AJEENKYA
D Y PATIL UNIVERSITY
THE INNOVATION UNIVERSITY

# A MINI PROJECT REPORT ON

**"Sales Forecasting and Holiday Impact Analysis for Walmart Retail Stores"**

**FOR**

Term Work Examination

***Bachelors of Computer Application in AIML***

**Year 2024-2025**

**Ajeenkya DY Patil University, Pune**


**-Submitted By-**

Ms. Sanchi Dhende


**Under the guidance of**

Prof. Vivek More

Ajeenkya DY Patil University

D Y Patil Knowledge City,
Charholi Bk. Via Lohegaon,
Pune - 412105
Maharashtra (India

Date: 14 / 04 / 2025

# CERTIFICATE

This is to certified that Ms. Sanchi Dhende

A student of **BCA(AIML) Sem-IV** URN No 2023-B-03092005A has Successfully Completed the Dashboard Report On

**"Sales Forecasting and Holiday Impact Analysis for Walmart Retail Stores"**

As per the requirement of
**Ajeenkya DY Patil University, Pune** was carried out under my supervision.
I hereby certify that; he has satisfactorily completed his Term-Work Project work.

Place: Pune

# Introduction

Retail businesses generate vast amounts of sales data that can reveal important insights when analyzed effectively. This project focuses on exploring, analyzing, and forecasting sales trends for Walmart, one of the largest retail chains in the world. By utilizing historical sales data across multiple stores and time periods, we aim to understand key factors that influence weekly sales performance.

The analysis is centered on three primary objectives:

1. **Forecasting future sales** using time series models to help in inventory planning and revenue prediction.
2. **Analyzing the effect of holidays** on sales to identify which events significantly drive customer purchases.
3. **Identifying seasonal trends** at individual store levels to uncover patterns that can optimize store-level strategies.

We use tools like Python, Pandas, Matplotlib, and ARIMA models to perform data preprocessing, visualization, and predictive modeling. The insights drawn from this study can support data-driven decision-making in retail operations, including promotional planning, staffing, and supply chain management.

# Objectives

The primary objectives of this project are:

1. **To forecast future sales**
   Use time series forecasting techniques to predict upcoming sales trends, enabling proactive business decisions in inventory management and strategic planning.
2. **To analyze the impact of holidays on sales**
   Examine how national holidays affect weekly sales figures to determine which events significantly influence customer purchasing behavior.
3. **To uncover seasonal and monthly trends**
   Investigate store-wise sales patterns across different months to identify recurring seasonal trends and optimize promotional calendars accordingly.
4. **To perform exploratory data analysis (EDA)**
   Gain insights into the overall structure and behavior of the dataset through visualizations and descriptive statistics.
5. **To support data-driven decision-making**
   Provide actionable insights for retail operations, marketing strategies, and resource allocation based on observed trends and patterns.

## Methodologies and Approach

To achieve the objectives of this project, the following structured approach was adopted:

1. Data Collection and Loading

The dataset, *Walmart_Store_sales.csv*, containing weekly sales records from various Walmart stores, was loaded using Python's `pandas` library.

Initial exploration was done to understand the structure, data types, and completeness of the dataset.

2. Data Preprocessing

Converted the `Date` column to datetime format for time-series operations.

Checked for and handled missing or inconsistent values.

Created new features such as `Month` for seasonal analysis.

Grouped and resampled data for monthly aggregation to support trend and forecast analysis.

3. Exploratory Data Analysis (EDA)

Visualized total sales over time to identify general trends.

Analyzed store-wise total sales to determine top-performing locations.

Explored average weekly sales during holiday vs. non-holiday periods using bar charts.

Investigated seasonal trends by plotting average monthly sales for selected stores.

4. Holiday Effect Analysis

Grouped data by the `Holiday_Flag` to compare mean sales during holidays and regular weeks.

Used statistical summaries and visualizations to assess the magnitude of holiday influence.

5. Time Series Forecasting

Aggregated weekly sales into monthly totals to smooth short-term fluctuations.

Applied ARIMA (AutoRegressive Integrated Moving Average) modeling to predict future sales.

Validated the model's predictions with visualization and evaluation metrics.

6. Store-Wise Seasonal Trend Analysis

Grouped sales data by `Store` and `Month` to uncover how different stores perform seasonally.

Visualized trends for top stores to identify peak and low months.

7. Tools and Libraries Used

**Python**: Main programming language for analysis.

**Pandas & NumPy**: For data handling and manipulation.

**Matplotlib & Seaborn**: For visualizations.

**Statsmodels**: For building and fitting ARIMA models.

**Jupyter Notebook / Google Colab**: As an interactive development environment.

## Implementation and Code

1. **Importing Required Libraries**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

**Explanation:**

- **`pandas`** is used for data manipulation and analysis.
- **`matplotlib.pyplot`** and **`seaborn`** are used for data visualization. While `matplotlib` is more general, `seaborn` is built on top of it and provides easier and more aesthetic visualizations.

