

# Walmart Sales Analysis For Retail Industry

## With Machine Learning

Harshit Raj(21BBS0219)

Syed Ausaf Ejaz(21BML0151)

Dev Patel(21BCE3670)

Pranaya Khunteta(21BBS0186)

Category: Machine Learning

### 1. INTRODUCTION

#### 1.1 Project Overview:

Sales forecasting is the process of estimating future sales. Accurate sales forecasts enable companies to make informed business decisions and predict short-term and long-term performance. Companies can base their forecasts on past sales data, industry-wide comparisons, and economic trends. Here, the company is Walmart. Walmart is a renowned retail corporation that operates a chain of hypermarkets. Walmart has provided data by combining the data of 45 stores including store information and monthly sales. Walmart runs several promotional markdown events throughout the year. These markdowns precede prominent holidays, the four largest of which are the Super Bowl, Labor Day, Thanksgiving, and Christmas. The weeks including these holidays are weighted five times higher in the evaluation than non-holiday weeks. The data is provided on a weekly basis.

#### 1.2 Purpose:

We have to find the impact of holidays on the sales of the store. The holidays included are Christmas, Thanksgiving, Super Bowl and Labour Day. We will be using algorithms such as Random Forest, Decision tree, XgBoost and ARIMA. We will train and test the data with these algorithms. Flask integration and Azure deployment will also be done.

### 2. LITERATURE SURVEY

#### 2.1 Existing problem:

The task at hand is to analyze and predict the impact of holidays on the sales of Walmart, a prominent retail corporation operating a chain of hypermarkets. This analysis is crucial for making informed business decisions and effectively planning for promotional markdown events associated with major holidays, including Christmas, Thanksgiving, Super Bowl, and Labor Day.

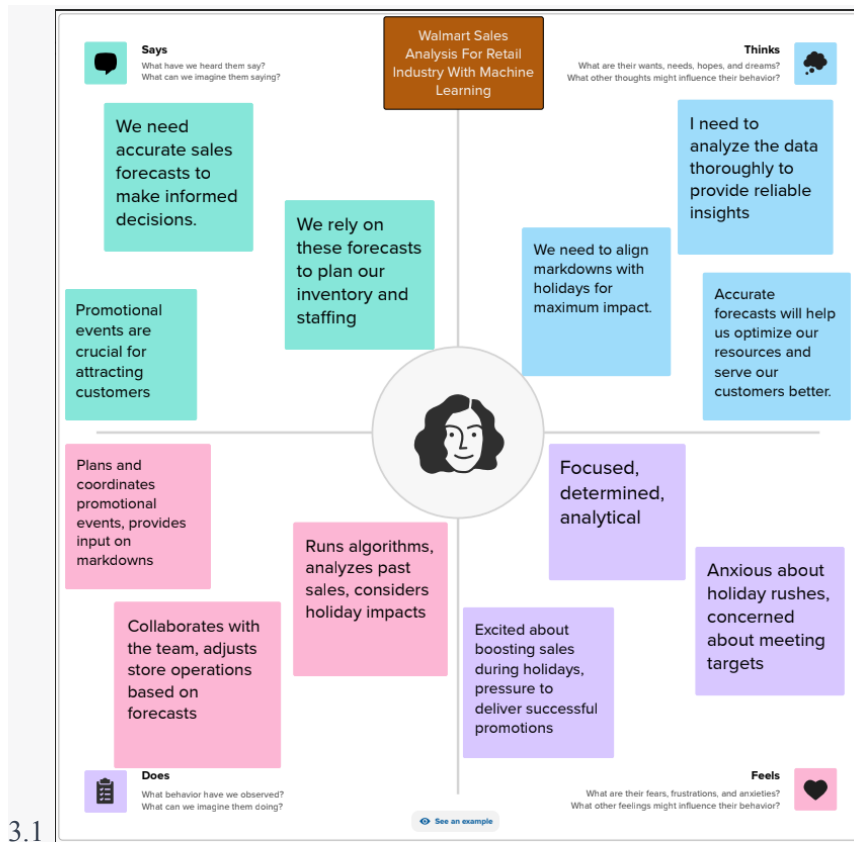
## 2.2 References:

1. Brownlee, J. (2016). Introduction to Time Series Forecasting with Python. Machine Learning Mastery. [Link](#)
2. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794).
3. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
4. sklearn.ensemble.RandomForestRegressor. scikit-learn Documentation. [Link](#)

## 2.3 Problem Statement Definition:

The problem at hand revolves around Walmart, a renowned retail corporation operating a chain of hypermarkets. The objective is to conduct a comprehensive analysis of sales data to accurately forecast future sales, with a particular focus on understanding the impact of major holidays, including Christmas, Thanksgiving, Super Bowl, and Labor Day.

### 3. IDEATION & PROPOSED SOLUTION



### 3.2 Brainstorm & Idea Prioritization:



## Conducting a brainstorm

Executing a brainstorm isn't unique; holding a productive brainstorm is. Great brainstorms are ones that set the stage for fresh and generative thinking through simple guidelines and an open and collaborative environment. Use this when you're just kicking-off a new project and want to hit the ground running with big ideas that will move your team forward.

- 🕒 15 minutes to prepare
- 🕒 30-60 minutes to collaborate
- 👤 3-8 people recommended



#### Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

🕒 15 minutes



##### Choose your best "How Might We" Questions

Create 5 HMW statements before the activity to propose them to the team.



##### Set the stage for creativity and inclusivity

Go over the brainstorming rules and keep them in front of your team while brainstorming to encourage collaboration, optimism, and creativity.

1. **Encourage wild ideas** (If none of the ideas sound a bit ridiculous, then you are filtering yourself too much.)
2. **Defer judgement** (This can be as direct as harsh words or as subtle as a condescending tone or talking over one another.)
3. **Build on the ideas of others** ("I want to build on that idea" or the use of "yes, and...")
4. **Stay focused on the topic at hand**
5. **Have one conversation at a time**
6. **Be visual** (Draw and/or upload to show ideas, whenever possible.)
7. **Go for quantity**



##### Interested in learning more?

Check out the Meta Think Kit website for additional tools and resources to help your team collaborate, innovate and move ideas forward with confidence.

[Open the website](#) →



#### "How Might We"

Certainly, here are the top 5 brainstorming questions. Let the group select the most promising one to begin the idea generation process based on their impact and relevance to your goals:

How might we use data analytics to optimize inventory management and reduce costs while ensuring Walmart's shelves are well-stocked with popular products?

How might we use machine learning to predict sales trends and optimize inventory management for Walmart, reducing overstock and understock issues?

How might we leverage customer data and machine learning algorithms to personalize marketing and promotions for Walmart's online and in-store customers, increasing sales and customer satisfaction?

How might we analyze historical sales data to identify the impact of external factors such as holidays, weather, and economic conditions on Walmart's sales, and use this analysis to make informed business decisions?

