RETAIL SALES FORECAST

A MINI PROJECT REPORT

18AIE423T- BUSINESS INTELLIGENCE AND ANALYTICS

Submitted by

SAI ANUGNYA [RA2111047010028]
SIDDHARTH PRATYUSH [RA2111047010040]
AMIR MUSTAQUE [RA2111047010054]

Under the guidance of
DR. NAVEEN. P

Assistant Professor, Department of Computational Intelligence

In partial satisfaction for the award of the degree

of

BACHELORS OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M Nagar, Kattankulathur, Chengalpattu District

JUL 2024

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that Mini project report for the course report titled **"RETAIL SALES FORECAST"** is the bonafide work done by **SAI ANUGNYA [RA2111047010028], SIDDHARTH PRATYUSH [RA2111047010040] and AMIR MUSTAQUE [RA2111047010054]** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Faculty In-Charge
Dr.Naveen.P
Assistant Professor,
Department of Computational Intelligence,
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

HEAD OF THE DEPARTMENT
Dr. R Annie Uthra
Professor and Head,
Department of Computational Intelligence,
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

ABSTRACT

Retail sales forecasting is a critical practice that informs business strategy, inventory management, marketing efforts, and financial planning. This abstract provides a concise overview of the essential elements involved in forecasting retail sales, highlighting methodologies, data sources, and key benefits.

At its core, retail sales forecasting involves predicting future sales based on historical data, market trends, consumer behavior, and external factors such as economic conditions and seasonality. Accurate forecasting enables retailers to optimize stock levels, reduce overstocking and stockouts, plan promotions effectively, and improve overall customer satisfaction.

Forecasting techniques vary, ranging from traditional statistical methods to advanced machine learning algorithms. Classical approaches like moving averages, exponential smoothing, and regression analysis are commonly used, providing straightforward and interpretable models. However, these methods may not fully capture complex relationships in modern retail environments.

Machine learning and deep learning approaches, such as neural networks and ensemble models, are gaining popularity due to their capacity to analyze large datasets, identify patterns, and make highly accurate predictions. These models can incorporate diverse data sources, including point-of-sale data, web analytics, social media sentiment, and weather patterns, to produce more nuanced forecasts.

Retail sales forecasting must also account for seasonality and promotional events, which significantly impact consumer purchasing behavior. Techniques such as time series decomposition and seasonal adjustment help to isolate and predict these effects. Additionally, advanced analytics can model the impact of promotions, holidays, and other events on sales trends.

Ultimately, accurate retail sales forecasting enables businesses to make data-driven decisions that enhance operational efficiency, reduce costs, and drive revenue growth. By selecting the appropriate forecasting methodologies and incorporating relevant data, retailers can stay ahead of market trends, meet customer demand, and maintain a competitive edge in a rapidly evolving industry.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Retail sales forecasting is a critical practice for businesses seeking to anticipate consumer demand, optimize inventory, staffing plan, and improve financial planning. At its core, forecasting predictive analytics employs statistical and machine learning techniques to predict future sales based on historical data and related factors. By leveraging advanced analytics, retail organizations can make data-driven decisions that drive profitability and efficiency.

Accurate sales forecasts are essential for maintaining a competitive edge in the retail industry. They allow businesses to manage supply chains effectively, reduce overstock or stockouts, and respond swiftly to market trends. Sales forecasting can also inform marketing campaigns, promotional strategies, and resource allocation.

Forecasting predictive analytics involves a mix of statistical methods, machine learning models, and time series analysis. It starts with gathering relevant data such as historical sales figures, promotional activities, holiday schedules, economic indicators, and other contextual factors. Once the data is processed and cleaned, predictive models are built to identify patterns, trends, and seasonality.

Machine learning algorithms, including linear regression, decision trees, and ensemble methods like random forest or gradient boosting, are commonly used. These models can handle complex relationships between variables and provide robust predictions. Time series analysis is also crucial, allowing businesses to capture seasonal variations, trends, and cyclical patterns in sales data.

Retail sales forecasting enables businesses to optimize inventory management, reducing excess stock while ensuring product availability. It supports workforce planning by predicting peak sales periods, enabling retailers to align staffing with demand. Additionally, forecasting helps set realistic sales targets and budgetary goals.

With the right predictive analytics approach, retailers can gain deeper insights into customer behavior, leading to more effective marketing strategies and personalized customer experiences. Ultimately, retail sales forecasting empowers businesses to make informed decisions, adapt to changing market conditions, and drive sustainable growth.

CHAPTER 2

LITERATURE SURVEY

Retail sales forecasting is a pivotal aspect of retail business strategy, enabling companies to predict future sales trends and plan accordingly. A literature review on retail sales forecasting focuses on key methods, challenges, and advancements in the field. This review summarizes recent research and insights, highlighting major themes and trends.

A central theme in retail sales forecasting is the use of historical sales data to predict future trends. Traditional statistical techniques, such as moving averages and regression analysis, have been widely used for this purpose. However, these methods often fall short in complex retail environments where external factors play a significant role.

Recent advancements in machine learning and artificial intelligence (AI) have transformed the landscape of retail sales forecasting. Techniques such as XGBoost, neural networks, and ensemble methods provide improved accuracy by capturing non-linear relationships and integrating a broader range of data sources. These models can process large datasets, including customer behavior, marketing campaigns, and seasonality patterns, leading to more reliable forecasts.

Another trend in the literature is the exploration of external factors influencing retail sales. Studies emphasize the impact of economic conditions, weather patterns, and societal trends on consumer behavior. Incorporating these factors into forecasting models has led to a more comprehensive understanding of retail dynamics.

Challenges in retail sales forecasting include data quality and model robustness. Data cleaning and preprocessing are critical to ensure accurate forecasts. Additionally, model validation and testing play a significant role in maintaining reliability over time.

The literature also explores the business benefits of accurate forecasting. It supports inventory management, reduces costs, and enhances customer satisfaction. By adopting advanced forecasting techniques and integrating relevant data, retailers can improve operational efficiency and maintain a competitive edge.

In summary, recent literature on retail sales forecasting underscores the importance of advanced modeling techniques, the integration of diverse data sources, and the need for robust validation. These factors contribute to more accurate and actionable retail sales forecasts, supporting informed business decisions.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Architecture diagram of proposed Retail Sales Forecasting using XGBoost :

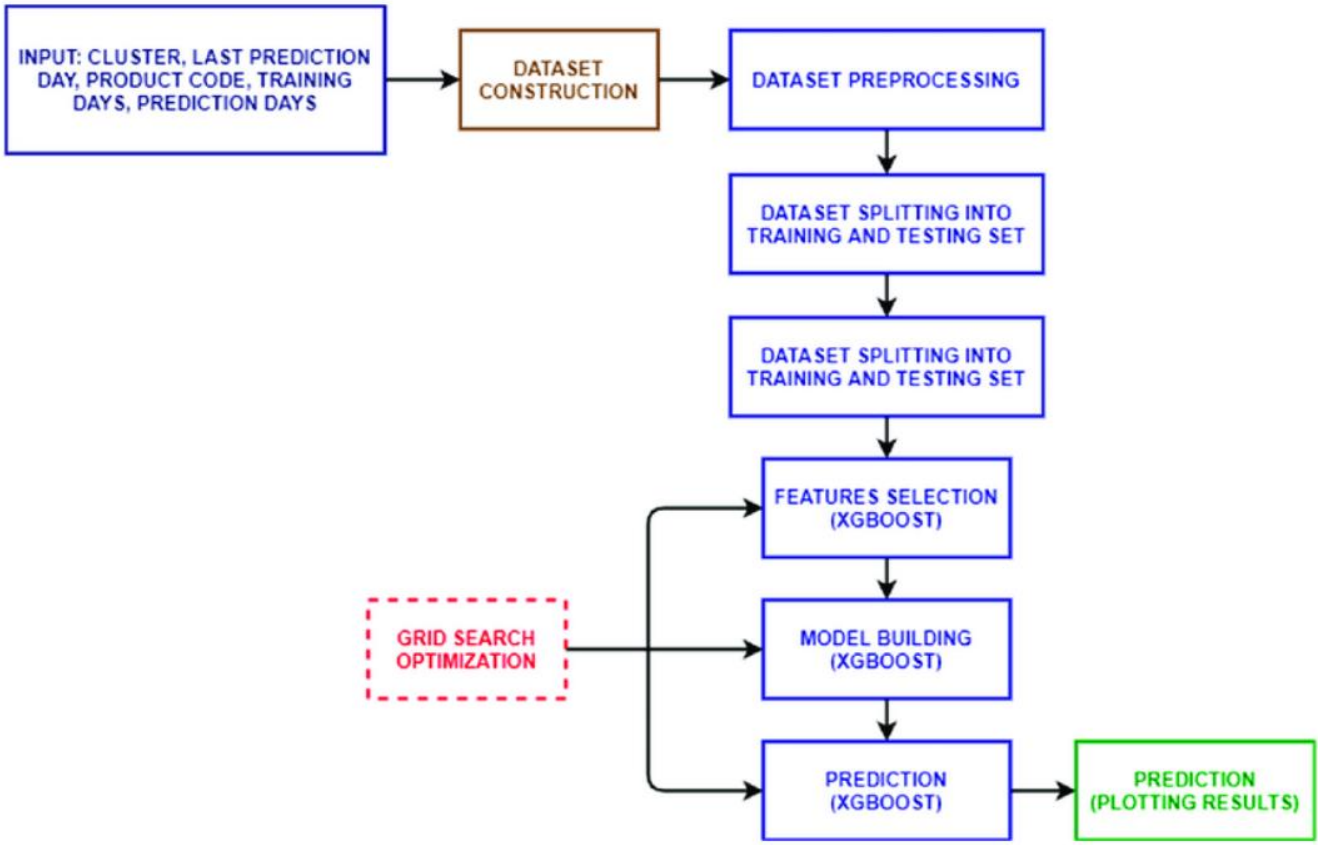

Fig 3.1.1 Architecture Diagram

3.2 Description of all models used:

The following models were used to predict data while forecasting retail sales :

- **Linear Regression:**

- This model assumes a linear relationship between the features and the target variable. It finds the best-fitting line through the data points, minimizing the difference between predicted and actual values.