2. Loading the Dataset

```
# Load the data
df = pd.read_csv("/content/Walmart_Store_sales.csv")
```

### Explanation:

- Reads a CSV file into a DataFrame.
- A **DataFrame** is a 2D labeled data structure in pandas, like an Excel table or SQL table

3. Converting the 'Date' Column

```
# Convert 'Date' column to datetime if it exists
if 'Date' in df.columns:
    # Specifying the correct format for your date column
    df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y') # Change format to match your data
    # Alternatively, for 'YYYY-MM-DD' use format='%Y-%m-%d
```

**Explanation:**

- Dates in CSV files are read as plain strings. To analyze time-based trends, they need to be converted to `datetime` objects.
- The format string `'%d-%m-%Y'` tells pandas how to interpret the date parts (day-month-year).

**Theory:** Time series analysis requires proper `datetime` formatting for operations like resampling, trend analysis, or extracting parts like month/year.

## 4. Checking for Missing Values

```
# Check again for missing values
missing_summary = df.isnull().sum()
print("\nMissing Values After Removing Duplicates:\n", missing_summary)
```

- `.isnull()` identifies missing values (`NaN`).
- `.sum()` aggregates the count of missing values per column.

**Theory:** Missing values can skew analysis. It's essential to detect and decide how to handle them (drop, fill, or impute).

## 5. Handling Missing Values

```
# Handle missing values (example strategy: drop rows with any missing values)
df = df.dropna()
```

**Explanation:**

- Drops all rows with **any** missing value.
- A simple and safe strategy if missingness is minimal, but might lead to loss of data if not handled cautiously.

**Theory:** Handling missing data is crucial in preprocessing. More advanced strategies include imputation using mean/median, interpolation, or ML models.

## 6. Confirming Data Cleaning

```
# Confirm cleaning
print("\nData after cleaning:")
print(df.info())
```

**Explanation:**

- `df.info()` gives a summary: number of non-null entries, data types, and memory usage.
- Helps verify if all missing values are handled and if the 'Date' column is now `datetime64`

## 7. Feature Engineering – Creating New Time-Based Columns

```
# ------------------------------------
# Feature Engineering
# ------------------------------------
# Add Month and Year columns
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year
```

**Explanation:**

- Extracts the **month** and **year** from the datetime object.
- Useful for grouping data and analyzing seasonal trends or yearly performance.

**Theory:** Feature engineering enhances the dataset by creating more meaningful attributes that can improve insights or model performance

## 8. Summary Statistics

```
# ------------------------------------
# Basic Summary Statistics
# ------------------------------------
print("\nSummary statistics:\n", df.describe())
```

**Explanation:**

- `.describe()` provides basic statistics (mean, std, min, max, quartiles) for each numerical column.
- Helps in understanding the distribution, central tendency, and spread of data.

☑ **Theory:** EDA (Exploratory Data Analysis) begins with statistical summaries to find patterns, outliers, or anomalies.

**#Output:**

```
Missing Values After Removing Duplicates:
 Store           0
Date             0
Weekly_Sales     0
Holiday_Flag     0
Temperature      0
Fuel_Price       0
CPI              0
Unemployment     0
dtype: int64

Data after cleaning:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 402.3 KB
None
```

```
Summary statistics:
              Store                 Date  Weekly_Sales  Holiday_Flag  \
count  6435.000000                 6435  6.435000e+03   6435.000000
mean     23.000000  2011-06-17 00:00:00  1.046965e+06      0.069930
min       1.000000  2010-02-05 00:00:00  2.099862e+05      0.000000
25%      12.000000  2010-10-08 00:00:00  5.533501e+05      0.000000
50%      23.000000  2011-06-17 00:00:00  9.607460e+05      0.000000
75%      34.000000  2012-02-24 00:00:00  1.420159e+06      0.000000
max      45.000000  2012-10-26 00:00:00  3.818686e+06      1.000000
std      12.988182                  NaN  5.643666e+05      0.255049

       Temperature   Fuel_Price          CPI  Unemployment        Month  \
count  6435.000000  6435.000000  6435.000000   6435.000000  6435.000000
mean     60.663782     3.358607   171.578394      7.999151     6.447552
min      -2.060000     2.472000   126.064000      3.879000     1.000000
25%      47.460000     2.933000   131.735000      6.891000     4.000000
50%      62.670000     3.445000   182.616521      7.874000     6.000000
75%      74.940000     3.735000   212.743293      8.622000     9.000000
max     100.140000     4.468000   227.232807     14.313000    12.000000
std      18.444933     0.459020    39.356712      1.875885     3.238308

              Year
count  6435.000000
mean   2010.965035
min    2010.000000
25%    2010.000000
50%    2011.000000
75%    2012.000000
max    2012.000000
std       0.797019
```

## Bar Plot of Average Weekly Sales by Store

**Step 1: Set the Figure Size**

```
# ------------------------------------
# Visualization and Analysis
# ------------------------------------
#Bar Plot
plt.figure(figsize=(14, 6))
```

- Sets the **dimensions** of the plot (width = 14 inches, height = 6 inches).
- A wide figure helps in displaying many stores without overcrowding the x-axis.