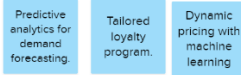
How might we leverage historical sales data to identify and predict sales trends for different product categories within Walmart?

2

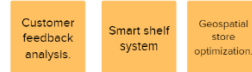
### Brainstorm solo

In the "solo brainstorm space," participants are encouraged to silently generate ideas and populate a template. This approach fosters diversity of thought, mitigating groupthink, and caters to both introverts and extroverts. It's important to establish a time limit to maintain focus and productivity, while also promoting a quantity-over-quality mindset to spur creativity.

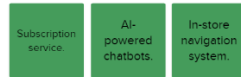
#### Person 1



#### Person 2



#### Person 3



#### Person 4

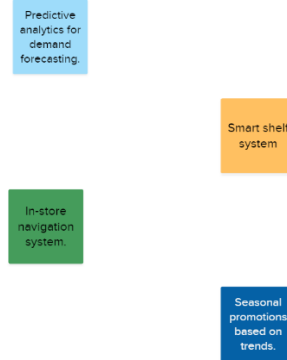


3

### Brainstorm as a group



Transitioning to the "group sharing space," participants move their ideas for collective review. The team silently reads through these contributions and collaboratively organizes them into thematic categories or according to similarities. During this process, it's essential to address any questions that may arise. Encouraging the "Yes, and..." approach fosters idea expansion and collaboration, where team members build upon each other's suggestions to create a more comprehensive brainstorming session.



4

### Decide your focus

Give each person two icons to vote which idea should your team focus on.

⌚ 5 minutes

#### Person 1



#### Person 2



#### Person 3



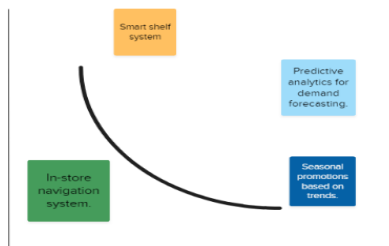
#### Person 4



→

### Prioritize your idea

A brainstorm like this typically results in a handful of promising ideas that you can carry forward and act upon.



## 4. REQUIREMENT ANALYSIS

### 4.1 Functional requirement:

#### Data Ingestion and Preprocessing:

- The system should be able to ingest and process the provided dataset, which includes information about 45 Walmart stores, along with monthly sales data organized on a weekly basis.

#### Data Cleaning and Validation:

- The system should perform data cleaning operations, including handling missing or erroneous values, ensuring consistency, and validating the integrity of the dataset.

#### Feature Engineering:

- The system should automatically identify and engineer relevant features that may influence sales, such as store attributes, historical sales data, and promotional events.

#### Holiday Impact Assessment:

- The system should implement a method to assess the impact of holidays on sales, considering factors like proximity to the holiday, duration of the holiday season, and historical sales performance during previous holiday periods.

#### Holiday Weighting:

- The system should apply a weighting scheme that gives higher importance to weeks surrounding major holidays (Super Bowl, Labor Day, Thanksgiving, and Christmas) during the analysis.

#### Model Selection and Training:

- The system should support the selection and training of multiple machine learning models, including Random Forest, Decision Tree, XGBoost, and ARIMA, for accurate sales forecasting.

#### Model Evaluation:

- The system should evaluate the performance of each trained model using appropriate metrics, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

#### Flask Interface:

- The system should integrate a user-friendly Flask interface for interacting with the trained models. This interface should allow users to input relevant parameters, such as store ID and date, and receive sales predictions.

#### Azure Cloud Deployment:

- The system should facilitate the deployment of the final model(s) on the Azure Cloud platform for easy access, scalability, and availability.

### 4.2 Non Functional Requirement:

#### Performance:

- The system should be able to handle large datasets and provide timely responses to user queries, ensuring smooth and efficient operation.

#### Scalability:

- The system should be designed to scale with increased data volume and user load, allowing for growth in both the size of the dataset and the number of users.

#### Reliability:

- The system should be reliable and available, with minimal downtime or disruptions in service. It should also have mechanisms in place for automatic recovery in case of

failures.

Usability:

- The user interface should be intuitive, easy to navigate, and user-friendly, requiring minimal training or technical expertise for users to effectively interact with the system.

Accessibility:

- The system should be accessible to a diverse user base, including those with disabilities, by adhering to accessibility standards and providing features like screen reader compatibility.

Security:

- The system should implement robust security measures to protect sensitive data and user information, including encryption, access controls, and secure authentication methods.

Privacy:

- The system should comply with privacy regulations and policies, ensuring that user data is handled and stored securely and in accordance with legal requirements.

Compliance:

- The system should adhere to relevant industry standards, regulations, and best practices, ensuring compliance with data protection laws and other relevant guidelines.

Maintainability:

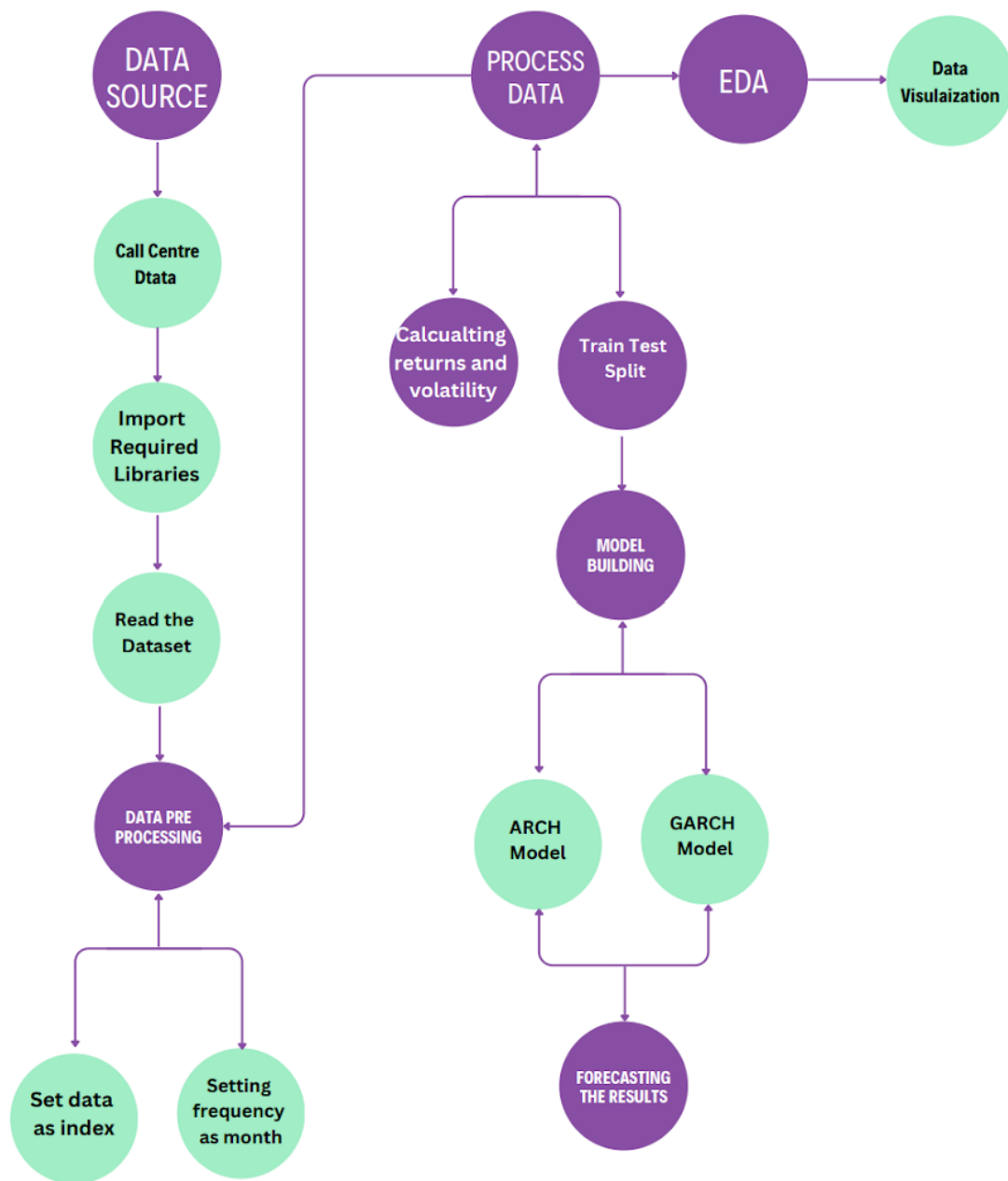
- The system should be designed with clean and well-documented code, making it easy for developers to understand, modify, and maintain the system in the future.

