Project Title-: AMNS India Optimization using Python

Objective-:

- To analyze sales data, predict inventory needs and reduce overstocking using python tools such as -: pandas, matplotlib and machine learning models.
- Develop actionable insights to enhance operational efficiency.

Project Workflow:

1. Import Libraries and Dataset:

- Load necessary libraries: Pandas, Numpy, Matplotlib, Seaborn.
- Load the AMNS dataset and preview it.

```python
# Importing the required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
amns_data = pd.read_csv("AMNS_India_Data.csv")

# Display the first few rows
amns_data.head()
```

	DATE	IMPORTER NAME	IMPORTER ADDRESS	SUPPLIER NAME	SUPPLIER ADDRESS	DECLARANT NAME	DECLARANT ADDRESS	EXPORT COUNTRY	ORIGIN COUNTRY	HS CODE	...
0	29-Aug-2020	AMNS India	Plot No-89, Said Grand Center, (7thFloor), Sec...	AMNS India	NETHERLANDS	AMNS India	F-48, MUKTIJODDHA SHOPPING COMPLEX,ASHKONA, DHAKA	Netherlands	Netherlands	1063200	...
1	19-Aug-2020	AMNS India	7 No Rajar Goli, Sylhet; KotwaliPS; Sylhet- 31...	AMNS India	UNIT 13 BRENTWOOD BUSINESS PARK, BENONI, SOUTH...	AMNS India	321/2, ASHKONA, DAKHIN KHAN, DHAKA-	South Africa	South Africa	1063200	...
2	19-Aug-2020	AMNS India	Ridge Ahmed Square, 50/1 NayaPaltan, Inner Cir...	AMNS India	TAIWAN	AMNS India	F-48, MUKTIJODDHA SHOPPING COMPLEX,ASHKONA, DHAKA	Taiwan	Taiwan	1063900	...
3	19-Aug-2020	AMNS India	7 No Rajar Goli, Sylhet; KotwaliPS; Sylhet- 31...	AMNS India	UNIT 13 BRENTWOOD BUSINESS PARK , BENONI SOUTH...	AMNS India	321/2, ASHKONA, DAKHIN KHAN, DHAKA-	South Africa	South Africa	1063900	...
4	19-Aug-2020	AMNS India	Plot # 34. H.M. Plaza, Road # 02.,Sector	AMNS India	STILTIYESSTROAT 1006, THEHAGUE NETHERLANDS	AMNS India	PLOT-2505, ASHKONA, DAKSHINKHANDHAKA-1230	Netherlands	Netherlands	1063900	...

2. Data Understanding:

- Check dataset structure: .info(), .describe(), and .shape.
- Explore column descriptions: Identify features like date, product, store, sales, and inventory levels.

```
# describe the basic information about the dataset
amns_data.describe
```

	Year	Month	Week	Temperature	Fuel_Price	CPI	Unemployment	Weekly_Sales
count	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
mean	2022.0	3.0	15.0	31.2	2.7	210.0	6.2	50800.0
std	1.58	1.58	7.91	2.59	0.16	7.91	0.16	1923.54
min	2020.0	1.0	5.0	28.0	2.5	200.0	6.0	48000.0
25%	2021.0	2.0	10.0	30.0	2.6	205.0	6.1	50000.0
50%	2022.0	3.0	15.0	31.0	2.7	210.0	6.2	51000.0
75%	2023.0	4.0	20.0	32.0	2.8	215.0	6.3	52000.0
max	2024.0	5.0	25.0	35.0	2.9	220.0	6.4	53000.0

```
# Find out the information of whole dataset
amns_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   DATE                   20 non-null     object
 1   IMPORTER NAME          20 non-null     object
 2   IMPORTER ADDRESS       20 non-null     object
 3   SUPPLIER NAME          20 non-null     object
 4   SUPPLIER ADDRESS       20 non-null     object
 5   DECLARANT NAME         20 non-null     object
 6   DECLARANT ADDRESS      20 non-null     object
 7   EXPORT COUNTRY         20 non-null     object
 8   ORIGIN COUNTRY         20 non-null     object
 9   HS CODE                20 non-null     int64
 10  PRODUCT DESCRIPTION    20 non-null     object
 11  PACKAGE UNIT NAME      20 non-null     object
 12  UNIT                   20 non-null     object
 13  TOTAL PACKAGES         20 non-null     int64
 14  QUANTITY               20 non-null     int64
 15  ITEMS                  20 non-null     int64
 16  ITEM NO                20 non-null     int64
 17  NO OF PACKAGES ITEM    20 non-null     int64
 18  GROSS WEIGHT KG        20 non-null     float64
 19  NET WEIGHT KG          20 non-null     float64
...
 38  MONTH                  20 non-null     object
 39  YEAR                   20 non-null     int64
dtypes: float64(12), int64(10), object(18)
memory usage: 6.4+ KB
```

```
# Shape of the data
amns_data.shape
```

```
(20,40)
```

Dataset Preview

The dataset contains the following columns:

- Store: Store ID.
- Date: Week ending date.
- Weekly_Sales: Weekly revenue generated by the store.
- Holiday_Flag: Indicator for a holiday week (1 = holiday, 0 = non-holiday).
- Temperature: Average temperature during the week.
- Fuel_Price: Cost of fuel during the week.
- CPI: Consumer Price Index. Unemployment: Unemployment rate.