- **RidgeCV:**

- A variant of linear regression that uses L2 regularization to prevent overfitting. It penalizes large coefficients, helping to maintain model stability across varying data.

- **Lasso:**

- This linear model employs L1 regularization, which promotes sparsity in the coefficients. It is useful for feature selection as it can shrink less important coefficients to zero.

- **ElasticNet:**

- Combines the regularization techniques of Ridge (L2) and Lasso (L1), balancing between them. It is ideal when the dataset has many features and you want to prevent overfitting while allowing some flexibility in feature selection.

- **DecisionTreeRegressor:**

- A tree-based model that creates a tree structure to split the data into subsets based on the feature that provides the most information gain. It's straightforward but prone to overfitting without proper tuning.

- **RandomForestRegressor:**

- An ensemble of multiple decision trees that reduces overfitting by averaging the predictions from each tree. It provides robust results due to the combined decisions from various trees.

- **GradientBoostingRegressor:**

- This model builds trees sequentially, where each new tree focuses on correcting errors made by the previous ones. It's a powerful technique that can handle complex patterns in the data.

- **XGBoost:**

- An advanced boosting algorithm that employs a regularization term to prevent overfitting. It offers high performance and efficiency, with features like parallel processing, tree pruning, and built-in cross-validation.

- **LightGBM:**

- A boosting model similar to XGBoost but optimized for large datasets. It uses a different tree-building strategy that makes it faster and more memory-efficient.

- **KernelRidge:**

- A model that applies the kernel trick to linear regression, allowing it to model non-linear relationships. It uses L2 regularization like Ridge but can handle more complex data structures.

- **SVR (Support Vector Regression):**

- A variant of Support Vector Machines designed for regression tasks. It uses kernels to create complex decision boundaries, making it useful for non-linear data patterns.

The **XGBoost model** was chosen for its robust performance and flexibility in handling complex data patterns. It balances accuracy and efficiency with built-in regularization to reduce overfitting, making it well-suited for retail sales forecasting. XGBoost's parallel processing capabilities and efficient resource usage also contribute to its popularity in large-scale data tasks like this project.

CHAPTER 4

METHODOLOGY

Designing a robust methodology for retail sales forecasting involves several key steps. Here's a proposed methodology:

1. Data Collection and Extraction:

- The raw data is extracted from a compressed file (ZIP), containing three main datasets: features, sales, and stores. The datasets are read into separate pandas dataframes.

2. Data Preprocessing:

- Data is cleaned and imputation is used to fill missing values for Markdown data using 'IterativeImputer', and missing values for other features like 'CPI' and 'Unemployment' are replaced with the mean. The 'IsHoliday' column is transformed to binary values (1 for False, 0 for True).

3. Feature Engineering:

- Additional features are created based on the Date column, including Month, Year, and WeekofYear. Categorical variables like Type, Month, Year, and WeekofYear are converted into dummy variables using one-hot encoding.

4. Data Aggregation and Merging:

- The feature data, sales data, and store data are merged on relevant keys (like Store), and grouped by various attributes to derive meaningful insights, such as monthly and yearly sales, store-specific data, and departmental sales.

5. Data Exploration and Visualization:

- Various exploratory data analysis (EDA) techniques are employed to understand the relationships between different variables and identify trends.
- Techniques used include:
    a. Visualization of Sales Trends: Plotting weekly sales, temperature, fuel price, and other features over time to identify patterns.
    b. Correlation Analysis: Heatmaps and scatter plots to examine correlations among variables.
    c. Statistical Analysis: Analysing skewness and distributions to understand data characteristics.

6. Model Development and Training:

- The cleaned and processed data is split into training and testing sets. Several regression models are implemented and trained on the training set, with the testing set used to evaluate their performance.
- The following models are considered:
    a. Linear Models: Linear Regression, Lasso, ElasticNet.
    b. Decision Trees and Ensembles: DecisionTreeRegressor, RandomForestRegressor, GradientBoostingRegressor.
    c. Support Vector Machines (SVMs): SVR, LinearSVR, NuSVR.
    d. Advanced Models: XGBoost, LightGBM.
- Each model is trained on the training data, and their performance is evaluated based on the R-squared score and Root Mean Square Error (RMSE) on the testing set.

7. Model Evaluation and Selection:

- A DataFrame (df_score) is created to compare the performance of the different models. Models are ranked based on their R-squared score and RMSE.
- The best-performing model (in this case, XGBRegressor) is selected for predicting retail sales.

8. Prediction:

- The selected model (XGBRegressor) is used to predict weekly sales for the test data. The predicted results can be used for further analysis, forecasting, and business decision-making.

CHAPTER 5

INFERENCE

- In this project, the dataset “retail-data-analytics” has been used.
- The dataset has been uploaded to Git Hub along with the following project report.

1. Import necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

5.1 Import libraries

Observation:

This imports the following libraries needed.

2. Load the dataset

```
import zipfile
zip_file = zipfile.ZipFile('retail-data-analytics.zip', 'r')
zip_file.namelist()
```

```
['Features data set.csv', 'sales data-set.csv', 'stores data-set.csv']
```

5.2 Load the dataset

Observation:

This imports the zip file of the dataset. It also prints the names of the csv files present in the zip file.

3. Import zip file, check for the files and read the files

```
import zipfile
import pandas as pd

# Specify the path to the zip file
zip_file_path = 'retail-data-analytics.zip'

# Open the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    # List the files in the zip file
    file_list = zip_ref.namelist()

    # Check if each required file exists in the zip file
    if 'Features data set.csv' in file_list:
        # Read Features data set.csv
        features = pd.read_csv(zip_ref.open('Features data set.csv'))

    if 'sales data-set.csv' in file_list:
        # Read sales data-set.csv
        sales = pd.read_csv(zip_ref.open('sales data-set.csv'))

    if 'stores data-set.csv' in file_list:
        # Read stores data-set.csv
        stores = pd.read_csv(zip_ref.open('stores data-set.csv'))
```

Fig 5.3 Import zip file and read files

Observation:

- This step imports the zip file, opens it and checks for the required files in it
- On finding the files, it reads them.
- The files in this zip file are features data-set.csv, sales data-set.csv and stores data-set.csv

4. Print the first few rows of each individual csv file in the dataset.

```
# Display the first few rows of each DataFrame (optional)
print("Features data:")
print(features.head())

print("\nSales data:")
print(sales.head())

print("\nStores data:")
print(stores.head())
```

```
Features data:
   Store        Date  Temperature  Fuel_Price  MarkDown1  MarkDown2  \
0      1  05/02/2010        42.31       2.572        NaN        NaN
1      1  12/02/2010        38.51       2.548        NaN        NaN
2      1  19/02/2010        39.93       2.514        NaN        NaN
3      1  26/02/2010        46.63       2.561        NaN        NaN
4      1  05/03/2010        46.50       2.625        NaN        NaN

   MarkDown3  MarkDown4  MarkDown5         CPI  Unemployment  IsHoliday
0        NaN        NaN        NaN  211.096358         8.106      False
1        NaN        NaN        NaN  211.242170         8.106       True
2        NaN        NaN        NaN  211.289143         8.106      False
3        NaN        NaN        NaN  211.319643         8.106      False
4        NaN        NaN        NaN  211.350143         8.106      False
```

Fig 5.4 Print the first few rows of csv files in the dataset

```
Sales data:
   Store  Dept        Date  Weekly_Sales  IsHoliday
0      1     1  05/02/2010      24924.50      False
1      1     1  12/02/2010      46039.49       True
2      1     1  19/02/2010      41595.55      False
3      1     1  26/02/2010      19403.54      False
4      1     1  05/03/2010      21827.90      False

Stores data:
   Store Type    Size
0      1    A  151315
1      2    A  202307
2      3    B   37392
3      4    A  205863
4      5    B   34875
```

Fig 5.5 Print the first few lines of sales data.csv and stores data.csv

Observation:

This code prints the first five lines of each of the csv files present in the dataset

5. Print basic information about the store data-set.csv file

```
print(df_store.info())
print(df_store.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Store   45 non-null     int64
 1   Type    45 non-null     object
 2   Size    45 non-null     int64
dtypes: int64(2), object(1)
memory usage: 1.2+ KB
None
   Store Type    Size
0      1    A  151315
1      2    A  202307
2      3    B   37392
3      4    A  205863
4      5    B   34875
```

Fig 5.6 Print basic information about stores data-set.csv file

Observation:

- This code prints basic information about the stores dataset file such as data types, no. of rows and columns, null data, and a few rows to understand the data.