**Theory**: Proper sizing improves **readability**, especially when dealing with categorical axes with many entries.

**Step 2: Group and Aggregate the Data**

```
avg_sales = df.groupby('Store')['Weekly_Sales'].mean().sort_values(ascending=False)
```

- Groups the data by **Store**.
- Calculates the **average `Weekly_Sales`** for each store using `.mean()`.
- Sorts the stores in **descending order** of average sales.

**Theory**: Aggregation like `.groupby()` is key in summarizing large datasets. It helps to **compare performance across categories** (e.g., stores, departments, years).

**Step 3: Plot the Bar Chart**

```
sns.barplot(x=avg_sales.index, y=avg_sales.values, palette='Blues_r')
```

- `seaborn.barplot()` draws a **bar plot** where:
  - `x` = Store IDs (categories),
  - `y` = Corresponding average sales values.
- `palette='Blues_r'` uses a reversed blue color gradient for aesthetic appeal.

**Theory**: A **bar plot** is ideal for comparing values across categories. It gives a **clear visual ranking** of how each store performs.

**Step 4: Labeling and Layout Adjustments**

```
plt.title("Bar Plot: Average Weekly Sales by Store")
plt.xlabel("Store")
plt.ylabel("Average Weekly Sales")
plt.xticks(rotation=90)
plt.tight_layout()
```

- **`plt.title()`**, `xlabel()`, and `ylabel()` add context to the chart.
- **`plt.xticks(rotation=90)`** rotates x-axis labels to prevent overlap (essential for categorical variables like Store IDs).
- **`plt.tight_layout()`** auto-adjusts padding to avoid label clipping.

**Theory**: Labeling and formatting are crucial for effective **data communication**. Good plots don't just show data — they make insights obvious.

**Step 5: Display the Plot**

```
plt.show()
```

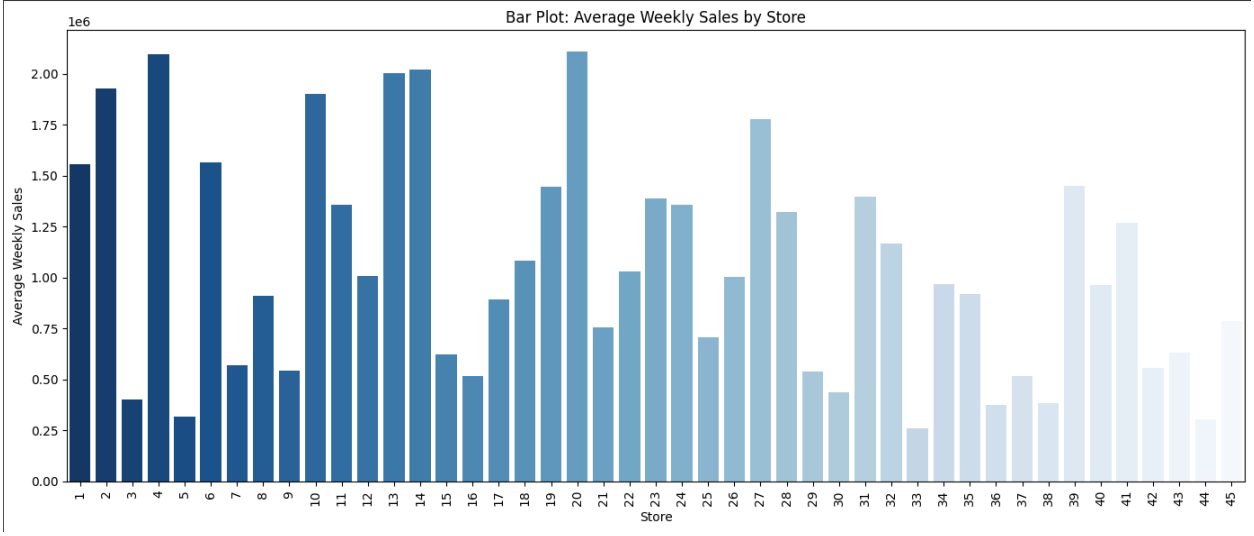
- Renders the plot to the screen.