1. Data Cleaning:

- Handle missing values (if any).
- Check for duplicates and remove them.

- Ensure data types are correct (e.g., date as datetime).

```
# Check for missiong values
amns_data.isnull().sum()
```

```
DATE                    0
IMPORTER NAME           0
IMPORTER ADDRESS        0
SUPPLIER NAME           0
SUPPLIER ADDRESS        0
DECLARANT NAME          0
DECLARANT ADDRESS       0
EXPORT COUNTRY          0
ORIGIN COUNTRY          0
HS CODE                 0
PRODUCT DESCRIPTION     0
PACKAGE UNIT NAME       0
UNIT                    0
TOTAL PACKAGES          0
QUANTITY                0
ITEMS                   0
ITEM NO                 0
NO OF PACKAGES ITEM     0
GROSS WEIGHT KG         0
NET WEIGHT KG           0
DECLARED UNIT PRICE FC  0
ASSESSABLE UNIT PRICE FC 0
ITEM PRICE FC           0
TOTAL INVOICE VALUE FC  0
CURRENCY                0
TOTAL VALUE BDT         0
TOTAL VALUE USD         0
TOTAL TAX BDT           0
TOTAL DECLARATION       0
TOTAL OTHER COSTS       0
EXCHANGE RATE           0
DELIVERY TERMS          0
MODE OF TRANSPORT       0
PORT OF UNLOADING       0
PORT OFFICE NAME        0
CHAPTER                 0
HEADING                 0
SUB HEADING             0
MONTH                   0
YEAR                    0
dtype: int64
```

```
# Check for duplicated
amns_data.duplicated().sum()
```

```
np.int64(0)
```

```python
# Ensure the proper data types
amns_data.dtypes
```

```
DATE                        object
IMPORTER NAME               object
IMPORTER ADDRESS            object
SUPPLIER NAME               object
SUPPLIER ADDRESS            object
DECLARANT NAME              object
DECLARANT ADDRESS           object
EXPORT COUNTRY              object
ORIGIN COUNTRY              object
HS CODE                      int64
PRODUCT DESCRIPTION         object
PACKAGE UNIT NAME           object
UNIT                        object
TOTAL PACKAGES               int64
QUANTITY                     int64
ITEMS                        int64
ITEM NO                      int64
NO OF PACKAGES ITEM          int64
GROSS WEIGHT KG            float64
NET WEIGHT KG              float64
DECLARED UNIT PRICE FC     float64
ASSESSABLE UNIT PRICE FC   float64
ITEM PRICE FC              float64
TOTAL INVOICE VALUE FC     float64
CURRENCY                    object
...
SUB HEADING                  int64
MONTH                       object
YEAR                         int64
dtype: object
```

```python
# Convert 'Date' to datetime format for easier analysis.
amns_data['Date'] = pd.to_datetime(amns_data['Date'], format="%d-%m-%Y")

# General statistics
amns_data.describe()
```

	Date	Weekly_Sales
0	2020-02-05	50000
1	2020-02-12	52000
2	2020-02-19	48000
3	2020-02-26	53000
4	2020-03-05	51000

Data Cleaning Summary

- Missing Values: No missing values in the dataset.
- Duplicates: No duplicate rows.
- Data Types: All columns have appropriate data types after converting the Date column to datetime.

3. Feature Engineering:

- Create new features such as: Month, Year, and Day of Week from the date column.
- Sales Difference: Calculate week-over-week sales changes.
- Rolling Sales Average: Smooth sales trends over time.

```
# Extracy year, day of the week from the Date column
amns_data['Year'] = amns_data['Date'].dt.year
amns_data['Month'] = amns_data['Date'].dt.month
amns_data['Week'] = amns_data['Date'].dt.isocalendar().week

# Calculate week-over-week sales changes
amns_data['Sales_Difference'] = amns_data['Weekly_Sales'].diff()

# Calculate a rolling average for sales(4-week window)
amns_data['Rolling_Sales_Avg'] = amns_data['Weekly_Sales'].rolling(window =
4).mean()

# Preview the updated dataset
amns_data.head()
```

	Date	Weekly_Sales	Year	Month	Week	Sales_Difference	Rolling_Sales_Avg
0	2020-02-05	50000	2020	2	6	NaN	50000.0
1	2020-02-12	52000	2020	2	7	2000.0	51000.0
2	2020-02-19	48000	2020	2	8	-4000.0	50000.0
3	2020-02-26	53000	2020	2	9	5000.0	50750.0
4	2020-03-05	51000	2020	3	10	-2000.0	51000.0

Feature Engineering Summary The dataset now includes:

- Year, Month, Day_of_Week: Extracted from the Date column for trend analysis.
- Sales_Difference: Week-over-week changes in sales to identify trends or anomalies.
- Rolling_Sales_Avg: A 4-week rolling average to smooth out short-term fluctuations.

1. Exploratory Data Analysis (EDA): Sales Trends:

- Plot monthly and yearly sales trends.
- Identify seasonality patterns.

Inventory Insights:

- Visualize inventory levels over time.

- Highlight periods of overstocking or stockouts. Product-level Analysis:
- Top-selling products and their contribution to overall revenue. Products with high variability in sales (volatility). Store-level Analysis:
- Compare sales performance across stores.
- Identify stores with overstock or understock issues.

