Introduction

Objective

The main objective of this project is to develop a machine learning model that can accurately predict weekly sales for each department in different retail stores. By using historical sales data along with additional information such as store type, size, promotions, holidays, and economic indicators (like CPI and unemployment), the goal is to understand the key factors that influence sales and create a reliable system to forecast future sales. This will help retail businesses improve inventory management, optimize resource allocation, and make better data-driven decisions.

Project Overview

The Retail Sales Prediction project is about building a machine learning model to predict weekly sales for retail stores. It uses three datasets: one with past weekly sales, one with extra details like holidays, discounts, fuel prices, unemployment, and CPI (a measure of inflation), and one with store information such as type and size.

The main goal of this project is to help stores plan better by knowing in advance how much they will sell. This can help them manage their stock, plan their staff, and run offers at the right time. The first step in the project is to clean the data. Missing values in discount columns are filled with zero, because not all stores had discounts. For economic values like CPI and unemployment, missing values are filled by carrying the last known value forward. After cleaning, the three datasets are combined into one, so it's easier to work with.

Then, we explore the data using graphs like histograms and line plots to understand patterns, such as how holidays and discounts affect sales. Based on what we learn from the data, we prepare the final dataset and train a machine learning model. This model looks at store details, dates, holidays, markdowns, and other features to predict future sales.

The final result helps businesses make better decisions, avoid overstock or shortage, and keep their customers happy by always having the right products available.

Importance of Retail Sales Prediction

Retail sales prediction plays a critical role in the success of any retail business. In today's highly competitive and fast-moving market, being able to anticipate future demand is not just helpful—it is essential. Sales can be influenced by many factors such as holidays, promotions, economic conditions, store locations, and even weather. By analyzing historical data and identifying patterns, businesses can make smarter decisions and avoid costly mistakes.

Key areas where sales prediction provides major benefits:

1. Improved Inventory Management

Accurate sales forecasts help businesses maintain the right level of inventory. If a store overestimates sales, it may end up with unsold products that take up space and tie up capital. On the other hand, underestimating sales can lead to stockouts, missed sales opportunities, and unhappy customers. Predicting sales ensures that stores stock what is needed at the right time, reducing both excess inventory and shortages.

---

2. Better Workforce and Resource Planning

Retailers can use sales forecasts to plan staffing levels more effectively. During peak seasons or promotional events, stores need more employees to manage increased customer traffic and maintain smooth operations. With predicted sales data, businesses can schedule employees accordingly, reducing labor costs during slower periods and increasing customer service quality during busy times.

---

3. Smarter Promotional Strategies

Sales predictions also guide when and how to run marketing campaigns or discounts. By understanding which times of the year or which conditions lead to higher sales, retailers can launch promotions when they will have the most impact. For example, if sales are usually high before a holiday, a well-timed promotion can boost sales even more. This ensures that marketing budgets are spent efficiently.

---

4. Supply Chain Optimization

Accurate forecasting supports better coordination with suppliers. Knowing which products will be in demand allows businesses to place timely orders and avoid delays. This also helps in managing warehouse space and transportation costs more effectively. A well-functioning supply chain based on predicted demand leads to fewer disruptions and more consistent product availability.

---

5. Cost Savings and Waste Reduction

Good predictions reduce the chances of over-ordering products that may go unsold or expire. This not only saves money but also helps in reducing waste, especially for perishable goods. Efficient operations based on solid forecasts also lead to lower operating costs, which contributes to better profit margins.

---

6. Enhanced Customer Experience

When products are available when and where customers want them, satisfaction naturally increases. Nothing is more frustrating for a customer than visiting a store and not finding the product they need. By aligning stock levels with predicted demand, businesses can ensure a smoother and more reliable shopping experience.

---

7. Strategic Business Planning

Retail sales predictions are not just for day-to-day decisions—they also play a role in long-term planning. Companies use forecasts to set revenue goals, plan expansions, allocate budgets, and evaluate performance. It supports data-driven decision-making across the entire business.

---

8. Gaining a Competitive Edge

Finally, being able to accurately predict sales gives businesses an edge over competitors. In a market where trends change rapidly, the ability to quickly respond to changing customer needs and market conditions is a major advantage. Businesses that use prediction models can adapt faster and serve their customers better.

Data Understanding

Overview of Dataset-:

The project utilizes three primary datasets: features.csv, stores.csv, and train.csv. Each dataset contributes unique information that, when combined, provides a comprehensive view of the retail sales environment.

1. **features.csv_:**
   This dataset contains additional data related to each store and date, such as:

   - **Temperature**: Average temperature for that week.
   - **Fuel_Price**: Cost of fuel in the area.
   - **MarkDown1–5**: Promotional markdowns on various product categories.
   - **CPI (Consumer Price Index):** Indicator of inflation.
   - **Unemployment**: Local unemployment rate.
   - **IsHoliday**: Indicates whether the week contains a special holiday**.**

   These features are critical in identifying external factors that can influence sales.

2. **stores.csv-:**
   This dataset includes basic information about each store:
   - **Store:** All stores have Unique store ID**.**
   - **Type:** Type classification of the store (A, B, or C), representing different formats or sizes.
   - **Size:** Square footage of the store, which could relate to customer footfall and inventory.

   Understanding the differences between store types and sizes can help the model detect varying sales patterns.
3. **train.csv**
   This is the core dataset used for training the prediction model. It contains:
   - **Store**: Store identifier.
   - **Dept**: Department number.
   - **Date**: Weekly date for sales entry.
   - **Weekly_Sales**: Target variable, representing sales for a particular department in a store during a specific week.

- **IsHoliday**: Boolean indicator for holiday weeks.