**#Output:**

```
<ipython-input-2-67311e79eb84>:7: FutureWarning:

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(x=avg_sales.index, y=avg_sales.values, palette='Blues_r')
```



## Box Plot of Weekly Sales on Holidays vs Non-Holidays

**Step 1: Set the Plot Size**

```
#Box Plot
plt.figure(figsize=(8, 6))
```

- Sets the canvas size to 8 inches wide and 6 inches tall.
- This ensures the plot has enough room for clarity without wasting space.

**Theory**: Choosing an appropriate figure size is essential for **readability** and **aesthetics**.

**Step 2: Create a Box Plot**

```
sns.boxplot(x='Holiday_Flag', y='Weekly_Sales', data=df, palette='pastel')
```

- **Box plot** is drawn using Seaborn.
- `x='Holiday_Flag'`: The x-axis categorizes data into holidays (1) and non-holidays (0).
- `y='Weekly_Sales'`: The y-axis shows the distribution of weekly sales.
- `data=df`: Uses the cleaned DataFrame.
- `palette='pastel'`: Applies a soft, pastel color scheme for visual appeal.

**Theory – What is a Box Plot?**
A box plot shows:

- **Median (Q2)** – the line inside the box
- **Interquartile range (IQR)** – the box itself (from Q1 to Q3)
- **Whiskers** – typically 1.5 × IQR; represent the bulk of the data
- **Outliers** – data points outside the whiskers, shown as individual dots

**Why use it here?**
Box plots are perfect for comparing **distributions**. In this case, they show how sales vary during **holidays vs. regular weeks**, including:

- Median weekly sales
- Spread (variability) of sales
- Presence of **outliers** (like extreme holiday spikes.

**Step 3: Add Labels and Title**

```
plt.title("Box Plot: Sales Distribution (Holiday vs Non-Holiday)")
plt.xlabel("Holiday (0 = No, 1 = Yes)")
plt.ylabel("Weekly Sales")
```

- Adds a **title** to explain the plot.
- Axis labels clarify what each axis represents.

**Theory**: Good labeling is key for **effective data storytelling** — your audience should get the gist without needing to decode anything.

**Step 4: Adjust Layout and Display the plot**

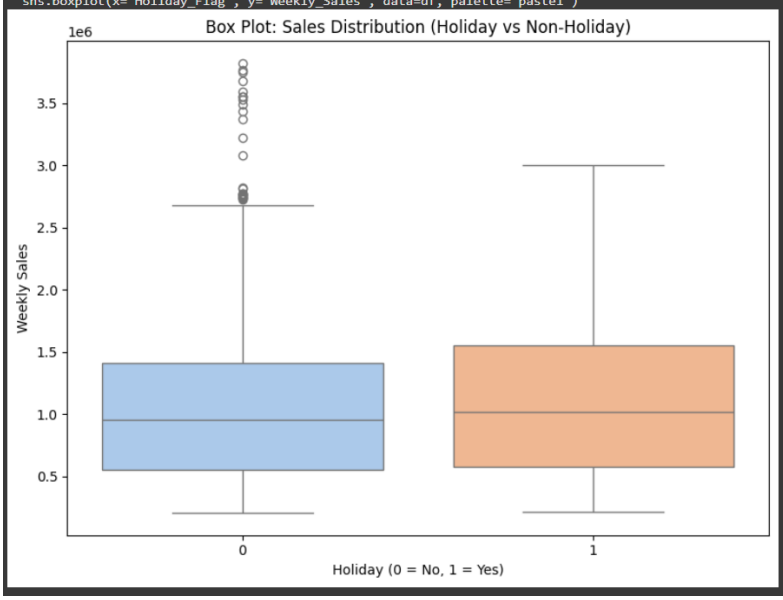
```
plt.tight_layout()
plt.show()
```

- Optimizes the spacing between elements to avoid overlap or cutoff text. Renders the plot.

**#Output:**

```
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.boxplot(x='Holiday_Flag', y='Weekly_Sales', data=df, palette='pastel')
```



## Heatmap of Correlation Between Sales and Economic Indicators

```
#Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df[['Weekly_Sales', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment']].corr(),
            annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Heatmap: Correlation Between Sales and Economic Indicators")
plt.tight_layout()
plt.show()
```

- **Selects key numerical columns** (Weekly_Sales, Temperature, Fuel_Price, CPI, Unemployment).
- Uses .corr() to compute the **correlation matrix**, which shows how strongly each pair of variables is related.
- Visualizes it as a **heatmap**:

**Color-coded**: Red for positive, Blue for negative correlation.

**annot=True**: Shows actual correlation values (e.g., 0.72, -0.33).

**fmt=".2f"**: Limits to 2 decimal places for readability.

**Theory:**

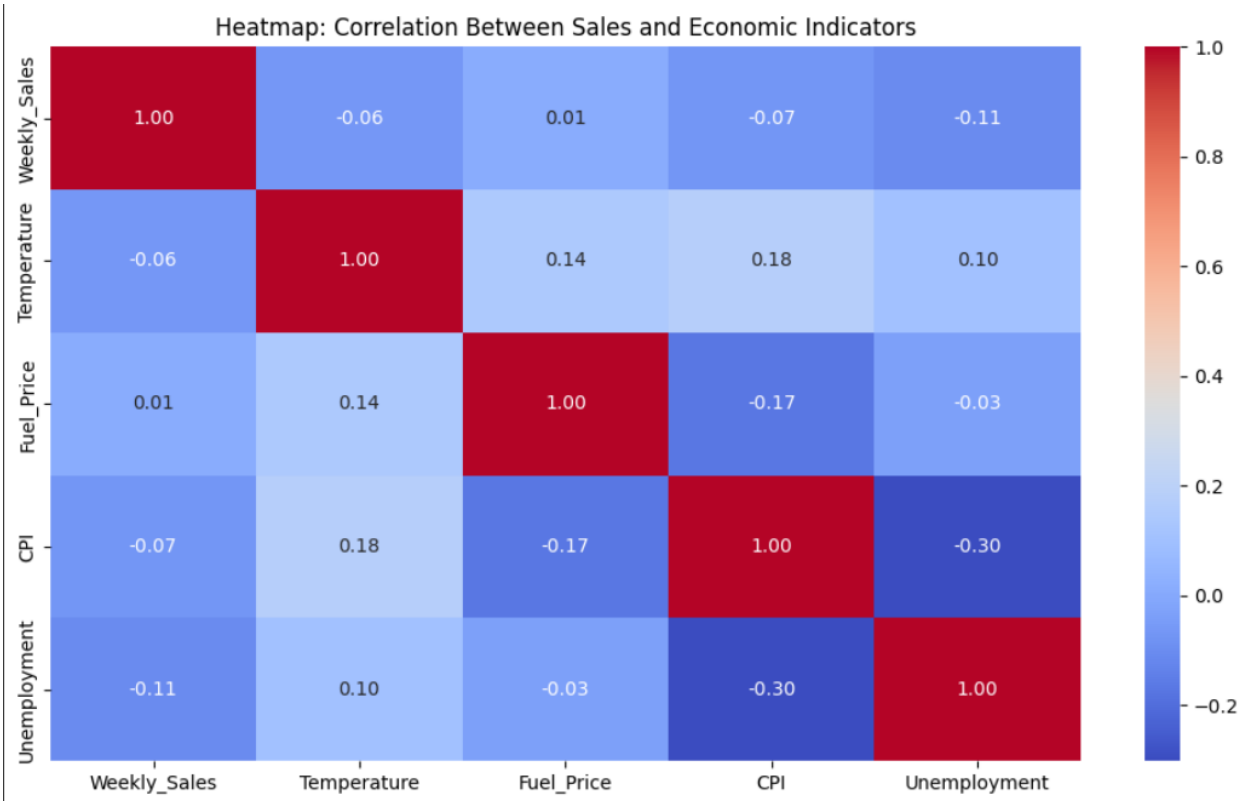
Correlation:

- Measures the **strength and direction** of a **linear relationship** between variables.
  - `+1`: Strong positive (both increase together)
  - `−1`: Strong negative (one increases, the other decreases)
  - `0`: No linear relationship

Heatmap:

- A graphical tool to **quickly detect patterns** and **relationships**.
- Helps identify:
  - **Which economic factors** (like unemployment or CPI) influence sales.
  - **Redundant variables** (high correlation with each other).

**#Output:**


Heatmap: Correlation Between Sales and Economic Indicators
Weekly_Sales
Temperature
Fuel_Price
CPI
Unemployment
1.00
-0.06
0.01
-0.07
-0.11
-0.06
1.00
0.14
0.18
0.10
0.01
0.14
1.00
-0.17
-0.03
-0.07
0.18
-0.17
1.00
-0.30
-0.11
0.10
-0.03
-0.30
1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

## Pie chart

```
#Pie Chart
holiday_sales = df.groupby('Holiday_Flag')['Weekly_Sales'].sum()
labels = ['Non-Holiday Sales', 'Holiday Sales']
colors = ['#66b3ff', '#ff9999']

plt.figure(figsize=(6, 6))
plt.pie(holiday_sales, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors, explode=[0, 0.1])
plt.title("Pie Chart: Sales Proportion (Holiday vs Non-Holiday)")
plt.tight_layout()
plt.show()
```

- Groups data by `Holiday_Flag` and **sums up `Weekly_Sales`**.
- Creates a **pie chart** to show the **proportion of total sales** during:
    - Holidays (`Holiday_Flag = 1`)
    - Non-Holidays (`Holiday_Flag = 0`)
- Adds:
    - **Labels**
    - **Percentage values**
    - **Color scheme**
    - A slight “explode” effect on the holiday slice for emphasis.