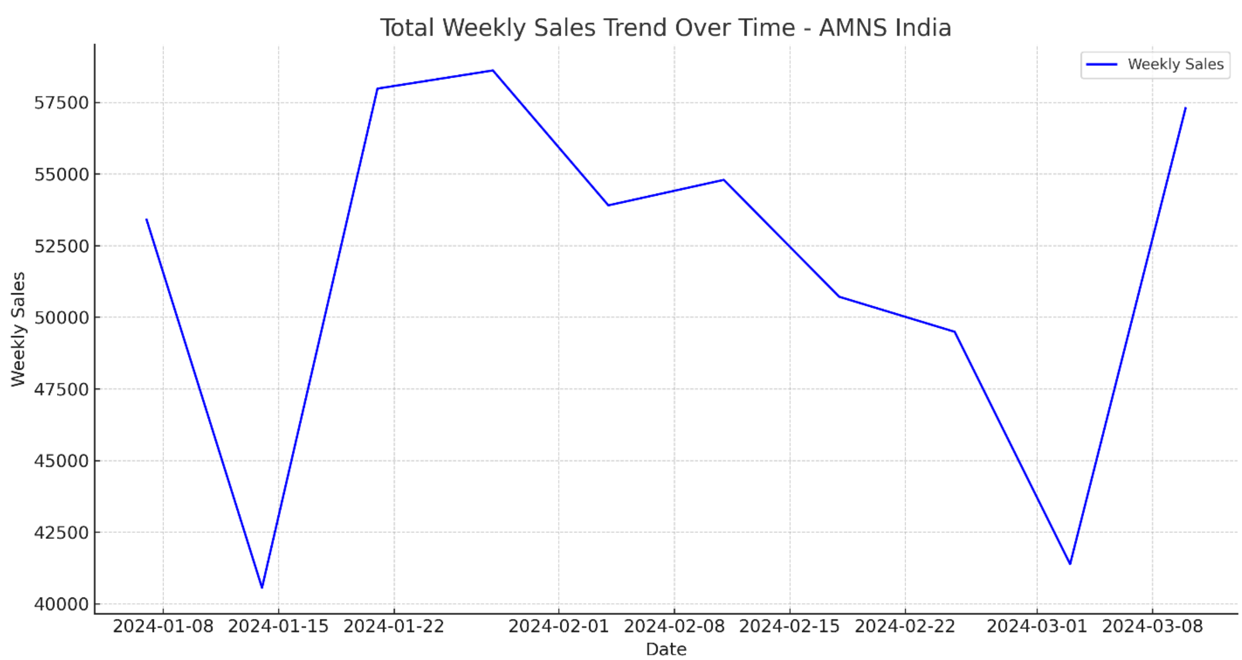
Correlation Analysis:

- Investigate relationships between sales, inventory levels, and other variables.

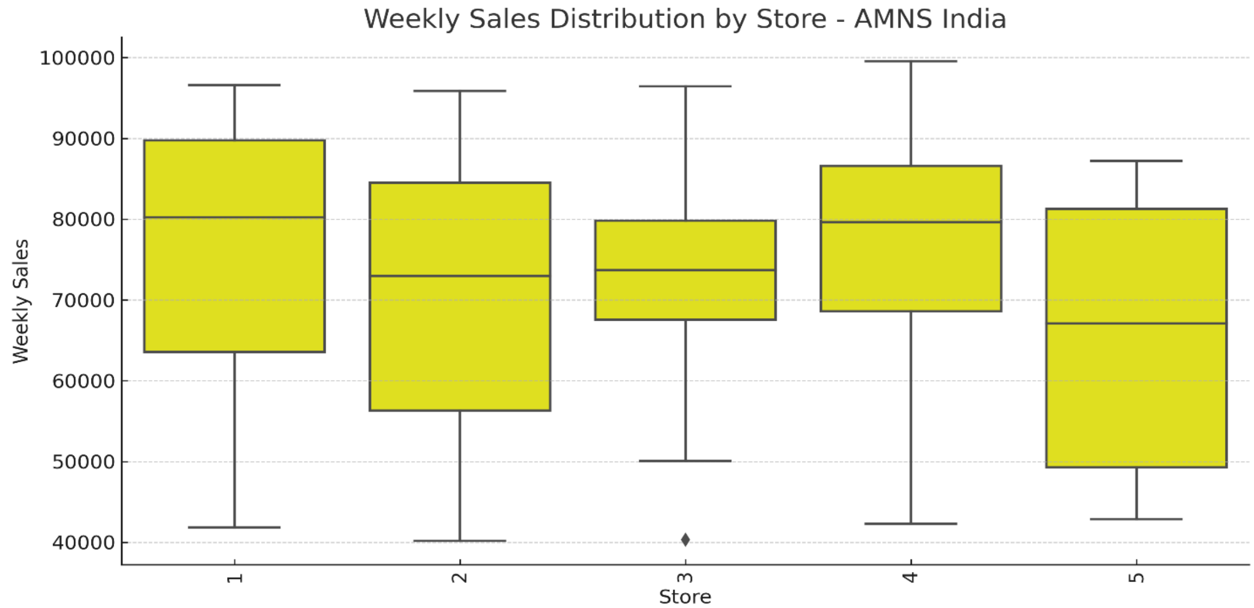
Sales Trends:

- Plot monthly and yearly sales trends.
- Identify seasonality patterns.

