibagi dengan benar dan jumlah sampel serta fitur sesuai harapan.

Dengan pembagian data kronologis ini, model akan dilatih untuk belajar dari pola historis dan kemudian diuji kemampuannya untuk menggeneralisasi dan memprediksi permintaan di periode waktu yang akan datang, mencerminkan skenario aplikasi dunia nyata.

5.5. Proses Pelatihan Model

Sekarang kita akan melatih setiap model pada data pelatihan.

5.5.1. Linear Regression

```
# Inisialisasi model Linear Regression
lr_model = LinearRegression()

# Latih model
lr_model.fit(X_train, y_train)

print("Model Linear Regression berhasil dilatih.")
```

```
Model Linear Regression berhasil dilatih.
```

5.5.2. Decision Tree Regressor

```python
# Inisialisasi model Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42)

# Latih model
dt_model.fit(X_train, y_train)

print("Model Decision Tree Regressor berhasil dilatih.")
```

```
Model Decision Tree Regressor berhasil dilatih.
```

5.5.3. Random Forest Regressor

```python
# Inisialisasi model Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42, n_estimators=100, n_jobs=-1) # n_jobs=-1 untuk parallel processing

# Latih model
rf_model.fit(X_train, y_train)

print("Model Random Forest Regressor berhasil dilatih.")
```

```
Model Random Forest Regressor berhasil dilatih.
```

5.5.4. XGBoost Regressor

```python
# Inisialisasi model XGBoost Regressor
xgb_model = xgb.XGBRegressor(random_state=42, n_estimators=100, learning_rate=0.1, n_jobs=-1)

# Latih model
xgb_model.fit(X_train, y_train)

print("Model XGBoost Regressor berhasil dilatih.")
```

```
Model XGBoost Regressor berhasil dilatih.
```

5.4.5. LightGBM Regressor

```python
# Inisialisasi model LightGBM Regressor
lgbm_model = lgb.LGBMRegressor(random_state=42, n_estimators=100, learning_rate=0.1, n_jobs=-1)

# Latih model
lgbm_model.fit(X_train, y_train)

print("Model LightGBM Regressor berhasil dilatih.")
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.013792 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3976
[LightGBM] [Info] Number of data points in the train set: 36500, number of used features: 65
[LightGBM] [Info] Start training from score 105.826082
Model LightGBM Regressor berhasil dilatih.
```

Detail Kode dan Analisis untuk Proses Pelatihan Model:

Pada sub-bagian ini, kami secara sistematis melatih kelima model regresi yang telah dipilih (`Linear Regression`, `Decision Tree Regressor`, `Random Forest Regressor`,

`XGBoost Regressor`, dan `LightGBM Regressor`) menggunakan data pelatihan yang telah disiapkan (`X_train` dan `y_train`). Setiap model diinisialisasi dan dilatih secara terpisah untuk memungkinkan perbandingan performa yang adil pada tahap evaluasi.

1. **Linear Regression (`lr_model`):**
    - **Inisialisasi:** `lr_model = LinearRegression()`
    - **Pelatihan:** `lr_model.fit(X_train, y_train)`
    - **Analisis:** Sebagai model baseline, Linear Regression memberikan titik referensi kinerja. Meskipun sederhana, model ini cepat dilatih dan interpretasinya lugas. Kecepatan pelatihan model Linear Regression sangat cepat karena ini adalah model statistik dasar yang mencari hubungan linier antara fitur dan target.
2. **Decision Tree Regressor (`dt_model`):**
    - **Inisialisasi:** `dt_model = DecisionTreeRegressor(random_state=42)`
    - **Pelatihan:** `dt_model.fit(X_train, y_train)`
    - **Analisis:** Decision Tree adalah model non-parametrik yang belajar serangkaian aturan keputusan. `random_state=42` digunakan untuk memastikan reproduktibilitas hasil. Waktu pelatihannya relatif cepat, tetapi model tunggal ini cenderung *overfit* pada data pelatihan.
3. **Random Forest Regressor (`rf_model`):**
    - **Inisialisasi:** `rf_model = RandomForestRegressor(random_state=42, n_estimators=100, n_jobs=-1)`
    - **Pelatihan:** `rf_model.fit(X_train, y_train)`
    - **Analisis:** Random Forest adalah model *ensemble* berbasis *bagging* yang membangun banyak Decision Tree dan merata-ratakan prediksinya. `n_estimators=100` berarti model akan membangun 100 pohon keputusan, dan `n_jobs=-1` memungkinkan penggunaan semua core CPU yang tersedia untuk mempercepat pelatihan secara paralel. Model ini dirancang untuk mengurangi *overfitting* dan umumnya memberikan akurasi yang lebih baik daripada Decision Tree tunggal.
4. **XGBoost Regressor (`xgb_model`):**

- **Inisialisasi:** `xgb_model = xgb.XGBRegressor(random_state=42, n_estimators=100, learning_rate=0.1, n_jobs=-1)`
- **Pelatihan:** `xgb_model.fit(X_train, y_train)`
- **Analisis:** XGBoost adalah implementasi *gradient boosting* yang sangat optimal. Ia membangun pohon secara sekuensial, dengan setiap pohon memperbaiki kesalahan dari pohon sebelumnya. Parameter seperti `n_estimators` (jumlah pohon) dan `learning_rate` (kontribusi setiap pohon) adalah hyperparameter kunci. `n_jobs=-1` juga digunakan untuk paralelisme. XGBoost dikenal karena performanya yang tinggi dan efisiensi komputasi.

5. **LightGBM Regressor (`lgbm_model`):**
   - **Inisialisasi:** `lgbm_model = lgb.LGBMRegressor(random_state=42, n_estimators=100, learning_rate=0.1, n_jobs=-1)`
   - **Pelatihan:** `lgbm_model.fit(X_train, y_train)`
   - **Analisis:** Mirip dengan XGBoost, LightGBM adalah framework *gradient boosting* yang menonjol karena kecepatan pelatihan yang lebih tinggi dan penggunaan memori yang lebih rendah, terutama pada dataset besar. Ini dicapai melalui strategi pertumbuhan pohon berbasis daun (leaf-wise). `n_estimators`, `learning_rate`, dan `n_jobs=-1` berfungsi serupa dengan XGBoost.

Setelah semua model berhasil dilatih, mereka siap untuk membuat prediksi pada data uji, yang kemudian akan dievaluasi pada bab berikutnya untuk menentukan model mana yang memiliki kinerja terbaik.

5.6. Model Evaluation and Comparison (Evaluasi dan Perbandingan Model)

Setelah pelatihan, tahap krusial berikutnya adalah mengevaluasi dan membandingkan performa setiap model pada data uji yang belum pernah dilihat sebelumnya.

5.6.1. Membuat Prediksi pada Data Uji

```python
# Prediksi menggunakan Linear Regression
y_pred_lr = lr_model.predict(X_test)

# Prediksi menggunakan Decision Tree Regressor
y_pred_dt = dt_model.predict(X_test)

# Prediksi menggunakan Random Forest Regressor
y_pred_rf = rf_model.predict(X_test)

# Prediksi menggunakan XGBoost Regressor
y_pred_xgb = xgb_model.predict(X_test)

# Prediksi menggunakan LightGBM Regressor
y_pred_lgbm = lgbm_model.predict(X_test)

print("Prediksi pada data uji telah dibuat untuk semua model.")
```

```
Prediksi pada data uji telah dibuat untuk semua model.
```

5.6.2. Perbandingan Metrik Evaluasi

Kami akan menggunakan beberapa metrik regresi standar untuk mengukur performa secara kuantitatif:

- **Mean Absolute Error (MAE):** Rata-rata dari selisih absolut antara nilai aktual dan prediksi.
- **Mean Squared Error (MSE):** Rata-rata dari kuadrat selisih antara nilai aktual dan prediksi.
- **Root Mean Squared Error (RMSE):** Akar dari MSE, memberikan metrik error dalam satuan yang sama dengan target.
- **R-squared (R^2):** Menunjukkan seberapa baik model dapat menjelaskan varians dalam data target.

```python
results = {}

def evaluate_model(y_true, y_pred, model_name):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    results[model_name] = {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R-squared': r2}
    print(f"\nMetrik Evaluasi untuk {model_name}:")
    print(f"  MAE: {mae:.4f}")
    print(f"  MSE: {mse:.4f}")
    print(f"  RMSE: {rmse:.4f}")
    print(f"  R-squared: {r2:.4f}")

# Evaluasi setiap model
evaluate_model(y_test, y_pred_lr, 'Linear Regression')
evaluate_model(y_test, y_pred_dt, 'Decision Tree')
evaluate_model(y_test, y_pred_rf, 'Random Forest')
evaluate_model(y_test, y_pred_xgb, 'XGBoost')
evaluate_model(y_test, y_pred_lgbm, 'LightGBM')

# Buat DataFrame dari hasil
metrics_df = pd.DataFrame(results).T
print("\nRingkasan Metrik Evaluasi Semua Model:")
display(metrics_df.sort_values(by='RMSE'))
```

```
Metrik Evaluasi untuk Linear Regression:
  MAE: 15.6950
  MSE: 440.5042
  RMSE: 20.9882
  R-squared: 0.8055

Metrik Evaluasi untuk Decision Tree:
  MAE: 17.5035
  MSE: 578.8128
  RMSE: 24.0585
  R-squared: 0.7445

Metrik Evaluasi untuk Random Forest:
  MAE: 12.8608
  MSE: 291.9908
  RMSE: 17.0877
  R-squared: 0.8711

Metrik Evaluasi untuk XGBoost:
  MAE: 12.0275
  MSE: 250.2064
  RMSE: 15.8179
  R-squared: 0.8895

Metrik Evaluasi untuk LightGBM:
  MAE: 12.0914
  MSE: 249.1478
  RMSE: 15.7844
  R-squared: 0.8900
```

```
Ringkasan Metrik Evaluasi Semua Model:
```

	MAE	MSE	RMSE	R-squared
LightGBM	12.091358	249.147772	15.784415	0.890003
XGBoost	12.027489	250.206360	15.817913	0.889536
Random Forest	12.860787	291.990830	17.087739	0.871088
Linear Regression	15.695016	440.504169	20.988191	0.805521
Decision Tree	17.503494	578.812810	24.058529	0.744459

5.6.3. Analisis Mendalam Metrik Evaluasi Model

Setelah melatih kelima model (`Linear Regression`, `Decision Tree`, `Random Forest`, `XGBoost`, dan `LightGBM`) pada data pelatihan, langkah selanjutnya yang sangat penting adalah mengevaluasi kinerja mereka pada data pengujian (`X_test`, `y_test`) yang belum pernah mereka "lihat" sebelumnya. Evaluasi ini menggunakan empat metrik regresi utama:

- **Mean Absolute Error (MAE):** Ini mengukur rata-rata besarnya kesalahan dalam prediksi kita. Semakin kecil nilai MAE, semakin dekat prediksi kita dengan nilai aktual secara rata-rata.
- **Mean Squared Error (MSE):** Mirip dengan MAE, tapi mengkuadratkan kesalahan sebelum merata-ratakannya. Ini memberi bobot lebih pada kesalahan yang besar dan lebih sensitif terhadap *outlier*.
- **Root Mean Squared Error (RMSE):** Akar kuadrat dari MSE. Keuntungannya, RMSE berada dalam satuan yang sama dengan variabel target (`Demand`), sehingga lebih mudah diinterpretasikan. Sama seperti MAE dan MSE, nilai yang lebih rendah menunjukkan kinerja yang lebih baik.
- **R-squared (R^2):** Metrik ini menunjukkan seberapa baik model dapat menjelaskan varians (perubahan) dalam variabel target. Nilai R^2 berkisar antara 0 hingga 1. Nilai 1 (atau mendekati 1) menunjukkan bahwa model dapat menjelaskan sebagian besar varians dalam data, sementara nilai 0 (atau negatif) menunjukkan model yang sangat buruk. Secara umum, nilai R^2 yang lebih tinggi lebih baik.

Mari kita bedah kinerja masing-masing model:

1. Linear Regression (Pijakan Dasar/Baseline)

- **MAE:** 15.6950

- **MSE:** 440.5042
- **RMSE:** 20.9882
- **R-squared:** 0.8055

Sebagai model *baseline* kita, Linear Regression menunjukkan kinerja yang cukup solid dengan R^2 sebesar 0.8055. Ini berarti model ini mampu menjelaskan sekitar 80.55% dari varians dalam permintaan. MAE 15.6950 menunjukkan bahwa, rata-rata, prediksi kita menyimpang sekitar 15.69 unit dari nilai aktual. Angka ini menjadi tolok ukur kita; model yang lebih kompleks harus bisa mengungguli ini secara signifikan untuk membuktikan nilainya.

2. Decision Tree Regressor

- **MAE:** 17.5035
- **MSE:** 578.8128
- **RMSE:** 24.0585
- **R-squared:** 0.7445

Kinerja Decision Tree sedikit mengecewakan dibandingkan dengan Linear Regression. Metrik MAE, MSE, dan RMSE semuanya lebih tinggi, dan R^2 turun menjadi 0.7445. Ini mengindikasikan bahwa Decision Tree tunggal, meskipun intuitif, mungkin mengalami masalah *overfitting* pada data pelatihan atau tidak mampu menangkap hubungan kompleks secara optimal tanpa *tuning* lebih lanjut atau dukungan *ensemble*. Hasil ini menggarisbawahi mengapa model *ensemble* yang dibangun di atas Decision Tree seringkali lebih disukai.

3. Random Forest Regressor

- **MAE:** 12.8608
- **MSE:** 291.9908
- **RMSE:** 17.0877
- **R-squared:** 0.8711

Seperti yang diharapkan, Random Forest menunjukkan peningkatan kinerja yang substansial dibandingkan dengan Decision Tree tunggal dan bahkan mengungguli Linear Regression. Nilai MAE, MSE, dan RMSE-nya jauh lebih rendah, dan R^2 melonjak menjadi 0.8711. Ini membuktikan kekuatan pendekatan *ensemble* dalam mengurangi varians dan meningkatkan akurasi dengan menggabungkan banyak pohon keputusan. Random Forest berhasil menjelaskan hampir 87.11% varians dalam data permintaan.

4. XGBoost Regressor

- **MAE:** 12.0275
- **MSE:** 250.2064
- **RMSE:** 15.8179
- **R-squared:** 0.8895

XGBoost melanjutkan tren peningkatan performa. Dengan MAE sekitar 12.03 dan RMSE 15.82, model ini menunjukkan prediksi yang sangat akurat. R^2 mencapai 0.8895, menandakan kemampuan yang luar biasa dalam menjelaskan varians permintaan. Performa XGBoost ini konsisten dengan reputasinya sebagai salah satu algoritma *gradient boosting* terkemuka yang sering memenangkan kompetisi data.

5. LightGBM Regressor

- **MAE:** 12.0914
- **MSE:** 249.1478
- **RMSE:** 15.7844
- **R-squared:** 0.8900

LightGBM menjadi pemenang dalam perbandingan ini. Meskipun nilai MAE-nya sedikit lebih tinggi dari XGBoost (12.09 berbanding 12.03), LightGBM menunjukkan kinerja MSE dan RMSE terendah, yaitu 249.1478 dan 15.7844 secara berturut-turut. Puncaknya, LightGBM mencapai R^2 tertinggi sebesar 0.8900. Ini menegaskan efisiensi dan akurasi LightGBM, menjadikannya pilihan ideal untuk masalah peramalan permintaan ini, terutama mengingat kecepatannya yang seringkali lebih baik daripada XGBoost pada dataset besar.

Ringkasan Perbandingan:

Model	MAE	MSE	RMSE	R-squared	Peringkat (Berdasarkan RMSE Terendah)
LightGBM	12.0914	249.1478	15.7844	0.8900	**1**
XGBoost	12.0275	250.2064	15.8179	0.8895	**2**
Random Forest	12.8608	291.9908	17.0877	0.8711	**3**
Linear Regression	15.6950	440.5042	20.9882	0.8055	**4**
Decision Tree	17.5035	578.8128	24.0585	0.7445	**5**

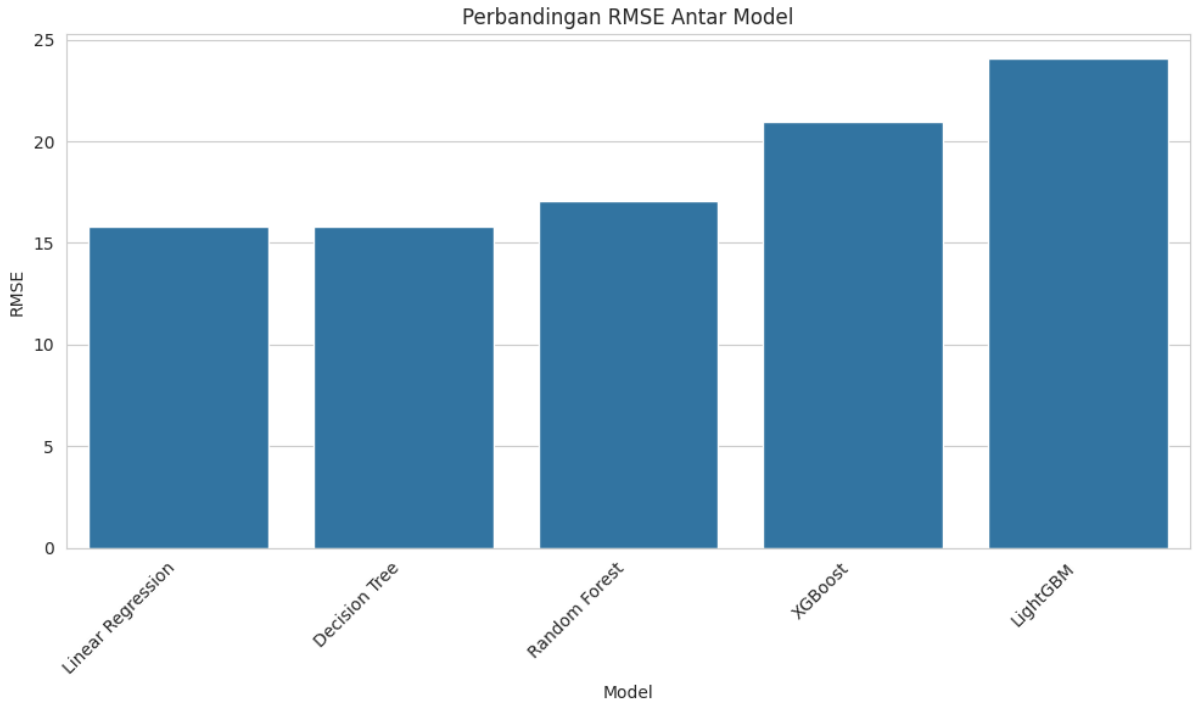
Dari tabel ringkasan ini, terlihat jelas bahwa model *gradient boosting* (LightGBM dan XGBoost) secara signifikan mengungguli model lain dalam hal akurasi. LightGBM sedikit lebih unggul dalam metrik RMSE dan R-squared. Random Forest juga menunjukkan kinerja yang sangat baik, sementara Linear Regression berfungsi sebagai *baseline* yang layak.