**Theory:**

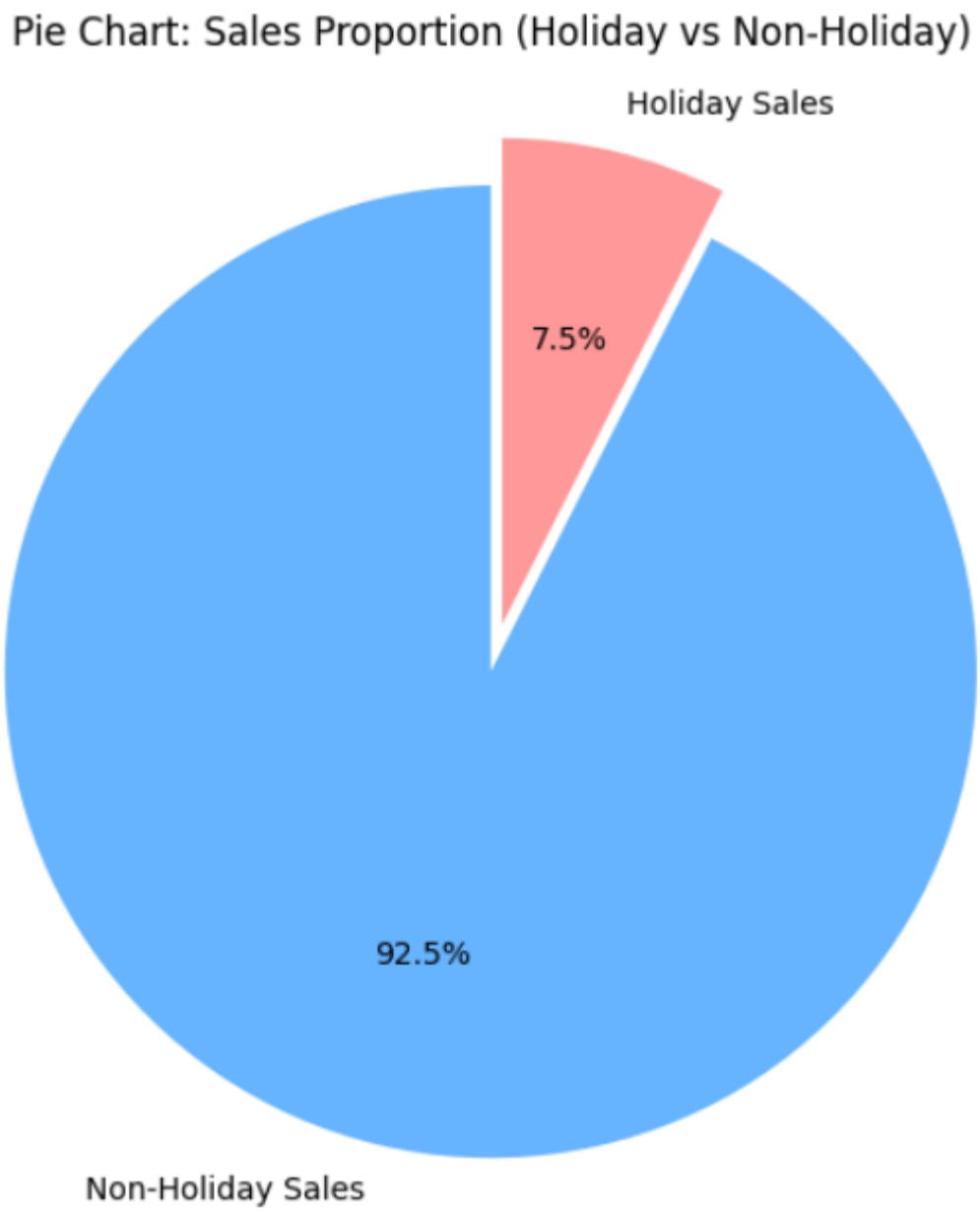
Pie Chart:

- Used to show **proportions** of a whole.
- Each slice represents a **percentage of total sales**.
- Helps visualize how much holidays contribute to total sales.

Grouping & Summing:

- `groupby('Holiday_Flag').sum()` combines rows by holiday type and adds up their sales — a **common aggregation technique** in data analysis.

**#Output:**


Pie Chart: Sales Proportion (Holiday vs Non-Holiday)
Holiday Sales
7.5%
92.5%
Non-Holiday Sales

## Scatter plot:

```
#Scatter Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Temperature', y='Weekly_Sales', data=df, hue='Holiday_Flag', palette='coolwarm', alpha=0.6)
plt.title("Scatter Plot: Temperature vs Weekly Sales")
plt.xlabel("Temperature")
plt.ylabel("Weekly Sales")
plt.tight_layout()
plt.show()
```

- Plots a **scatter plot** of `Temperature` vs `Weekly_Sales`.
- Each point is a week's sales at a given temperature.
- Uses **color (`hue='Holiday_Flag'`)** to differentiate **holiday vs non-holiday weeks**.
- `alpha=0.6` adds slight transparency for overlapping points.