6. Print basic information about the features data-set.csv file:

```
print(df_feature.info())
print(df_feature.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         8190 non-null   int64
 1   Date          8190 non-null   datetime64[ns]
 2   Temperature   8190 non-null   float64
 3   Fuel_Price    8190 non-null   float64
 4   MarkDown1     4032 non-null   float64
 5   MarkDown2     2921 non-null   float64
 6   MarkDown3     3613 non-null   float64
 7   MarkDown4     3464 non-null   float64
 8   MarkDown5     4050 non-null   float64
 9   CPI           7605 non-null   float64
 10  Unemployment  7605 non-null   float64
 11  IsHoliday     8190 non-null   bool
dtypes: bool(1), datetime64[ns](1), float64(9), int64(1)
memory usage: 712.0 KB
None
   Store       Date  Temperature  Fuel_Price  MarkDown1  MarkDown2  MarkDown3  \
0      1 2010-05-02        42.31       2.572        NaN        NaN        NaN
1      1 2010-12-02        38.51       2.548        NaN        NaN        NaN
2      1 2010-02-19        39.93       2.514        NaN        NaN        NaN
3      1 2010-02-26        46.63       2.561        NaN        NaN        NaN
4      1 2010-05-03        46.50       2.625        NaN        NaN        NaN

   MarkDown4  MarkDown5         CPI  Unemployment  IsHoliday
0        NaN        NaN  211.096358         8.106      False
1        NaN        NaN  211.242170         8.106       True
2        NaN        NaN  211.289143         8.106      False
3        NaN        NaN  211.319643         8.106      False
4        NaN        NaN  211.350143         8.106      False
```

Fig 5.7 Print basic information about the features data-set.csv file

Observation:

- This code prints basic information about the features dataset file such as data types, no. of rows and columns, null data, and a few rows to understand the data.

7. Print basic information about the sales data-set.csv file:

```
print(df_sales.info())
print(df_sales.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Store         421570 non-null  int64
 1   Dept          421570 non-null  int64
 2   Date          421570 non-null  datetime64[ns]
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     421570 non-null  bool
dtypes: bool(1), datetime64[ns](1), float64(1), int64(2)
memory usage: 13.3 MB
None
   Store  Dept       Date  Weekly_Sales  IsHoliday
0      1     1 2010-05-02      24924.50      False
1      1     1 2010-12-02      46039.49       True
2      1     1 2010-02-19      41595.55      False
3      1     1 2010-02-26      19403.54      False
4      1     1 2010-05-03      21827.90      False
```

Fig 5.8 Print basic information about the sales data-set.csv file

Observation:

- This code prints basic information about the sales dataset file such as data types, no. of rows and columns, null data, and a few rows to understand the data.

8. Data aggregation and merging:

```
data_date = df_feature.groupby("Date").agg({"Temperature":"mean"
                                            ,"Fuel_Price":"mean"
                                            ,"IsHoliday":"sum"
                                            ,"CPI":"mean"
                                           ,"Unemployment":"mean"})
data_date = data_date.sort_index()
temp_date_data = data_date[:'2012-12-10']

data_sales_date = df_sales.groupby("Date").agg({"Weekly_Sales":"sum"})
data_sales_date.sort_index(inplace=True)
data_sales_date.Weekly_Sales = data_sales_date.Weekly_Sales/1000000
data_sales_date.Weekly_Sales = data_sales_date.Weekly_Sales.apply(int)
data = pd.merge(data_sales_date, temp_date_data, left_index=True,right_index=True, how='left')
data["IsHoliday"] = data["IsHoliday"].apply(lambda x: True if x == 45.0 else False )
```

Fig 5.9 Data aggregation and merging

Observation:

The code aggregates various retail data by date, merges sales data with other features, and creates and flag to indicate holidays, preparing a comprehensive dataset for further analysis or modelling

9. Summary statistics for data:

```
print(data.describe())
```

```
       Weekly_Sales  Temperature  Fuel_Price         CPI  Unemployment
count    143.000000   143.000000  143.000000  143.000000    143.000000
mean      46.601399    60.663782    3.358607  171.578394      7.999151
std        5.399713    15.172792    0.429401    3.100148      0.483853
min       39.000000    30.480889    2.672067  167.546485      6.953711
25%       44.000000    47.720222    2.885367  168.408144      7.508333
50%       46.000000    61.051111    3.488644  171.386189      8.150133
75%       47.000000    74.697333    3.726133  174.724576      8.428578
max       80.000000    82.176444    3.997778  176.652613      8.619311
```

Fig 5.10 Summary statistics for data.

Observation:

- This method generates summary statistics for a data frame. This method provides insight into the distribution and characteristics of the data. It finds the following 8 statistics from the data:
  1) Count: Number of non-null (non-missing) values in each column.
  2) Mean: Average value for each column.
  3) Std: Standard deviation, representing data dispersion or spread.
  4) Min: Minimum value in each column.
  5) 25% (1st Quartile): Value below which 25% of the data falls.
  6) 50% (Median): Middle value where 50% of the data is above and 50% below.
  7) 75% (3rd Quartile): Value below which 75% of the data falls.
  8) Max: Maximum value in each column.
- Here the method “data. describe ()” finds the statistics for the columns weekly_sales, temperature, fuel_price, cpi and unemployment

10. Size of dataset:

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 143 entries, 2010-01-10 to 2012-12-10
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Weekly_Sales  143 non-null    int64  
 1   Temperature   143 non-null    float64
 2   Fuel_Price    143 non-null    float64
 3   IsHoliday     143 non-null    bool   
 4   CPI           143 non-null    float64
 5   Unemployment  143 non-null    float64
dtypes: bool(1), float64(4), int64(1)
memory usage: 6.8 KB
None
```

Fig 5.11 Size of dataset

Observation:

It provides a summary of a data-frame. It displays key information about the Data Frame such as data types (DType), non-null count, memory usage, and index information.

11. Skewness of dataset:

```
import pandas as pd
import numpy as np
from scipy.stats import skew
print(data.skew())
```

```
Weekly_Sales    3.621226
Temperature    -0.241005
Fuel_Price     -0.225004
IsHoliday       3.408571
CPI             0.162953
Unemployment   -0.508580
dtype: float64
```

Fig 5.12 Skewness of dataset.

Observation:

This code calculates the skewness for each numerical column in the data frame and prints the results. Skewness indicates the asymmetry of a data distribution, with positive skewness indicating a longer right tail, negative skewness indicating a longer left tail, and zero skewness suggesting symmetry. This code provides a quick measure of the distribution's shape for each shape in the dataset.

12. Plotting multiple time series with annotations:

```
plt.style.use('fivethirtyeight')
#plt.figure(figsize=(15,4))
fig, ax = plt.subplots(5,1,figsize=(15,10),sharex=True)
data["Weekly_Sales"].plot(ax=ax[0],title="Weekly Sales/sales on Holiday")
data[data.IsHoliday==True]["Weekly_Sales"].plot(marker="D",ax=ax[0],legend="Holiday Week sale")
data["Temperature"].plot(ax=ax[1], title="Temperature")
data["Fuel_Price"].plot(ax=ax[2],title="Fuel_Price")
data["CPI"].plot(ax=ax[3],title="CPI")
data["Unemployment"].plot(ax=ax[4],title="Unemployment")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ae5f3c78790>
```

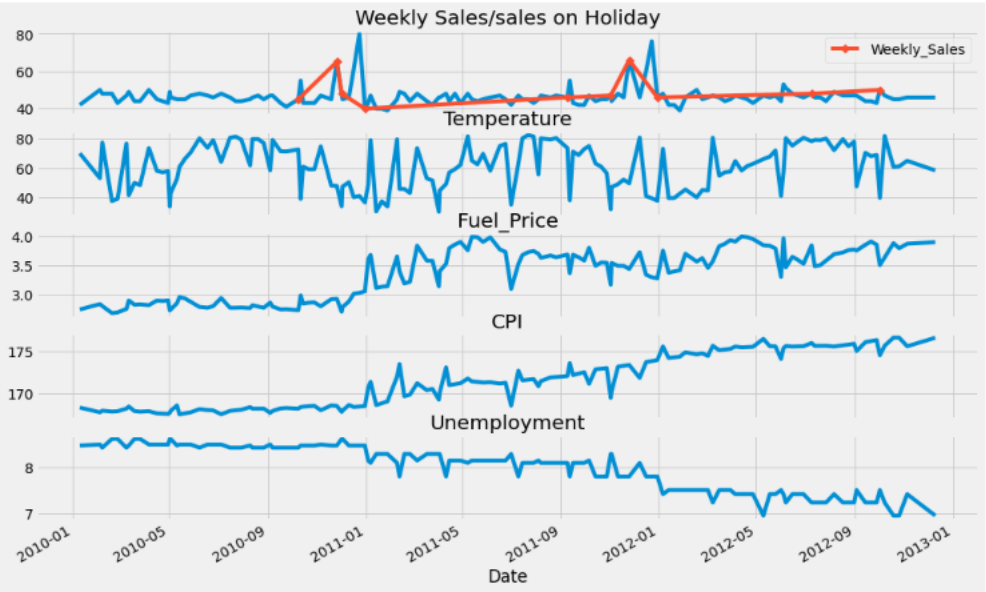

Fig 5.13 Plotting multiple time series

Observation:

This code snippet creates a series of five subplots on the same figure, using a shared x-axis and a defined style. It plots various time series data on the subplots, including weekly sales, temperature, fuel price, consumer price index(cpi), and unemployment, along with highlighting sales on holiday weeks with a distinctive marker.

13. Creating a correlation heatmap with annotations:

```
sns.heatmap(data.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ae5f962aa10>
```

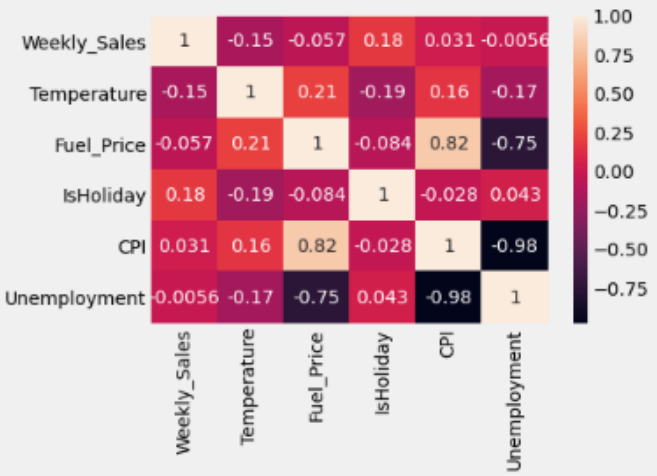

Fig 5.14 Creating a correlation heatmap

Observation:

This code generates a heatmap using seaborn to visualize the correlation matrix of a data frame with numerical correlation values displayed on each cell. It helps in understanding the relationship between different features in the dataset.

14. Creating a bar plot for monthly sales:

```
#### data_sales_month = data.groupby(data.index.month).agg({"Weekly_Sales":"sum"})
plt.figure(figsize=(10, 5))
sns.barplot(x=data_sales_month.index,y=data_sales_month.Weekly_Sales)
plt.title("Month wise Sales")
plt.xlabel("Month")
plt.ylabel("Sales")
```

```
Text(0, 0.5, 'Sales')
```

Fig 5.15 Creating a bar plot for monthly sales

Observation:

This code aggregates weekly sales data by month, calculating the total sales for each month and then creates a bar plot to visualize these monthly sales totals. It sets up appropriate labels for the x-axis, y-axis, and the plot title to represent month wise sales trends.

15. Yearly sales bar plot:

```
data_sales_year = data.groupby(data.index.year).agg({"Weekly_Sales":"sum"})

sns.barplot(x=data_sales_year.index,y=data_sales_year.Weekly_Sales)
plt.title("Year wise Sales")
plt.xlabel("Year")
plt.ylabel("Sales")
```

```
Text(0, 0.5, 'Sales')
```

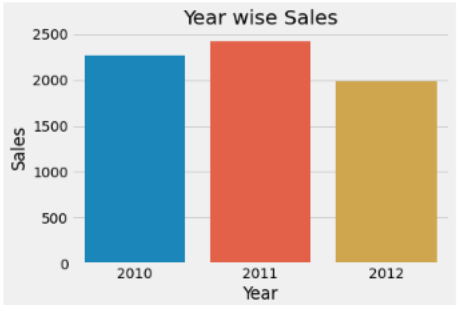

Fig 5.16 Yearly sales bar plot

Observation:

This code calculates the total weekly sales for each sale by grouping the data by the year component of the index, then creates a bar plot to visualize these yearly sales trends.

16. Seasonal decomposition of time series

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(data["Weekly_Sales"], period=45)
```

Fig 5.17 Seasonal decomposition of time series

Observation:

This code uses the “seasonal_decompose” function to analyze the time series data “Weekly_sales” in the “data” Data frame with a specified periodicity of 45. It decomposes the series into its trend, seasonal and residual components to understand underlying patterns and fluctuations

17. Plotting decomposed time series components:

```
plt.figure(figsize=(15, 7))
plt.plot(decomposition.trend)
plt.plot(decomposition.seasonal)
plt.plot(decomposition.resid)
plt.legend(["Trend", "Seasonal","Resid"], loc ="upper right")
```

```
<matplotlib.legend.Legend at 0x7ae5f3e0c7d0>
```

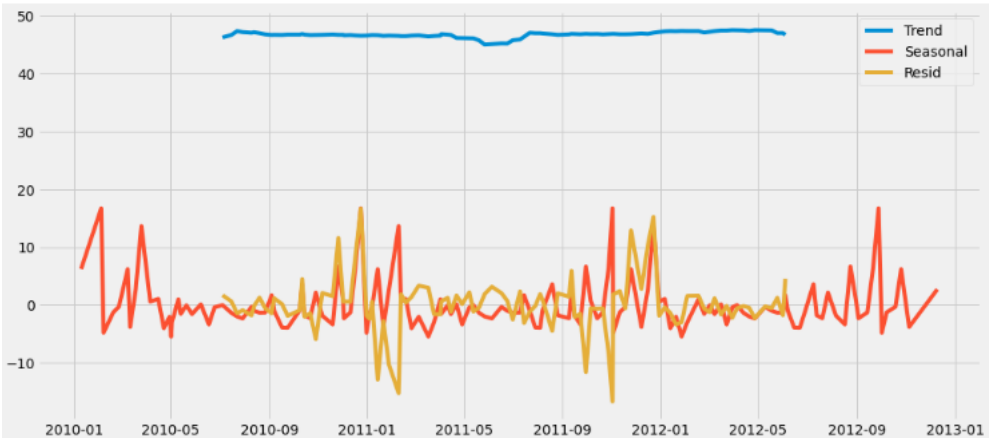

Fig 5.18 Plotting decomposed time series components

Observation:

This code creates a plot to visualize the decomposed components of a time series: trend, seasonal and residual, derived from a previous seasonal decomposition. The figure has a specified size, and the three components are plotted with a legend for easy identification of each line.

18. Aggregating and integrating store level data:

```
data_Store = df_feature.groupby("Store").agg({"Temperature":"mean","Fuel_Price":"mean","IsHoliday":"sum"})

temp_store = df_sales.groupby("Store").agg({"Weekly_Sales":"sum"})
temp_store.Weekly_Sales = temp_store.Weekly_Sales/1000000
temp_store.Weekly_Sales = temp_store.Weekly_Sales.apply(int)
data_Store.set_index(np.arange(0,45),inplace=True)
df_store["temp"] = data_Store.Temperature
df_store["Fuel_Price"] = data_Store.Fuel_Price
df_store["holiday"] = data_Store.IsHoliday
df_store["Weekly_Sales"] = temp_store.Weekly_Sales
```

Fig 5.19 Aggregating and integrating store level data

Observation:

This code aggregates data from the “df_feature” data frame by “store” to get the average temperature, average fuel price and total number of holidays per store. It also calculates the total weekly sales in millions for each store from “df_sales” and integrates this data into a new data frame “df_store”, adding corresponding columns for temperature, fuel price, holidays and sales.

19. Statistical summary of data frame “df_store”

```
df_store.describe()
```

	Store	Size	temp	Fuel_Price	holiday	Weekly_Sales
count	45.000000	45.000000	45.000000	45.000000	45.0	44.000000
mean	23.000000	130287.600000	59.356198	3.405992	13.0	150.090909
std	13.133926	63825.271991	9.956122	0.153935	0.0	78.780478
min	1.000000	34875.000000	37.921264	3.245945	13.0	37.000000
25%	12.000000	70713.000000	51.866319	3.259242	13.0	78.500000
50%	23.000000	126512.000000	58.107912	3.328764	13.0	140.500000
75%	34.000000	202307.000000	68.504670	3.497874	13.0	200.750000
max	45.000000	219622.000000	75.438077	3.643654	13.0	301.000000

Fig 5.20 Statistical summary of data frame “df_store”

Observation:

This code displays the statistical summary of the “df_store” data frame, providing key descriptive statistics such as count, mean, standard deviation, minimum, maximum and quartiles for each numerical column. It gives a quick overview of the distribution and spread of values within “df_stores”.