Scalability:

- The system should be capable of handling an increasing number of users and growing data volumes without significant degradation in performance or functionality.

## 5. PROJECT DESIGN

### 5.1 Data Flow Diagrams & User Stories:



## User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Retail)	Sales Analysis	WMSA-1	Access Walmart sales data for a specific time period	View sales data for selected time period	High	Sprint-1



Analyst )						
		WMSA-2	Filter sales data by product category for analysis	View sales data filtered by product category	High	Sprint-1
		WMSA-3	Generate visualizations (charts, graphs) based on sales data	Generate visualizations representing sales trends and patterns	Medium	Sprint-2
		WMSA-4	Export analyzed sales data and visualizations for reporting	Export data in CSV or PDF format	Medium	Sprint-2
		WMSA-5	Set custom alerts for specific sales thresholds	Receive notifications for defined thresholds	Low	Sprint-3
Data Scientist	Data Modeling	WMSA-6	Access raw Walmart sales data for machine learning modeling	Download raw sales data for machine learning analysis	High	Sprint-1
		WMSA-7	Train machine learning models on Walmart sales data	Predict future sales trends through trained models	High	Sprint-2
		WMSA-8	Evaluate accuracy of machine learning models based on historical data	Assess model accuracy and adjust parameters as needed	Medium	Sprint - 4

## 5.2 Solution Architecture:

The sales forecasting system for Walmart requires a robust architecture to handle data processing, model training, user interaction, and deployment. Below is a proposed solution architecture:

### Data Ingestion and Storage:

- Data Sources: Walmart's provided dataset containing store information and monthly sales data.
- Data Ingestion: Use tools like Apache Kafka, AWS Kinesis, or Google Cloud Pub/Sub for real-time data ingestion. For batch processing, tools like Apache Airflow or AWS Glue can be used.
- Data Storage: Store the ingested data in a scalable and reliable database such as Amazon Redshift, Google BigQuery, or a NoSQL database like MongoDB.

### Data Preprocessing and Feature Engineering:

- Preprocessing: Use tools like Apache Spark or Python libraries like Pandas for data cleaning, normalization, and handling missing values.
- Feature Engineering: Implement feature extraction techniques to derive relevant features from the raw data, considering factors like store attributes, historical sales, and holiday information.

### Holiday Impact Assessment and Weighting:

- Implement custom logic or algorithms to assess the impact of holidays on sales and apply the appropriate weighting scheme.

#### Model Training and Evaluation:

- Model Selection: Use machine learning libraries like scikit-learn, XGBoost, and statsmodels for implementing Random Forest, Decision Tree, XGBoost, and ARIMA models.
- Model Training: Utilize distributed computing frameworks like Apache Spark for training models on large datasets.
- Model Evaluation: Evaluate models using appropriate metrics (e.g., MAE, MSE, RMSE) through techniques like cross-validation and holdout testing.

#### Flask Web Application:

- Develop a Flask-based web application for user interaction.
- User Interface: Create input fields for parameters like store ID and date, and display predicted sales results.
- Communication: Flask handles user requests and communicates with the backend for model predictions.

#### Azure Cloud Deployment:

- Web Application Deployment: Host the Flask web application on Azure Web Services or a container service.

## 6. PROJECT PLANNING & SCHEDULING

### 6.1 Technical Architecture

#### Data Ingestion and Storage:

- Data Sources: Walmart's provided dataset containing store information and monthly sales data.
- Data Ingestion Component:
  - Use Azure Event Hubs for real-time data ingestion.
  - For batch processing, leverage Azure Data Factory to schedule and automate data ingestion tasks.
- Data Storage:
  - Utilize Azure Blob Storage for scalable and reliable storage of raw and processed data.
  - Implement Azure Synapse Analytics as a data warehouse for structured data.

#### Data Preprocessing and Feature Engineering:

- Preprocessing Component:
  - Leverage Azure Databricks for distributed data processing, cleaning, and normalization.
  - Use Azure Machine Learning services for data preprocessing pipelines.
- Feature Engineering Component:

- Implement custom Python scripts within Azure Databricks to derive relevant features from the raw data.
- Utilize Azure ML Feature Engineering capabilities for automated feature selection and extraction.

#### Holiday Impact Assessment and Weighting:

- Develop custom algorithms or scripts using Azure Functions or Azure Machine Learning pipelines to assess the impact of holidays and apply appropriate weighting.

#### Model Training and Evaluation:

- Model Training Component:
  - Utilize Azure Machine Learning for training machine learning models, with options for distributed training.
- Model Evaluation Component:
  - Implement custom evaluation scripts or use Azure Machine Learning capabilities for model performance assessment.

#### Flask Web Application:

- Develop a Flask-based web application hosted on Azure App Service for user interaction and model predictions.
- Utilize Azure Functions for serverless capabilities to handle specific tasks or endpoints.

#### Azure Deployment:

- Deploy the trained models using Azure Machine Learning models or Azure Kubernetes Service (AKS).
- Host the Flask application on Azure App Service for easy scaling and management.