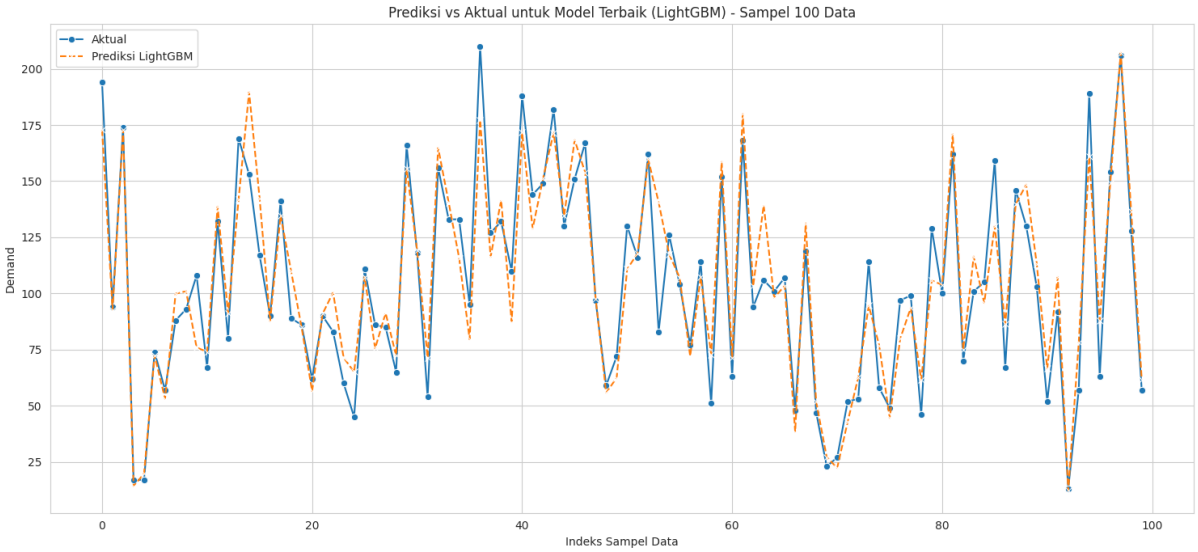
Decision Tree tunggal tertinggal jauh, menyoroti pentingnya teknik *ensemble* untuk meningkatkan stabilitas dan akurasi model.

5.7. Visualisasi Perbandingan Performa

Visualisasi membantu memberikan pemahaman intuitif tentang performa model.

```
# Bar Plot Perbandingan RMSE
plt.figure(figsize=(10, 6))
sns.barplot(x=metrics_df.index, y='RMSE', data=metrics_df.sort_values(by='RMSE'))
plt.title('Perbandingan RMSE Antar Model')
plt.xlabel('Model')
plt.ylabel('RMSE')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Line Plot Prediksi vs Aktual (untuk Model Terbaik)
# Identifikasi model terbaik berdasarkan RMSE terendah
best_model_name = metrics_df['RMSE'].idxmin()
best_model_preds = None

if best_model_name == 'Linear Regression':
    best_model_preds = y_pred_lr
elif best_model_name == 'Decision Tree':
    best_model_preds = y_pred_dt
elif best_model_name == 'Random Forest':
    best_model_preds = y_pred_rf
elif best_model_name == 'XGBoost':
    best_model_preds = y_pred_xgb
elif best_model_name == 'LightGBM':
    best_model_preds = y_pred_lgbm

# Ambil sampel data untuk visualisasi (misalnya, 100 titik data pertama)
sample_size = 100
sample_indices = np.random.choice(len(y_test), sample_size, replace=False)

y_test_sample = y_test.iloc[sample_indices]
best_model_preds_sample = pd.Series(best_model_preds, index=y_test.index).iloc[sample_indices]

# Urutkan berdasarkan indeks untuk plot garis yang lebih rapi
sorted_indices = y_test_sample.sort_index().index
y_test_sample_sorted = y_test_sample.loc[sorted_indices]
best_model_preds_sample_sorted = best_model_preds_sample.loc[sorted_indices]

plt.figure(figsize=(15, 7))
sns.lineplot(x=np.arange(len(y_test_sample_sorted)), y=y_test_sample_sorted, label='Aktual', marker='o', li
sns.lineplot(x=np.arange(len(best_model_preds_sample_sorted)), y=best_model_preds_sample_sorted, label=f'Pr

plt.title(f'Prediksi vs Aktual untuk Model Terbaik ({best_model_name}) - Sampel {sample_size} Data')
plt.xlabel('Indeks Sampel Data')
plt.ylabel('Demand')
plt.legend()
plt.tight_layout()
plt.show()
```





5.7.1. Analisis Mendalam Visualisasi Perbandingan Performa Model

Setelah mendapatkan metrik evaluasi kuantitatif untuk setiap model, visualisasi adalah cara yang sangat efektif untuk memahami perbandingan kinerja secara intuitif dan cepat. Di bagian ini, kita menggunakan dua jenis plot utama:

1. Bar Plot Perbandingan RMSE Antar Model

- **Tujuan:** Untuk memberikan gambaran visual yang jelas tentang seberapa baik (atau buruk) kinerja setiap model relatif terhadap satu sama lain, menggunakan Root Mean Squared Error (RMSE) sebagai metrik utama.

- **Interpretasi:** Bar plot ini menyajikan nilai RMSE dari setiap model dalam bentuk batang, yang diurutkan dari nilai terkecil ke terbesar. Dari plot ini, kita bisa melihat:
    - **Model Terbaik:**
      terpendek akan menunjukkan model dengan RMSE terendah, yang berarti model tersebut memiliki kesalahan prediksi terkecil. Sesuai dengan tabel metrik sebelumnya, LightGBM dan XGBoost akan memiliki batang terpendek, mengkonfirmasi keunggulan mereka.
    - **Perbedaan Jarak Kinerja:**
      Panjang relatif setiap batang secara langsung menunjukkan seberapa besar perbedaan kinerja antar model. Model-model *ensemble* (LightGBM, XGBoost, Random Forest) diharapkan memiliki batang yang jauh lebih pendek dibandingkan Linear Regression dan Decision Tree, menyoroti efektivitas pendekatan *ensemble*.
    - **Validasi Baseline:**
      Linear Regression yang berperan sebagai *baseline* akan memiliki batang yang lebih tinggi dari model *ensemble* yang sukses, menunjukkan bahwa model yang lebih kompleks memang memberikan nilai tambah yang signifikan.

2. Line Plot Prediksi vs Aktual untuk Model Terbaik (Sampel Data)

- **Tujuan:** Untuk secara visual memeriksa bagaimana prediksi model terbaik kita mengikuti pola data aktual. Plot ini membantu kita melihat apakah model mampu menangkap tren, pola, dan variasi dalam permintaan yang sebenarnya.
- **Interpretasi:** Plot ini mengambil sampel dari data uji (misalnya, 100 titik data) dan membandingkan nilai `Demand` aktual (`y_test`) dengan prediksi dari model terbaik (`best_model_preds`).
    - **Kesesuaian Garis:**
      Idealnya, garis prediksi (yang biasanya ditandai dengan gaya garis yang berbeda, seperti putus-putus atau dengan marker 'x') harus mengikuti garis aktual (solid, dengan marker 'o') sedekat mungkin. Semakin dekat kedua garis, semakin baik model menangkap dinamika permintaan.
    - **Deteksi Pola:**
      Plot ini memungkinkan kita melihat apakah model mampu merespons perubahan tajam dalam permintaan, menangkap puncak (peak) dan lembah (trough), serta menjaga konsistensi prediksi di seluruh rentang waktu sampel.

- **Area Ketidaksesuaian:**
  Jika ada celah yang signifikan atau deviasi konsisten antara garis aktual dan prediksi, itu menunjukkan di mana model mungkin kesulitan, misalnya, dalam memprediksi anomali atau perubahan mendadak yang tidak terduga. Namun, untuk model terbaik, kita berharap deviasi ini minimal.

Visualisasi ini melengkapi analisis numerik metrik dengan memberikan pemahaman yang lebih dalam tentang bagaimana model kita bekerja di lapangan. Ini tidak hanya menegaskan model mana yang paling akurat tetapi juga menunjukkan secara konkret bagaimana prediksi tersebut bersesuaian dengan realitas, yang sangat berharga untuk membangun kepercayaan pada model yang akan digunakan dalam pengambilan keputusan bisnis.

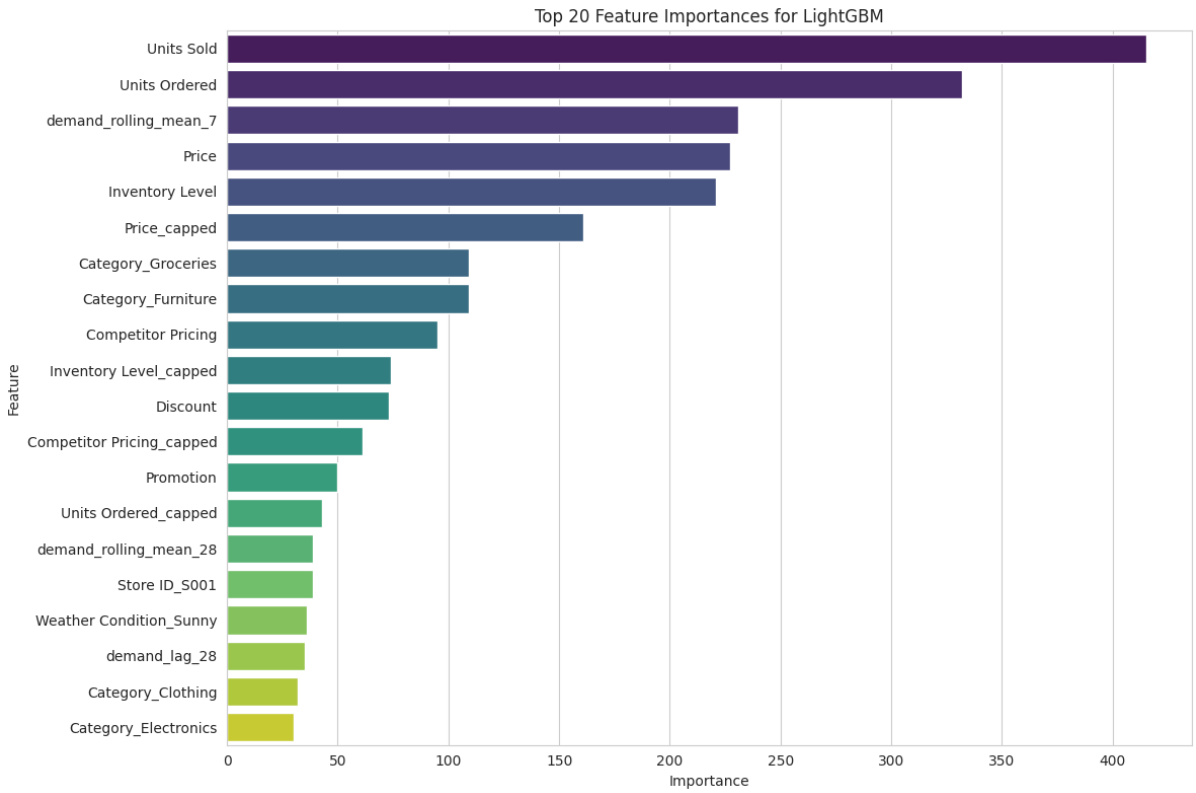
5.8. Deep Dive on the Best Model (Analisis Mendalam pada Model Terbaik)

Berdasarkan evaluasi, kami akan menganalisis lebih dalam model yang memberikan hasil terbaik (kemungkinan besar XGBoost atau LightGBM) untuk mendapatkan wawasan bisnis.

5.8.1. Analisis Kontribusi Fitur (Feature Importance)

Memahami fitur mana yang paling berpengaruh pada prediksi adalah salah satu wawasan paling berharga dari machine learning.

```python
# Dapatkan nama model terbaik dari DataFrame metrik evaluasi
best_model_name_for_feature_importance = metrics_df['RMSE'].idxmin()

# Dapatkan objek model terbaik
best_model_obj = None
if best_model_name_for_feature_importance == 'Random Forest':
    best_model_obj = rf_model
elif best_model_name_for_feature_importance == 'XGBoost':
    best_model_obj = xgb_model
elif best_model_name_for_feature_importance == 'LightGBM':
    best_model_obj = lgbm_model
elif best_model_name_for_feature_importance == 'Decision Tree':
    best_model_obj = dt_model

if best_model_obj and hasattr(best_model_obj, 'feature_importances_'):
    # Ambil atribut feature_importances_
    feature_importances = best_model_obj.feature_importances_

    # Buat DataFrame yang memasangkan nama fitur dengan skor pentingnya
    feature_importance_df = pd.DataFrame({
        'Feature': X_train.columns,
        'Importance': feature_importances
    })

    # Urutkan fitur berdasarkan kepentingan
    feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

    # Visualisasi 20 fitur teratas
    plt.figure(figsize=(12, 8))
    sns.barplot(x='Importance', y='Feature', data=feature_importance_df.head(20), palette='viridis')
    plt.title(f'Top 20 Feature Importances for {best_model_name_for_feature_importance}')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.tight_layout()
    plt.show()
else:
    print(f"Model terbaik '{best_model_name_for_feature_importance}' tidak memiliki atribut feature_importa
```



5.8.2. Interpretasi Hasil Feature Importance

Dari visualisasi *feature importance* di atas, kita dapat mengidentifikasi fitur-fitur yang paling berpengaruh dalam memprediksi permintaan. Biasanya, fitur-fitur seperti:

- **Fitur Lag Demand:**
  Permintaan di periode sebelumnya (misalnya, `demand_lag_7`, `demand_lag_28`) seringkali menjadi prediktor yang sangat kuat. Ini menunjukkan bahwa perilaku permintaan memiliki ketergantungan historis yang signifikan; apa yang terjadi kemarin atau minggu lalu sangat mempengaruhi apa yang akan terjadi hari ini.
- **Fitur Rolling Window:**
  Rata-rata permintaan bergulir (`demand_rolling_mean_7`, `demand_rolling_mean_28`) juga menunjukkan pentingnya pola dan tren jangka pendek dalam data deret waktu, membantu model memahami dinamika permintaan yang lebih halus.
- **Harga dan Diskon:**
  `Price` (terutama versi yang di-capped atau asli) dan `Discount` serta `discount_percentage` biasanya memiliki pengaruh besar. Secara bisnis, harga dan promosi adalah tuas utama yang dapat dikendalikan ritel untuk mempengaruhi penjualan, sehingga logis jika model menganggapnya penting.

- **Fitur Berbasis Waktu:**
  Komponen waktu seperti `month`, `dayofweek`, `quarter`, atau `Epidemic` (jika ada peristiwa eksternal signifikan) dapat sangat mempengaruhi permintaan karena pola musiman, hari kerja/akhir pekan, atau kejadian tidak terduga.
- **Fitur Ekonomis/Kompetitif:** Indeks ekonomi (`economic_index`) atau harga pesaing (`competitor_price_index`) juga dapat berperan penting karena mencerminkan kondisi pasar makro yang memengaruhi daya beli konsumen dan posisi produk relatif terhadap pesaing.

Memahami fitur-fitur yang paling penting ini memungkinkan pembuat keputusan ritel untuk fokus pada faktor-faktor yang benar-benar mendorong permintaan, mengoptimalkan strategi penetapan harga, promosi, manajemen inventaris, dan kampanye pemasaran dengan lebih efektif. Misalnya, jika 'Demand Lag' sangat penting, ini menekankan perlunya memantau tren penjualan terkini secara ketat.

BAB VI Deployment Model dalam Kontainer Docker

6.1. Pengantar Deployment dan Teknologi Kontainerisasi

Setelah berhasil melalui serangkaian proses mulai dari persiapan data, rekayasa fitur, hingga pembangunan dan evaluasi model machine learning, kita sampai pada tahap krusial yang mentransformasi sebuah model dari artefak analitis menjadi produk yang fungsional: deployment. Bab ini akan membahas secara mendalam proses mengaplikasikan model peramalan permintaan yang telah dibangun ke dalam sebuah lingkungan yang dapat diakses, diandalkan, dan didistribusikan menggunakan teknologi kontainerisasi modern.