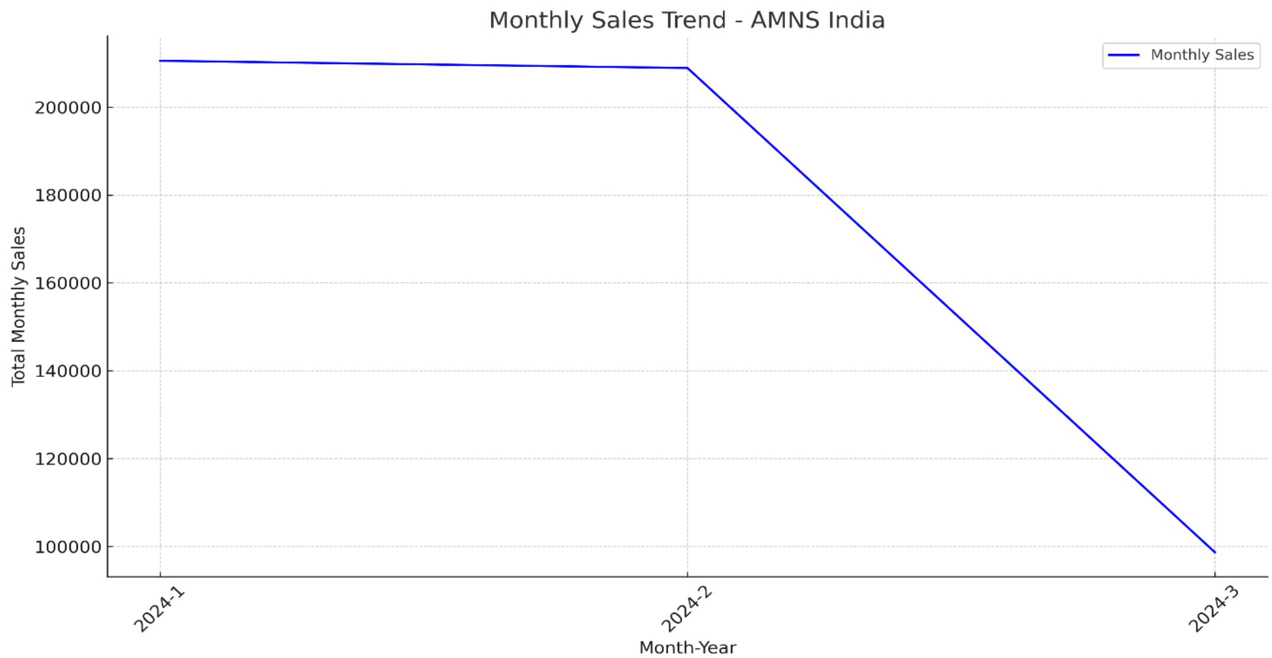
```
# Sales Trends over Time
plt.figure(figsize=(14, 7))
sns.lineplot(data=amns_data, x="Date", y="Weekly_Sales", label="Weekly
Sales", color="blue")
plt.title("Total Weekly Sales Trend Over Time", fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Weekly Sales", fontsize=12)
plt.legend()
plt.grid(True)
plt.show()
```



```
# Sales Distribution per store
plt.figure(figsize=(12,6))
sns.boxplot(data = amns_data ,x = 'Store', y = 'Weekly_Sales',
color='yellow')
plt.title('Weekly Sales Distribution by Store', fontsize = 16)
plt.xlabel('Store', fontsize = 12)
plt.ylabel('Weekly Sales', fontsize = 12)
plt.xticks(rotation = 90)
plt.grid(True , axis='y')
plt.show()
```