## 6.2 Sprint Planning & Estimation

### Sprint 1: Getting Data Ready

- Task 1: Get the data ready for analysis. (Estimate: 5 days)
- Task 2: Check for any missing or incorrect information. (Estimate: 3 days)

### Sprint 2: Understanding Sales Trends

- Task 1: Look for patterns in sales data. (Estimate: 6 days)
- Task 2: See how holidays affect sales. (Estimate: 4 days)

### Sprint 3: Building Prediction Models

- Task 1: Use techniques to make predictions about future sales. (Estimate: 7 days)
- Task 2: Test and make sure the predictions are accurate. (Estimate: 5 days)

### Sprint 4: Creating a Simple Website

- Task 1: Make a simple website where you can enter information. (Estimate: 6 days)
- Task 2: Connect the website to the prediction models. (Estimate: 4 days)

#### Sprint 5: Making it User-Friendly

- Task 1: Improve the website's design and layout. (Estimate: 6 days)
- Task 2: Test the website with some people to make sure it's easy to use. (Estimate: 4 days)

#### Sprint 6: Getting Ready to Share

- Task 1: Make sure everything is secure and safe. (Estimate: 5 days)
- Task 2: Create a plan for what to do if something goes wrong. (Estimate: 3 days)

#### Sprint 7: Final Testing and Launch

- Task 1: Test everything one more time to be sure it's all working. (Estimate: 7 days)
- Task 2: Launch the system for everyone to use. (Estimate: 3 days)

#### 6.3 Sprint Delivery Schedule:

#### Sprint 1: Getting Data Ready

- Start Date: October 9, 2023
- End Date: October 11, 2023

#### Sprint 2: Understanding Sales Trends

- Start Date: October 12, 2023
- End Date: October 14, 2023

#### Sprint 3: Building Prediction Models

- Start Date: October 15, 2023
- End Date: October 17, 2023

#### Sprint 4: Creating a Simple Website

- Start Date: October 18, 2023
- End Date: October 20, 2023

#### Sprint 5: Making it User-Friendly

- Start Date: October 21, 2023
- End Date: October 23, 2023

#### Sprint 6: Getting Ready to Share

- Start Date: October 24, 2023
- End Date: October 26, 2023

#### Sprint 7: Final Testing and Launch

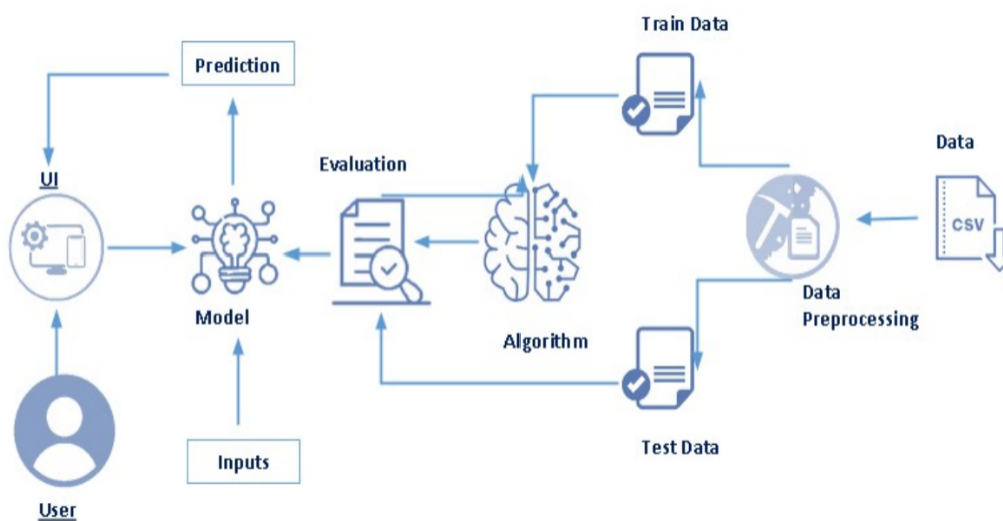
- Start Date: October 27, 2023
- End Date: October 29, 2023

#### Sprint 8: Helping and Improving

- Start Date: October 30, 2023
- End Date: November 9, 2023

### 7. CODING & SOLUTIONING:

#### Architecture:



#### Pre-requisites

To complete this project, knowledge of the following software, concepts and packages is required.

- Anaconda navigator

- Python packages
  - o Open anaconda prompt as administrator
  - o Type “pip install numpy” and click enter.
  - o Type “pip install pandas” and click enter.
  - o Type “pip install matplotlib” and click enter.
  - o Type “pip install seaborn” and click enter.
  - o Type “pip install pmdarima” and click enter.

### **Prior Knowledge**

You must have prior knowledge of following topics to complete this project.

- ML Concepts
  - o Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
  - o Unsupervised learning: <https://www.javatpoint.com/unsupervised-machine-learning>
  - o Regression and classification
  - o Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
  - o Xgboost: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
  - o Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>  
<https://towardsdatascience.com/what-are-rmse-and-mae-e405ce230383>

### **Project Objectives**

1. **Accurate Sales Forecasting:** Develop a robust model to predict Walmart store sales accurately, considering various factors such as historical sales data, seasonality, promotions, and external factors.
2. **Optimize Inventory Management:** Utilize sales forecasts to optimize inventory levels, ensuring that each store has the right amount of stock to meet demand without excess, minimizing both stockouts and overstock situations.
3. **Improve Demand Planning:** Enhance the ability to plan and allocate resources efficiently based on anticipated demand patterns, allowing for better workforce management and resource allocation.
4. **Enhance Decision-Making:** Provide actionable insights to decision-makers within Walmart by analyzing sales forecasts and identifying key drivers, enabling informed decision-making for promotions, pricing, and inventory management.

5. **Real-Time Monitoring:** Implement a system for real-time monitoring of sales performance against forecasts, allowing for quick adjustments and proactive management in response to unexpected changes in the market or internal factors.

### **Project Flow:**

#### **1.Data Collection:**

- Gather historical sales data from Walmart stores, including information on promotions, pricing, and other relevant factors.

- Collect external data such as economic indicators, weather patterns, and local events that may impact sales.

#### **2. Data Cleaning and Preprocessing:**

- Clean and preprocess the collected data to handle missing values, outliers, and ensure data consistency.

- Explore relationships between different variables to identify potential features for the forecasting model.

#### **3.Exploratory Data Analysis (EDA):**

- Conduct EDA to gain insights into the data distribution, trends, and correlations.

- Identify seasonality patterns, peak sales periods, and any other significant patterns that may influence sales.

#### **4.Feature Engineering:**

- Create new features or transform existing ones to improve the model's predictive power.

- Incorporate external factors such as holidays, economic indicators, and local events.

### **Data Collection**

ML depends heavily on data. It is most crucial aspect that makes algorithm training possible.

So this section allows you to download the required dataset.

#### **Download the dataset**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, Walmart-dataset.zip is used data which has features.csv, stores.csv, train.csv,

and test.csv. This data is downloaded from kaggle.com. Please refer the link given below to download the dataset.

### Visualizing and analysing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

There are numerous techniques for understanding the data. But here some of them are used. In addition, other and multiple techniques can also be used.