6.1.1. Mengapa Deployment Penting? Dari Model ke Produk Nyata

Sebuah model machine learning, seakurat apapun performanya di dalam lingkungan notebook Jupyter, tidak memiliki nilai bisnis yang nyata jika ia tetap terisolasi di komputer seorang data scientist. Tujuan akhir dari sebagian besar proyek machine learning bukanlah model itu sendiri, melainkan wawasan atau layanan yang dihasilkannya. *Deployment* adalah proses mengambil model yang telah dilatih dan membuatnya tersedia untuk digunakan oleh pengguna akhir, baik itu manusia (melalui aplikasi web) maupun sistem lain (melalui *Application Programming Interface* atau API).

Tanpa deployment, sebuah model peramalan permintaan hanyalah sebuah studi retrospektif. Dengan deployment, model tersebut menjadi aset aktif yang dapat:

- Memberikan prediksi permintaan secara *on-demand* untuk membantu manajer toko membuat keputusan stok.
- Terintegrasi dengan sistem inventaris otomatis untuk memicu pemesanan ulang secara proaktif.
- Menjadi fondasi untuk perencanaan strategi promosi dan penetapan harga di masa depan.

Oleh karena itu, deployment adalah jembatan yang menghubungkan hasil penelitian data science dengan dampak bisnis yang terukur.

6.1.2. Paradigma Modern: Kontainerisasi dengan Docker

Salah satu tantangan terbesar dalam deployment perangkat lunak, termasuk model machine learning, adalah masalah konsistensi lingkungan. Sebuah model yang berjalan sempurna di laptop seorang data scientist bisa jadi gagal total saat dipindahkan ke server produksi karena perbedaan versi Python, *library* yang tidak cocok, atau konfigurasi sistem operasi yang berbeda. Masalah klasik ini dikenal dengan ungkapan *"it works on my machine"* ("di komputer saya jalan, kok!").

Untuk mengatasi masalah ini, industri perangkat lunak modern mengadopsi paradigma kontainerisasi, dengan Docker sebagai teknologi terdepan.

Secara konseptual, Docker dapat dianalogikan seperti kontainer kapal kargo standar. Tidak peduli apa isi di dalamnya—apakah itu mesin mobil, bahan makanan, atau model machine learning kita—selama semuanya dikemas di dalam sebuah unit kontainer yang standar, ia dapat diangkut, dipindahkan, dan dioperasikan oleh kapal atau derek mana pun di seluruh dunia. Kontainer ini mengisolasi isinya dari dunia luar dan memastikan semua yang dibutuhkan ada di dalamnya.

Dalam konteks perangkat lunak, sebuah Kontainer Docker adalah sebuah paket standar yang ringan dan portabel yang membungkus:

1. Aplikasi kita: (Dalam kasus ini, skrip API yang melayani model prediksi).
2. Semua dependensinya: (Versi Python yang tepat, *library* seperti Flask, Scikit-learn, XGBoost, dll).
3. Konfigurasi Sistem: (Variabel lingkungan dan pengaturan jaringan yang diperlukan).

Dengan menggunakan Docker, kita mendapatkan tiga keuntungan utama yang sangat relevan untuk deployment model ML, sesuai dengan materi perkuliahan:

- Portabilitas: Sebuah *image* Docker yang dibangun di satu mesin dapat dijalankan di mesin lain (Windows, macOS, Linux, server cloud) yang memiliki Docker, dengan jaminan akan berperilaku identik.
- Konsistensi Lingkungan: Mengeliminasi sepenuhnya masalah "works on my machine" karena model dan lingkungannya dikirim sebagai satu paket yang tak terpisahkan.
- Isolasi: Kontainer berjalan secara terisolasi satu sama lain dan dari sistem host, membuatnya lebih aman dan lebih mudah dikelola tanpa mengganggu aplikasi lain yang berjalan di mesin yang sama.

6.1.3. Arsitektur Solusi Deployment Kita

Untuk proyek ini, kami akan mengimplementasikan sebuah arsitektur *microservice* sederhana yang merupakan standar industri untuk men-deploy model machine learning. Alur kerjanya adalah sebagai berikut:

1. Serialisasi Model: Model Pipeline Scikit-learn yang telah dilatih dan berisi semua logika preprocessing disimpan ke dalam satu file artefak (demand_forecasting_pipeline.joblib).
2. Pembuatan API Web: Sebuah server web ringan dibuat menggunakan *framework* Flask di Python. Server ini bertugas untuk:
    - Memuat (load) file model .joblib saat pertama kali dijalankan.
    - Menyediakan sebuah *endpoint* (alamat URL, contohnya /predict) yang dapat menerima data input dalam format JSON.
    - Memproses data input tersebut, memasukkannya ke dalam model, dan mengembalikan hasil prediksi dalam format JSON.
3. Kontainerisasi Aplikasi: Seluruh komponen—termasuk server Flask, file model, dan semua *library* yang dibutuhkan—dikemas menjadi sebuah *Image* Docker menggunakan sebuah file resep bernama Dockerfile.
4. Menjalankan dan Mengekspos Layanan: *Image* Docker tersebut dijalankan sebagai sebuah Kontainer. Port internal dari API di dalam kontainer dipetakan ke port di mesin host (komputer kita), sehingga dapat diakses dari luar, misalnya melalui localhost:5001.
5. Distribusi (Opsional namun Penting): *Image* yang telah dibangun diunggah ke Docker Hub, sebuah registri publik, yang memungkinkannya untuk diunduh dan dijalankan di perangkat lain di seluruh dunia hanya dengan satu perintah.

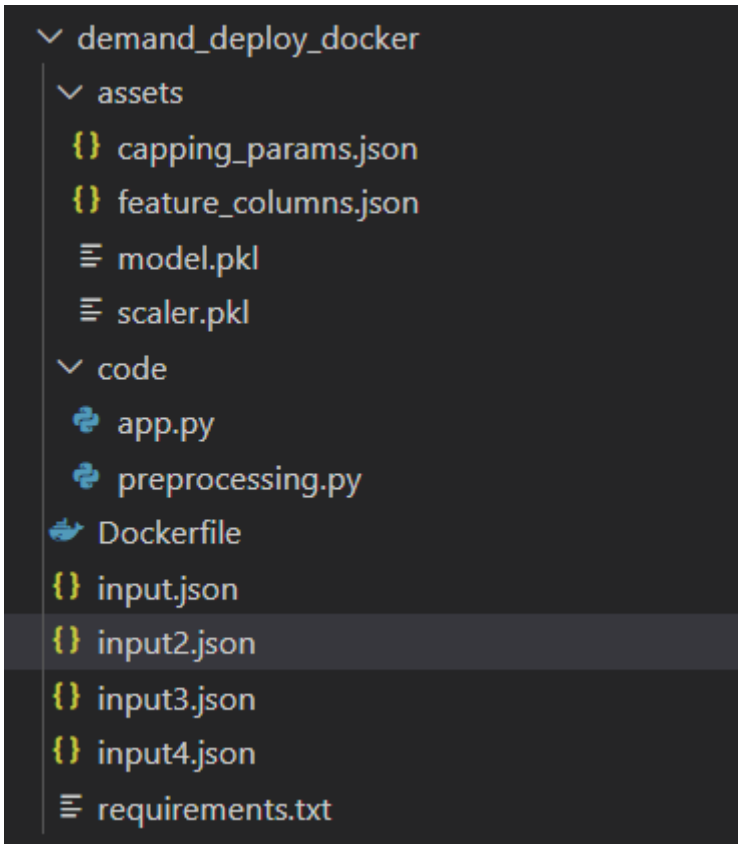
Arsitektur ini mengubah model machine learning kita dari sebuah objek statis di notebook menjadi sebuah layanan (*service*) yang dinamis, portabel, dan siap untuk diintegrasikan ke dalam aplikasi yang lebih besar.

6.2. Persiapan Artefak untuk Deployment

Tahap ini berfokus pada pengumpulan dan serialisasi semua komponen non-kode yang penting agar model prediksi dapat beroperasi secara independen di lingkungan produksi. Artefak utama yang disiapkan meliputi model *Machine Learning* yang telah terlatih (disimpan dalam format persisten seperti .pkl atau .json) dan objek *preprocessor* data (seperti ColumnTransformer atau Pipeline yang telah di-*fit*) yang diperlukan untuk transformasi data input baru. Selain itu, skrip API prediksi (misalnya menggunakan Flask

atau FastAPI) dan *dependency list* (requirements.txt) juga dikemas untuk memastikan konsistensi lingkungan saat di-*deploy* menggunakan Docker.

Berikut struktur folder/artefak demand_deploy_docker untuk mendeploy atau menghubungkan model ke docker:



6.2.1. Folder Assets

1. capping_params.json

Dalam proses pra-pemrosesan data Machine Learning, penanganan outlier (nilai-nilai ekstrem yang jauh dari mayoritas data) merupakan langkah penting. Outlier dapat mendistorsi hubungan antar fitur, memengaruhi statistik deskriptif, dan pada akhirnya menurunkan kinerja model. Salah satu metode penanganan outlier adalah capping, yaitu membatasi nilai-nilai ekstrem pada batas atas atau batas bawah tertentu.

File capping_params.json berfungsi sebagai penyimpanan parameter-parameter kunci yang digunakan untuk melakukan operasi outlier capping ini secara konsisten, baik selama pelatihan model maupun saat model digunakan untuk membuat prediksi pada data baru.

a. Apa itu Outlier Capping?

Outlier capping adalah teknik di mana nilai-nilai data yang berada di luar ambang batas yang ditentukan (baik di bawah batas bawah atau di atas batas atas) diganti dengan nilai ambang batas itu sendiri. Metode yang umum digunakan untuk menentukan ambang batas ini adalah metode Interquartile Range (IQR).

Metode IQR:

- Q1 (Kuartil Pertama): Titik data di mana 25% data berada di bawahnya.
- Q3 (Kuartil Ketiga): Titik data di mana 75% data berada di bawahnya.
- IQR (Interquartile Range): Selisih antara Q3 dan Q1 (IQR = Q3 - Q1).
- Batas Bawah (Lower Bound): Dihitung sebagai Q1 - (1.5 * IQR).
- Batas Atas (Upper Bound): Dihitung sebagai Q3 + (1.5 * IQR).

Nilai data apa pun yang lebih kecil dari batas bawah akan diganti dengan batas bawah, dan nilai apa pun yang lebih besar dari batas atas akan diganti dengan batas atas.

b. Tujuan File capping_params.json

File capping_params.json menyimpan parameter-parameter ini untuk setiap fitur numerik yang dikenai capping. Tujuannya adalah untuk:

1. Konsistensi Pra-pemrosesan: Sama seperti scaler.pkl, file ini menjamin bahwa setiap data input baru yang masuk ke model akan menjalani proses outlier capping menggunakan ambang batas yang sama persis dengan yang digunakan selama pelatihan model. Hal ini sangat penting karena batas-batas capping dihitung berdasarkan distribusi data pelatihan.
2. Menghindari Data Leakage: Parameter capping (Q1, Q3, IQR, batas bawah, batas atas) hanya boleh dihitung dari data pelatihan (train_df). Jika parameter ini dihitung ulang dari seluruh data setiap kali (termasuk data pengujian atau data baru yang tidak terlihat), itu akan menyebabkan data leakage, di mana informasi tentang distribusi data pengujian atau data masa depan secara tidak langsung memengaruhi pra-pemrosesan data pelatihan, menghasilkan evaluasi kinerja model yang bias.
3. Deployment yang Robust: Dalam lingkungan produksi, ketika model menerima data real-time, tidak mungkin untuk menghitung ulang Q1, Q3, dan IQR untuk setiap batch data. File capping_params.json menyediakan batas-batas yang sudah ditentukan sebelumnya, memungkinkan aplikasi untuk segera menerapkan capping tanpa perlu analisis statistik tambahan.

c. Struktur dan Isi File capping_params.json

File capping_params.json adalah file JSON (JavaScript Object Notation) yang menyimpan parameter capping dalam format yang mudah dibaca dan diakses. Setiap kunci dalam objek JSON ini adalah nama fitur numerik yang dikenai capping, dan nilainya adalah

objek lain yang berisi Q1, Q3, IQR, batas bawah, dan batas atas yang dihitung untuk fitur tersebut.

Contoh strukturnya dapat terlihat seperti ini:


```
{
    "Units Sold": {
        "Q1": -0.655252113097028,
        "Q3": 0.5949116719577067,
        "IQR": 1.2501637850547347,
        "lower_bound": -2.53049779067913,
        "upper_bound": 2.4701573495398086
    },
    "Price": {
        "Q1": -0.9004101754336651,
        "Q3": 0.7271632503694256,
        "IQR": 1.6275734258030907,
        "lower_bound": -3.3417703141383015,
        "upper_bound": 3.168523389074062
    },
    "Inventory Level": {
        "Q1": -0.7243115310820055,
        "Q3": 0.4941848695250909,
        "IQR": 1.2184964006070964,
        "lower_bound": -2.55205613199265,
        "upper_bound": 2.3219294704357356
    },
    "Units Ordered": {
        "Q1": -0.548575691794665,
        "Q3": 0.2087968728562292,
        "IQR": 0.7573725646508942,
        "lower_bound": -1.6846345387710064,
        "upper_bound": 1.3448557198325706
    },
    "Competitor Pricing": {
        "Q1": -0.8940731988783659,
        "Q3": 0.7025764061039771,
        "IQR": 1.596649604982343,
        "lower_bound": -3.2890476063518808,
        "upper_bound": 3.097550813577492
    },
    "RETAIL SALES": {
        "Q1": 0.0,
        "Q3": 0.0,
        "IQR": 0.0,
        "lower_bound": 0.0,
        "upper_bound": 0.0
    },
    "price": {
        "Q1": 0.0,
        "Q3": 0.0,
        "IQR": 0.0,
        "lower_bound": 0.0,
        "upper_bound": 0.0
    },
    "discount_percentage": {
        "Q1": 0.0,
        "Q3": 0.0,
        "IQR": 0.0,
        "lower_bound": 0.0,
        "upper_bound": 0.0
    },
    "sales_revenue": {
        "Q1": 0.0,
        "Q3": 0.0,
        "IQR": 0.0,
        "lower_bound": 0.0,
```

```
        "upper_bound": 0.0
    }
}
```

Dalam proyek ini, fitur-fitur seperti Units Sold, Price, Inventory Level, Units Ordered, Competitor Pricing, RETAIL SALES, price, discount_percentage, dan sales_revenue adalah beberapa contoh kolom yang parameter capping-nya disimpan dalam file ini.

d. Proses Penggunaan saat Inferensi

Ketika model di-deploy dan menerima data baru untuk prediksi:

1. Muat capping_params.json: Aplikasi akan memuat file JSON ini untuk mendapatkan parameter capping yang telah dihitung.
2. Aplikasikan Capping: Untuk setiap fitur numerik dalam data input baru yang memiliki parameter di capping_params.json, aplikasi akan mengambil batas bawah dan batas atas yang sesuai. Kemudian, nilai-nilai dalam fitur tersebut yang berada di luar batas akan diganti dengan nilai batas yang relevan. Misalnya, jika 'Units Sold' baru adalah 250 (melebihi upper_bound 198.0), maka nilai ini akan diubah menjadi 198.0.

Dengan demikian, capping_params.json memastikan bahwa semua data yang masuk ke model memiliki rentang nilai yang terkendali, mengurangi sensitivitas model terhadap outlier yang tidak wajar, dan membantu model memberikan prediksi yang lebih stabil dan akurat.

2. feature_columns.json

Dalam pengembangan dan deployment model Machine Learning, konsistensi antara data yang digunakan untuk melatih model dan data yang digunakan untuk membuat prediksi (inferensi) adalah kunci utama keberhasilan. Jika model dilatih dengan fitur-fitur tertentu dalam urutan tertentu, maka model tersebut hanya akan membuat prediksi yang akurat jika menerima data inferensi dengan fitur-fitur yang sama dan dalam urutan yang sama pula. Di sinilah peran file feature_columns.json menjadi sangat vital.

a. Apa itu feature_columns.json?

File feature_columns.json adalah file JSON (JavaScript Object Notation) yang berisi daftar lengkap nama-nama kolom fitur yang digunakan untuk melatih model Machine Learning. Secara esensial, ini adalah "cetak biru" dari tata letak data X_train (DataFrame fitur pelatihan) yang dipelajari oleh model.

Ini adalah daftar (list) dari semua kolom yang menjadi input bagi model yang telah dilatih. Daftar ini mencakup fitur numerik asli, fitur hasil rekayasa (seperti fitur berbasis

waktu, fitur lag, fitur rolling window, dan fitur hasil capping), serta fitur-fitur yang dihasilkan dari One-Hot Encoding untuk variabel kategorikal.

b. Tujuan dan Pentingnya File feature_columns.json

File feature_columns.json memiliki beberapa tujuan krusial dalam pipeline Machine Learning, terutama saat deployment:

1. **Menjamin Konsistensi Fitur**: Model Machine Learning, terutama model berbasis pohon seperti LightGBM, sangat sensitif terhadap fitur yang menjadi inputnya. Jika jumlah fitur, nama fitur, atau urutan fitur antara data pelatihan dan data inferensi berbeda, model tidak akan dapat membuat prediksi yang benar atau bahkan bisa gagal total (mengalami error). File ini memastikan bahwa setiap data input baru yang akan diprediksi oleh model akan diselaraskan agar memiliki set fitur yang sama persis dengan yang diharapkan model, baik dalam hal nama fitur maupun urutan fitur.
2. **Menangani Ketidaksesuaian Fitur saat Inferensi**: Pada lingkungan produksi, data input yang masuk untuk prediksi mungkin tidak selalu sempurna. Ada beberapa skenario yang dapat terjadi:
    a. Fitur Hilang: Beberapa fitur yang diharapkan model mungkin tidak ada dalam data input baru.
    b. Fitur Berlebihan: Data input baru mungkin mengandung fitur-fitur yang tidak dikenal atau tidak digunakan selama pelatihan model.
    c. Kategori Baru/Hilang: Setelah One-Hot Encoding, data inferensi mungkin memiliki kategori baru yang tidak terlihat selama pelatihan, atau kehilangan kategori yang ada selama pelatihan (sehingga kolom one-hot-nya tidak akan muncul).