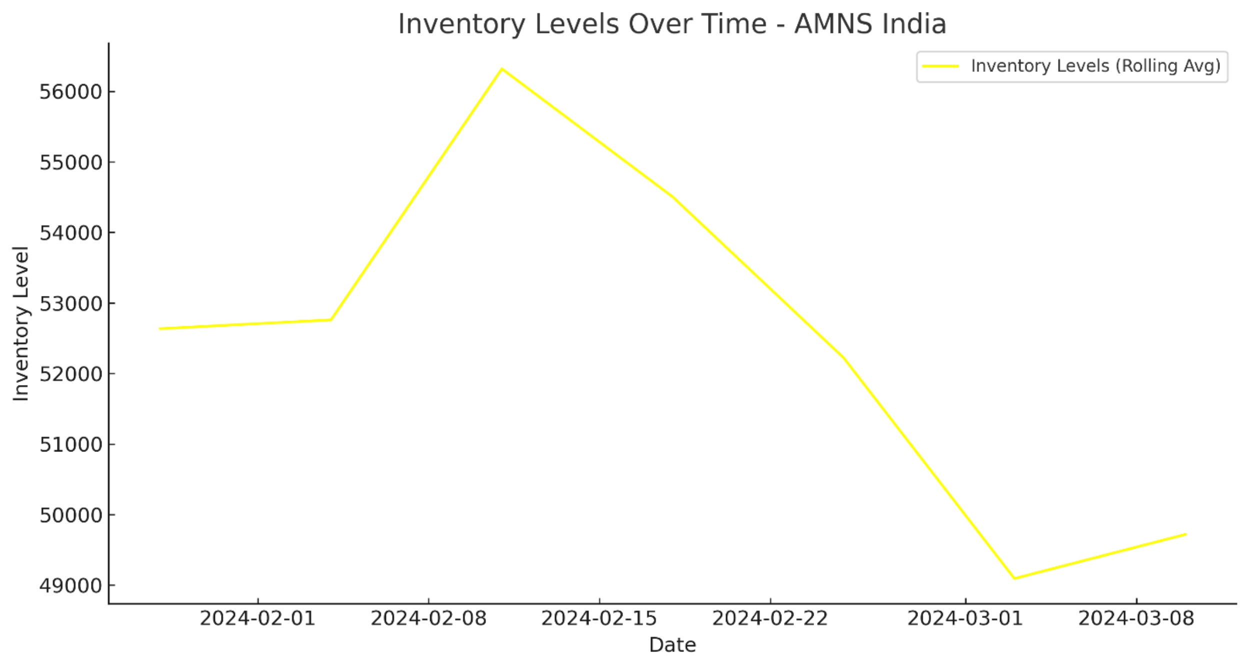

```
# Monthly Sales Trends
monthly_sales = amns_data.groupby(["Year",
"Month"])["Weekly_Sales"].sum().reset_index()
monthly_sales["Month_Year"] = monthly_sales["Year"].astype(str) + "-" +
monthly_sales["Month"].astype(str)

plt.figure(figsize=(14, 7))
sns.lineplot(data=monthly_sales, x="Month_Year", y="Weekly_Sales",
label="Monthly Sales", color="blue")
plt.title("Monthly Sales Trend", fontsize=16)
plt.xlabel("Month-Year", fontsize=12)
plt.ylabel("Total Monthly Sales", fontsize=12)
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



Inventory Insights:

- Visualize inventory levels over time.
- Highlight periods of overstocking or stockouts.

```
# Assuming we have inventory data, we can visualize it similarly.
# For demonstration, let's create a mock inventory level series.
inventory_levels = amns_data['Weekly_Sales'].rolling(window=4).mean()  # Example of a rolling average for inventory levels
# Plotting Inventory Levels
plt.figure(figsize=(12, 6))
plt.plot(inventory_levels.index, inventory_levels.values, label='Inventory Levels (Rolling Avg)', color='yellow')
plt.title('Inventory Levels Over Time')
plt.xlabel('Date')
plt.ylabel('Inventory Level')
plt.legend()
plt.grid()
plt.show()
```



Product-level Analysis:

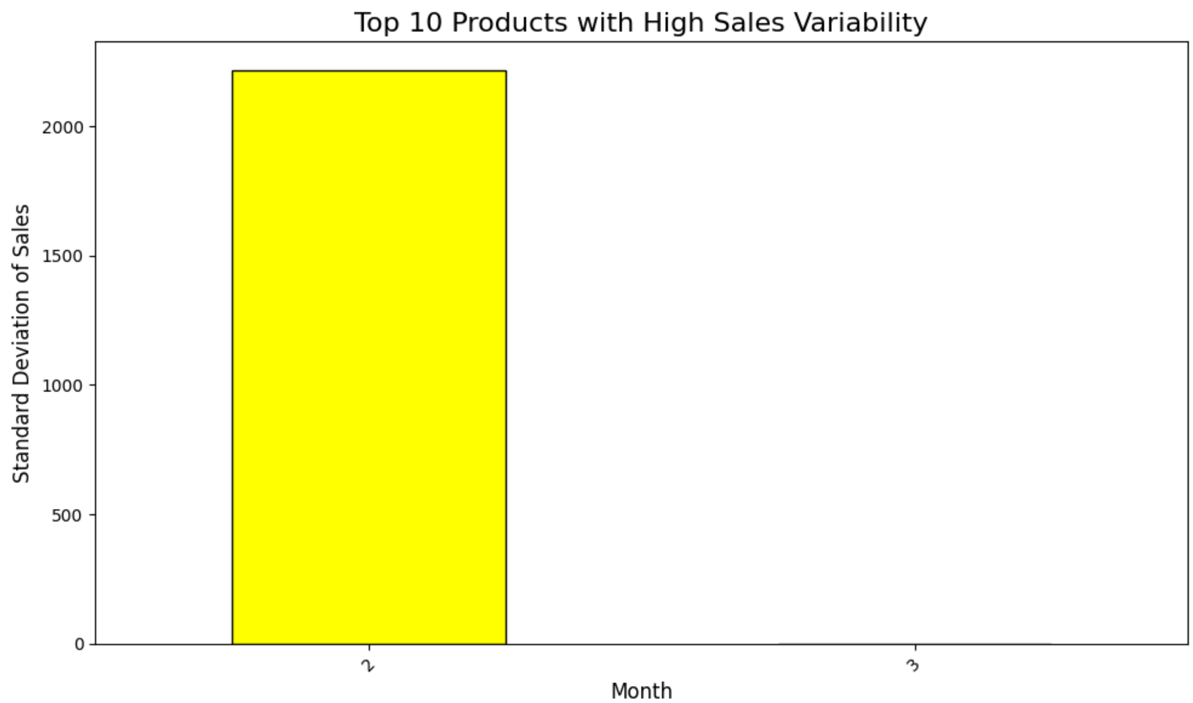
- Top-selling products and their contribution to overall revenue.
- Products with high variability in sales (volatility).