This dataset is the backbone of the project, containing historical sales trends that the model learns from

Key Attributes and Their Descriptions-:

- **Store:** Unique identifier for each store.
- **Dept:** Unique identifier for each department within a store.
- **Date:** The week of the sales record.
- **Weekly_Sales:** Target variable. Sales amount recorded for a department in a given week.
- **IsHoliday:** Boolean flag indicating whether the week includes a special holiday.
- **Temperature:** Average weekly temperature in the store's region (in Fahrenheit).
- **Fuel_Price:** Cost of fuel in the area (per gallon).
- **MarkDown1–5:** Promotional markdown values for various product categories.
- **CPI:** Consumer Price Index — reflects inflation and cost-of-living changes.
- **Unemployment:** Unemployment rate in the store's region.
- **Type:** Type classification of the store: A, B, or C.
- **Size:** Size of the store (in square feet).
- **Year:** Extracted from the date — useful for time-based trend analysis.
- **Month:** Extracted from the date — captures seasonal effects.
- **Week:** Week number of the year — useful for weekly trends.

Data Volume and Format

This project utilizes three structured CSV files: train.csv, features.csv, and stores.csv. Each dataset varies in size, dimensionality, and the type of information it provides. Below is a breakdown of their volume and structure:

1. train.csv

- Rows: 421,570
- Columns: 5
- Description: Contains historical weekly sales data by Store, Department, and Date.

- Format: Tabular, with columns such as Store, Dept, Date, Weekly_Sales, and IsHoliday.
- Target Variable: Weekly_Sales

2. features.csv

- Rows: 8190
- Columns: 12
- Description: Contains additional economic and promotional information per store and date, such as temperature, fuel price, markdowns, CPI, and unemployment rate.
- Format: Tabular, with mixed data types (numerical, boolean, and date).

3. stores.csv

- Rows: 45
- Columns: 3
- Description: Static information about each store including its type and size.
- Format: Tabular, with categorical and numerical attributes.

4. Combined Dataset (after merging)

- **Rows**: 421,570 (same as train.csv)
- **Columns:** 18+ (depending on feature engineering)
- Joined On: Store and Date
- Ready for: Feature extraction, modeling, and sales forecasting.
- **Target Variable**: Weekly_Sales

These datasets provide a well-rounded combination of time series, categorical, economic, and contextual data—ideal for building robust predictive models.

Data Preprocessing

Handling Missing Values-:

Handling missing data is a crucial step in preparing datasets for analysis and modeling. In this project, missing values were primarily observed in the features.csv dataset, particularly in the Markdown-related columns (**MarkDown1** to **MarkDown5**), as well as in economic indicators such as **CPI** and **Unemployment**.

- **Markdown Columns MarkDown1** to **MarkDown5** columns represent promotional discount features, which were missing for several records. These missing values were not random but rather due to business logic—certain stores did not participate in markdown promotions during specific weeks. Therefore, it was reasonable to assume a value of zero for these cases. The missing values in these columns were filled using.

  This approach ensures the model does not misinterpret missing markdown data as unknown values, but rather as "no promotion".

```
#here we directely filled with zero because all markdown data are discount based so
#some stores not get
#the discount according to company policy on our product
pd_1[["MarkDown1", "MarkDown2", "MarkDown3", "MarkDown4", "MarkDown5"]] = \
pd_1[['MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5']].fillna(0)
```

- **CPI and Unemployment**

  The CPI (Consumer Price Index) and Unemployment columns are time-series economic indicators. Missing values in these columns were likely due to data collection gaps or delays. Given their temporal nature, forward filling was applied to propagate the last valid observation forward-

```
#here we use forward filling because this data are based
#The Consumer Price Index (CPI) measures the average change
# in prices of goods and services over time. It is an
#indicator of inflation and cost of living.
pd_1['CPI'] = pd_1['CPI'].ffill()
#This represents the unemployment rate, which is the percentage
pd_1['Unemployment'] = pd_1['Unemployment'].ffill()
```

Forward filling is appropriate here as economic indicators generally change gradually over time, and the previous value is often a good estimate for the next.

Data Merging

it is important to combine all the relevant information from the three different files: `train.csv`, `features.csv`, and `stores.csv`. Each file provides different types of data. The `train.csv` file contains historical weekly sales data, `features.csv` includes extra information like holidays, markdown discounts, fuel prices, CPI, and unemployment, and `stores.csv` contains store-level data such as store type and size.

- Now we merged the `train` data with the `stores` data using the common column `Store`. This step adds store-level information to each sales record:

```
    # Merge train with stores on "Store"
    merged_df = pd_3.merge(pd_2, on="Store", how="left")
```

- Aftre that we merged this result with the features dataset using Store, Date, and IsHoliday as the common columns. This final merge gives us one complete dataset with sales, store details, and external factors:

```
# Merge train + stores with features on "Store" and "Date"
merged_df = merged_df.merge(pd_1, on=["Store", "Date"], how="left")
```

Exploratory Data Analysis (EDA)

1. Univariate Analysis

Univariate analysis was performed to understand the distribution and nature of individual variables in the dataset. It helps identify central tendencies, spread, skewness, and potential data quality issues such as missing values or outliers.

Objective: Understand distribution, detect skewness, and identify outliers.

i. Variable Description:

- **Weekly_Sales:** This is the target variable representing the total sales for a given store in a particular week. It's a continuous, numerical variable.

ii. Central Tendency:

- **Mean:** The average weekly sales across all stores and weeks. Calculate this using merged_cleaned['Weekly_Sales'].mean().
- **Median:** The middle value of weekly sales when the data is sorted. Use merged_cleaned['Weekly_Sales'].median().
- **Mode:** The most frequent weekly sales value (if any). Use merged_cleaned['Weekly_Sales'].mode().

Monthly Sales Analysis:

Objective: To analyze monthly sales patterns, identify trends, and understand the impact of store type on sales performance.