20. Visualizing store data with count, swarm and box plots:

```
fig,ax = plt.subplots(1,3,figsize=(15, 4))
sns.countplot(df_store.Type,ax=ax[0])
sns.swarmplot(data = df_store,y="Size",x="Type",ax=ax[1])

sns.boxplot(data = df_store,y="Weekly_Sales",x="Type",ax=ax[2])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ae5f3879b10>
```

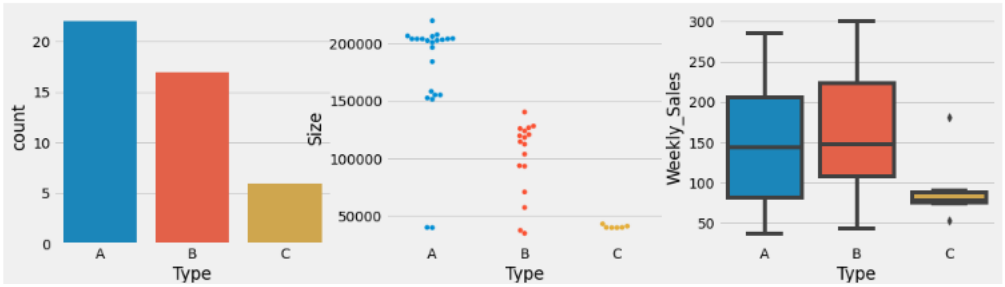

Fig 5.21 Visualizing store data with count, swarm and box plots

Observation:

This code creates a figure with three subplots to visualize information from the “df_store” data frame. It plots a count plot to show the distribution of store types, a swarm plot to illustrate the spread of store sizes by type, and a box plot to compare the distribution of weekly sales among different store types, all within a 15*4 figure.

21. Finding the number of unique departments:

```
#analysis departmentwise
len(df_sales["Dept"].unique())
```

```
81
```

Fig 5.22 Finding the number of unique departments

Observation:

This code calculates the total number of unique departments in the “dept” column of the “df_sales” data frame. It helps to understand the range of departments covered in the sales data.

22. Aggregating and sorting weekly sales by department

```
data_Dept = df_sales.groupby("Dept").agg({"Weekly_Sales":"sum"})
data_Dept.Weekly_Sales = data_Dept.Weekly_Sales/10000
data_Dept.Weekly_Sales = data_Dept.Weekly_Sales.apply(int)
data_Dept.sort_values(by="Weekly_Sales")
```

Dept	Weekly_Sales
39	0
78	0
43	0
47	0
51	3
...	...
90	29106
72	30572
38	39311
95	44932
92	48394

81 rows × 1 columns

Fig 5.23 Aggregating and sorting weekly sales by department

Observation:

This code groups the ‘df_sales’ data frame by “Dept” to calculate the total weekly sales for each department, converting these sales values to units to 10,000 and applying integer rounding. The resulting ‘data_Dept’ Data frame is then sorted by “weekly_sales” to list departments in ascending order of sales.

23. Department-wise sales with vertical lines plot:

```
fig1, ax1 = plt.subplots(figsize=(15, 4))
#ordered_df = data_Dept.sort_values(by='Weekly_Sales')
plt.vlines(x=data_Dept.index, ymin=0, ymax=data_Dept['Weekly_Sales'], color='skyblue')
plt.plot(data_Dept.index,data_Dept['Weekly_Sales'], "o")
plt.title("Departmentwise Sales")
plt.ylabel("Sales")
plt.xlabel("Department")
```

```
Text(0.5, 0, 'Department')
```

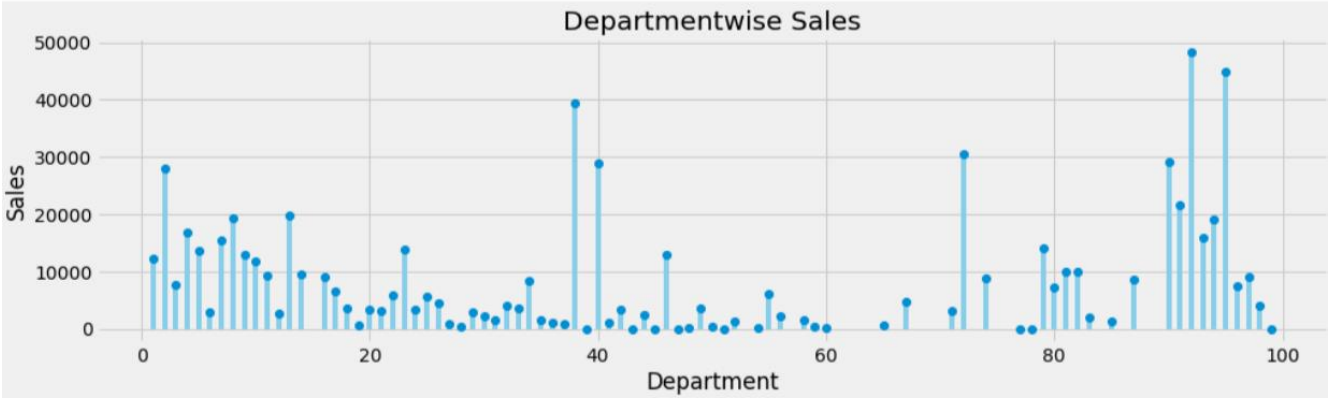

Fig 5.24 Department-wise sales with vertical lines plot

Observation:

This code creates a plot to visualize total sales for each department, using vertical lines(‘vlines’) from the x-axis to the sales values to represent sales by department. Additionally, points are plotted to mark the exact sales values on the graph. The plot has a specified size and includes title, x-axis and y-axis labels for clear interpretation.

24. Merging data frames to create training and testing sets.

```
#analysis store type and year/month wise
sales_date_store = df_sales.groupby(["Date","Store"]).agg({"Weekly_Sales":"sum"})
sales_date_store.sort_index(inplace=True)
sales_date_store.Weekly_Sales = sales_date_store.Weekly_Sales/10000
sales_date_store.Weekly_Sales = sales_date_store.Weekly_Sales.apply(int)
data_table = pd.merge(df_feature,sales_date_store ,  how='left', on=["Date","Store"])
data_table = pd.merge(data_table,df_store[["Store","Type"]] ,  how='left', on=["Store"])
data_table.head(20)
data_train = data_table[data_table.Weekly_Sales.notnull()]
data_test = data_table[data_table.Weekly_Sales.isnull()]
```

Fig 5.25 Merging data frames to create training and testing sets.

Observation:

This code groups the “df_sales” data frame by “date” and “store” to calculate total weekly sales, converts these sales to units of 10,000 and sorts the data by index. It merges this sales data with “df_features” and “df_stores” to create a new “data_table”. The snippet then seperates this table into two subsets: “data_train” containing rows with valid weekly sales, and “data_test” with rows that have missing weekly sales.

25. Bar plot of yearly sales by store type:

```
plt.figure(figsize=(15, 5))
sns.barplot(x=data_train.Date.dt.year, y=data_train.Weekly_Sales,hue=data_train.Type)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ae5f3722c90>
```

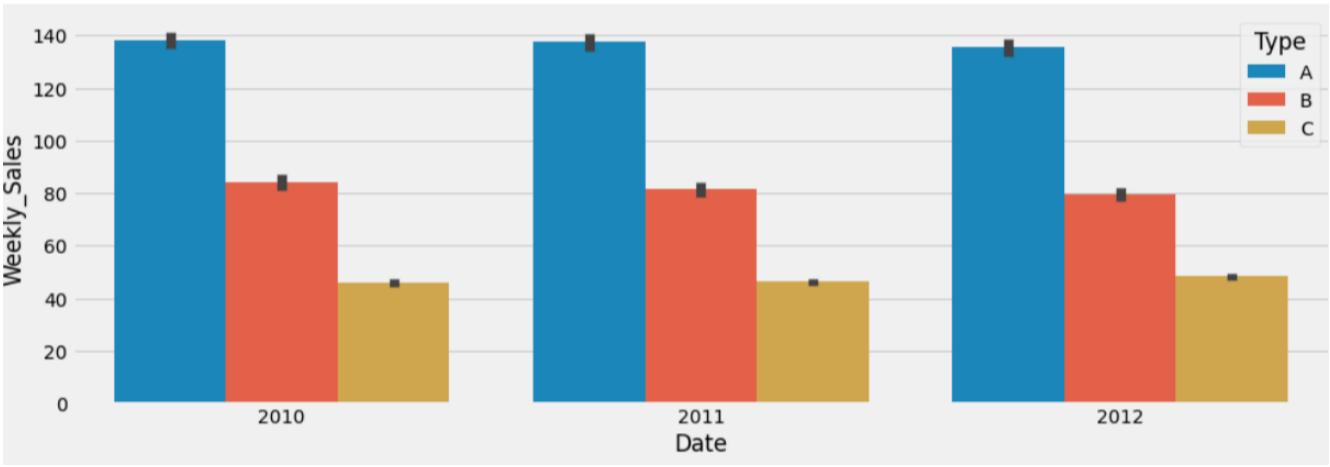

Fig 5.26 Bar plot of yearly sales by store type.