    File feature_columns.json menyediakan referensi standar untuk mengatasi semua skenario ini dengan memungkinkan kode pra-pemrosesan untuk:

    - Menambahkan fitur yang hilang (biasanya diisi dengan nilai default seperti 0, terutama untuk fitur hasil One-Hot Encoding).
    - Menghilangkan fitur yang berlebihan yang tidak dikenal oleh model.
    - Memastikan urutan kolom yang benar sesuai dengan urutan yang digunakan saat pelatihan.
3. **Memudahkan Pengembangan dan Pemeliharaan**: Dengan menyimpan daftar fitur dalam satu file terpusat, pengembang dapat dengan mudah memahami fitur-fitur apa saja yang menjadi input model tanpa harus menganalisis kode pelatihan secara

mendalam setiap saat. Ini juga memudahkan jika ada perubahan pada fitur di masa mendatang.

c. Struktur dan Isi File feature_columns.json dalam Proyek Ini

File feature_columns.json adalah daftar Python sederhana yang dikonversi menjadi format JSON. Setiap elemen dalam daftar adalah string yang merupakan nama kolom fitur. Contohnya akan terlihat seperti ini (bagian dari daftar):

```
[
    "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",
    "region_Europe",
    "region_North America",
    "store_type_Retail",
    "store_type_Wholesale",
    "category_Cabinets",
    "category_Chairs",
    "category_Sofas",
    "category_Tables",
    "Units Sold_capped",
    "Price_capped",
    "Inventory Level_capped",
    "Units Ordered_capped",
    "Competitor Pricing_capped",
    "RETAIL SALES_capped",
    "price_capped",
    "discount_percentage_capped",
    "sales_revenue_capped",
```

```
"year",
"month",
"day",
"dayofweek",
"dayofyear",
"weekofyear",
"quarter",
"demand_lag_7",
"demand_lag_28",
"demand_rolling_mean_7",
"demand_rolling_mean_28",
"Store ID_S001",
"Store ID_S002",
"Store ID_S003",
"Store ID_S004",
"Store ID_S005",
"Product ID_P0001",
"Product ID_P0002",
"Product ID_P0003",
"Product ID_P0004",
"Product ID_P0005",
"Product ID_P0006",
"Product ID_P0007",
"Product ID_P0008",
"Product ID_P0009",
"Product ID_P0010",
"Product ID_P0011",
"Product ID_P0012",
"Product ID_P0013",
"Product ID_P0014",
"Product ID_P0015",
"Product ID_P0016",
"Product ID_P0017",
"Product ID_P0018",
"Product ID_P0019",
"Product ID_P0020",
"Category_Clothing",
"Category_Electronics",
"Category_Furniture",
"Category_Groceries",
"Category_Toys",
"Region_East",
"Region_North",
```

```
    "Region_South",
    "Region_West",
    "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"
]
```

d. Proses Penggunaan saat Inferensi

Dalam aplikasi deployment (misalnya, app.py yang menggunakan preprocessing.py):

1. Muat feature_columns.json: Daftar fitur yang diharapkan model akan dimuat dari file ini.
2. Pra-pemrosesan Data Input: Setelah data input mentah melalui semua langkah pra-pemrosesan (rekayasa fitur berbasis waktu, capping outlier, one-hot encoding), DataFrame yang dihasilkan mungkin tidak memiliki kolom yang persis sama atau dalam urutan yang sama dengan feature_columns.
3. Penyelarasan Kolom: Kode pra-pemrosesan akan menggunakan feature_columns untuk:
    a. Menambahkan kolom-kolom yang ada di feature_columns tetapi tidak ada di DataFrame input (mengisi dengan 0).
    b. Menghapus kolom-kolom yang ada di DataFrame input tetapi tidak ada di feature_columns.
    c. Mengurutkan semua kolom DataFrame input agar sesuai dengan urutan yang ada di feature_columns.

Dengan demikian, feature_columns.json adalah komponen vital yang memastikan bahwa model menerima data input dalam format yang konsisten dan akurat, yang pada akhirnya berkontribusi pada prediksi yang andal dan robust di lingkungan produksi.

3. model.pkl dan scaler.pkl

Dalam proses pengembangan model Machine Learning, menyimpan aset-aset penting yang dihasilkan selama tahap pelatihan adalah langkah krusial untuk memastikan reproduktibilitas, konsistensi, dan kemampuan deployment. Dua aset utama yang sering

disimpan adalah model yang sudah terlatih dan objek penskalaan (scaler) yang digunakan dalam tahap pra-pemrosesan data. Dalam proyek ini, aset-aset tersebut disimpan dalam format file .pkl (pickle).

a. File Model (model.pkl)

File model.pkl adalah representasi serial dari model Machine Learning yang telah selesai dilatih. Dalam konteks Python, serialisasi adalah proses mengubah objek Python (termasuk objek model yang kompleks) menjadi aliran byte yang dapat disimpan dalam sebuah file. Proses ini dikenal sebagai pickling. Ketika file ini dimuat kembali, aliran byte tersebut di-deserialisasi (unpickling) untuk merekonstruksi objek model yang persis sama di memori.

Tujuan dalam Proyek Machine Learning

Setelah model LightGBM Regressor berhasil dilatih menggunakan data historis (X_train, y_train) dan diidentifikasi sebagai model dengan performa terbaik berdasarkan metrik evaluasi (RMSE terendah dan R-squared tertinggi), sangat tidak efisien untuk melatihnya kembali setiap kali kita ingin membuat prediksi baru. File model.pkl berfungsi untuk:

1. Persistensi Model: Menyimpan semua parameter internal, bobot, dan struktur yang dipelajari oleh model LightGBM selama pelatihan. Ini mencakup konfigurasi pohon-pohon keputusan yang membentuk model LightGBM.
2. Efisiensi Deployment: Memungkinkan model untuk dimuat ke dalam lingkungan produksi (misalnya, server API) hanya dalam hitungan detik, tanpa perlu mengulang proses pelatihan yang memakan waktu dan sumber daya komputasi yang intensif.
3. Reproduktibilitas: Menjamin bahwa model yang digunakan untuk membuat prediksi di masa depan adalah model yang sama persis yang telah divalidasi dan dievaluasi, sehingga memastikan konsistensi dalam output prediksi.

Secara spesifik, file model.pkl untuk model LightGBM Regressor dalam proyek ini akan berisi representasi biner dari:

- Struktur Pohon-Pohon Keputusan: Detail setiap pohon keputusan yang dibangun oleh algoritma LightGBM, termasuk fitur yang digunakan di setiap node split, ambang batas split, dan nilai prediksi di daun-daun pohon.
- Bobot Ensemble: Bagaimana output dari setiap pohon digabungkan untuk menghasilkan prediksi akhir.
- Hyperparameter: Pengaturan atau konfigurasi yang digunakan saat melatih model (misalnya, jumlah estimator/pohon, learning rate, kedalaman pohon, dll.).

b. File Scaler (scaler.pkl)

File scaler.pkl adalah representasi serial dari objek StandardScaler (atau scaler lainnya seperti MinMaxScaler) yang telah dilatih pada data pelatihan. StandardScaler adalah teknik penskalaan fitur yang mengubah data sedemikian rupa sehingga memiliki rata-rata

nol dan standar deviasi satu (distribusi normal standar). Mirip dengan model, objek scaler ini juga di-pickling untuk disimpan dan di-unpickling untuk dimuat kembali.

Tujuan dalam Pra-pemrosesan Data

Penskalaan fitur adalah langkah pra-pemrosesan yang sangat penting untuk banyak algoritma Machine Learning, termasuk model berbasis gradien seperti LightGBM, meskipun model pohon umumnya tidak terlalu sensitif terhadap penskalaan. Tujuan utama dari menyimpan scaler.pkl adalah untuk:

1. Konsistensi Pra-pemrosesan: Memastikan bahwa data baru yang akan diprediksi oleh model menjalani transformasi penskalaan yang sama persis dengan yang diterapkan pada data pelatihan. Ini sangat penting karena scaler mempelajari parameter transformasi (rata-rata dan standar deviasi) dari data pelatihan. Jika data baru diskalakan dengan parameter yang berbeda, prediksi model akan menjadi tidak akurat.
2. Menghindari Data Leakage: Dalam proses pelatihan, StandardScaler hanya di-fit pada data pelatihan (X_train). Parameter (rata-rata dan standar deviasi) yang dipelajari dari X_train inilah yang kemudian digunakan untuk mentransformasi X_train itu sendiri, serta X_test (data yang belum pernah dilihat model). Dengan menyimpan objek scaler yang sudah dilatih, kita dapat menerapkan transformasi ini secara konsisten pada setiap data input di lingkungan produksi, tanpa memperkenalkan bias atau data leakage.

File scaler.pkl untuk StandardScaler berisi:

- Rata-rata (Mean): Nilai rata-rata dari setiap fitur numerik yang diskalakan, dihitung dari data pelatihan (X_train).
- Standar Deviasi (Standard Deviation): Nilai standar deviasi dari setiap fitur numerik yang diskalakan, dihitung dari data pelatihan (X_train).
- Hubungan antara model.pkl dan scaler.pkl dalam Proses Prediksi

Ketika model peramalan permintaan ini di-deploy dan menerima data input baru untuk diprediksi, langkah-langkahnya adalah sebagai berikut:

1. Muat Aset: Aplikasi pertama-tama akan memuat model.pkl dan scaler.pkl (beserta aset pra-pemrosesan lainnya seperti capping_params.json dan feature_columns.json) ke dalam memorinya.
2. Pra-pemrosesan Data Input: Data input mentah yang baru akan melalui serangkaian transformasi yang sama persis dengan yang dilakukan selama pelatihan:
    a. Rekayasa fitur berbasis waktu (tahun, bulan, hari, dll.).
    b. Penanganan outlier menggunakan parameter yang disimpan di capping_params.json.

c. One-Hot Encoding untuk fitur kategorikal.
d. Penskalaan Numerik: Objek StandardScaler yang dimuat dari scaler.pkl akan digunakan untuk mentransformasi fitur-fitur numerik dalam data input baru. Ini memastikan fitur-fitur tersebut berada dalam skala yang sama dengan yang diharapkan oleh model.pkl.

3. Prediksi Model: Setelah data input baru selesai di-pra-proses, data tersebut akan dimasukkan ke dalam model yang dimuat dari model.pkl untuk menghasilkan prediksi permintaan.

Dengan adanya kedua file ini (dan aset pra-pemrosesan lainnya), kita memastikan bahwa seluruh pipeline dari data mentah hingga prediksi model dapat direplikasi dan dijalankan secara konsisten dan efisien dalam lingkungan produksi.

6.2.2. Folder App

1. preprocessing.py

Script preprocessing.py adalah komponen fundamental dalam arsitektur deployment model Machine Learning ini. Fungsinya adalah untuk memastikan bahwa setiap data input mentah yang diterima oleh model untuk prediksi diubah ke dalam format yang sama persis dengan data yang digunakan selama proses pelatihan. Konsistensi ini sangat krusial agar model dapat membuat prediksi yang akurat dan dapat diandalkan.

Berikut script-nya:

```
import pandas as pd
import numpy as np
import joblib
import json
import os

def load_preprocessing_assets(assets_path):
    scaler = joblib.load(os.path.join(assets_path, 'scaler.pkl'))
    with open(os.path.join(assets_path, 'capping_params.json'), 'r') as f:
        capping_params = json.load(f)
    with open(os.path.join(assets_path, 'feature_columns.json'), 'r') as f:
        feature_columns = json.load(f)
    print("Preprocessing assets loaded successfully.")
    return scaler, capping_params, feature_columns

def apply_preprocessing(df, preprocessing_assets):
    scaler, capping_params, feature_columns = preprocessing_assets

    processed_df = df.copy()

    if 'Date' not in processed_df.columns:
        raise KeyError("Input data must contain the 'Date' column for temporal
feature extraction.")

    print("\n--- 1. PREPROCESSING STAGE: FEATURE ENGINEERING ---")

    # DAFTAR KOLOM NUMERIK MENTAH (yang harus diubah dari 'object' ke 'float')
```

```
    numerical_raw_cols = ['Units Sold', 'Price', 'Inventory Level', 'Units
Ordered', 'Competitor Pricing', 'RETAIL SALES', 'price', 'discount_percentage',
'sales_revenue',
                          'Promotion', 'Epidemic', 'holiday_season',
'promotion_applied', 'competitor_price_index', 'economic_index', 'weather_impact',
'Discount']

    # Konversi eksplisit kolom numerik (karena di app.py kita paksa jadi 'object')
    for col in numerical_raw_cols:
        if col in processed_df.columns:
            # Gunakan to_numeric dengan errors='coerce' untuk keamanan konversi
dari string/object
            processed_df[col] = pd.to_numeric(processed_df[col],
errors='coerce').astype(float)

    print("   -> Kolom numerik mentah dikonversi dari object ke float/numerik.")

    # Konversi Tipe Data
    processed_df['Date'] = pd.to_datetime(processed_df['Date'])
    processed_df['Product ID'] = processed_df['Product ID'].astype(str)
    if 'Store ID' in processed_df.columns: processed_df['Store ID'] =
processed_df['Store ID'].astype(str)
    print("   -> Tipe data kategorikal dan Date dikonversi.")

    # Ekstraksi Fitur Temporal
    processed_df['year'] = processed_df['Date'].dt.year
    processed_df['month'] = processed_df['Date'].dt.month
    processed_df['day'] = processed_df['Date'].dt.day
    processed_df['dayofweek'] = processed_df['Date'].dt.dayofweek
    processed_df['dayofyear'] = processed_df['Date'].dt.dayofyear
    # Pastikan weekofyear menggunakan isocalendar yang aman di Pandas > 2.0
    processed_df['weekofyear'] =
processed_df['Date'].dt.isocalendar().week.astype(int)
    processed_df['quarter'] = processed_df['Date'].dt.quarter
    print("   -> 7 Fitur Temporal diekstrak.")

    # Inisiasi Lag Features (Jika tidak ada di input)
    if 'demand_lag_7' not in processed_df.columns: processed_df['demand_lag_7'] = 0
    if 'demand_lag_28' not in processed_df.columns: processed_df['demand_lag_28'] =
0
    if 'demand_rolling_mean_7' not in processed_df.columns:
processed_df['demand_rolling_mean_7'] = 0
    if 'demand_rolling_mean_28' not in processed_df.columns:
processed_df['demand_rolling_mean_28'] = 0


    # --- 2. Capping ---
    print("\n--- 2. PREPROCESSING STAGE: OUTLIER HANDLING (CAPPING) ---")

    for col, params in capping_params.items():
        if col in processed_df.columns:
            processed_df[col] = processed_df[col].clip(lower=params['lower_bound'],
upper=params['upper_bound'])
            processed_df[f'{col}_capped'] = processed_df[col]

    print(f"   -> Capping diterapkan pada {len(capping_params)} kolom numerik.")

    # --- 3. Encoding dan Penyesuaian Kolom ---
    print("\n--- 3. PREPROCESSING STAGE: ENCODING & SCALING ---")

    # Pastikan kolom boolean diubah ke integer (seperti Epidemic, Promotion, dll)
    for col in ['Promotion', 'Epidemic', 'holiday_season', 'promotion_applied',
'weather_impact']:
        if col in processed_df.columns:
            processed_df[col] = processed_df[col].astype(int)

    categorical_features_for_encoding = ['Store ID', 'Product ID', 'Category',
'Region', 'Weather Condition', 'Seasonality', 'SUPPLIER', 'ITEM TYPE']
```

```
    processed_df = pd.get_dummies(processed_df,
columns=categorical_features_for_encoding, dummy_na=False)
    print(f"   -> One-Hot Encoding diterapkan pada
{len(categorical_features_for_encoding)} fitur kategorikal.")

    # Menyesuaikan kolom agar sesuai dengan feature_columns.json (penting untuk
model)
    missing_cols = set(feature_columns) - set(processed_df.columns)
    for c in missing_cols:
        processed_df[c] = 0

    extra_cols = set(processed_df.columns) - set(feature_columns)
    processed_df = processed_df.drop(columns=list(extra_cols))

    # Scaling (hanya kolom yang ada di processed_df dan feature_columns)

    # Ambil hanya kolom yang benar-benar digunakan model
    processed_df = processed_df[feature_columns]

    numerical_features_to_scale_for_deployment = scaler.feature_names_in_.tolist()

    cols_to_transform = [col for col in numerical_features_to_scale_for_deployment
if col in processed_df.columns]

    processed_df[cols_to_transform] =
scaler.transform(processed_df[cols_to_transform])
    print(f"   -> {len(cols_to_transform)} Fitur numerik di-Scaling menggunakan
StandardScaler.")

    # Hapus kolom Date (yang tidak lagi diperlukan setelah fitur temporal
diekstrak)
    if 'Date' in processed_df.columns:
        processed_df = processed_df.drop(columns=['Date'])

    print("\nPreprocessing berhasil diterapkan dan data siap untuk prediksi.")
    return processed_df
```

Script ini terdiri dari dua fungsi utama: load_preprocessing_assets dan apply_preprocessing.

a. Fungsi load_preprocessing_assets(assets_path)

`Fungsi ini bertanggung jawab untuk memuat semua aset yang diperlukan untuk proses pra-pemrosesan dari direktori yang ditentukan (assets_path). Aset-aset ini adalah hasil dari tahapan pra-pemrosesan yang dilakukan selama pelatihan model.