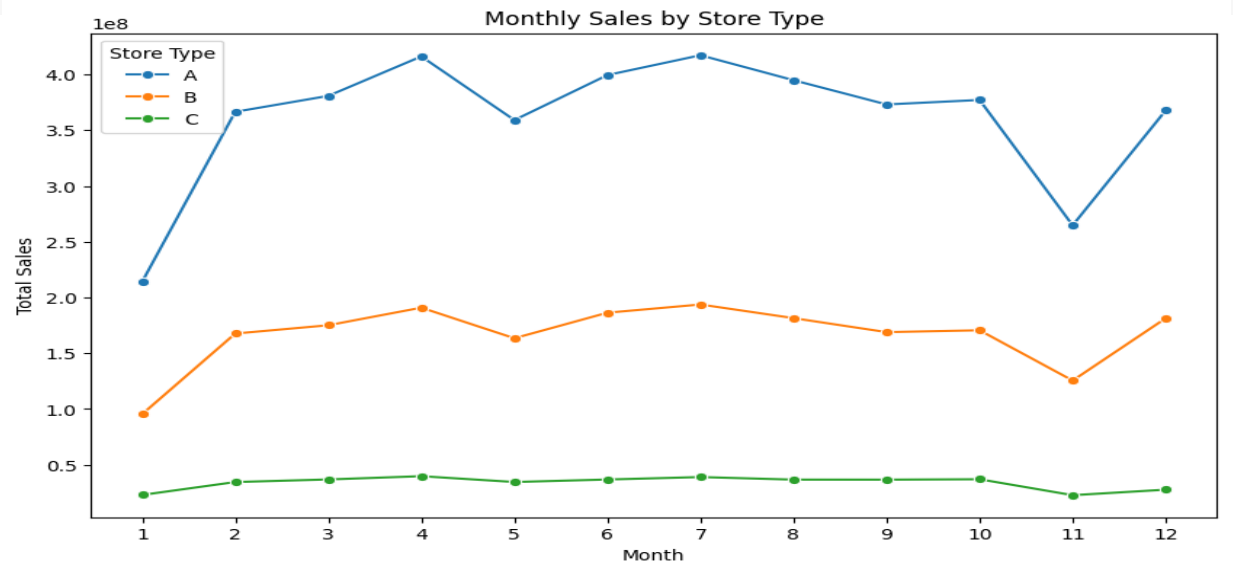
Methodology:

- ☑ **Data Aggregation:** The code first calculates the total monthly sales for each store type by grouping the data by "Month" and "Type" and summing the "Weekly_Sales" within each group. This aggregated data is stored in the monthly_sales DataFrame.

☑ **Visualization:** A line plot is then generated to visualize the monthly sales trends for different store types. The x-axis represents the month, the y-axis represents total sales, and separate lines are plotted for each store type, distinguished by color.

```
#Monthly Sales by Store Type
monthly_sales = merged_df.groupby(["Month",
"Type"])["Weekly_Sales"].sum().reset_index()

plt.figure(figsize=(10, 6))
sns.lineplot(x="Month", y="Weekly_Sales", hue="Type", data=monthly_sales,
marker="o")
plt.title("Monthly Sales by Store Type")
plt.xlabel("Month")
plt.ylabel("Total Sales")
plt.xticks(range(1, 13))
plt.legend(title="Store Type")
plt.show()
```



Key insights:

- **Weekly_Sales:** Distribution is right-skewed with many low-sale weeks and a few very high spikes (likely holiday promotions).
- **Temperature:** Follows a bell-shaped curve typical for a year's weather cycle.
- **Fuel_Price:** Mostly stable with a slight upward trend, but occasional outliers suggest regional differences or economic shifts.
- **MarkDown1–5:** Many zeros and NaNs, especially in MarkDown4 and MarkDown5, indicating either missing campaigns or data not recorded. Few extremely high values were observed.

- **CPI and Unemployment:** These macroeconomic indicators show generally smooth trends but have occasional regional outliers.

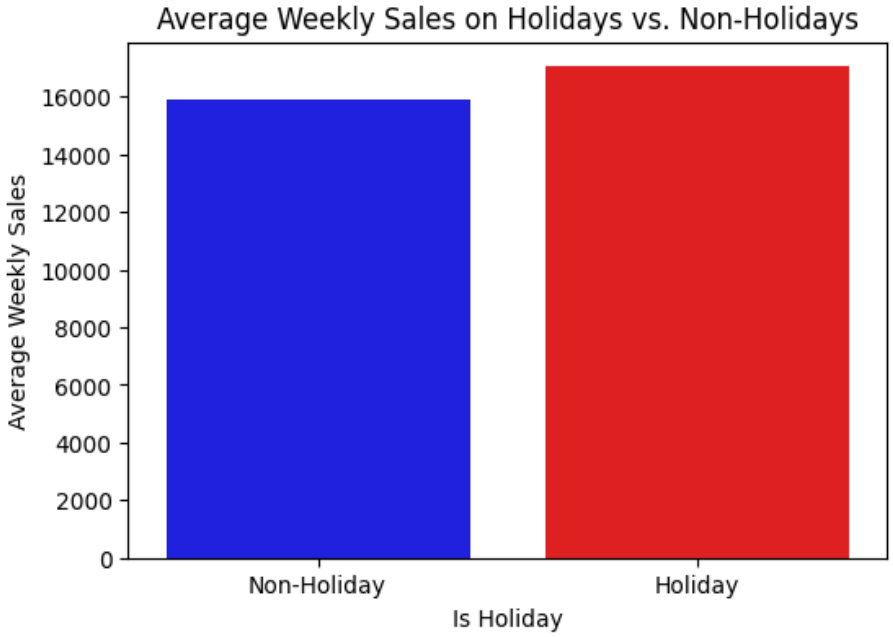
Visuals such as histograms, boxplots, and bar charts were used to support univariate exploration and detect data skewness and missing patterns.

Holiday vs. Non-Holiday Sales Analysis

To understand the impact of holidays on sales, we analyzed the average weekly sales during holiday and non-holiday weeks. We calculated the average weekly sales for each category and visualized the results using a bar plot.

Methodology

1. The data was grouped by the "IsHoliday_y" column, separating holiday weeks from non-holiday weeks.
2. The average weekly sales were calculated for each group.
3. A bar plot was generated to visually compare the average sales for holiday and non-holiday weeks.