**Theory:**

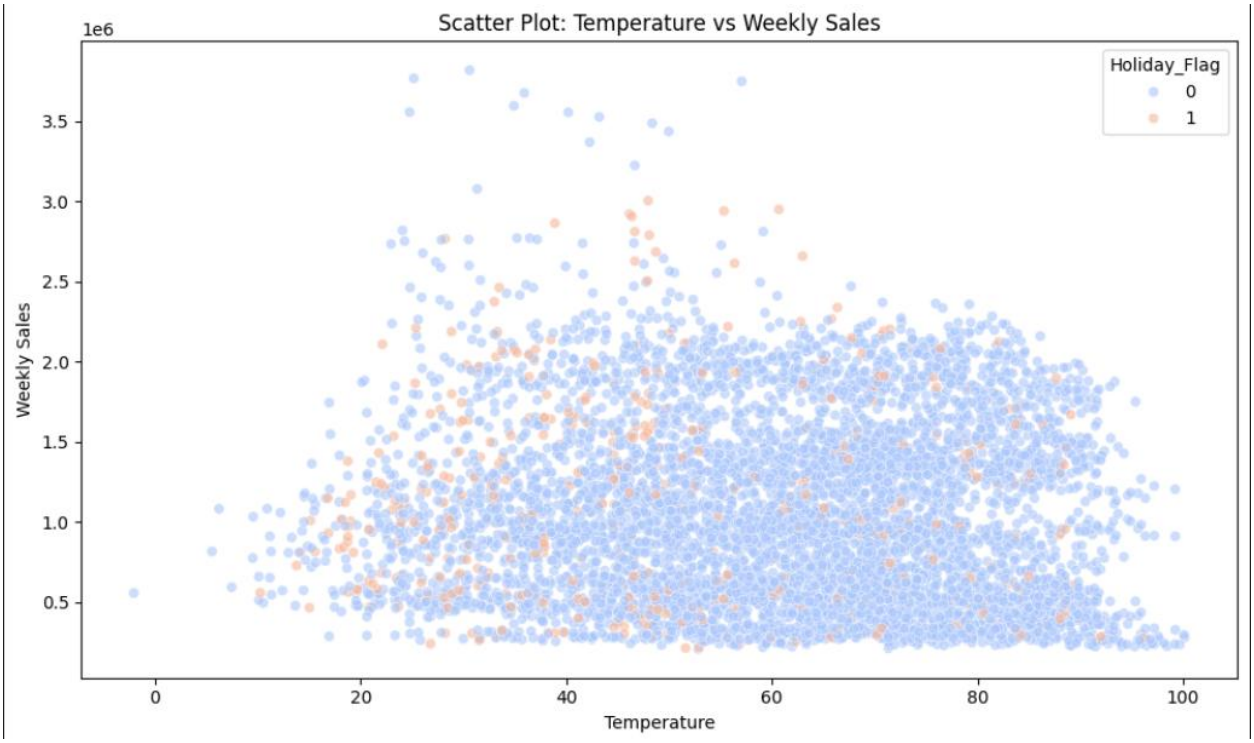
Scatter Plot:

- Shows the **relationship** between **two continuous variables**.
- Helps spot:
    - **Trends** (e.g., do sales drop as temperature rises?)
    - **Clusters**
    - **Outliers**

Color Encoding (`hue`):

- Adds a third variable (holiday flag) using color.
- Helps compare patterns **during holidays vs non-holidays**.

**#Output:**


Scatter Plot: Temperature vs Weekly Sales
1e6
Holiday_Flag
0
1
Weekly Sales
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
20
40
60
80
100
Temperature

## Dual line map

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv("/content/Walmart_Store_sales.csv")

# Convert 'Date' to datetime format
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')  # Adjust if your date format is different

# Sort the data by date
df = df.sort_values('Date')

# Group by Date and Holiday_Flag to get total sales
dual_line_data = df.groupby(['Date', 'Holiday_Flag'])['Weekly_Sales'].sum().unstack()