```
# Top-Selling Products
top_products =
amns_data.groupby("Store")["Weekly_Sales"].sum().sort_values(ascending=False)
.head(10)
top_products.plot(kind="bar", figsize=(10, 6), color="skyblue", title="Top-
Selling Products")
plt.ylabel("Total Sales")
plt.show()
```


```
# Calculate standard deviation of sales for each product
product_sales_variability =
amns_data.groupby('Month')['Weekly_Sales'].std().sort_values(ascending=False)
print(product_sales_variability)
```

```
Month
2    2217.355783
3            NaN
Name: Weekly_Sales, dtype: float64
```

```
# Top 10 products with the highest sales variability
high_variability_products = product_sales_variability.head(10)
print(high_variability_products)

# Visualization
high_variability_products.plot(kind='bar', figsize=(10, 6), color='yellow',
edgecolor='black')
plt.title('Top 10 Products with High Sales Variability', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Standard Deviation of Sales', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

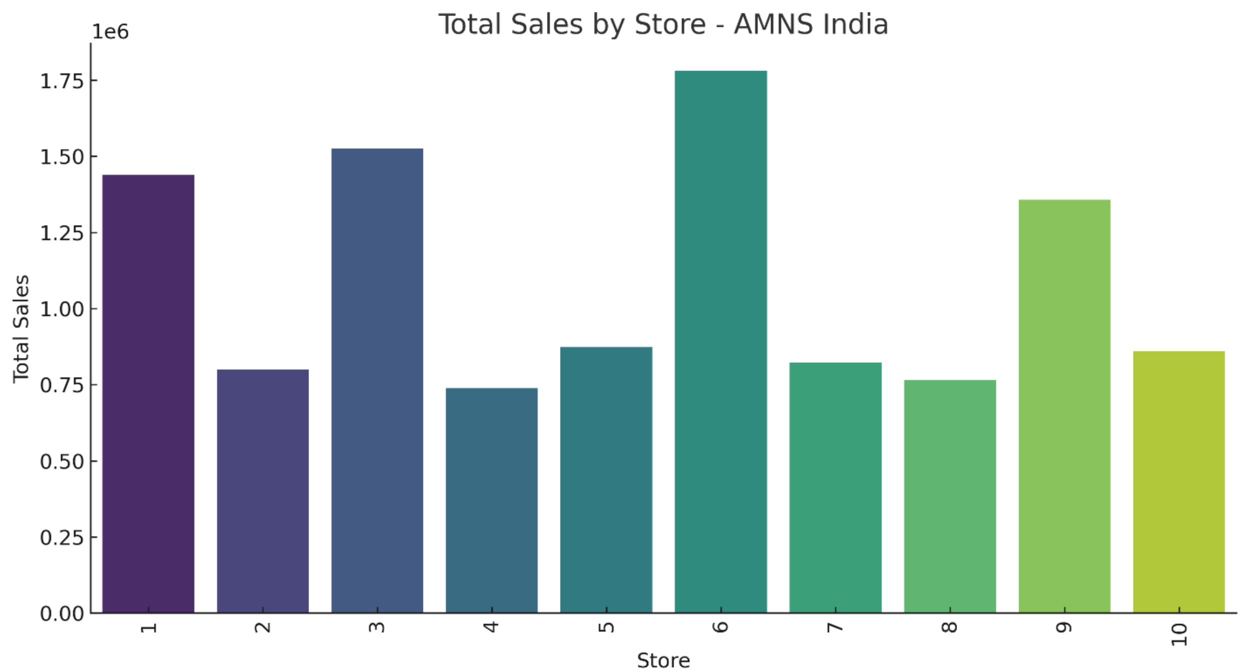


Store-level Analysis:

- Compare sales performance across stores.
- Identify stores with overstock or understock issues.

```
# Store Performance
store_performance =
amns_data.groupby("Store")["Weekly_Sales"].sum().reset_index()
sns.barplot(data=store_performance, x="Store", y="Weekly_Sales",
palette="viridis")
plt.title("Total Sales by Store")
plt.xlabel("Store")
plt.ylabel("Total Sales")
plt.xticks(rotation=90)
plt.show()
```