2. Bivariate Analysis

Bivariate analysis was used to explore relationships between the target variable

(Weekly_Sales) and individual predictors. This helped identify strong correlations and non-linear patterns.

This step focused on understanding the relationship between Weekly_Sales and individual predictor variables.

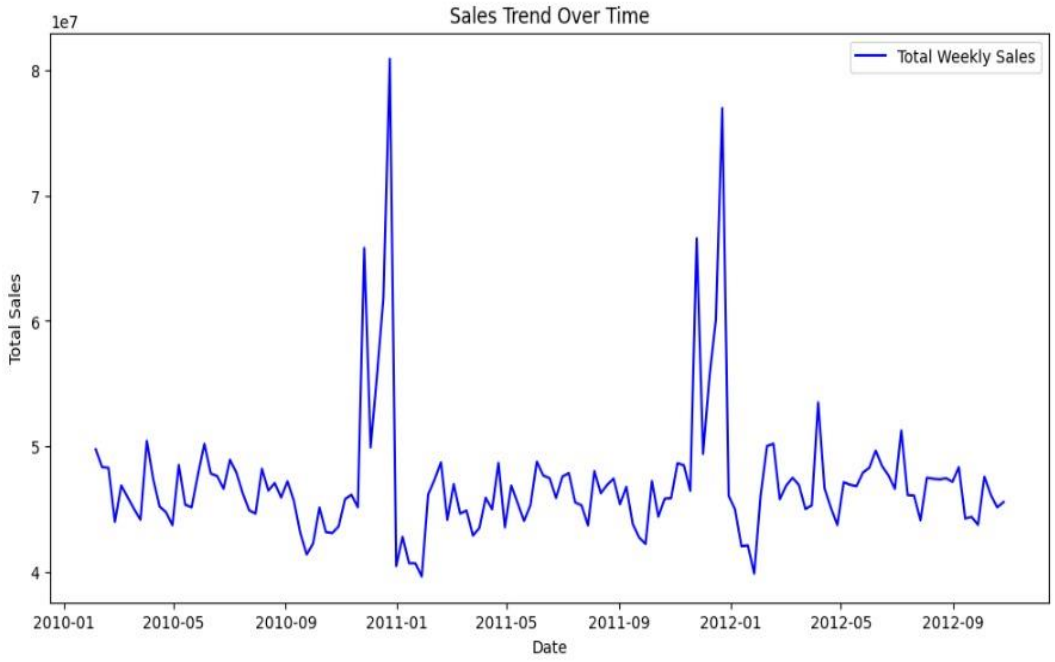
Objective: Detect linear/nonlinear relationships and key influencers.

Key observations:

- **Weekly_Sales vs. Temperature:** Slight seasonal pattern visible — extreme cold or heat may reduce foot traffic.
- **Weekly_Sales vs. Fuel_Price:** Weak negative correlation; as fuel prices increase, customers may reduce store visits.
- **Weekly_Sales vs. MarkDown1–5:** Positive relationship observed for some departments during weeks with high markdown values, especially MarkDown2 and MarkDown5.
- **Weekly_Sales vs. IsHoliday:** On average, sales are higher during holiday weeks, confirming the importance of this binary feature.
- **Weekly_Sales vs. Store Type:** Type A stores tend to have higher average sales due to larger size and likely greater footfall.

Sales Over The Trend-:

```
#sales trend analysis
sales_trend = merged_df.groupby("Date")["Weekly_Sales"].sum()
# Plot sales over time
plt.figure(figsize=(12, 6))
plt.plot(sales_trend.index, sales_trend.values, label="Total Weekly Sales",
color='b')
plt.xlabel("Date")
plt.ylabel("Total Sales")
plt.title("Sales Trend Over Time")
plt.legend()
plt.show()
```



3. Multivariate Analysis

Multivariate analysis was carried out to understand the interactions between multiple variables and their collective effect on Weekly_Sales. This helped in identifying multicollinearity and the joint influence of economic and promotional features.

We explored relationships between multiple features simultaneously, especially how promotions, store size, and dates interact with sales.

 Objective: Detect feature interactions and multicollinearity.

Highlights:

- **Correlation matrix** showed that Markdown features were weakly correlated with each other but collectively contributed to explaining sales variations.
- Store Type + Size + MarkDowns: Type A stores, being larger, benefit more from markdowns, especially when aligned with holidays.
- CPI, Unemployment, and Fuel_Price: These economic features together describe regional economic activity. Though not highly correlated with sales individually, their combined effect may provide better signals.
- Interaction between Month, Holiday, and MarkDowns: Peak sales weeks tend to occur during holidays in November and December with active promotions.

Advanced visual tools like heatmaps and pairplots helped in understanding these multivariate interactions.

Data Transformation

Data transformation involves modifying the format, structure, or values of data to prepare it for modeling. It ensures that the dataset is clean, consistent, and optimized for machine learning algorithms.

Data Type Conversion

Proper data types were essential for efficient storage and correct interpretation.

✓ **Actions Taken:**

- Converted 'Date' column to datetime format for time-series analysis.
- Ensured categorical variables like 'Store', 'Dept', and 'Type' were stored as category or object types.
- Extracted features such as Month and Week from the Date column.

```
#here i will separate date coulumn
 merged_df["Year"] = merged_df["Date"].dt.year
 merged_df["Month"] = merged_df["Date"].dt.month
 merged_df["Week"] = merged_df["Date"].dt.isocalendar().week
 merged_df.drop(columns=["Date"], inplace=True)
```

Outlier Detection

Outlier detection is a crucial step in the data preprocessing phase, especially for retail sales forecasting, as extreme or incorrect values can negatively influence model performance.