Importing the libraries

Import the necessary libraries as shown in the image.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
!pip install pmdarima
import pickle
```

### Read the Dataset

The dataset format might be in .csv, excel files, .txt, .json, etc. So, the dataset can be read with the help of pandas. In pandas we have a function called read\_csv() to read the dataset. As a parameter we have to give the directory of csv file.

All the datasets are read in the same way.

```
df_train=pd.read_csv("/content/train.csv")
df_features=pd.read_csv("/content/features.csv")
df_stores=pd.read_csv("/content/stores.csv")
df_test=pd.read_csv("/content/test.csv")
```

After reading the datasets, we will be viewing them by using the .head() function which will by default display first 5 rows of the dataset.

df\_train.head()

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

df\_features.head()



	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2 \
0	1	2010-02-05	42.31	2.572	NaN	NaN
1	1	2010-02-12	38.51	2.548	NaN	NaN
2	1	2010-02-19	39.93	2.514	NaN	NaN
3	1	2010-02-26	46.63	2.561	NaN	NaN
4	1	2010-03-05	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

```
df_stores.head()
```

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

```
df=df_train.merge(df_features,how='left',indicator=True).merge(df_stores,how='left')
```

```
df.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price \
0	1	1	2010-02-05	24924.50	False	42.31	2.572
1	1	1	2010-02-12	46039.49	True	38.51	2.548
2	1	1	2010-02-19	41595.55	False	39.93	2.514
3	1	1	2010-02-26	19403.54	False	46.63	2.561
4	1	1	2010-03-05	21827.90	False	46.50	2.625

	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI \
0	NaN	NaN	NaN	NaN	NaN	211.096358
1	NaN	NaN	NaN	NaN	NaN	211.242170
2	NaN	NaN	NaN	NaN	NaN	211.289143
3	NaN	NaN	NaN	NaN	NaN	211.319643
4	NaN	NaN	NaN	NaN	NaN	211.350143

	Unemployment	_merge	Type	Size
0	8.106	both	A	151315
1	8.106	both	A	151315
2	8.106	both	A	151315
3	8.106	both	A	151315
4	8.106	both	A	151315

Data Pre-processing

As we seen and understood the description of the data, lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so, the dataset has to be cleaned properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

#### Checking for null values

- To find the data type, .info() function is used.

**df.isnull().sum()**

```
Store      0
Dept       0
Date       0
Weekly_Sales    0
IsHoliday    0
Temperature    0
Fuel_Price    0
Markdown1    270889
Markdown2    310322
Markdown3    284479
Markdown4    286603
Markdown5    270138
CPI          0
Unemployment    0
_merge      0
Type        0
Size        0
dtype: int64
```

```
df2=df.drop(['Markdown1','Markdown2','Markdown3','Markdown4','Markdown5'],axis=1)
```

```
df2.isnull().sum()
```

```
Store      0
Dept       0
Date       0
Weekly_Sales  0
IsHoliday  0
Temperature 0
Fuel_Price 0
CPI        0
Unemployment 0
_merge     0
Type       0
Size       0
dtype: int64
```

```
df2[df2.Weekly_Sales<0]
```

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature \
846	1	6	2012-08-10	-139.65	False	85.05
2384	1	18	2012-05-04	-1.27	False	75.55
6048	1	47	2010-02-19	-863.00	False	39.93
6049	1	47	2010-03-12	-698.00	False	57.79
6051	1	47	2010-10-08	-58.00	False	63.93
...	...	...	...	...	...	...
419597	45	80	2010-02-12	-0.43	True	27.73
419598	45	80	2010-02-19	-0.27	False	31.27
419603	45	80	2010-04-16	-1.61	False	54.28
419614	45	80	2010-07-02	-0.27	False	76.61
419640	45	80	2011-02-11	-0.24	True	30.30

	Fuel_Price	CPI	Unemployment	_merge	Type	Size
846	3.494	221.958433	6.908	both	A	151315
2384	3.749	221.671800	7.143	both	A	151315
6048	2.514	211.289143	8.106	both	A	151315
6049	2.667	211.380643	8.106	both	A	151315
6051	2.633	211.746754	7.838	both	A	151315
...	...	...	...	...	...	...
419597	2.773	181.982317	8.992	both	B	118221
419598	2.745	182.034782	8.992	both	B	118221
419603	2.899	181.692477	8.899	both	B	118221
419614	2.815	182.318780	8.743	both	B	118221
419640	3.239	183.701613	8.549	both	B	118221

```
[1285 rows x 12 columns]
```

```
df3=pd.DataFrame(df2[df2.Weekly_Sales>0])
df3.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price \
0	1	1	2010-02-05	24924.50	False	42.31	2.572
1	1	1	2010-02-12	46039.49	True	38.51	2.548

2	1	1	2010-02-19	41595.55	False	39.93	2.514
3	1	1	2010-02-26	19403.54	False	46.63	2.561
4	1	1	2010-03-05	21827.90	False	46.50	2.625

	CPI	Unemployment	_merge	Type	Size
0	211.096358	8.106	both	A	151315
1	211.242170	8.106	both	A	151315
2	211.289143	8.106	both	A	151315
3	211.319643	8.106	both	A	151315
4	211.350143	8.106	both	A	151315

```
df3[df3.Weekly_Sales<0].sum()
```

<ipython-input-49-391912987fe8>:1: FutureWarning: The default value of numeric\_only in DataFrame.sum is deprecated. In a future version, it will default to False. In addition, specifying 'numeric\_only=None' is deprecated. Select only valid columns or specify the value of numeric\_only to silence this warning.

```
df3[df3.Weekly_Sales<0].sum()
```

```
Store      0.0
Dept       0.0
Date       0.0
Weekly_Sales  0.0
IsHoliday  0.0
Temperature 0.0
Fuel_Price  0.0
CPI        0.0
Unemployment 0.0
Type       0.0
Size       0.0
dtype: float64
```

### Exploratory Data Analysis

From the description of the stores dataset. We can observe that there are 3 types of stores:

Type A, Type B, and Type C. Since there are 45 stores in total, we will plot a pie-chart to see

the percentage of each store type.