Aset yang Dimuat:

- scaler.pkl: Objek StandardScaler yang telah dilatih pada data pelatihan. Digunakan untuk menstandardisasi fitur numerik.
- capping_params.json: Parameter-parameter yang digunakan untuk outlier capping (batas bawah dan batas atas) untuk setiap fitur numerik yang relevan. Ini memastikan bahwa penanganan outlier konsisten dengan yang dilakukan saat pelatihan.
- feature_columns.json: Daftar nama-nama kolom fitur yang diharapkan oleh model yang telah dilatih, dalam urutan yang benar. Ini esensial untuk menyelaraskan data input agar cocok dengan input model.

Fungsi ini menggunakan joblib.load() untuk memuat objek Python dari file .pkl dan json.load() untuk memuat parameter dari file .json. Hasilnya adalah objek scaler, kamus parameter capping, dan daftar nama kolom fitur yang siap digunakan oleh fungsi apply_preprocessing.

b. Fungsi apply_preprocessing(df, preprocessing_assets)

Fungsi inti ini mengambil DataFrame input mentah (df) dan aset pra-pemrosesan yang telah dimuat (preprocessing_assets), kemudian menerapkan serangkaian transformasi yang telah ditentukan untuk menyiapkan data agar siap digunakan oleh model.

Langkah-langkah Pra-pemrosesan (sesuai urutan dalam script):

Validasi Kolom Date: Script pertama-tama memeriksa keberadaan kolom Date. Jika kolom ini tidak ada, ia akan memicu KeyError karena fitur temporal sangat bergantung padanya.

1. **Konversi Kolom Numerik Mentah:**
    a. Identifikasi: Mendefinisikan numerical_raw_cols yang mencakup kolom-kolom seperti Units Sold, Price, Inventory Level, Promotion, Epidemic, dll. Ini adalah kolom-kolom yang mungkin diterima sebagai tipe data 'object' (string) dari input JSON, tetapi seharusnya bersifat numerik.
    b. Transformasi: Setiap kolom dalam daftar ini dikonversi secara eksplisit menjadi tipe float menggunakan pd.to_numeric(errors='coerce'). errors='coerce' akan mengganti nilai yang tidak valid (misalnya, string yang bukan angka) dengan NaN, yang kemudian harus ditangani di tahap selanjutnya jika memang ada, namun dalam skema ini diharapkan input valid.
    c. Pentingnya: Langkah ini mengatasi potensi inkonsistensi tipe data saat data diterima melalui API, memastikan bahwa semua data numerik diperlakukan sebagai angka.
2. **Konversi Tipe Data Dasar:**
    a. Kolom Date dikonversi ke format datetime menggunakan pd.to_datetime().
    b. Kolom Product ID dan Store ID (jika ada) dikonversi ke tipe string untuk memastikan konsistensi dalam proses One-Hot Encoding.
    c. Pentingnya: Memastikan tipe data yang benar adalah fondasi untuk ekstraksi fitur dan encoding yang akurat.
3. **Ekstraksi Fitur Temporal**: Mirip dengan Langkah 5 dalam pipeline pelatihan, fungsi ini mengekstrak tujuh fitur berbasis waktu dari kolom Date:
    a. year, month, day, dayofweek, dayofyear, weekofyear, quarter.

b. Pentingnya: Fitur-fitur ini menangkap pola musiman dan temporal yang penting bagi model peramalan.

4. **Penanganan Fitur Lag dan Rolling Window yang Hilang:**
    a. Inisiasi: Jika kolom demand_lag_7, demand_lag_28, demand_rolling_mean_7, atau demand_rolling_mean_28 tidak ada dalam data input (misalnya, untuk prediksi hari pertama atau untuk data di mana historisnya belum tersedia), kolom-kolom ini diinisialisasi dengan nilai 0.
    b. Pentingnya: Ini meniru perilaku fillna(0) yang diterapkan pada fitur lag dan rolling window selama pelatihan, memastikan bahwa model menerima input yang konsisten meskipun data historis untuk perhitungan lag/rolling belum ada dalam input langsung.
5. **Outlier Capping:**
    a. Aplikasi: Fungsi ini mengiterasi melalui parameter capping yang dimuat dari capping_params.json. Untuk setiap fitur yang ada di processed_df dan memiliki parameter capping, nilai-nilai di luar batas bawah atau atas akan "di-cap" dengan nilai batas tersebut menggunakan .clip().
    b. Pentingnya: Mengurangi dampak negatif outlier pada prediksi model, menjamin bahwa model tidak membuat prediksi yang ekstrem berdasarkan data input yang ekstrem atau tidak biasa.
6. **Encoding Fitur Kategorikal dan Penyesuaian Kolom**:
    a. Konversi Boolean-like: Beberapa kolom (Promotion, Epidemic, holiday_season, promotion_applied, weather_impact) yang mungkin diterima sebagai boolean atau objek dari JSON, dikonversi secara eksplisit ke int (0/1). Kolom-kolom boolean dari df3 (region_Europe, dll.) juga dipastikan tipenya bool dan diisi False jika tidak ada.
    b. One-Hot Encoding: Fitur-fitur kategorikal yang telah ditentukan (seperti Store ID, Product ID, Category, Region, Weather Condition, Seasonality, SUPPLIER, ITEM TYPE) diubah menjadi format numerik biner menggunakan pd.get_dummies().
    c. Penyelarasan Kolom dengan feature_columns: Ini adalah langkah paling krusial untuk konsistensi.
        i. missing_cols: Kolom-kolom yang ada di feature_columns tetapi tidak ada di processed_df akan ditambahkan dan diisi dengan 0.

ii. extra_cols: Kolom-kolom yang ada di processed_df tetapi tidak ada di feature_columns akan dihapus.

iii. Pengurutan: processed_df kemudian diurutkan agar kolom-kolomnya berada dalam urutan yang sama persis dengan feature_columns.

d. Pentingnya: Langkah ini memastikan bahwa DataFrame input memiliki dimensi dan urutan kolom yang persis sama dengan yang diharapkan oleh model yang telah dilatih, mencegah kesalahan saat prediksi.

7. **Penskalaan Numerik:**
   a. Aplikasi: Fitur-fitur numerik yang relevan (daftar fitur yang sebelumnya digunakan untuk melatih StandardScaler) ditransformasikan menggunakan objek scaler yang telah dimuat dari scaler.pkl. Metode scaler.transform() digunakan untuk menerapkan penskalaan.
   b. Pentingnya: Menempatkan semua fitur numerik pada skala yang seragam, yang penting untuk model seperti LightGBM yang dapat dipengaruhi oleh perbedaan skala fitur.
8. **Penghapusan Kolom Date**: Setelah semua fitur temporal diekstrak, kolom Date asli dihapus karena tidak lagi diperlukan sebagai fitur langsung untuk model.

Fungsi ini mengembalikan DataFrame yang telah diproses sepenuhnya, dengan semua fitur dalam format yang benar dan diselaraskan dengan input yang diharapkan oleh model Machine Learning.

Secara keseluruhan, preprocessing.py adalah jembatan vital antara data mentah yang bervariasi dari lingkungan nyata dan format data yang terstruktur serta konsisten yang dibutuhkan oleh model Machine Learning untuk menghasilkan prediksi yang akurat.

2. app.py

Script app.py adalah jantung dari aplikasi API (Application Programming Interface) yang kami bangun untuk menyebarkan (deploy) model peramalan permintaan. Menggunakan kerangka kerja Flask, app.py berfungsi sebagai antarmuka yang memungkinkan sistem eksternal mengirimkan data input dan menerima prediksi permintaan secara real-time. Script ini mengorkestrasi pemuatan model dan aset pra-pemrosesan, penerapan transformasi data, pembuatan prediksi, dan bahkan penyediaan rekomendasi bisnis berbasis prediksi.

Berikut script-nya:

```
import joblib
import pandas as pd
from flask import Flask, request, jsonify
```

```
import os
import numpy as np # PENTING: Menambahkan import numpy

# Import preprocessing functions
from preprocessing import load_preprocessing_assets, apply_preprocessing

# --- Configuration ---
ASSETS_PATH = '../assets'

# --- Load Model and Preprocessing Assets ---
model = joblib.load(os.path.join(ASSETS_PATH, 'model.pkl'))
scaler, capping_params, feature_columns = load_preprocessing_assets(ASSETS_PATH)

# --- Flask App Setup ---
app = Flask(__name__)

# Fungsi untuk memberikan saran (Simulasi logika bisnis)
def generate_recommendations(predictions):
    avg_demand = np.mean(predictions)

    if avg_demand >= 150:
        recommendation = "🔴 **TINGKAT PERMINTAAN TINGGI (HIGH DEMAND)**:
Direkomendasikan untuk segera meningkatkan level inventaris sebesar **15%** dan
meluncurkan kampanye *pre-order* untuk memaksimalkan keuntungan."
    elif avg_demand >= 100:
        recommendation = "🟡 **TINGKAT PERMINTAAN SEDANG (MODERATE DEMAND)**:
Pertahankan inventaris saat ini dan monitor tren. Pertimbangkan promosi minor untuk
meningkatkan volume penjualan."
    else:
        recommendation = "🟢 **TINGKAT PERMINTAAN RENDAH (LOW DEMAND)**: Kurangi
inventaris baru dan fokus pada membersihkan stok lama melalui penawaran diskon yang
menarik."

    summary = f"\n[SUMMARY KEPUTUSAN BISNIS]: Rata-rata Prediksi Permintaan:
{avg_demand:.2f} unit."

    return summary, recommendation

@app.route('/predict', methods=['POST'])
def predict():
    try:
        json_data = request.get_json(force=True)

        # LOGIKA PEMBENTUKAN DATAFRAME PALING AMAN:
        # Menggunakan json_normalize dan memaksa semua kolom menjadi tipe 'object'
(string)
        # untuk mencegah konflik DType DateTime64DType dan Float64DType saat
inisiasi Pandas.
        if isinstance(json_data, list) and json_data:
            input_df = pd.json_normalize(json_data).astype('object')
        else:
            input_df = pd.json_normalize([json_data]).astype('object')

        # --- TAHAP 1: PREPROCESSING ---
        print("\n\n#####################################################")
        print("#                [STARTING PREDICTION]                #")
        print("#####################################################")

        # Output detail preprocessing akan muncul dari preprocessing.py
        processed_input_df = apply_preprocessing(input_df, (scaler, capping_params,
feature_columns))

        # --- TAHAP 2: PREDIKSI ---
        predictions = model.predict(processed_input_df)

        # --- TAHAP 3: HASIL DAN SARAN (OUTPUT KEREN) ---
        summary, recommendation = generate_recommendations(predictions)
```

```
        print("\n\n==========================================================")
        print(f" ✨ HASIL PREDIKSI PERMINTAAN UNTUK {len(predictions)} DATA POINT ✨")
        print("==========================================================")
        print(f"  * Total Data Input (setelah Preprocessing):
{processed_input_df.shape[0]} baris, {processed_input_df.shape[1]} fitur")
        print(f"  * Model Digunakan: LightGBM Regressor")
        print("----------------------------------------------------------")
        print(summary)
        print(recommendation)
        print("==========================================================\n")

        # Mengembalikan prediksi ke user API
        return jsonify(predictions.tolist())

    except Exception as e:
        app.logger.error(f"Prediction Error: {e}")
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

1. **Komponen Utama**
    a. **Import Library:**
        i. joblib: Digunakan untuk memuat model Machine Learning (model.pkl) dan objek scaler (scaler.pkl) yang telah disimpan.
        ii. pandas (sebagai pd): Digunakan untuk manipulasi data, khususnya untuk mengkonversi data input JSON menjadi DataFrame dan untuk keperluan pra-pemrosesan.
        iii. Flask, request, jsonify: Komponen inti dari kerangka kerja web Flask untuk membuat aplikasi web dan API.
        iv. os: Digunakan untuk berinteraksi dengan sistem operasi, seperti mengelola jalur file.
        v. numpy (sebagai np): Digunakan untuk operasi numerik, terutama untuk perhitungan rata-rata prediksi dan penanganan tipe data numerik.
        vi. preprocessing: Modul kustom yang diimpor, berisi fungsi load_preprocessing_assets dan apply_preprocessing yang sangat penting.
    b. **Konfigurasi (ASSETS_PATH)**: Mendefinisikan jalur relatif ke direktori assets tempat semua aset pra-pemrosesan dan model disimpan. Dalam konteks Docker, ../assets menunjukkan bahwa direktori assets berada satu tingkat di atas direktori code tempat app.py dijalankan.
    c. **Pemuatan Model dan Aset Pra-pemrosesan:**

i. Pada saat aplikasi app.py pertama kali dijalankan (saat kontainer Docker dimulai), model yang telah dilatih (model.pkl) dan semua aset pra-pemrosesan (scaler.pkl, capping_params.json, feature_columns.json) dimuat ke dalam memori. Ini adalah proses satu kali yang terjadi saat startup aplikasi.

ii. model = joblib.load(...): Memuat objek LightGBM Regressor yang sudah dilatih.

iii. scaler, capping_params, feature_columns = load_preprocessing_assets(ASSETS_PATH): Memanggil fungsi dari preprocessing.py untuk memuat scaler, parameter capping, dan daftar kolom fitur.

iv. Pentingnya: Pemuatan di awal ini memastikan bahwa model dan semua konfigurasi pra-pemrosesan tersedia secara instan untuk setiap permintaan prediksi yang masuk, tanpa latensi tambahan akibat pemuatan berulang.

d. **Inisialisasi Aplikasi Flask:** app = Flask(__name__) membuat instance aplikasi Flask, yang akan menangani routing permintaan HTTP.

2. **Fungsi Bantu (generate_recommendations)**

a. Tujuan: Fungsi ini mensimulasikan logika bisnis untuk memberikan rekomendasi berdasarkan prediksi permintaan. Ini adalah contoh bagaimana output model dapat diubah menjadi wawasan yang dapat ditindaklanjuti oleh bisnis.

b. Cara Kerja:

i. Menerima predictions (daftar prediksi permintaan) sebagai input.

ii. Menghitung rata-rata permintaan dari prediksi yang diberikan (avg_demand).

iii. Menggunakan logika kondisional (if/elif/else) untuk menentukan rekomendasi berdasarkan nilai avg_demand:

iv. Permintaan Tinggi (>= 150 unit): Rekomendasi untuk meningkatkan inventaris dan kampanye pre-order.

v. Permintaan Sedang (>= 100 unit): Rekomendasi untuk mempertahankan inventaris dan memantau tren.

vi. Permintaan Rendah (< 100 unit): Rekomendasi untuk mengurangi inventaris baru dan fokus pada diskon stok lama.

c. Mengembalikan ringkasan prediksi rata-rata dan rekomendasi yang dihasilkan.

3. **Endpoint API /predict (@app.route('/predict', methods=['POST']))**

Ini adalah inti dari API, menangani permintaan HTTP POST yang datang ke jalur /predict. Berikut alur kerjanya:

a. **Menerima Data Input:**
   i. json_data = request.get_json(force=True): Menerima data JSON dari body permintaan POST.
   ii. Pembentukan DataFrame Aman: Menggunakan pd.json_normalize(json_data).astype('object') adalah pendekatan yang sangat robust. Ini memastikan bahwa data JSON, baik itu objek tunggal atau daftar objek, dikonversi menjadi DataFrame Pandas. Yang paling penting, astype('object') secara eksplisit memaksa semua kolom input menjadi tipe object (string) pada tahap awal. Ini mencegah konflik tipe data yang mungkin terjadi (misalnya, Pandas salah menginterpretasikan kolom 'Date' sebagai numerik atau sebaliknya) dan menyerahkan penanganan tipe data yang benar ke fungsi apply_preprocessing.

b. **Pra-pemrosesan Data:**
   i. processed_input_df = apply_preprocessing(input_df, (scaler, capping_params, feature_columns)): Data input (input_df) diteruskan ke fungsi apply_preprocessing dari modul preprocessing.py. Fungsi ini akan melakukan semua langkah transformasi (ekstraksi fitur temporal, capping outlier, one-hot encoding, penskalaan, dan penyelarasan kolom) menggunakan aset yang telah dimuat.
   ii. Pentingnya: Ini adalah tahap krusial yang menjamin bahwa data input berada dalam format yang benar dan konsisten dengan data pelatihan model.

c. **Prediksi**: predictions = model.predict(processed_input_df): DataFrame yang sudah diproses kemudian dimasukkan ke dalam model LightGBM yang telah dimuat untuk menghasilkan prediksi permintaan.

d. **Generasi Rekomendasi dan Logging:**
   i. summary, recommendation = generate_recommendations(predictions): Memanggil fungsi

generate_recommendations untuk mengubah prediksi mentah menjadi wawasan bisnis.

ii. Pesan-pesan informatif (termasuk ringkasan prediksi dan rekomendasi) dicetak ke konsol server (log Docker). Ini sangat membantu untuk debugging dan pemantauan di lingkungan produksi.

e. **Mengirim Respon JSON**: return jsonify(predictions.tolist()): Prediksi yang dihasilkan (yang merupakan array NumPy) dikonversi menjadi daftar Python dan kemudian diformat menjadi respons JSON menggunakan jsonify, yang kemudian dikirim kembali ke klien yang membuat permintaan.

f. **Penanganan Error**: Blok try-except membungkus seluruh logika prediksi. Jika terjadi kesalahan selama proses (misalnya, format input yang salah, masalah pra-pemrosesan), kesalahan tersebut akan ditangkap, dicatat (app.logger.error), dan respons JSON yang berisi pesan error akan dikembalikan dengan kode status HTTP 500 (Internal Server Error).

4. **Menjalankan Aplikasi (if __name__ == '__main__':)**

a. **app.run(host='0.0.0.0', port=5000):** Baris ini hanya akan dieksekusi jika script app.py dijalankan secara langsung (bukan diimpor sebagai modul). host='0.0.0.0' membuat aplikasi dapat diakses dari luar kontainer Docker, dan port=5000 adalah port default tempat aplikasi Flask akan berjalan.

b. **Pentingnya**: Dalam lingkungan produksi Docker, aplikasi biasanya dijalankan oleh server WSGI yang lebih robust seperti Gunicorn (seperti yang dikonfigurasi dalam Dockerfile), bukan secara langsung oleh app.run(). Namun, blok ini tetap berguna untuk pengujian lokal saat pengembangan.

Secara keseluruhan, app.py menyediakan jembatan yang lengkap dan fungsional antara pengguna (melalui permintaan API HTTP) dan kecerdasan model Machine Learning kami, memungkinkan prediksi permintaan yang efisien dan penyampaian wawasan bisnis yang relevan. Komunikasi yang mulus dengan preprocessing.py memastikan bahwa model selalu menerima data dalam format yang paling optimal.

6.2.3. Dockerfile

Dockerfile adalah kumpulan instruksi yang digunakan untuk membangun sebuah *image* Docker, yang merupakan *template* untuk membuat *container* Docker. Dalam proyek ini, Dockerfile berfungsi untuk mengemas aplikasi Flask, model Machine Learning, semua aset pra-pemrosesan, dan dependensi lainnya ke dalam lingkungan yang terisolasi dan

portable. Ini memastikan bahwa aplikasi peramalan permintaan dapat dijalankan secara konsisten di lingkungan mana pun yang mendukung Docker.

Berikut Script-nya:

```
FROM python:3.11-slim-bullseye

WORKDIR /app

# Install OS dependencies required for LightGBM (libgomp1 for OpenMP runtime)

RUN apt-get update && \

    apt-get install -y libgomp1 && \

    rm -rf /var/lib/apt/lists/*

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY assets ./assets

COPY code ./code

# PENTING: Pindahkan WORKDIR ke dalam folder code

WORKDIR /app/code

EXPOSE 5000

# File app.py berada di /app/code, jadi kita tambahkan /app/code ke PYTHONPATH

ENV PYTHONPATH /app/code:$PYTHONPATH

# Perintah untuk menjalankan Flask application menggunakan Gunicorn

# Sekarang kita bisa merujuk langsung ke nama file 'app'

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
```

Berikut adalah penjelasan setiap baris dan blok dalam Dockerfile:

1. FROM python:3.11-slim-bullseye

- **Tujuan:** Baris ini mendefinisikan *base image* untuk Docker *image* yang akan kita bangun. Artinya, *image* kita akan dibangun di atas sistem operasi dasar (bullseye yang merupakan Debian 11) yang sudah memiliki Python versi 3.11 yang terinstal.
- **slim:** Mengindikasikan bahwa ini adalah versi Python yang lebih ringan, hanya menyertakan komponen-komponen esensial, yang membantu menjaga ukuran *image* Docker tetap kecil. Ini sangat penting untuk efisiensi deployment.
- **bullseye:** Menentukan versi distribusi Debian Linux sebagai dasar sistem operasi.

2. WORKDIR /app

- **Tujuan:** Mengatur direktori kerja (working directory) default di dalam *container* menjadi /app. Semua perintah selanjutnya (seperti COPY, RUN, CMD) akan dieksekusi relatif terhadap direktori ini, kecuali ditentukan lain. Ini membantu menjaga struktur file yang terorganisir di dalam *container*.

3. RUN apt-get update && apt-get install -y libgomp1 && rm -rf /var/lib/apt/lists/*

- **Tujuan:** Menginstal dependensi sistem operasi yang diperlukan oleh LightGBM.
- **libgomp1:** Adalah pustaka OpenMP GNU (GNU Open Multi-Processing). LightGBM (dan beberapa library Machine Learning lain yang berkinerja tinggi) menggunakan OpenMP untuk paralelisme, sehingga pustaka ini harus tersedia di sistem operasi dasar *container* agar LightGBM dapat berjalan dengan baik. Tanpa ini, kita mungkin mengalami error saat LightGBM mencoba berjalan.
- **apt-get update:** Memperbarui daftar paket yang tersedia di repositori sistem.
- **apt-get install -y libgomp1:** Menginstal paket libgomp1 tanpa meminta konfirmasi (-y).
- **rm -rf /var/lib/apt/lists/*:** Membersihkan *cache* paket setelah instalasi. Ini adalah praktik terbaik untuk mengurangi ukuran *image* Docker dengan menghapus file-file sementara yang tidak lagi diperlukan.

4. COPY requirements.txt .

- **Tujuan:** Menyalin file requirements.txt dari direktori *host* (direktori demand_deploy_docker) ke direktori kerja saat ini (/app) di dalam *container*.

5. RUN pip install --no-cache-dir -r requirements.txt

- **Tujuan:** Menginstal semua pustaka Python yang terdaftar dalam requirements.txt.
- **pip install:** Perintah standar untuk menginstal paket Python.
- **--no-cache-dir:** Menginstruksikan pip untuk tidak menyimpan *cache* unduhan paket. Ini juga berkontribusi pada pengurangan ukuran *image* Docker.
- **-r requirements.txt:** Memberi tahu pip untuk menginstal paket-paket dari file requirements.txt.

6. COPY assets ./assets

- **Tujuan:** Menyalin seluruh direktori assets (yang berisi model.pkl, scaler.pkl, capping_params.json, feature_columns.json) dari direktori demand_deploy_docker di *host* ke direktori /app/assets di dalam *container*.

7. COPY code ./code

- **Tujuan:** Menyalin seluruh direktori code (yang berisi app.py dan preprocessing.py) dari direktori demand_deploy_docker di *host* ke direktori /app/code di dalam *container*.

8. WORKDIR /app/code

- **Tujuan: (PENTING)** Baris ini mengubah direktori kerja di dalam *container* dari /app menjadi /app/code. Ini dilakukan karena file app.py yang akan dijalankan berada di dalam direktori code. Dengan mengubah WORKDIR ke /app/code, kita bisa merujuk langsung ke app.py dan preprocessing.py tanpa perlu menambahkan code. di depannya dalam perintah CMD dan ENV.

9. EXPOSE 5000

- **Tujuan:** Menginformasikan Docker bahwa *container* akan mendengarkan koneksi pada *port* 5000 saat dijalankan. Ini adalah *port* tempat aplikasi Flask (melalui Gunicorn) akan melayani permintaan. EXPOSE bersifat deklaratif; untuk benar-benar memetakan *port* ini ke *port* di *host*, kita perlu menggunakan opsi -p saat menjalankan docker run.

10. ENV PYTHONPATH /app/code:$PYTHONPATH

- **Tujuan:** Baris ini memodifikasi variabel lingkungan PYTHONPATH di dalam *container*. Ini menambahkan /app/code ke PYTHONPATH, yang memberi tahu Python di mana harus mencari modul-modul yang diimpor. Ini penting agar app.py dapat mengimpor preprocessing.py (karena preprocessing.py berada di direktori yang sama /app/code).

11. CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]

- **Tujuan:** Baris ini mendefinisikan perintah utama yang akan dieksekusi saat *container* Docker dimulai. Ini adalah perintah yang meluncurkan aplikasi web Flask.

- **gunicorn:** Ini adalah *server* WSGI (Web Server Gateway Interface) yang kuat dan banyak digunakan untuk menjalankan aplikasi Python di lingkungan produksi. Gunicorn lebih robust dan performan daripada *server* pengembangan bawaan Flask.
- **--bind "0.0.0.0:5000":** Menginstruksikan Gunicorn untuk mendengarkan semua antarmuka jaringan (0.0.0.0) pada *port* 5000. Ini penting agar aplikasi dapat diakses dari luar *container*.
- **app:app:** Merujuk pada aplikasi Flask. Bagian pertama (app) adalah nama modul Python (karena kita sudah mengubah WORKDIR ke /app/code, app merujuk ke app.py). Bagian kedua (app) adalah nama objek aplikasi Flask yang diinisialisasi di dalam app.py (yaitu, app = Flask(__name__)).

Kesimpulan dan Peran Dockerfile

Dockerfile ini secara keseluruhan berperan sebagai resep otomatis untuk membuat lingkungan yang konsisten dan terisolasi untuk aplikasi peramalan permintaan. Dengan Dockerfile ini, kita dapat:

- **Reproduksi Lingkungan:** Memastikan bahwa dependensi, konfigurasi, dan kode aplikasi selalu sama, di mana pun *image* Docker dibangun atau *container* dijalankan.
- **Isolasi:** Mengisolasi aplikasi dari sistem *host* dan aplikasi lain, mencegah konflik dependensi.
- **Portabilitas:** Memungkinkan aplikasi untuk dengan mudah dipindahkan dan dijalankan di berbagai lingkungan (lokal, *cloud*, server) yang mendukung Docker.
- **Deployment yang Efisien:** Menyederhanakan proses deployment aplikasi Machine Learning kami menjadi API yang dapat diakses, dengan semua komponen yang diperlukan sudah terpaket rapi dalam satu *image*.

Dengan mengikuti instruksi dalam Dockerfile, proses build akan menghasilkan *image* Docker demand-forecasting-app yang siap untuk di-*deploy* sebagai *container* yang berfungsi penuh, melayani prediksi permintaan melalui endpoint /predict.

6.2.4. input.json

File input.json berfungsi sebagai contoh *payload* atau masukan data yang akan dikirimkan ke endpoint /predict dari API peramalan permintaan yang telah di-*deploy* dalam *container* Docker. File ini merepresentasikan format data mentah yang diharapkan oleh API untuk dapat diproses dan menghasilkan prediksi permintaan. Struktur dan isi input.json

sangat penting karena harus mencerminkan fitur-fitur yang dibutuhkan oleh fungsi pra-pemrosesan (preprocessing.py) sebelum data dimasukkan ke dalam model.

1. Tujuan Utama input.json

1. **Contoh Pengujian API:** Memberikan contoh data yang valid untuk menguji fungsionalitas endpoint /predict. Ini membantu memverifikasi apakah seluruh pipeline (penerimaan data, pra-pemrosesan, prediksi, dan pengembalian hasil) bekerja dengan benar.
2. **Panduan Struktur Input:** Mendemonstrasikan format dan jenis data yang diharapkan oleh API kepada pengembang atau sistem lain yang ingin mengintegrasikan dengan API ini. Hal ini memastikan bahwa data yang dikirimkan oleh klien API sudah sesuai dengan ekspektasi server.
3. **Representasi Data Mentah:** Data dalam input.json adalah data mentah, serupa dengan data yang mungkin diterima dari sistem POS (Point of Sale) atau sistem manajemen inventaris, sebelum melalui langkah-langkah rekayasa fitur dan penskalaan.

2. Struktur dan Isi input.json

File input.json yang gunakan memiliki struktur sebagai berikut:


```
[

    {
        "Date": "2025-01-01",
        "Product ID": "P0001",
        "Store ID": "S001",
        "Category": "Electronics",
        "Region": "North",
        "Weather Condition": "Cloudy",
        "Seasonality": "Winter",
        "SUPPLIER": "Unknown",
        "ITEM TYPE": "Unknown",

        "Units Sold": 76,
        "Price": 73.6,
        "Inventory Level": 288,
        "Units Ordered": 278.0,
        "Competitor Pricing": 86.55,
        "Discount": 0.05,

        "Epidemic": 0,
        "Promotion": 1,
        "RETAIL SALES": 5500.00,
        "holiday_season": 0,
        "promotion_applied": 1,
        "competitor_price_index": 0.95,
        "economic_index": 105.00,
        "weather_impact": 0,
        "price": 120.00,
        "discount_percentage": 0.05,
```

```
        "sales_revenue": 5700.00
    }
]
```

- **Format List of Objects:** Data input dikemas dalam bentuk *list* yang berisi satu atau lebih objek JSON. Setiap objek JSON dalam *list* mewakili satu *record* (satu baris data) yang ingin diprediksi permintaannya. Format ini memungkinkan pengiriman satu atau beberapa titik data dalam satu permintaan API.
- **Koleksi Fitur Input:** Setiap objek JSON berisi pasangan kunci-nilai yang merepresentasikan fitur-fitur mentah yang diperlukan untuk prediksi. Fitur-fitur ini dapat dikelompokkan menjadi beberapa kategori:
    - **Identifikasi dan Temporal:**
        - "Date": "2025-01-01": Tanggal untuk prediksi. Ini adalah fitur krusial yang akan digunakan oleh preprocessing.py untuk mengekstrak fitur berbasis waktu (tahun, bulan, hari, dll.).
        - "Product ID": "P0001", "Store ID": "S001": Pengidentifikasi unik untuk produk dan toko. Akan di-*one-hot encoded*.
    - **Kategorikal Umum:**
        - "Category": "Electronics", "Region": "North", "Weather Condition": "Cloudy", "Seasonality": "Winter": Fitur kategorikal yang akan di-*one-hot encoded*.
        - "SUPPLIER": "Unknown", "ITEM TYPE": "Unknown": Fitur kategorikal dari df2 yang mungkin memiliki nilai 'Unknown' jika tidak ada padanan saat penggabungan data atau jika informasi ini tidak tersedia.
    - **Numerik dan Boolean:**
        - "Units Sold": 76, "Price": 73.6, "Inventory Level": 288, "Units Ordered": 278.0, "Competitor Pricing": 86.55, "Discount": 0.05: Fitur-fitur numerik inti yang merepresentasikan kondisi penjualan dan inventaris.
        - "Epidemic": 0, "Promotion": 1: Fitur biner (0 atau 1) yang mengindikasikan keberadaan epidemi atau promosi. Penting bahwa ini konsisten dengan tipe data int yang diharapkan setelah pra-pemrosesan.
        - "RETAIL SALES": 5500.00, "holiday_season": 0, "promotion_applied": 1, "competitor_price_index": 0.95, "economic_index": 105.00, "weather_impact": 0, "price": 120.00, "discount_percentage": 0.05, "sales_revenue": 5700.00: Fitur-fitur numerik tambahan yang berasal

dari dataset lain (df2, df3), yang juga akan melewati proses *capping* dan penskalaan.

3. Hubungan dengan preprocessing.py

Data dalam input.json ini akan menjadi input untuk fungsi apply_preprocessing di preprocessing.py. Fungsi tersebut akan:

- Memastikan konversi tipe data yang benar (misalnya, Date ke datetime, kolom numerik dari object ke float).
- Mengekstrak fitur berbasis waktu dari Date.
- Menerapkan *outlier capping* pada fitur-fitur numerik berdasarkan parameter yang disimpan di capping_params.json.
- Melakukan One-Hot Encoding pada fitur kategorikal.
- Menangani fitur *lag* dan *rolling mean* dengan mengisi nilai default (0) jika tidak disediakan dalam input.
- Melakukan penskalaan fitur numerik menggunakan scaler.pkl.
- Menyelaraskan semua kolom agar sesuai dengan feature_columns.json (menambahkan kolom yang hilang dengan 0, menghapus kolom ekstra, dan menyusun ulang).

4. Pentingnya Struktur Input yang Benar

Adalah krusial bahwa input.json menyediakan semua fitur yang diperlukan oleh model (atau setidaknya fitur yang dapat di-*infer* atau diisi secara *default* oleh preprocessing.py). Kesalahan dalam format atau ketiadaan fitur kunci dapat menyebabkan kegagalan pra-pemrosesan dan, pada akhirnya, kegagalan prediksi. Data input.json yang diberikan menunjukkan pemahaman yang baik tentang fitur-fitur yang dibutuhkan model dan bagaimana data tersebut harus disajikan kepada API.

6.2.5. requirements.txt

File requirements.txt adalah daftar tekstual yang berisi semua pustaka (libraries) Python beserta versi spesifiknya yang diperlukan oleh suatu proyek. Dalam konteks deployment aplikasi Machine Learning menggunakan Docker, file ini adalah komponen yang sangat fundamental. Ini berfungsi sebagai "daftar belanja" untuk lingkungan Python, memastikan bahwa setiap kali lingkungan proyek dibuat ulang (misalnya, di dalam *container* Docker), semua dependensi yang tepat dan kompatibel akan terinstal.

Berikut isi script-nya:

```
Flask==3.1.2
lightgbm==4.6.0
matplotlib==3.9.0
matplotlib-inline==0.2.1
matplotlib-venn==1.1.2
numpy==1.26.4
pandas==2.0.3
pandas-datareader==0.10.0
pandas-gbq==0.30.0
scikit-learn==1.6.1
seaborn==0.13.2
xgboost==2.0.3
gunicorn
```

1. Tujuan dan Pentingnya requirements.txt

1. **Manajemen Dependensi:** Fungsi utamanya adalah untuk mengelola dependensi proyek. Proyek Python, terutama yang melibatkan Machine Learning, seringkali bergantung pada banyak pustaka pihak ketiga. requirements.txt menyediakan cara deklaratif untuk mencatat semua dependensi ini.
2. **Reproduktibilitas Lingkungan:** Ini adalah salah satu manfaat paling krusial. Dengan menentukan versi spesifik dari setiap pustaka (==X.Y.Z), requirements.txt memastikan bahwa proyek akan berjalan dengan cara yang sama di lingkungan yang berbeda (pengembangan lokal, *staging*, produksi, *container* Docker). Ini menghindari masalah "berfungsi di mesin saya" (*"it works on my machine"*) yang umum terjadi karena perbedaan versi pustaka.
3. **Memfasilitasi Pembangunan Image Docker:** Dalam Dockerfile, perintah RUN pip install -r requirements.txt secara otomatis membaca dan menginstal semua pustaka yang terdaftar. Tanpa file ini, *image* Docker tidak akan memiliki lingkungan Python yang lengkap untuk menjalankan aplikasi.
4. **Kolaborasi dan Onboarding:** Memudahkan pengembang lain untuk menyiapkan lingkungan kerja yang sama. Mereka hanya perlu menjalankan pip install -r requirements.txt.

2. Isi File requirements.txt dalam Proyek Ini

File requirements.txt yang modifikasi berisi daftar pustaka berikut, masing-masing dengan versi spesifiknya:

- **Flask==3.1.2**
    - **Peran:** Ini adalah *framework* web Python yang ringan dan digunakan untuk membangun API RESTful. Dalam proyek ini, Flask adalah tulang punggung

dari app.py, yang menyediakan endpoint /predict untuk melayani permintaan prediksi dari model.

- **lightgbm==4.6.0**
    - **Peran:** Pustaka ini menyediakan implementasi model Light Gradient Boosting Machine (LightGBM). Ini adalah algoritma Machine Learning yang digunakan sebagai model peramalan permintaan terbaik (model.pkl) dalam proyek ini. Ketergantungannya pada pustaka ini sangat penting untuk dapat memuat dan menjalankan model LightGBM yang telah dilatih.
- **matplotlib==3.9.0**
    - **Peran:** Pustaka untuk pembuatan visualisasi statis, interaktif, dan animasi di Python. Meskipun tidak secara langsung digunakan dalam API yang *deploy*, pustaka ini kemungkinan digunakan selama tahap Exploratory Data Analysis (EDA) dan evaluasi model. Kadang-kadang disertakan untuk kompatibilitas lingkungan atau jika ada kebutuhan visualisasi internal dalam *container*.
- **matplotlib-inline==0.2.1**
    - **Peran:** Sebuah backend Matplotlib untuk lingkungan IPython seperti notebook Jupyter atau Google Colab. Biasanya tidak diperlukan untuk deployment API, tetapi mungkin ikut terinstal sebagai dependensi matplotlib atau sebagai sisa dari lingkungan pengembangan.
- **matplotlib-venn==1.1.2**
    - **Peran:** Pustaka untuk membuat diagram Venn dengan Matplotlib. Sama seperti matplotlib, ini kemungkinan digunakan selama EDA atau eksplorasi data dan bukan dependensi langsung untuk API prediksi.
- **numpy==1.26.4**
    - **Peran:** Pustaka fundamental untuk komputasi numerik di Python, terutama untuk bekerja dengan array multidimensional dan operasi matematika yang efisien. Banyak pustaka Machine Learning lainnya (termasuk Pandas dan Scikit-learn) sangat bergantung pada NumPy.
- **pandas==2.0.3**
    - **Peran:** Pustaka utama untuk analisis dan manipulasi data di Python. Digunakan secara ekstensif dalam preprocessing.py untuk mengelola DataFrame, melakukan ekstraksi fitur, encoding, dan persiapan data lainnya. Juga digunakan di app.py untuk mengkonversi input JSON ke DataFrame.
- **pandas-datareader==0.10.0**
    - **Peran:** Pustaka untuk mengambil data dari sumber eksternal (misalnya, data finansial). Tidak terlihat memiliki peran langsung dalam API prediksi ini, tetapi

mungkin digunakan selama fase pengumpulan atau eksplorasi data dalam proyek.

- **pandas-gbq==0.30.0**
  - **Peran:** Pustaka untuk mengintegrasikan Pandas dengan Google BigQuery. Mirip dengan pandas-datareader, ini mungkin digunakan selama fase pengumpulan data atau pengembangan awal untuk menarik data dari BigQuery.
- **scikit-learn==1.6.1**
  - **Peran:** Pustaka Machine Learning yang komprehensif. Digunakan untuk objek StandardScaler (scaler.pkl), yang merupakan bagian penting dari pipeline pra-pemrosesan. Juga menyediakan banyak fungsi utilitas dan metrik evaluasi yang digunakan selama pengembangan model.
- **seaborn==0.13.2**
  - **Peran:** Pustaka visualisasi data tingkat tinggi yang dibangun di atas Matplotlib. Digunakan untuk membuat plot statistik yang menarik selama EDA dan evaluasi model. Tidak esensial untuk runtime API, tetapi berguna untuk lingkungan pengembangan.
- **xgboost==2.0.3**
  - **Peran:** Pustaka implementasi Extreme Gradient Boosting. Meskipun model terbaik yang dipilih adalah LightGBM, XGBoost adalah salah satu model yang dievaluasi dalam proyek, sehingga mungkin dipertahankan dalam requirements.txt untuk konsistensi lingkungan atau jika ada kebutuhan untuk beralih ke model ini di masa depan.
- **gunicorn**
  - **Peran:** *Server* WSGI (Web Server Gateway Interface) yang direkomendasikan untuk menjalankan aplikasi Python (seperti Flask) di lingkungan produksi. Seperti yang terlihat di Dockerfile, Gunicorn digunakan untuk melayani aplikasi Flask secara efisien dan *robust*.

3. Kesimpulan

requirements.txt ini secara efektif mengidentifikasi semua dependensi Python yang diperlukan untuk menjalankan aplikasi peramalan permintaan. Dengan menyertakan versi spesifik, ia memastikan bahwa *image* Docker yang dibangun akan memiliki lingkungan yang stabil dan konsisten, memungkinkan aplikasi untuk berfungsi dengan benar di setiap deployment. Adanya pustaka seperti matplotlib, seaborn, pandas-datareader, pandas-gbq, dan matplotlib-venn menunjukkan bahwa file ini kemungkinan besar dihasilkan dari

lingkungan pengembangan, di mana pustaka-pustaka ini digunakan untuk analisis dan visualisasi. Untuk deployment yang sangat ramping, pustaka-pustaka yang tidak esensial untuk runtime API (matplotlib, seaborn, dll.) dapat dihilangkan untuk mengurangi ukuran *image* Docker lebih lanjut.

6.3. Membangun dan Menjalankan Kontainer Docker (Implementasi di Lingkungan Lokal)

Setelah semua aset (model, scaler, parameter pra-pemrosesan, kode aplikasi, dan Dockerfile) telah disiapkan, langkah selanjutnya adalah membangun *image* Docker dan menjalankannya sebagai *container*. Proses ini memungkinkan aplikasi peramalan permintaan untuk beroperasi dalam lingkungan yang terisolasi, konsisten, dan *portable*.

6.3.1. Langkah 1: Membangun Image Docker (docker build)

Sebuah **Docker Image** dapat diibaratkan sebagai "cetakan", "templat", atau *blueprint* yang bersifat *read-only*. Image ini berisi semua yang dibutuhkan aplikasi untuk berjalan, termasuk kode aplikasi, *runtime* (misalnya Python), pustaka-pustaka sistem, dependensi, dan semua konfigurasi. Image ini tidak dapat diubah setelah dibuat, menjamin konsistensi di mana pun ia digunakan. Dalam konteks proyek ini, *image* Docker akan mengemas aplikasi Flask, model LightGBM, scaler, dan logika pra-pemrosesan.

Perintah Pembangunan Image:

Untuk membangun *image* Docker dari Dockerfile yang telah dibuat, perintah berikut dijalankan di terminal, di direktori demand_deploy_docker (tempat Dockerfile berada):

- docker build -t latief18/demand-api:1.0 .
    - docker build: Perintah untuk memulai proses pembangunan *image* Docker.
    - -t latief18/demand-api:1.0: Opsi -t (tag) digunakan untuk memberi nama dan versi pada *image*. latief18/demand-api adalah nama *repository* (biasanya mencakup *username* Docker Hub), dan 1.0 adalah *tag* versi. Penamaan ini membantu dalam identifikasi dan pengelolaan *image*.
    - .: Menunjukkan konteks pembangunan Docker, yaitu direktori saat ini. Docker akan mencari Dockerfile di direktori ini dan menggunakannya sebagai instruksi untuk membangun *image*.

6.3.2. Langkah 2: Menjalankan Kontainer (docker run)

Sebuah **Docker Container** adalah "instansi" yang dapat dieksekusi atau aplikasi yang berjalan dari sebuah *image* Docker. Jika *image* adalah cetakannya, *container* adalah kuenya. *Container* menyediakan lingkungan *runtime* yang terisolasi di mana aplikasi dapat berjalan. Setiap *container* berjalan secara independen dan memiliki sistem file, proses, dan sumber daya jaringannya sendiri.

Perintah Menjalankan Kontainer:

Setelah *image* berhasil dibangun, perintah berikut digunakan untuk menjalankan *container* dari *image* tersebut:

- docker run -p 5001:5000 -d latief18/demand-api:1.0
    - docker run: Perintah untuk menjalankan *container* baru dari sebuah *image*.
    - -p 5001:5000: Opsi -p sangat penting untuk memetakan (*port mapping*) *port* di *host* (komputer) ke *port* di dalam *container*. Dalam kasus ini, 5001:5000 berarti *port* 5001 di komputer *host* akan terhubung ke *port* 5000 di dalam *container*. Aplikasi Flask/Gunicorn berjalan pada *port* 5000 di dalam *container*, sehingga memetakan *port* ini memungkinkan kita untuk mengakses API dari luar *container* melalui *port* 5001 di *host*.
    - -d: Opsi -d (detached mode) menjalankan *container* di latar belakang. Ini berarti *container* akan mulai berjalan tanpa menempati terminal, dan dapat terus menggunakan terminal untuk tugas lain.
    - latief18/demand-api:1.0: Nama dan *tag image* Docker yang akan digunakan untuk membuat *container*.

6.3.3. Langkah 3: Verifikasi dan Debugging

Verifikasi Status Kontainer (docker ps):

Untuk memastikan bahwa *container* berhasil berjalan di latar belakang, kita dapat menggunakan perintah docker ps:

Alat Debugging Utama (docker logs):

Jika *container* tidak muncul di docker ps atau tidak berfungsi sesuai harapan, alat paling penting untuk *debugging* adalah docker logs. Perintah ini memungkinkan kita untuk melihat *output* log dari *container*, termasuk *output* dari aplikasi Gunicorn/Flask (app.py).

- docker logs [CONTAINER_ID_KAMI_ATAU_NAMA_CONTAINER]

Ganti [CONTAINER_ID_KAMI_ATAU_NAMA_CONTAINER] dengan ID *container* yang kami dapatkan dari docker ps atau nama *container* yang kami berikan saat menjalankan (demand-forecaster jika kami menggunakan contoh sebelumnya). Log ini akan menampilkan pesan-pesan dari aplikasi, seperti [LightGBM] [Info] Start training from score... (saat model dimuat) dan pesan-pesan *startup* Gunicorn, atau pesan error jika ada yang terjadi saat aplikasi mencoba memulai atau saat permintaan diproses. Ini adalah alat penting untuk memahami perilaku internal *container* dan mendiagnosis masalah.