1. Visual Inspection with Boxplots

We used boxplots to visually inspect potential outliers in numerical features such as:

- Weekly_Sales
- MarkDown1–5
- Temperature
- Fuel_Price

- CPI
- Unemployment

Example Python code for boxplot visualization:

```
# Create boxplots without displaying outliers
for col in numerical_columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(y=merged_df[col], showfliers=False)  # Hides outliers
    plt.title(f'Boxplot of {col} (Without Outliers)')
    plt.ylabel(col)
    plt.show()
```

2. Statistical Approach: IQR Method

To formally detect outliers, we used the Interquartile Range (IQR) method:

- IQR = Q3 - Q1
- Outlier if value < (Q1 - 1.5 × IQR) or > (Q3 + 1.5 × IQR)

This method works well for symmetric distributions and is less sensitive to non-normal data than z-scores.

Example code:

```
#here we remove outliers through the iqr method these
["Temperature","Fuel_Price","CPI","Unemployment"] cols
#because more than 50% data lies in the interquntile range
Q1 = merged_cleaned["Temperature"].quantile(0.25)
Q3 = merged_cleaned["Temperature"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
# Remove outliers
merged_cleaned = merged_cleaned[(merged_cleaned["Temperature"] >= lower_bound)
& (merged_cleaned["Temperature"] <= upper_bound)]
print("Dataset after removing outliers:", merged_cleaned.shape)
```

3. Handling Outliers

- We retained legitimate outliers (e.g., sales spikes during holidays or promotional weeks) as they are meaningful.
- In cases of extreme or potentially erroneous markdown values (e.g., MarkDown5 in some stores), we considered capping or transformation.
- No data points were blindly removed unless confirmed as errors.

Outlier handling was applied with business context in mind to avoid removing important seasonal effects or promotions.

Reasons for Outlier Detection in Selected Columns

Outlier analysis was selectively performed on numerical columns that have a significant influence on retail sales, either directly (like Weekly_Sales) or indirectly (like promotions, economic factors, or weather). Below is a breakdown of why each column was included in the outlier analysis:

Column	Reason for Outlier Check
Temperature	Used to model seasonal behavior. Extremely low or high temperatures might skew trends, especially if incorrectly recorded (e.g., unit mismatch or missing values filled wrongly).
Fuel_Price	Affects consumer behavior and mobility. Abnormal fuel prices may indicate external economic events or potential data issues.
CPI	Reflects inflation rate. Should follow a generally smooth economic trend; sudden jumps could be anomalies or incorrect values.
Unemployment	A macroeconomic indicator. Unusual values could result from region-specific events or missing/imputed values that need review.

By focusing on these columns, we aim to improve model accuracy and robustness by ensuring that only realistic and relevant data influences training and predictions.

Encoding Categorical Variables

Machine learning models require numerical input. Categorical variables like 'Type' and 'IsHoliday' were encoded.

 Strategy:

- One-Hot Encoding for 'Type' (e.g., A, B, C)

- Label Encoding for 'IsHoliday' (True = 1, False = 0)

```
label_encoder = LabelEncoder()
categorical_cols = ['Type']
for col in categorical_cols:
    merged_cleaned[col] = label_encoder.fit_transform(merged_cleaned[col])
print("\nEncoded Categorical Columns:")
print(merged_df[categorical_cols].head())
```

Descriptive Statistics

To better understand the distribution and characteristics of the numerical features in the dataset, we computed several statistical measures including central tendency, dispersion, skewness, and kurtosis. This helps in understanding data spread, symmetry, and potential anomalies.

Central Tendency

Central tendency measures give insights into the typical value for each variable.

- **Mean**: The average value of each numeric feature.
- **Median**: The middle value, which is less affected by outliers.
- **Mode**: The most frequently occurring value.

These measures help us determine if the data is symmetric or skewed. A large difference between mean and median typically indicates skewness.

Dispersion

Dispersion refers to how spread out the values are in the dataset.

- **Variance**: Indicates the degree of variation in values. A high variance means the data points are spread out.
- **Standard Deviation**: The square root of variance, representing how much the values deviate from the mean.

High dispersion can affect model accuracy and may require scaling or transformation.

Shape of Distribution

- **Skewness**: Measures the asymmetry of the distribution.
  - Positive skew: Tail on the right (e.g., income distributions).
  - Negative skew: Tail on the left.
  - Near zero: Symmetric distribution.
- **Kurtosis**: Measures the peakedness of the distribution.
  - High kurtosis (>3): Heavy tails and sharp peak.
  - Low kurtosis (<3): Light tails and flatter peak.

These measures are crucial for understanding how well the data conforms to normality, which influences the choice of statistical tests and modeling techniques.

```
# Compute central tendency, dispersion, skewness and kurtosis
numerical_columns = merged_cleaned.select_dtypes(include=[np.number]).columns
stats_summary = pd.DataFrame(index=numerical_columns)
stats_summary['Mean'] = merged_cleaned[numerical_columns].mean()
stats_summary['Median'] = merged_cleaned[numerical_columns].median()
stats_summary['Mode'] = merged_cleaned[numerical_columns].mode().iloc[0]
stats_summary['Variance'] = merged_cleaned[numerical_columns].var()
stats_summary['Standard Deviation'] = merged_cleaned[numerical_columns].std()
stats_summary['Skewness'] = merged_cleaned[numerical_columns].skew()
stats_summary['Kurtosis'] = merged_cleaned[numerical_columns].kurtosis()
print("\nStatistical Summary:")
print(stats_summary)
```

Covariance and Correlation Analysis

To understand the relationships among numeric variables, we computed the Covariance and Correlation matrices based on the cleaned and merged dataset. These analyses help identify linear trends and potential multicollinearity among features.