```
df3["Type"].unique()
```

```
array(['A', 'B', 'C'], dtype=object)
```

```
df3["Type"].value_counts()
```

```
A    214961
B    162787
C     42464
Name: Type, dtype: int64
```

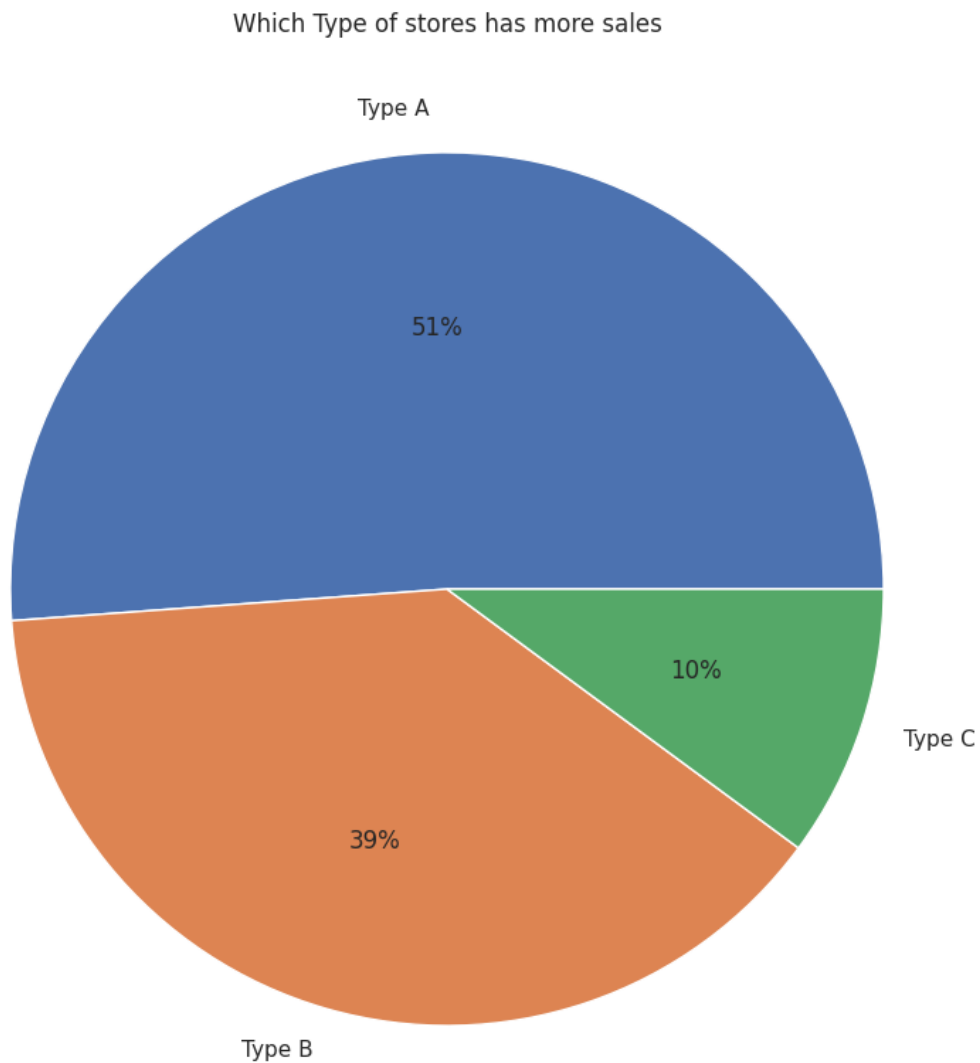
```

stores = ["Type A","Type B","Type C"]

data = df3["Type"].value_counts()

# Creating plot
fig, ax = plt.subplots()
plt.pie(data,labels=stores,autopct='%0.0f%%')
ax.set_title('Which Type of stores has more sales')
# show plot
plt.show()

```



```

df3['year'] = pd.DatetimeIndex(df3['Date']).year
df3['month'] = pd.DatetimeIndex(df3['Date']).month
df3['week'] = pd.DatetimeIndex(df3['Date']).week
df3.head()

```

```
<ipython-input-53-bac6e3b1d5c9>:3: FutureWarning: weekofyear and week have been deprecated, please
use DatetimeIndex.isocalendar().week instead, which returns a Series. To exactly reproduce the behavior
of week and weekofyear and return an Index, you may call pd.Int64Index(idx.isocalendar().week)
df3['week'] = pd.DatetimeIndex(df3['Date']).week
```

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price \
0	1	1	2010-02-05	24924.50	False	42.31	2.572
1	1	1	2010-02-12	46039.49	True	38.51	2.548
2	1	1	2010-02-19	41595.55	False	39.93	2.514
3	1	1	2010-02-26	19403.54	False	46.63	2.561
4	1	1	2010-03-05	21827.90	False	46.50	2.625

	CPI	Unemployment	_merge	Type	Size	year	month	week
0	211.096358	8.106	both	A	151315	2010	2	5
1	211.242170	8.106	both	A	151315	2010	2	6
2	211.289143	8.106	both	A	151315	2010	2	7
3	211.319643	8.106	both	A	151315	2010	2	8
4	211.350143	8.106	both	A	151315	2010	3	9

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

```
# import file with data
data = df3
```

```
# prints data that will be plotted
# columns shown here are selected by corr() since
# they are ideal for the plot
print(data.corr())
sns.set_theme(style="whitegrid")
# plotting correlation heatmap
dataplot = sns.heatmap(data.corr(), cmap="YlGnBu", annot=True)
sns.set(rc = {'figure.figsize':(40,12)})
```