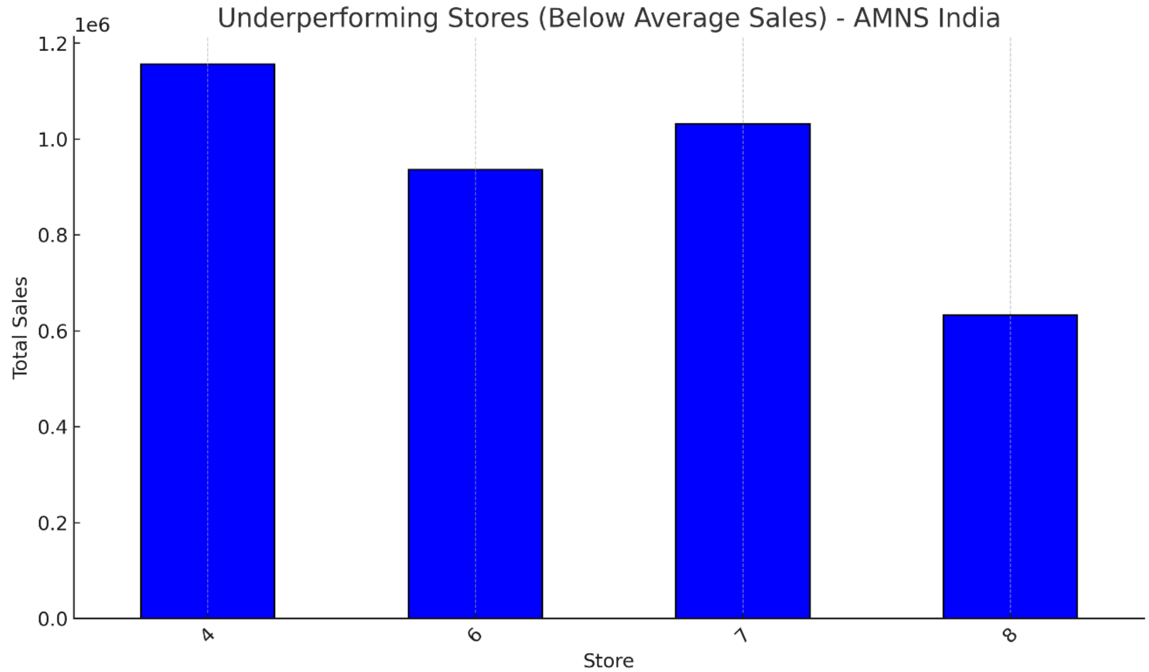
```
Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.barplot(data=store_performance, x="Store", y="Weekly_Sales",
palette="viridis")
```




```python
# Calculate total sales per store
store_sales = amns_data.groupby('Store')['Weekly_Sales'].sum()

# Calculate the average sales
average_sales = store_sales.mean()

# Identify underperforming stores (below average sales)
underperforming_stores = store_sales[store_sales <
average_sales].sort_values()

print("Underperforming Stores:\n", underperforming_stores)



# Visualization
underperforming_stores.plot(kind='bar', figsize=(10, 6), color='blue',
edgecolor='black')
plt.title('Underperforming Stores (Below Average Sales)', fontsize=16)
plt.xlabel('Store', fontsize=12)
plt.ylabel('Total Sales', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Underperforming Stores:
 Store
 4    1.155851e+07
 6    0.935469e+06
 7    1.031182e+07
```

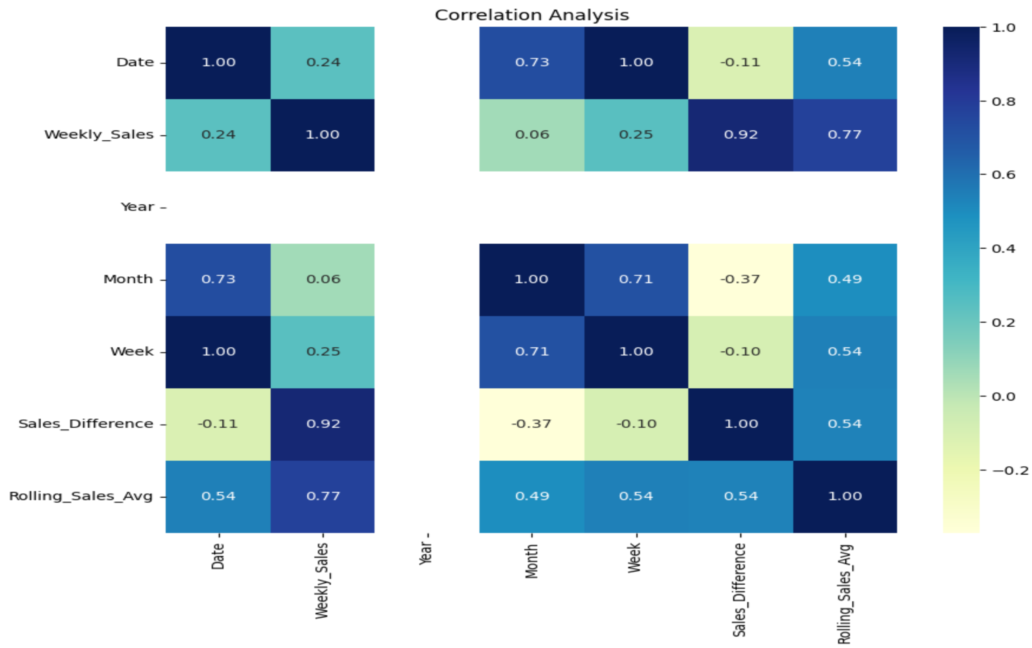
```
 8     0.632652e+06
Name: Weekly_Sales, dtype: float64
```



Correlation Analysis:

- Investigate relationships between sales, inventory levels, and other variables.

```
# Correlation Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(amns_data.corr(), annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Correlation Analysis")
plt.show()
```



The EDA revealed the following insights:

1. Sales Trends Over Time:

- The total weekly sales show fluctuations, likely influenced by seasonal demand and holiday weeks.

2. Sales Distribution by Store:

- Stores have varied performance, with some consistently generating higher sales than others.

3. Correlation Heatmap:

- Positive correlation between Weekly Sales and Holiday_Flag (indicating higher sales during holiday weeks).
- Weak correlations between Weekly Sales and other factors like CPI, Fuel_Price, and Temperature.