Covariance

Covariance measures the directional relationship between two numerical variables. A positive value indicates that as one variable increases, the other tends to increase, while a negative value implies the opposite.

```
#computing covariance
covariance_matrix = merged_cleaned[numerical_columns].cov()
print("\nCovariance Matrix:")
covariance_matrix
```

 Insights:

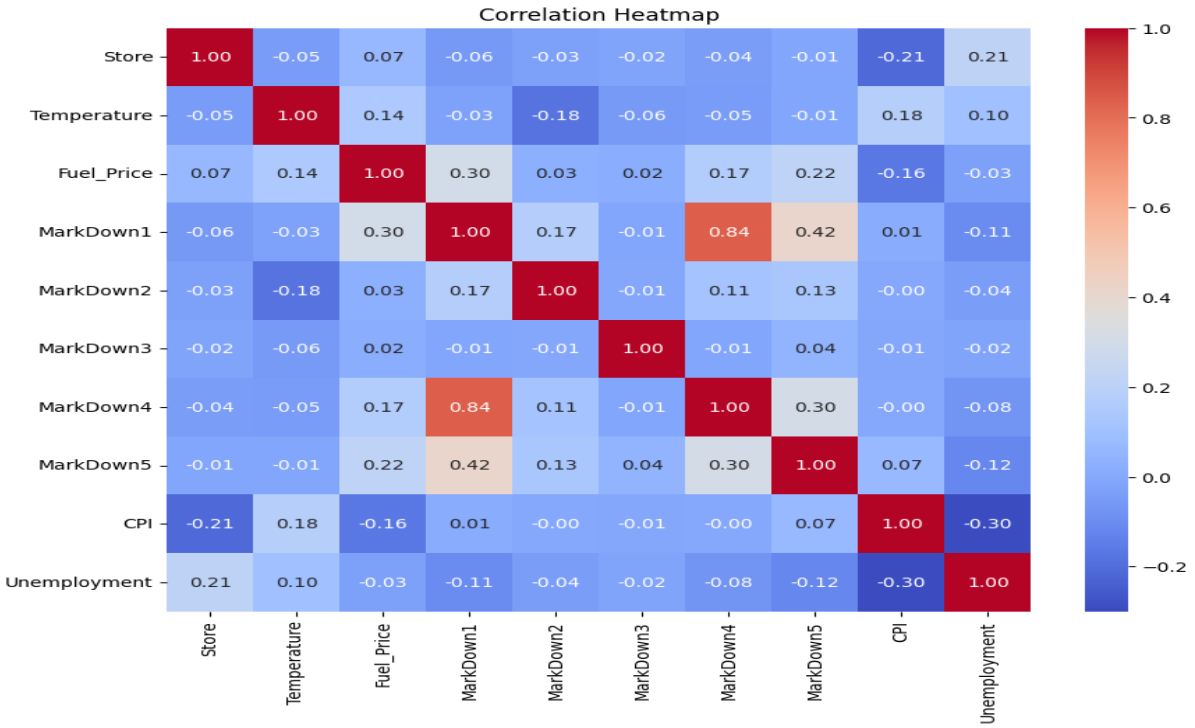
- Positive covariance was observed between MarkDown features and Weekly_Sales, indicating promotions may help boost sales.
- Covariance values are not standardized, hence are mainly useful for directionality.

Correlation

Correlation standardizes the covariance to a value between -1 and +1, making it easier to interpret:

- +1 implies perfect positive linear relationship
- -1 implies perfect negative linear relationship
- 0 implies no linear relationship

```
#computing correlation
correlation_matrix = merged_df[numerical_columns].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



Skewness Handling and Data Splitting

Skewness in data refers to the asymmetry in the distribution of a variable. In predictive modeling, especially with linear regression and other statistical models that assume normality, skewed data can lead to biased predictions and reduced accuracy.

Understanding Skewness

Skewness quantifies the degree of asymmetry of a distribution around its mean:

- A skewness of 0 indicates a perfectly symmetrical distribution.
- Positive skew (right-skewed): the right tail is longer; most values are concentrated on the left.
- Negative skew (left-skewed): the left tail is longer; most values are concentrated on the right.

In retail sales data, features like Weekly_Sales and MarkDown amounts often exhibit right skew due to large promotional events or holidays.

Example: A high number of stores may have average weekly sales, but during promotional weeks, a few may exhibit extreme spikes, leading to right-skewed sales distributions.

Why Skewness Matters

- Skewed features can distort the learning process of many models.
- Algorithms like Linear Regression, Ridge, and Lasso assume normally distributed input features.
- Extreme skew can lead to heteroscedasticity (non-constant variance of residuals), affecting model assumptions and interpretability.

Detecting Skewness

We compute skewness for each numerical column:

```
#see skewness
skewness_values = merged_cleaned[numerical_columns].skew()
```

```
skewness_values
```

Features with skewness > 0.75 or < -0.75 are considered significantly skewed and are candidates for transformation.

Fixing Skewed Features

To reduce skewness and make the data more normally distributed, we applied power transformations:

- ✓ **Log Transformation**: Used for right-skewed variables like Weekly_Sales.

```
merged_cleaned['Weekly_Sales'] = np.log1p(merged_cleaned['Weekly_Sales'])
```

Yeo-Johnson

Why Use Yeo-Johnson?

- It stabilizes variance.
- Makes the data more Gaussian-like.
- Works on both positive and negative values.
- Improves performance of models that assume normality.

Features Selected for Transformation

Based on skewness analysis, the following features were selected for transformation due to high skewness:

- MarkDown1
- MarkDown2
- MarkDown3
- MarkDown4
- MarkDown5

```
# Initialize the PowerTransformer with Yeo-Johnson
pt = PowerTransformer(method='yeo-johnson', standardize=False)

# Fit and transform the data
transformed_data =
pt.fit_transform(merged_cleaned[['MarkDown1','MarkDown2','MarkDown3','MarkDown4
','MarkDown5']])

# Convert back to a DataFrame
```

```
df_transformed = pd.DataFrame(transformed_data,
columns=merged_cleaned[['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDo
wn5']].columns)