```
# displaying heatmap
mp.show()
```

```
<ipython-input-54-e273f2bf13a4>:11: FutureWarning: The default value of numeric_only in
DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or
specify the value of numeric_only to silence this warning.
print(data.corr())
```

	Store	Dept	Weekly_Sales	IsHoliday	Temperature \
Store	1.000000	0.024258	-0.085117	-0.000522	-0.050230
Dept	0.024258	1.000000	0.148749	0.000663	0.004727
Weekly_Sales	-0.085117	0.148749	1.000000	0.012843	-0.002339
IsHoliday	-0.000522	0.000663	0.012843	1.000000	-0.155775
Temperature	-0.050230	0.004727	-0.002339	-0.155775	1.000000
Fuel_Price	0.065321	0.003544	0.000089	-0.078155	0.143700
CPI	-0.211261	-0.007178	-0.021162	-0.001933	0.182223
Unemployment	0.208759	0.007787	-0.025806	0.010555	0.096768

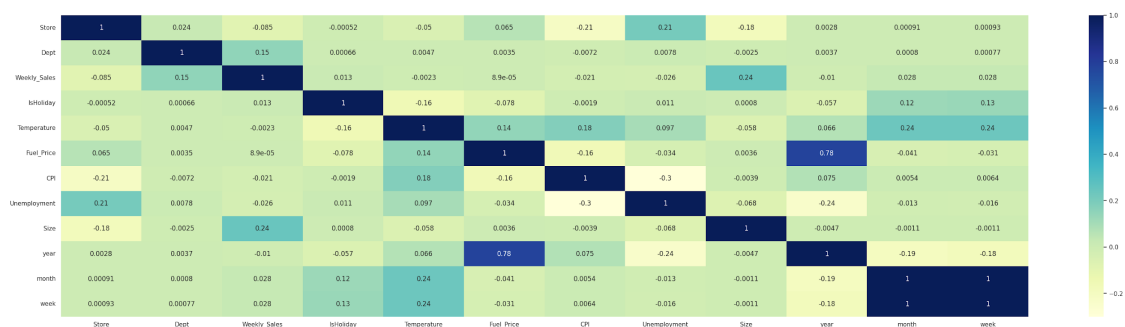
Size	-0.182763	-0.002491	0.244117	0.000797	-0.058413
year	0.002831	0.003716	-0.010015	-0.056572	0.065712
month	0.000907	0.000800	0.028401	0.123058	0.235957
week	0.000926	0.000767	0.027659	0.127846	0.236256

	Fuel_Price	CPI	Unemployment	Size	year \
Store	0.065321	-0.211261	0.208759	-0.182763	0.002831
Dept	0.003544	-0.007178	0.007787	-0.002491	0.003716
Weekly_Sales	0.000089	-0.021162	-0.025806	0.244117	-0.010015
IsHoliday	-0.078155	-0.001933	0.010555	0.000797	-0.056572
Temperature	0.143700	0.182223	0.096768	-0.058413	0.065712
Fuel_Price	1.000000	-0.164199	-0.033915	0.003632	0.779681
CPI	-0.164199	1.000000	-0.299887	-0.003903	0.074547
Unemployment	-0.033915	-0.299887	1.000000	-0.068335	-0.237210
Size	0.003632	-0.003903	-0.068335	1.000000	-0.004716
year	0.779681	0.074547	-0.237210	-0.004716	1.000000
month	-0.040931	0.005366	-0.012562	-0.001051	-0.194295
week	-0.031191	0.006428	-0.015614	-0.001130	-0.181804

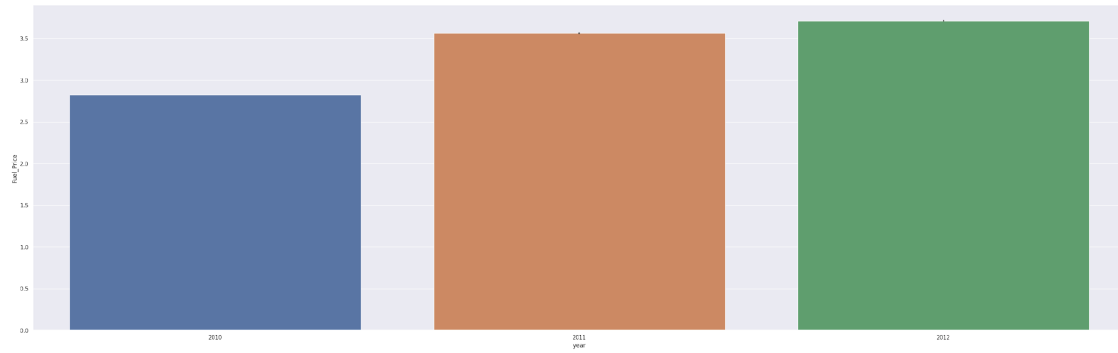
	month	week
Store	0.000907	0.000926
Dept	0.000800	0.000767
Weekly_Sales	0.028401	0.027659
IsHoliday	0.123058	0.127846
Temperature	0.235957	0.236256
Fuel_Price	-0.040931	-0.031191
CPI	0.005366	0.006428
Unemployment	-0.012562	-0.015614
Size	-0.001051	-0.001130
year	-0.194295	-0.181804
month	1.000000	0.996000
week	0.996000	1.000000

<ipython-input-54-e273f2bf13a4>:14: FutureWarning: The default value of numeric\_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric\_only to silence this warning.

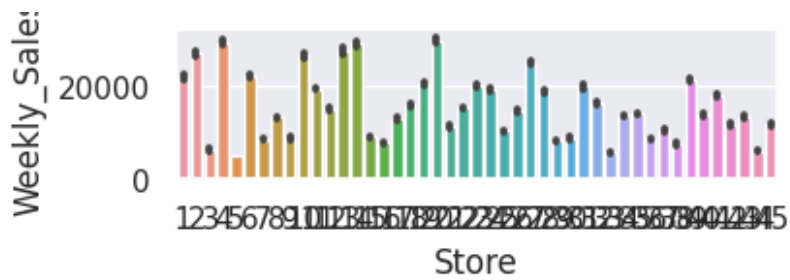
dataplot = sns.heatmap(data.corr(), cmap="YlGnBu", annot=True)



sns.barplot(x='year', y="Fuel\_Price", data=df3)  
sns.set(rc = {'figure.figsize':(4,1)})

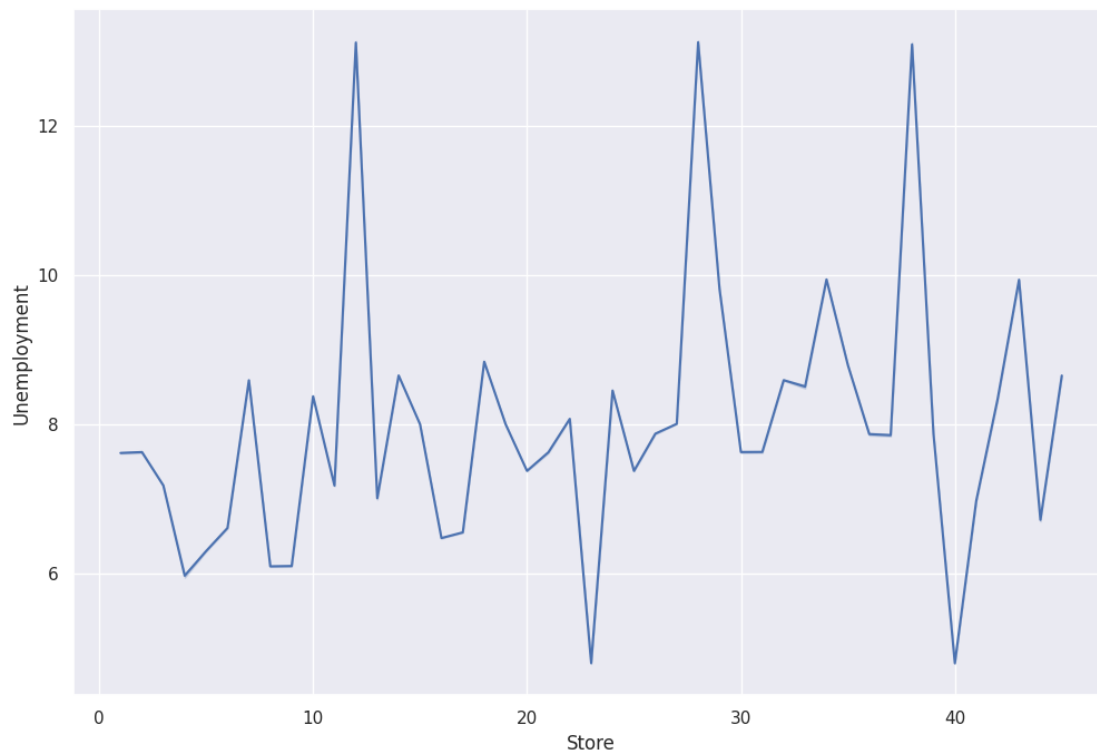


```
sns.barplot(x='Store', y="Weekly_Sales", data=df3)
sns.set(rc = {'figure.figsize':(12,8)})
```