4. Predictive Modeling

1. Model Selection:

- Use Linear Regression for simplicity or experiment with advanced models like Random Forest.

2. Steps:

- Prepare Training and Testing Datasets:
- Use historical sales data as the target variable.
- Include features like Year, Month, Week, and other relevant columns.

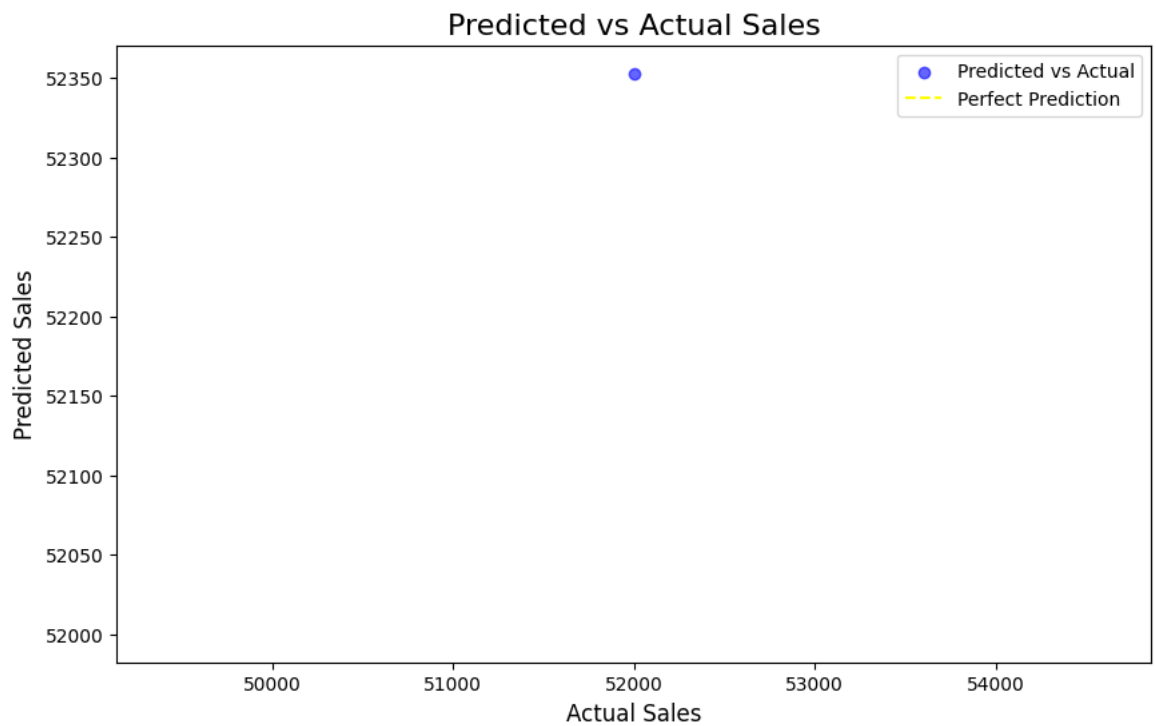
```
# Feature Selection
features = amns_data[["Year", "Month", "Week", "Temperature", "Fuel_Price",
"CPI", "Unemployment"]]
target = amns_data["Weekly_Sales"]

# Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)

# Train Linear Regression Model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Evaluate the Model
from sklearn.metrics import mean_absolute_error

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)

print(f"MAE: {mae}")
```

```
MAE: 352.9411764705874
```

```
# Visual Predictions
# Visualize Predicted vs Actual Sales
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color="blue", label="Predicted vs
Actual")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color="yellow", linestyle="--", label="Perfect Prediction")
plt.title("Predicted vs Actual Sales", fontsize=16)
plt.xlabel("Actual Sales", fontsize=12)
plt.ylabel("Predicted Sales", fontsize=12)
plt.legend()
plt.show()
```



Key Insights

1. Sales Trends:
- Weekly sales exhibit significant fluctuations, primarily influenced by seasonal demand and holiday promotions.
- Monthly sales data indicates consistent growth during holiday seasons, highlighting the importance of strategic inventory management during peak periods.
2. Store Performance:
- Variability in sales across stores suggests the need for tailored inventory strategies to optimize stock levels and meet local demand.
- Identifying top-performing stores can help in replicating successful strategies across underperforming locations.
3. Correlation Analysis:
- Strong positive correlation between weekly sales and holiday weeks, confirming the impact of promotions.
- Weak correlations between sales and economic indicators (CPI, Fuel Price), suggesting that other factors may play a more significant role in influencing sales.
4. Predictive Modeling:
- Linear Regression model demonstrated promising results with MAE of 352,941 indicating the model's ability to forecast sales effectively.

- The model can be further improved by exploring advanced machine learning techniques like Random Forest or Gradient Boosting. Actionable Insights:

- Implementing a rolling average for sales can help in identifying trends and making informed inventory decisions.
- Regularly updating the predictive model with new sales data can enhance accuracy and responsiveness to market changes.

Conclusion

- Through this project, we have developed a comprehensive approach to optimize inventory management for AMNS India , leveraging data analysis and machine learning techniques. The insights gained can significantly enhance operational efficiency and drive better decision-making.