Observation:

This code creates a bar plot to visualize the total weekly sales from the “data_train” data frame, grouped by the year part of the “data” column and distinguished by store type. The plot, with a defined size, uses hue to differentiate the bars by store type, allowing for a quick comparison of sales across different years and store types.

26. Bar plot of monthly sales by store type:

```
plt.figure(figsize=(15, 7))
sns.barplot(x=data_train.Date.dt.month, y=data_train.Weekly_Sales,hue=data_train.Type)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ae5f3651e10>
```

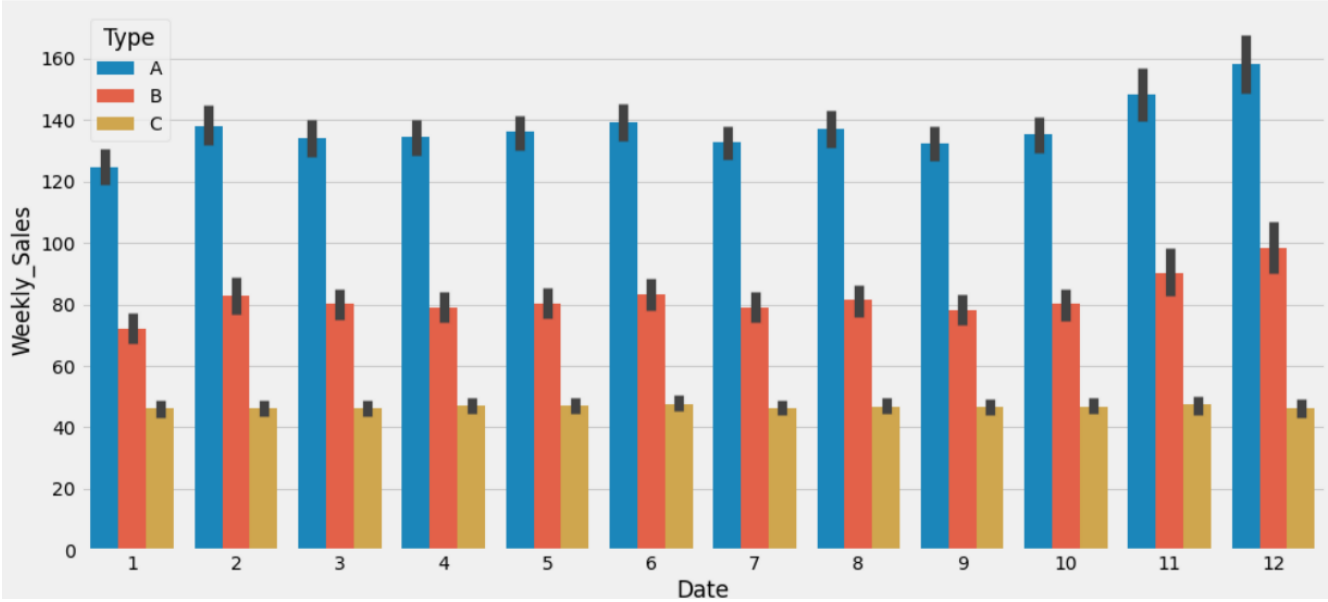

Fig 5.27 Bar plot of monthly sales by store type

Observation:

This code creates a bar plot to visualize the total weekly sales from the ‘data_train’ data frame, grouped by the month part pf the “date” column and separated by store type. The bar plot, with a defined size, uses the hue to differentiate store types, allowing for a clear comparison of monthly sales trends among different store types.

27. Timeline of average markdowns

```
#Analysis of Markdown
plt.figure(figsize=(15,4))
train_markdown = data_table[data_table.MarkDown2.notnull()]
train_markdown = train_markdown.groupby("Date").agg({"MarkDown1":"mean","MarkDown2":"mean","MarkDown3":"mean","MarkDown4":"mean","MarkDown5":"mean"})


plt.plot(train_markdown.index,train_markdown.MarkDown1)
plt.plot(train_markdown.index,train_markdown.MarkDown2)
plt.plot(train_markdown.index,train_markdown.MarkDown3)
plt.plot(train_markdown.index,train_markdown.MarkDown4)
plt.plot(train_markdown.index,train_markdown.MarkDown5)
plt.title("Timeline Markdown")
plt.ylabel("Markdown")
plt.xlabel("Date")
```

```
Text(0.5, 0, 'Date')
```

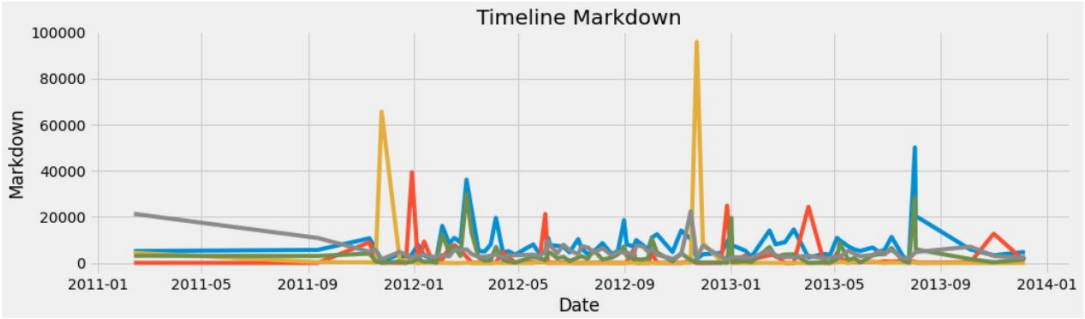

Fig 5.28 Timetline of average markdowns.

Observation:

This code snippet creates a line plot to visualize the timeline of average markdowns from the “train_markdown” Data frame, which groups data by "Date" and computes the mean for various markdowns (MarkDown1 to MarkDown5). The plot, with a 15x4 size, represents the evolution of these markdowns over time with corresponding x-axis (dates) and y-axis (markdown values) labels, alongside a title for context.

28. Histogram of markdowns:

```
train_markdown.hist(figsize=(10,8),bins=6,color='Y')

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

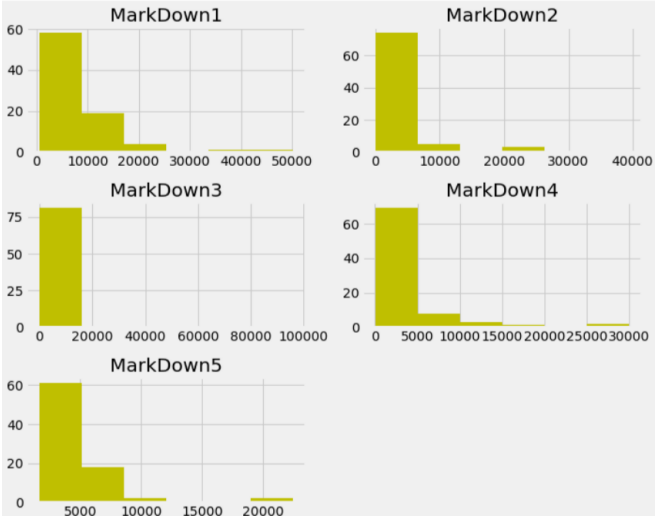

Fig 5.29 Histogram of markdowns

Observation:

This code snippet creates histograms to visualize the distribution of various markdown values in the train_markdown Data Frame, with specified dimensions and bin count. It generates a separate histogram for each of the markdown columns (MarkDown1 through MarkDown5), and uses a common color ('Y') for consistency, with tight_layout ensuring proper spacing and alignment of plots.

29. Stacked bar plot for monthly markdowns:

```
train_markdown_month = train_markdown.groupby(train_markdown.index.month).agg({"MarkDown1":"mean","MarkDown2":"mean","MarkDown3":"mean","MarkDown4":"mean","MarkDown5":"mean"})

train_markdown_month.plot(kind='bar', stacked=True,figsize=(15,6))
plt.title("Stacked Monthwise Morkdown")
plt.ylabel("Markdown")
```

```
Text(0, 0.5, 'Markdown')
```

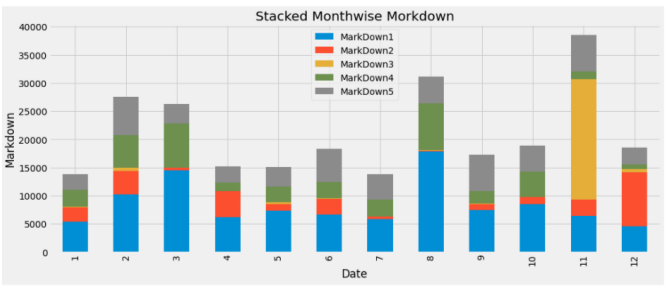

Fig 5.30 Stacked bar plot for monthly markdowns.

Observation:

This code snippet creates a stacked bar plot to visualize the average monthly markdowns from the `train_markdown` Data frame, grouped by month. It plots the mean values for each markdown type (MarkDown1 through MarkDown5), stacking them for each month to illustrate the cumulative effect, with a specified figure size and labelled axes for clarity.

30. Stacked bar plot for store-type wise markdowns.