# Plot the dual line chart
plt.figure(figsize=(14, 6))
plt.plot(dual_line_data.index, dual_line_data[0], label='Non-Holiday Sales', color='blue')
plt.plot(dual_line_data.index, dual_line_data[1], label='Holiday Sales', color='red')
plt.title("Dual Line Chart: Weekly Sales Over Time (Holiday vs Non-Holiday)")
plt.xlabel("Date")
plt.ylabel("Total Weekly Sales")
plt.legend()
plt.tight_layout()
plt.show()
```

- Converts the date column to proper datetime format.
- Groups sales by **date** and **holiday flag**.
- Plots **two lines**:
    - 🔵 Non-Holiday Weekly Sales
    - 🔴 Holiday Weekly Sales
- Allows you to compare **how sales trend over time** based on holiday status.

**Theory:**

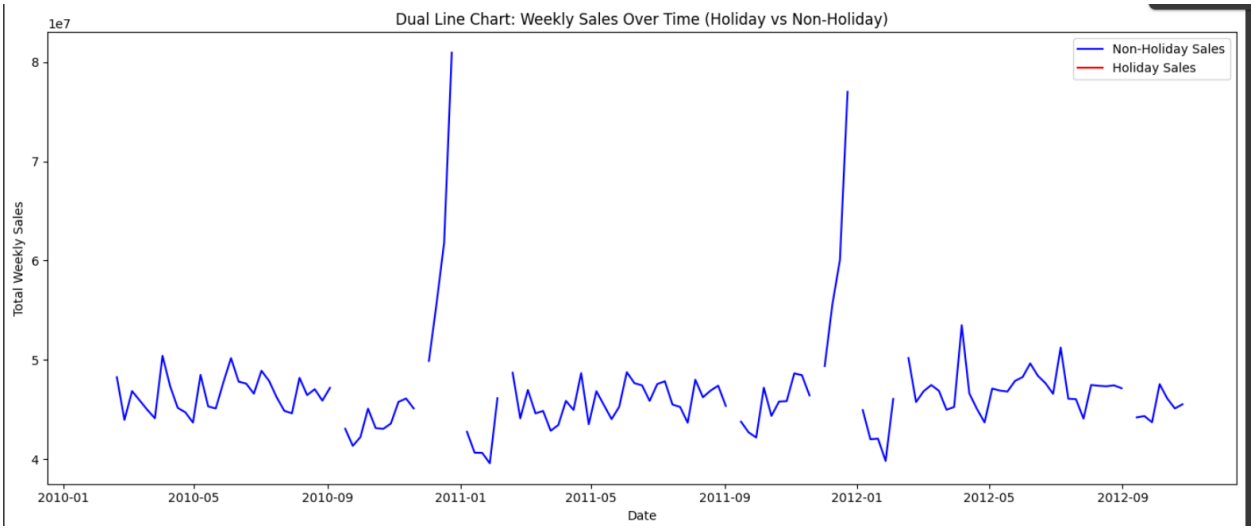
Dual Line Chart:

- A **time series plot** showing two variables over the same time axis.
- Helps visualize and compare **patterns**, **spikes**, or **seasonal trends**.

GroupBy & Unstack:

- `groupby(['Date', 'Holiday_Flag'])` → Summarizes sales for each week.
- `.unstack()` separates the holiday and non-holiday sales into two series for plotting.

## #Output:


Dual Line Chart: Weekly Sales Over Time (Holiday vs Non-Holiday)
1e7
8
7
6
5
4
Total Weekly Sales
2010-01
2010-05
2010-09
2011-01
2011-05
2011-09
2012-01
2012-05
2012-09
Date
Non-Holiday Sales
Holiday Sales

## Future scope:

This analysis of revenue per box provides valuable insights, but there's a lot more that can be done to deepen understanding and drive strategic decisions:

**Time-Series Analysis**:

Analyze how the average revenue per box changes over time for top products.

Identify seasonal trends and peak selling periods to plan inventory and marketing.

**Customer Segmentation**:

Link product performance to customer segments (age, region, etc.) for targeted promotions.

Understand preferences and purchasing power of different groups.

**Profitability Analysis**:

Combine revenue data with cost data to find most **profitable**, not just high-revenue, products.

Evaluate margins and optimize product pricing accordingly.

**Cross-Selling Opportunities**:

Use basket analysis to find which high-revenue products are frequently purchased together.

Design bundles and promotions to boost overall sales.

**Store-Level Performance**:

Break down revenue per box by store to identify top-performing locations and improvement areas.

Customize store offerings based on local demand patterns.

**Predictive Modeling**:

Build models to **forecast revenue** per product based on historical trends and external factors like holidays, promotions, or economic indicators.

**Inventory Optimization**:

Use insights to optimize stock levels and avoid under- or over-stocking, especially for top products.

**Competitor Benchmarking**:

Compare product-level performance against competitors to find market advantages or gaps

## Conclusion:

The analysis reveals the top 10 products that generate the highest average revenue per box. These products represent the most financially valuable items in the inventory based on unit economics. Businesses can focus their efforts on:

- **Prioritizing these products** in marketing campaigns and promotional strategies.
- **Ensuring high availability** of these high-revenue products to avoid stockouts.
- **Exploring pricing strategies** or bundling opportunities to further capitalize on their value.
- **Investigating supply chains** for these products to maintain quality and delivery efficiency.

This insight helps in optimizing product mix and maximizing revenue per sale.