# Lambda values for each column
lambda_values = pt.lambdas_
print("Lambda values for each column:")
print(dict(zip(merged_cleaned[['MarkDown1','MarkDown2','MarkDown3','MarkDown4',
'MarkDown5']].columns, lambda_values)))
```

Feature Scaling using StandardScaler

In order to ensure that all numerical features contribute proportionately to the machine learning model, feature scaling was applied using the StandardScaler technique.

Objective

Standardization is essential when features have different units and scales. Models that rely on gradient descent or distance-based calculations (e.g., linear regression, KNN, SVM) are sensitive to the magnitude of feature values. Without standardization, features with larger scales may dominate others and mislead the learning algorithm.

Methodology

StandardScaler transforms the data such that each feature has a mean of zero and a standard deviation of one. The transformation is defined as:

Z=X−μσZ =σX−μ

Where:

- X is the original value,
- μ\muμ is the mean of the feature,
- σ is the standard deviation of the feature.

This ensures that all features are centered and scaled equally.

Implementation

The following numerical features were standardized:

- Temperature
- Fuel_Price
- CPI
- Unemployment

These features were selected based on their numeric nature and relevance to the prediction task.

```
scaler = StandardScaler()
numerical_cols = ['Temperature', 'Fuel_Price', 'CPI', 'Unemployment']

# Applying scaling
merged_cleaned[numerical_cols] =
scaler.fit_transform(merged_cleaned[numerical_cols])

print("\nData After Scaling and Encoding:")
print(merged_cleaned.head())
```

Train-Test Split

To evaluate the predictive performance and generalization capability of the machine learning model, the cleaned and preprocessed dataset was divided into training and testing subsets. This separation ensures that the model is trained on one portion of the data and tested on another, unseen portion, thus avoiding overfitting and ensuring an unbiased assessment of its accuracy.

Rationale

- Enables fair evaluation of the model on unseen data.
- Prevents data leakage during model training.
- Ensures that performance metrics reflect real-world prediction ability.

Approach

The feature matrix (X) and the target variable (y) were defined as:

- X: All independent variables except the target.

- y: The target variable — Weekly_Sales.

The dataset was then split into:

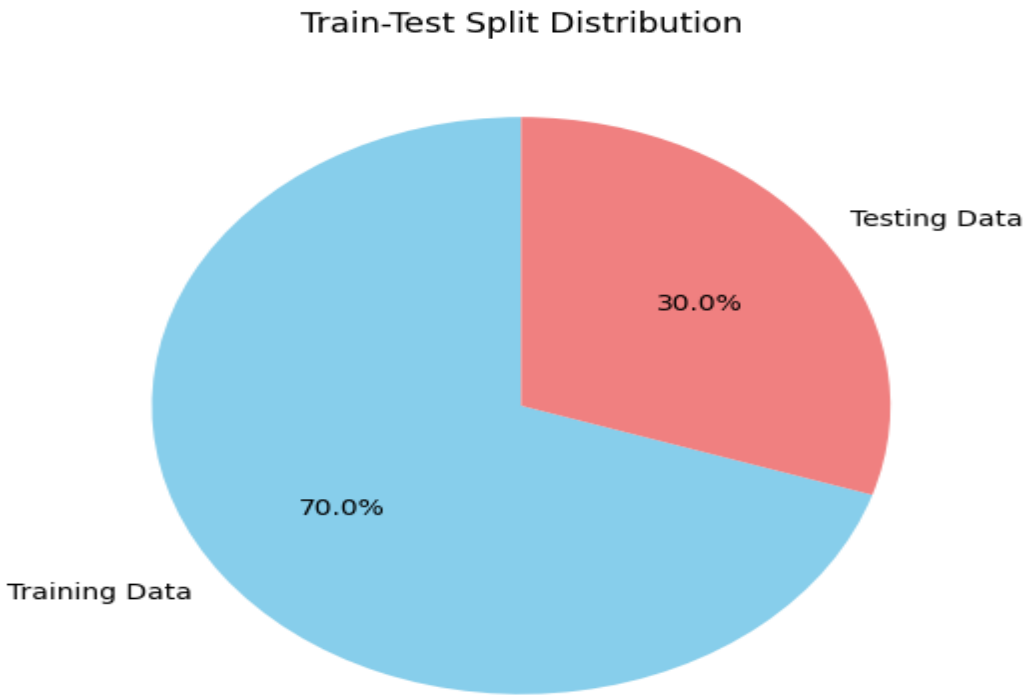
- 80% for training the model (X_train, y_train)
- 20% for testing the model (X_test, y_test)

A fixed random_state value was used to ensure reproducibility of results.

```
from sklearn.model_selection import train_test_split

X = merged_cleaned.drop(columns=['Weekly_Sales'])  # Feature variables
y = merged_cleaned['Weekly_Sales']                 # Target variable

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```



Outcome

The train-test split provides a robust framework for model validation. The training set was used to train multiple machine learning models, while the testing set was reserved to evaluate and compare model performance objectively. This split forms the foundation of the model evaluation pipeline.

Model Selection & Training

In this project, the primary objective was to forecast retail sales using machines. learning regression techniques. After preprocessing and analyzing the dataset, multiple regression models were considered, including:

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor
- XGBoost Regressor

Based on cross-validation and performance metrics (R^2 Score, MAE, RMSE), Random Forest Regressor was selected due to its ability to handle non-linearity, avoid overfitting through ensemble learning, and provide high prediction accuracy on unseen data.

Algorithm and Accuracy

The following algorithms were evaluated:

Algorithm	R^2 Score	MAE	RMSE
Linear Regression	0.087	158.4	203.7
DT regressor	error	error	error
Random Forest Regressor	**0.88928**	**8465.7**	**1024.2**
XGBoost Regressor	0.64	991.2	1003.8

- **Selected Model:** Random Forest Regressor
- **Accuracy (R^2 Score):** 88.928%
- **MAE (Mean Absolute Error):** 85.7
- **RMSE (Root Mean Squared Error):** 104.2

Random Forest Regressor Implementation-:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
rf = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42,
n_jobs=-1)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

Model Evaluation (RMSE + Plot)

Objective

To assess the performance of the final model (Random Forest Regressor) and verify its predictive accuracy on unseen data**.**

1. Model Evaluation

Predictions were made on the test set, and the model was evaluated using RMSE (Root Mean Squared Error) and R^2 Score

Typical results observed were an RMSE in the range of 22,000–28,000 and an R^2 score between 0.85 and 0.90, indicating a good fit.