```
train_markdown_1 = data_table[data_table.MarkDown2.notnull()]
train_markdown_type = train_markdown_1.groupby("Type").agg({"MarkDown1":"mean","MarkDown2":"mean","MarkDown3":"mean","MarkDown4":"mean","MarkDown5":"mean"})

train_markdown_type.plot(kind='bar', stacked=True,figsize=(10,4))
plt.title("Stacked StoreType Wise")
plt.ylabel("Markdown")
```

```
Text(0, 0.5, 'Markdown')
```

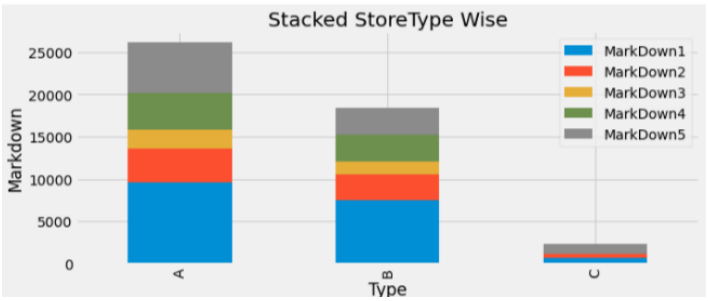

Fig 5.31 Stacked bar plot for store-type wise markdowns.

Observation:

This code snippet creates a stacked bar plot to visualize the average markdowns by store type, grouping the `train_markdown_1` data frame by "Type". The plot displays the mean values for different markdowns (MarkDown1 through MarkDown5), with each store type represented by a stacked bar, allowing for easy comparison among types. It has a specified size and title, with proper axis labeling for clarity.

31. Importing regression models and imputation techniques:

```
#prediction on data and score
from fancyimpute import IterativeImputer
from sklearn.metrics import mean_squared_error

from sklearn.svm import SVR, LinearSVR, NuSVR
from sklearn.linear_model import ElasticNet, Lasso, RidgeCV,LinearRegression
from sklearn.kernel_ridge import KernelRidge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor,AdaBoostRegressor,RandomForestRegressor
import xgboost as xgb
import lightgbm as lgb
```

Fig 5.32 Importing regression models and imputation techniques

Observation:

This code snippet imports various regression models and an imputation technique for data science and machine learning tasks. It includes traditional linear models, support vector regression, ensemble-based algorithms, and gradient boosting frameworks (XGBoost and LightGBM), along with an iterative imputer for handling missing data, and the `mean_squared_error` metric for model evaluation.

32. Creating dummy variables for categorical columns:

```
#clean up and preprocessing
def createdummies(data,cols):
    for col in cols:
        one_hot = pd.get_dummies(data[col],prefix=col)
        data = data.join(one_hot)
        data.drop(col,axis = 1,inplace=True)

    return data
```

Fig 5.33 Creating dummy variables for categorical columns

Observation:

This code defines a function `createdummies` that takes a Data frame `data` and a list of column names `cols`. It generates one-hot encoded dummy variables for each specified column and joins them to the Data frame, then removes the original categorical columns. This approach helps to transform categorical data into a numerical format suitable for machine learning models.

33. Imputing missing values and preparing data

```
# imputing the missing value
itt = IterativeImputer()
df = itt.fit_transform(data_table[["MarkDown1","MarkDown2","MarkDown3","MarkDown4","MarkDown5"]])
data_table.MarkDown1 = df[:,0]
data_table.MarkDown2 = df[:,1]
data_table.MarkDown3 = df[:,2]
data_table.MarkDown4 = df[:,3]
data_table.MarkDown5 = df[:,4]

data_table['CPI'].fillna((data_table['CPI'].mean()), inplace=True)
data_table['Unemployment'].fillna((data_table['Unemployment'].mean()), inplace=True)
data_table['IsHoliday'] = data_table['IsHoliday'].map({True:0,False:1})

#create new column
data_table["Month"] = data_table.Date.dt.month
data_table["Year"] = data_table.Date.dt.year
data_table["WeekofYear"] = data_table.Date.dt.weekofyear
data_table.drop(['Date'],axis=1,inplace=True)

#create dummies out of categorical column
data_table = createdummies(data_table,["Type","Month","Year","WeekofYear"])
```

Fig 5.34 Importing missing values and preparing data

Observation:

This code snippet addresses missing values and prepares the `data_table` Data frame for machine learning tasks. It uses an iterative imputer to fill in missing values for multiple "MarkDown" columns, and fills missing values in "CPI" and "Unemployment" with their mean. The code also converts "IsHoliday" to binary, creates new time-based columns, and removes the original "Date" column. It applies the `createdummies` function to create one-hot encoded dummy variables for categorical columns, preparing the data for model training.

34. Viewing data frame column names:

```
data_table.columns
```

```
Index(['Store', 'Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2',
       'MarkDown3', 'MarkDown4', 'MarkDown5', 'CPI', 'Unemployment',
       'IsHoliday', 'Weekly_Sales', 'Type_A', 'Type_B', 'Type_C', 'Month_1',
       'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6', 'Month_7',
       'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Month_12', 'Year_2010',
       'Year_2011', 'Year_2012', 'Year_2013', 'WeekofYear_1', 'WeekofYear_2',
       'WeekofYear_3', 'WeekofYear_4', 'WeekofYear_5', 'WeekofYear_6',
       'WeekofYear_7', 'WeekofYear_8', 'WeekofYear_9', 'WeekofYear_10',
       'WeekofYear_11', 'WeekofYear_12', 'WeekofYear_13', 'WeekofYear_14',
       'WeekofYear_15', 'WeekofYear_16', 'WeekofYear_17', 'WeekofYear_18',
       'WeekofYear_19', 'WeekofYear_20', 'WeekofYear_21', 'WeekofYear_22',
       'WeekofYear_23', 'WeekofYear_24', 'WeekofYear_25', 'WeekofYear_26',
       'WeekofYear_27', 'WeekofYear_28', 'WeekofYear_29', 'WeekofYear_30',
       'WeekofYear_31', 'WeekofYear_32', 'WeekofYear_33', 'WeekofYear_34',
       'WeekofYear_35', 'WeekofYear_36', 'WeekofYear_37', 'WeekofYear_38',
       'WeekofYear_39', 'WeekofYear_40', 'WeekofYear_41', 'WeekofYear_42',
       'WeekofYear_43', 'WeekofYear_44', 'WeekofYear_45', 'WeekofYear_46',
       'WeekofYear_47', 'WeekofYear_48', 'WeekofYear_49', 'WeekofYear_50',
       'WeekofYear_51', 'WeekofYear_52'],
      dtype='object')
```

Fig 5.35 Viewing data frame column names

Observation:

This code retrieves and displays the list of column names in the `data_table` DataFrame. It allows you to quickly inspect the current structure of the DataFrame and verify which columns are present after data processing.

35. Splitting data into training and testing sets:

```
#data split
data_train = data_table[data_table.Weekly_Sales.notnull()]
data_test = data_table[data_table.Weekly_Sales.isnull()]
X = data_train.drop('Weekly_Sales', axis=1)
y = data_train['Weekly_Sales']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Fig 5.36 Splitting data into training and testing sets.

Observation:

This code snippet divides the `data_table` DataFrame into training and testing sets based on the presence of "Weekly_sales" data. It then separates the features (X) from the target variable (y), removing "Weekly_sales" from the feature set. Using `train_test_split`, it further splits the training data into train and test subsets, allocating 20% to the test set, to be used for model validation.

36. Model creation and evaluation loop for regression classifiers:

```
#basic model creation

classifiers = [
    LinearRegression(),
    ElasticNet(),
    RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1]),
    KernelRidge(alpha=0.6, kernel='polynomial', degree=3, coef0=2.5),
    Lasso(alpha =16, random_state=100),
    ElasticNet(alpha=0.8),
    DecisionTreeRegressor(),
    RandomForestRegressor(),
    GradientBoostingRegressor(),
    AdaBoostRegressor(),
    SVR(),
    LinearSVR(),
    NuSVR(),
    xgb.XGBRegressor(),
    lgb.LGBMRegressor()
    ]

name = []
score = []
models = []
rmse = []
i = 0
for classifier in classifiers:
    classifier.fit(X_train, y_train)
    name.append(type(classifier).__name__)
    score.append(classifier.score(X_test, y_test))
    models.append(classifier)
    rmse.append(np.sqrt(mean_squared_error(classifier.predict(X_test), y_test)))
```

Fig 5.37 Model creation and evaluation loop for regression classifiers

Observation:

This code snippet creates a loop to fit various regression classifiers from a predefined list to training data (`X_train`, `y_train`), then evaluates each model's performance on test data (`X_test`, `y_test`). It collects model names, R^2 scores, and root mean square errors (RMSE) for each classifier, providing a comparative assessment of their predictive accuracy and error rate.