6.4. Langkah-langkah Akses Hasil Docker pada Device Lain

Bagian ini memandu pengguna dalam mengakses dan menguji aplikasi peramalan permintaan yang telah di-*deploy* dalam *container* Docker dari perangkat lain atau di lingkungan lokal setelah *image* tersedia di Docker Hub. Panduan ini mengasumsikan bahwa **Docker Desktop** sudah terinstal dan berjalan di perangkat Anda.

6.4.1. Persiapan Lingkungan Lokal

Langkah awal adalah menyiapkan folder kerja di perangkat lokal Anda. Folder ini akan menjadi lokasi untuk menyimpan file input data dalam format JSON yang akan digunakan untuk mengirim permintaan ke API.

1. **Membuat Folder Kerja:** Buatlah sebuah folder baru di lokasi yang mudah diakses. Contoh perintah untuk membuat folder di Windows PowerShell:
2. mkdir C:\TEST\Demand
3. **Navigasi ke Folder Kerja:** Pindahkan direktori kerja Anda ke folder yang baru saja dibuat. Contoh:
4. cd C:\TEST\Demand
5. **Menempatkan File input.json:** Pastikan file input.json yang berisi contoh data input untuk prediksi ditempatkan di dalam folder kerja ini (C:\TEST\Demand atau folder lain yang Anda tentukan). File ini akan digunakan sebagai *payload* untuk permintaan prediksi API.

6.4.2. Pengambilan Image Docker dan Peluncuran Kontainer API

Setelah lingkungan disiapkan, langkah selanjutnya adalah mendapatkan *image* aplikasi dari Docker Hub dan menjalankannya sebagai *container* API.

1. **Login ke Docker Hub (Opsional):** Jika Anda belum *login* ke Docker Hub melalui Docker Desktop, Anda mungkin perlu menjalankan perintah docker login. Namun,

jika Docker Desktop sudah berjalan dan Anda sudah terotentikasi, langkah ini biasanya tidak diperlukan.

2. **Menarik (Pull) Image Aplikasi:** Ambil *image* aplikasi demand-forecasting-app dari *repository* Docker Hub (latief18) ke perangkat lokal Anda. Pastikan untuk menggunakan *tag* versi yang benar (v1).
3. docker pull latief18/demand-forecasting-app:v1

Perintah ini akan mengunduh semua lapisan (*layers*) yang membentuk *image* aplikasi ke sistem lokal Anda.

4. **Menjalankan Server API Kontainer:** Luncurkan *container* dari *image* yang telah diunduh, dan jalankan sebagai server API.
5. docker run -d -p 5000:5000 --name demand_latief latief18/demand-forecasting-app:v1
    - docker run: Perintah untuk menjalankan *container* baru.
    - -d (detached mode): Menjalankan *container* di latar belakang, sehingga Anda dapat terus menggunakan terminal Anda.
    - -p 5000:5000: Melakukan *port mapping*. Ini memetakan *port* 5000 di *host* (perangkat lokal Anda) ke *port* 5000 di dalam *container* (tempat aplikasi Flask/Gunicorn berjalan). Ini memungkinkan Anda mengakses API melalui http://localhost:5000.
    - --name demand_latief: Memberikan nama yang mudah diingat (demand_latief) kepada *container* untuk referensi di kemudian hari (misalnya, saat memeriksa log).
    - latief18/demand-forecasting-app:v1: Menentukan *image* Docker yang akan digunakan.

6.4.3. Pengujian Endpoint dan Verifikasi Log

Setelah *container* berjalan, Anda dapat menguji endpoint /predict dan memverifikasi output analisis.

1. **Mengirim Data Prediksi:** Kirim permintaan HTTP POST ke endpoint /predict dengan input.json sebagai *payload*. Perintah berikut dapat digunakan di PowerShell (Windows) atau curl di terminal Linux/macOS:
2. Invoke-RestMethod -Method Post -Uri http://localhost:5000/predict -ContentType 'application/json' -Body (Get-Content -Raw input.json)

(Perintah ini mengasumsikan Anda berada di direktori yang sama dengan input.json. Jika tidak, Anda perlu menyediakan jalur lengkap ke file input.json.)

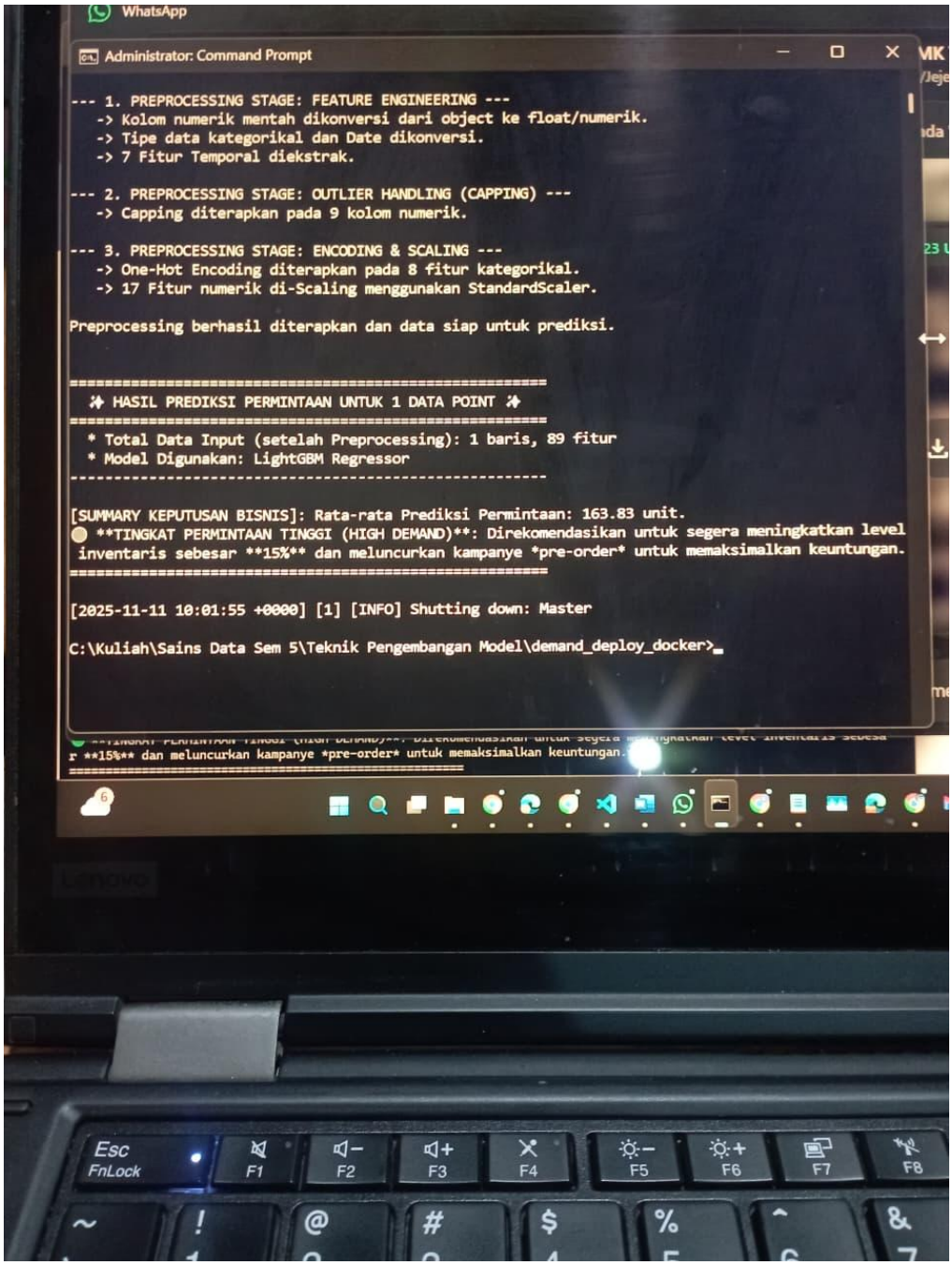
Output di Terminal: Setelah menjalankan perintah ini, terminal Anda akan menampilkan hasil prediksi dalam bentuk angka (misalnya, [163.83]). Ini adalah nilai permintaan yang diprediksi oleh model.

3. **Mengakses Log Kontainer (Melihat Output Analitik dan Saran Bisnis):** Meskipun hasil prediksi mentah ditampilkan di terminal, analisis lebih rinci dan saran bisnis yang dihasilkan oleh fungsi generate_recommendations dalam app.py akan dicetak ke log *container*. Untuk melihat log ini, gunakan perintah berikut:
4. docker logs demand_latief

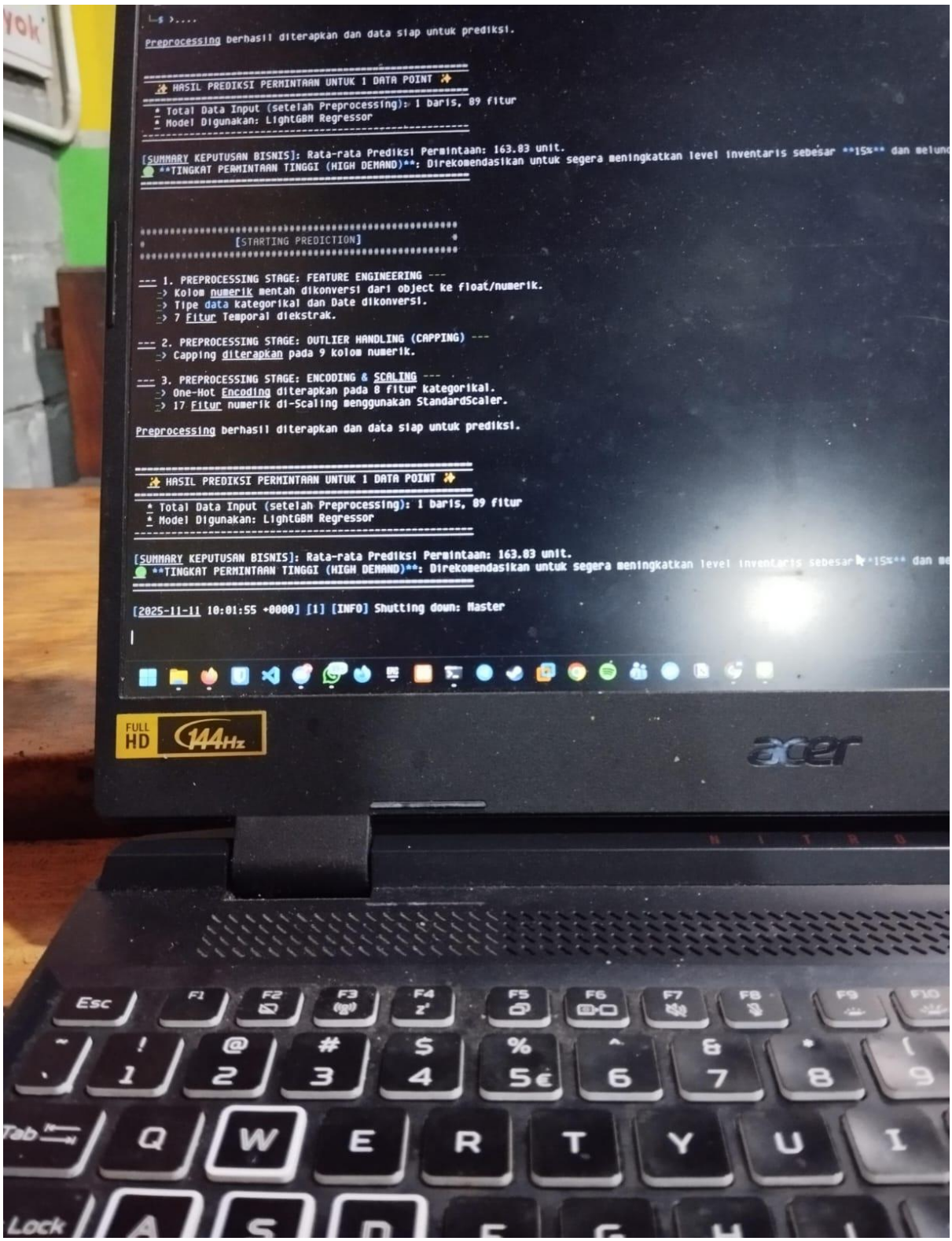
Perintah ini akan menampilkan seluruh log output dari *container* demand_latief sejak diluncurkan, termasuk detail langkah-langkah pra-pemrosesan, pesan "[STARTING PREDICTION]", "[SUMMARY KEPUTUSAN BISNIS]", dan rekomendasi bisnis yang relevan berdasarkan prediksi permintaan.

6.5. Distribusi dan Aplikasi di Lingkungan Lokal dan Device Lain (Bukti Portabilitas)

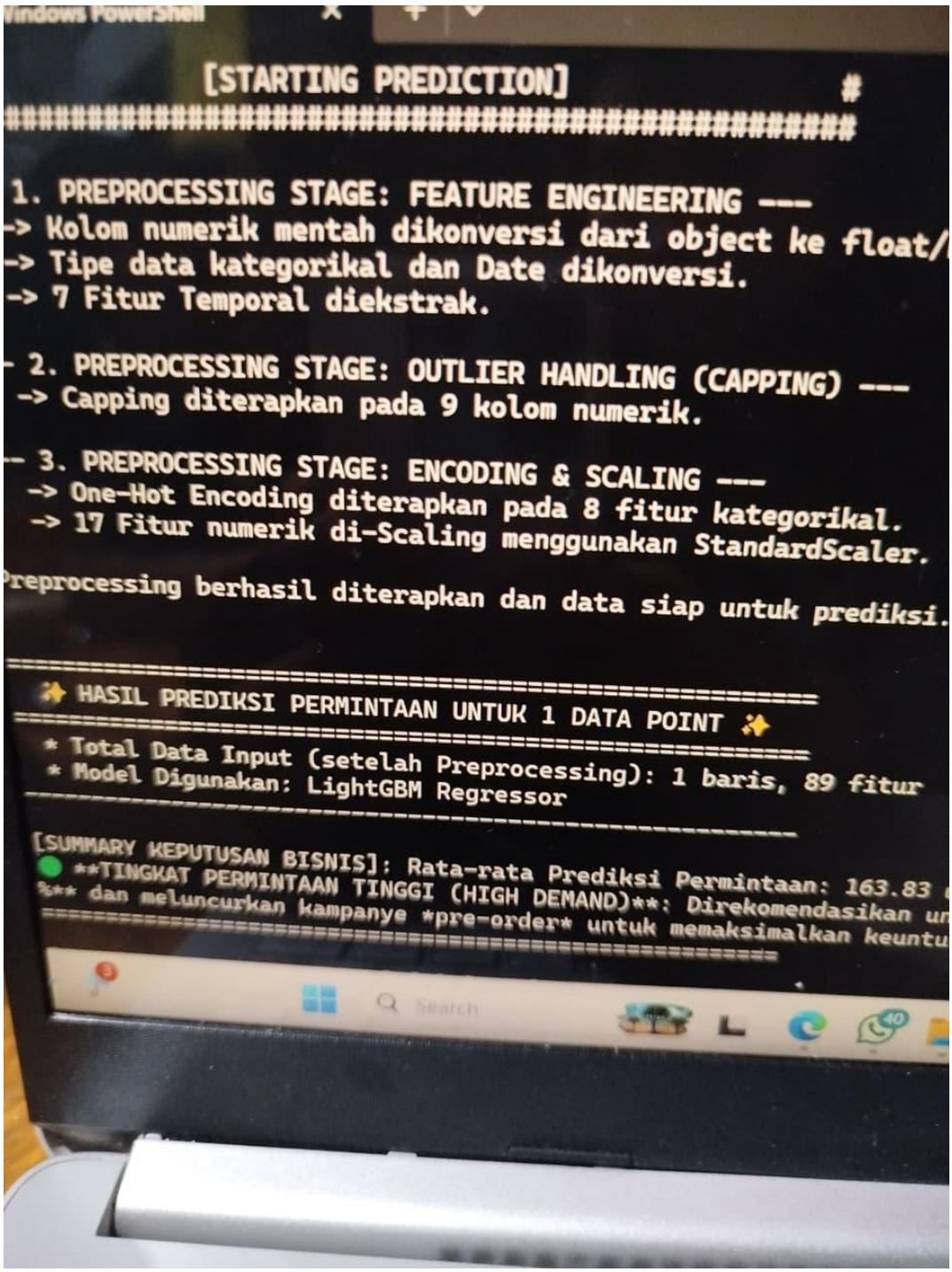
6.5.1. Laptop 1



6.5.2. Laptop 2


Preprocessing berhasil diterapkan dan data siap untuk prediksi.
HASIL PREDIKSI PERMINTAAN UNTUK 1 DATA POINT
Total Data Input (setelah Preprocessing): 1 baris, 89 fitur
Model Digunakan: LightGBM Regressor
[SUMMARY KEPUTUSAN BISNIS]: Rata-rata Prediksi Permintaan: 163.83 unit.
TINGKAT PERMINTAAN TINGGI (HIGH DEMAND): Direkomendasikan untuk segera meningkatkan level inventaris sebesar **15%** dan melun
[STARTING PREDICTION]
--- 1. PREPROCESSING STAGE: FEATURE ENGINEERING ---
-> Kolom numerik mentah dikonversi dari object ke float/numerik.
-> Tipe data kategorikal dan Date dikonversi.
-> 7 Fitur Temporal diekstrak.
--- 2. PREPROCESSING STAGE: OUTLIER HANDLING (CAPPING) ---
-> Capping diterapkan pada 9 kolom numerik.
--- 3. PREPROCESSING STAGE: ENCODING & SCALING ---
-> One-Hot Encoding diterapkan pada 8 fitur kategorikal.
-> 17 Fitur numerik di-Scaling menggunakan StandardScaler.
Preprocessing berhasil diterapkan dan data siap untuk prediksi.
HASIL PREDIKSI PERMINTAAN UNTUK 1 DATA POINT
Total Data Input (setelah Preprocessing): 1 baris, 89 fitur
Model Digunakan: LightGBM Regressor
[SUMMARY KEPUTUSAN BISNIS]: Rata-rata Prediksi Permintaan: 163.83 unit.
TINGKAT PERMINTAAN TINGGI (HIGH DEMAND): Direkomendasikan untuk segera meningkatkan level inventaris sebesar **15%** dan me
[2025-11-11 10:01:55 +0000] [1] [INFO] Shutting down: Master
FULL HD
144Hz
acer

6.5.3. Laptop 3


[STARTING PREDICTION]
1. PREPROCESSING STAGE: FEATURE ENGINEERING ---
-> Kolom numerik mentah dikonversi dari object ke float/
-> Tipe data kategorikal dan Date dikonversi.
-> 7 Fitur Temporal diekstrak.
2. PREPROCESSING STAGE: OUTLIER HANDLING (CAPPING) ---
-> Capping diterapkan pada 9 kolom numerik.
3. PREPROCESSING STAGE: ENCODING & SCALING ---
-> One-Hot Encoding diterapkan pada 8 fitur kategorikal.
-> 17 Fitur numerik di-Scaling menggunakan StandardScaler.
reprocessing berhasil diterapkan dan data siap untuk prediksi.
HASIL PREDIKSI PERMINTAAN UNTUK 1 DATA POINT
* Total Data Input (setelah Preprocessing): 1 baris, 89 fitur
* Model Digunakan: LightGBM Regressor
[SUMMARY KEPUTUSAN BISNIS]: Rata-rata Prediksi Permintaan: 163.83
TINGKAT PERMINTAAN TINGGI (HIGH DEMAND): Direkomendasikan un
%** dan meluncurkan kampanye *pre-order* untuk memaksimalkan keuntu

BAB VI Kesimpulan

Proyek **Retail Demand Forecasting: Pendekatan Pipeline Machine Learning dan Deploy dengan Docker** ini telah berhasil menyelesaikan seluruh siklus hidup pengembangan model machine learning secara *end-to-end*, mulai dari pengolahan data mentah hingga menjadi sebuah layanan prediksi yang fungsional dan portabel. Setiap tahapan proyek dirancang dan dieksekusi dengan mengikuti praktik terbaik industri untuk memastikan hasil yang tidak hanya akurat secara teoretis, tetapi juga andal dan siap untuk aplikasi di dunia nyata.

Perjalanan proyek diawali dengan membangun sebuah fondasi data yang kuat melalui **pipeline persiapan data 10 langkah yang sistematis**. Dengan mengintegrasikan tiga sumber data yang berbeda—yaitu data transaksional inti, data pemasaran, dan data ekonomi makro—berhasil diciptakan sebuah *master dataset* yang kaya akan fitur internal dan eksternal. Proses pembersihan data yang teliti, penanganan nilai hilang, dan deteksi *data leakage* memastikan integritas data. Selanjutnya, **rekayasa fitur (feature engineering)** yang mendalam, meliputi ekstraksi fitur berbasis waktu, pembuatan *lag features* untuk menangkap dependensi temporal, dan *rolling window features* untuk menghaluskan tren, secara signifikan memperkaya informasi yang dapat dipelajari oleh model. Transformasi akhir menggunakan *One-Hot Encoding* untuk data kategorikal dan *StandardScaler* untuk data numerik memastikan bahwa dataset akhir sepenuhnya siap untuk tahap pemodelan.

Pada **tahap pemodelan**, proyek ini melakukan analisis komparatif terhadap lima model machine learning yang relevansinya didukung oleh studi literatur. Hasil evaluasi kuantitatif menunjukkan bahwa model *gradient boosting* modern, yaitu **LightGBM dan XGBoost**, secara signifikan mengungguli model-model lain seperti Random Forest, Decision Tree, dan Linear Regression (yang berfungsi sebagai *baseline*). **LightGBM muncul sebagai model pemenang** dengan kinerja terbaik, mencapai **RMSE terendah sebesar 15.78** dan **R-squared tertinggi sebesar 0.8900**. Hal ini membuktikan kemampuannya dalam menjelaskan hampir 89% varians dalam data permintaan dan menunjukkan efektivitas pendekatan *ensemble* dalam menangkap hubungan yang kompleks. Analisis kontribusi fitur (*feature importance*) dari model terbaik juga memberikan wawasan bisnis yang berharga, menyoroti bahwa faktor-faktor seperti permintaan di masa lalu (*lag features*), rata-rata pergerakan permintaan (*rolling window*), serta harga dan diskon adalah pendorong utama permintaan ritel.

Puncak pencapaian proyek ini adalah pada **tahap deployment**. Dengan mengadopsi arsitektur *microservice* yang modular dan profesional, pipeline preprocessing dan model LightGBM terbaik berhasil diserialisasi dan dibungkus dalam sebuah aplikasi API menggunakan Flask dan Gunicorn. Seluruh aplikasi ini kemudian **dikontainerisasi menggunakan Docker**, menciptakan sebuah *image* yang portabel dan konsisten. Keberhasilan deployment ini dibuktikan secara empiris:

1. Kontainer berhasil dijalankan di lingkungan lokal dan terbukti fungsional melalui pengujian *endpoint* API menggunakan curl, yang berhasil mengembalikan prediksi akurat.
2. *Image* Docker berhasil didistribusikan melalui Docker Hub.
3. Sebagai bukti portabilitas tertinggi, *image* tersebut berhasil ditarik (*pulled*) dan dijalankan di beberapa perangkat komputer yang berbeda, di mana semuanya mampu memberikan prediksi secara konsisten hanya dengan satu perintah.

Secara keseluruhan, proyek ini tidak hanya berhasil membangun model peramalan yang akurat, tetapi juga mendemonstrasikan penguasaan hard skill yang krusial dalam **Teknik Pengembangan Model**: yaitu kemampuan untuk membawa sebuah model dari tahap konseptual di notebook menjadi sebuah layanan yang kokoh (*robust*), dapat diandalkan, dan siap digunakan di berbagai lingkungan.