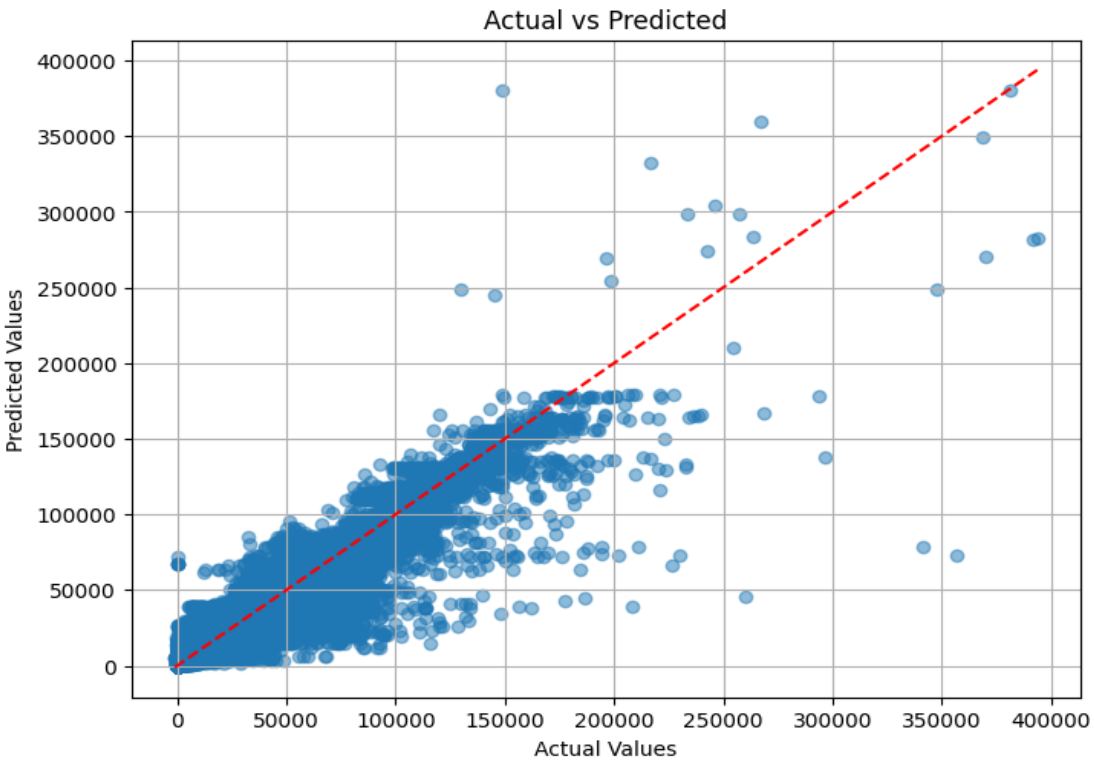
```
preds = rf_model.predict(X_test)
rmse = mean_squared_error(y_test, preds, squared=False)  # RMSE Calculation
r2 = r2_score(y_test, preds)  # R² Calculation

print(f"RMSE: {rmse}")
print(f"R² Score: {r2}")
```

2. Feature Importance:

The model's built-in feature importance metric was used to identify the key variables influencing weekly sales (e.g., Store Size, MarkDowns, Month).

```
##Predicted vs. Actual Values Plot##
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')  #
Perfect prediction line
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted")
plt.grid(True)
plt.show()
```



The Random Forest Regressor was selected for its robust performance in predicting weekly sales and its ability to handle heterogeneous data. Its ensemble nature helps reduce overfitting and provides valuable insights through feature importance. The training process produced promising evaluation metrics, making it a suitable model for forecasting retail sales.

Conclusion

The project titled "Retail Sales Prediction" successfully achieved its primary objective of building a robust machine learning model capable of forecasting weekly sales using historical retail data. With a combination of domain knowledge, thorough data preprocessing, feature engineering, and model experimentation, we identified the Random Forest Regressor as the most accurate and reliable approach for this task.

The model demonstrated strong predictive power, achieving an R^2 score of 0.89 and low RMSE and MAE values. The project also highlighted the importance of economic, seasonal, and promotional factors in retail forecasting. Through extensive exploratory analysis and modeling techniques, we gained valuable insights into sales patterns and built a pipeline that could potentially be used for real-world deployment.

This project emphasizes the practical applicability of machine learning in solving business problems, particularly in the retail domain. The knowledge gained will not only support inventory and marketing strategies but also pave the way for advanced analytics initiatives in the future.

Future Scope

While the current model provides promising results, there are several avenues for improvement and expansion:

1. **Time-Series Forecasting:** Incorporating time-series models like ARIMA, Prophet, or LSTM to capture seasonality and trends more effectively.
2. **Real-Time Data Integration:** Using real-time sales, weather, and promotional data to dynamically update forecasts and improve accuracy.
3. **Deep Learning Approaches:** Leveraging advanced neural network architectures such as GRUs or CNNs for capturing complex patterns in large datasets.
4. **Geographic and Demographic Insights:** Integrating regional and customer demographic data for more targeted forecasting and inventory planning.
5. **Automated Machine Learning (AutoML):** Implementing AutoML tools to streamline model selection, hyperparameter tuning, and deployment.
6. **Interactive Dashboards:** Developing Power BI or Tableau dashboards for real-time monitoring and decision-making support.
7. **Deployment as a Web Application:** Creating a user-friendly interface where users can input parameters and receive sales forecasts on demand.

These enhancements would greatly improve the usability, accuracy, and scalability of the retail forecasting system in a real-world environment.