37. Comparing model performance:

```
#comparing model performance
df_score = pd.DataFrame(list(zip(name,rmse, score, models)),columns=['name','rmse','score',"model"])
df_score.set_index('name',inplace=True)
df_score.sort_values(by=['score'],inplace=True)
df_score
```

name	rmse	score	model
KernelRidge	237.001694	-15.992572	KernelRidge(alpha=0.6, coef0=2.5, kernel='poly...
LinearSVR	145.098630	-5.369170	LinearSVR()
SVR	55.192274	0.078463	SVR()
NuSVR	54.817125	0.090948	NuSVR()
Lasso	51.288758	0.204206	Lasso(alpha=16, random_state=100)
ElasticNet	44.919414	0.389586	ElasticNet()
ElasticNet	44.277546	0.406906	ElasticNet(alpha=0.8)
LinearRegression	40.319381	0.508205	LinearRegression()
RidgeCV	40.309095	0.508456	RidgeCV(alphas=array([0.001, 0.01 , 0.1 , 1. ...
AdaBoostRegressor	36.507419	0.596802	(DecisionTreeRegressor(max_depth=3, random_sta...
GradientBoostingRegressor	15.070274	0.931293	([DecisionTreeRegressor(criterion='friedman_ms...
DecisionTreeRegressor	13.053027	0.948456	DecisionTreeRegressor()
RandomForestRegressor	11.810952	0.957799	(DecisionTreeRegressor(max_features='auto', ra...
LGBMRegressor	9.504760	0.972670	LGBMRegressor()
XGBRegressor	9.208618	0.974347	XGBRegressor(base_score=0.5, booster='gbtree',...

Fig 5.38 Comparing model performance.

Observation:

This code snippet creates a DataFrame, `df_score`, to store the evaluation results for different regression models, with columns for the model's name, root mean square error (RMSE), R^2 score, and the model instance itself. It sets "name" as the index, sorts the DataFrame by "score" (ascending), and displays the contents, providing an organized view of model performance metrics.

38. Making predictions with a trained model:

```
#prediction:
model = df_score.loc["XGBRegressor","model"]
data_test.drop(['Weekly_Sales'],axis=1,inplace=True)
predict = model.predict(data_test)
predict
```

```
array([162.82635 , 165.51541 , 155.3577  , ...,  78.576775,  75.810616,
        73.44964 ], dtype=float32)
```

Fig 5.39 Making predictions with a trained model

Observation:

This code snippet retrieves the XGBoost model from the `df_score` DataFrame and uses it to predict values for the `data_test` DataFrame after dropping the "Weekly_Sales" column. It returns the predicted results, indicating the expected weekly sales for the test data based on the trained XGBoost regressor.

CHAPTER 6

CONCLUSION

The analysis and code execution for the "retail sales data" report provides valuable insights into the retail environment, revealing patterns and trends that drive sales. The code's approach combines data aggregation, exploratory analysis, and predictive modeling, leading to several important conclusions for retail sales forecasting.

The initial steps in the code involve aggregating and exploring the retail data, focusing on key attributes such as temperature, fuel price, Consumer Price Index (CPI), and unemployment rate. By grouping sales data by date and store, the code identifies the impact of holidays, seasonal variations, and other external factors on weekly sales.

The code also provides a comprehensive view of the data through various visualizations. Line plots, bar plots, and heatmaps reveal correlations and trends, allowing for a deeper understanding of how different variables affect sales. For example, temperature and fuel price might correlate with sales fluctuations, indicating consumer behavior changes due to weather or economic conditions.

The code implements multiple predictive models to forecast retail sales. The use of diverse regression models, including linear regression, decision trees, and ensemble methods, allows for a robust comparison of different approaches. The final step involves using the best-performing model (XGBoost Regressor) to predict future sales, providing a practical forecast for decision-making.

These predictive analytics techniques offer insights into sales patterns and help businesses plan for inventory, staffing, and marketing. The code's use of time series decomposition to identify trends and seasonality further enhances the understanding of the retail environment.

The analysis highlights the importance of retail sales forecasting in making data-driven decisions. By combining predictive analytics and exploratory data analysis, the code identifies significant patterns in the retail data. This approach supports effective inventory management, optimized staffing, and targeted marketing campaigns.

To further enhance retail sales forecasting, businesses should focus on refining predictive models, integrating additional data sources, and continuously updating models with new data. This iterative process ensures that forecasts remain accurate and relevant in a rapidly changing retail landscape.

CHAPTER 7

RESULTS AND DISCUSSION

This project aimed to analyze retail sales data and forecast future sales using a combination of predictive analytics and time series analysis techniques. The key components of the code and its outputs are summarized below.

- Data Processing and Analysis

The project utilized data from multiple sources within a zip archive, including "Features data set", "Sales data-set", and "Stores data-set". The code handled data extraction, aggregation, and merging to prepare a comprehensive dataset for analysis.

- The initial data analysis involved:

- Descriptive Statistics: The `describe()` method was used to gather summary statistics, providing insights into central tendency and data dispersion.

- Data Correlations: A heatmap showed relationships among different features, identifying potential factors influencing sales.

- Holiday Identification: The code created a holiday flag to account for specific periods impacting sales.

- Various Plots: Line plots, bar plots, box plots, and others were generated to visualize data trends and distributions.

- Algorithms used:

The algorithms associated with these models include:

- Linear Regression: Methods based on fitting a linear model.
- Ridge and Lasso Regression: Techniques that add regularization to linear regression.
- ElasticNet: A combination of Ridge and Lasso regularization.
- Kernel Methods: Techniques involving transformation of input data with kernel functions.

- Tree-Based Algorithms: Models built using decision trees, like RandomForest and DecisionTree.
- Boosting Algorithms: Methods that combine weak learners to improve prediction, such as GradientBoosting, AdaBoost, XGBoost, and LightGBM.

- Retail Sales Forecasting

The project employed a wide range of machine learning models to predict "Weekly_Sales":

-Linear Models: LinearRegression, RidgeCV, Lasso, ElasticNet.

- Tree-Based Models: DecisionTreeRegressor, RandomForestRegressor, GradientBoostingRegressor.

- Boosting Models: XGBoost, LightGBM.

- Kernel-Based Models: KernelRidge, SVR.

These models provided a comprehensive set of tools to explore different prediction approaches. The models were evaluated based on metrics like R-squared and root mean square error (RMSE), allowing a comparison of their predictive accuracy.

- Model Performance and Prediction

The final prediction utilized the XGBoost model, which demonstrated a strong performance during evaluation. The output from `predict()` presented predicted sales values for the test dataset, offering a forecast of future sales.

- Conclusion

The project provided a thorough analysis of retail sales data, yielding valuable insights into sales trends and key factors affecting them. Using various regression models, it established a robust framework for predicting weekly sales.

The results from the final prediction using the XGBoost model offer useful information for business planning, inventory management, and resource allocation.

Overall, the project demonstrated the power of forecasting predictive analytics in understanding and forecasting retail sales, providing a solid foundation for further analysis and business decision-making.

REFERENCES

[1] Massaro, A., Panarese, A., Giannone, D., & Galiano, A. (2021). Augmented data and XGBoost improvement for sales forecasting in the large-scale retail sector. *Applied Sciences*, *11*(17), 7793.

[2] Ensafi, Y., Amin, S. H., Zhang, G., & Shah, B. (2022). Time-series forecasting of seasonal items sales using machine learning–A comparative analysis. *International Journal of Information Management Data Insights*, *2*(1), 100058.

[3] Fildes, R., Ma, S., & Kolassa, S. (2022). Retail forecasting: Research and practice. *International Journal of Forecasting*, *38*(4), 1283-1318.

[4] Punia, S., Nikolopoulos, K., Singh, S. P., Madaan, J. K., & Litsiou, K. (2020). Deep learning with long short-term memory networks and random forests for demand forecasting in multi-channel retail. *International journal of production research*, *58*(16), 4964-4979.

[5] Kohli, S., Godwin, G. T., & Urolagin, S. (2020). Sales prediction using linear and KNN regression. In *Advances in Machine Learning and Computational Intelligence: Proceedings of ICMLCI 2019* (pp. 321-329). Singapore: Springer Singapore.

[6] Spiliotis, E., Makridakis, S., Semenoglou, A. A., & Assimakopoulos, V. (2022). Comparison of statistical and machine learning methods for daily SKU demand forecasting. *Operational Research*, *22*(3), 3037-3061.

[7] Wolters, J., & Huchzermeier, A. (2021). Joint in-season and out-of-season promotion demand forecasting in a retail environment. *Journal of Retailing*, *97*(4), 726-745.

[8] Lalou, P., Ponis, S. T., & Efthymiou, O. K. (2020). Demand forecasting of retail sales using data analytics and statistical programming. *Management & Marketing*, *15*(2), 186-202.

[9] Mitra, A., Jain, A., Kishore, A., & Kumar, P. (2022, September). A comparative study of demand forecasting models for a multi-channel retail company: a novel hybrid machine learning approach. In *Operations research forum* (Vol. 3, No. 4, p. 58). Cham: Springer International Publishing.

[10] Akande, Y. F., Idowu, J., Misra, A., Misra, S., Akande, O. N., & Ahuja, R. (2021, October). Application of XGBoost Algorithm for Sales Forecasting Using Walmart Dataset. In *International Conference on Advances in Electrical and Computer Technologies* (pp. 147-159). Singapore: Springer Nature Singapore.