```
sns.lineplot(x="Store",y="Unemployment",data=data)
```

```
<Axes: xlabel='Store', ylabel='Unemployment'>
```

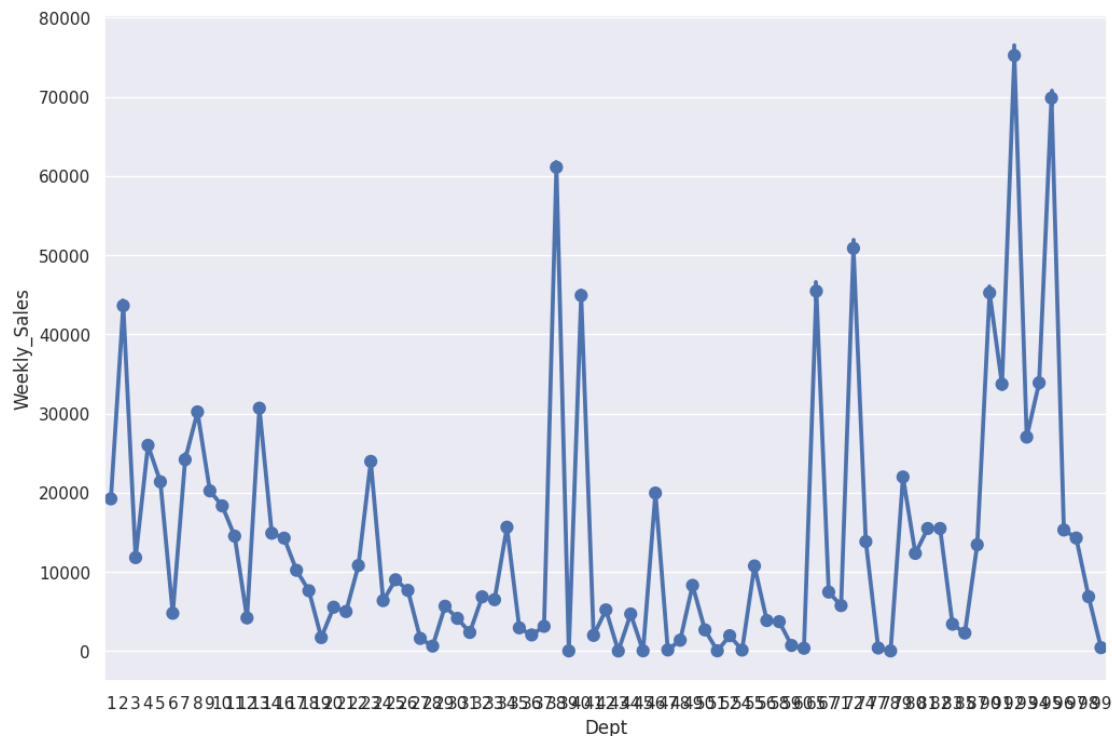




```
df3["Dept"].unique()
```

```
array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 40, 41, 42, 44, 45, 46, 47, 48, 49, 51, 52, 54, 55, 56,
       58, 59, 60, 67, 71, 72, 74, 77, 78, 79, 80, 81, 82, 83, 85, 87, 90,
       91, 92, 93, 94, 95, 96, 97, 98, 99, 39, 50, 43, 65])
```

```
sns.pointplot(x="Dept",y="Weekly_Sales",data=df3)
sns.set(rc = {'figure.figsize':(40,10)})
```



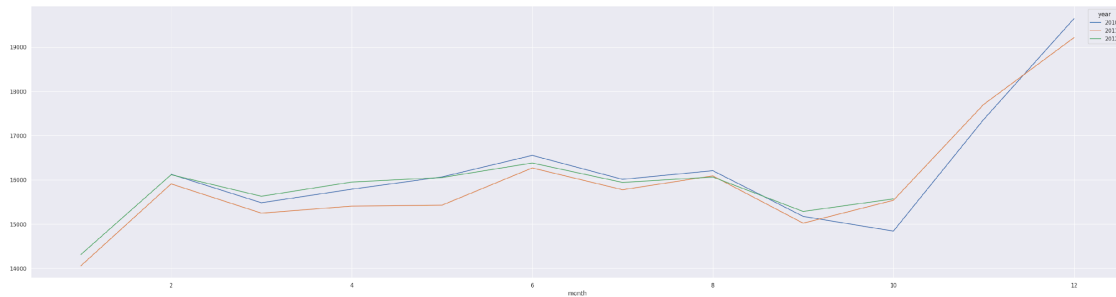
```
df4=df3.drop(columns=["Date"],axis=1)
df4.head()
```

	Store	Dept	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	CPI \
0	1	1	24924.50	False	42.31	2.572	211.096358
1	1	1	46039.49	True	38.51	2.548	211.242170
2	1	1	41595.55	False	39.93	2.514	211.289143
3	1	1	19403.54	False	46.63	2.561	211.319643
4	1	1	21827.90	False	46.50	2.625	211.350143

	Unemployment_merge	Type	Size	year	month	week
0	8.106	both	A 151315	2010	2	5
1	8.106	both	A 151315	2010	2	6
2	8.106	both	A 151315	2010	2	7
3	8.106	both	A 151315	2010	2	8
4	8.106	both	A 151315	2010	3	9

```
month_wise_sales = pd.pivot_table(df4, values = "Weekly_Sales", columns = "year", index = "month")
month_wise_sales.plot()
```

<Axes: xlabel='month'>



*# Import label encoder*

**from** sklearn **import** preprocessing

*# label\_encoder object knows how to understand word labels.*

```
le= preprocessing.LabelEncoder()
```

*# Encode labels in column 'species'.*

```
df4["IsHoliday"]= le.fit_transform(df4["IsHoliday"])
```

```
df4["Type"]= le.fit_transform(df4["Type"])
```

```
df4.isnull().sum()
```

```
Store      0
Dept       0
Weekly_Sales  0
IsHoliday  0
Temperature 0
Fuel_Price 0
CPI        0
Unemployment 0
_merge     0
Type       0
Size       0
year       0
month      0
week       0
dtype: int64
```

```
df4=df4.drop(columns=['_merge'],axis=1)
```

```
df4=df4.drop([309678])
```

```
df4
```

	Store	Dept	Weekly_Sales	IsHoliday	Temperature	Fuel_Price \
0	1	1	24924.50	0	42.31	2.572
1	1	1	46039.49	1	38.51	2.548
2	1	1	41595.55	0	39.93	2.514

3	1	1	19403.54	0	46.63	2.561
4	1	1	21827.90	0	46.50	2.625
...	...	...	...	...	...	...
421565	45	98	508.37	0	64.88	3.997
421566	45	98	628.10	0	64.89	3.985
421567	45	98	1061.02	0	54.47	4.000
421568	45	98	760.01	0	56.47	3.969
421569	45	98	1076.80	0	58.85	3.882

	CPI	Unemployment	Type	Size	year	month	week
0	211.096358	8.106	0	151315	2010	2	5
1	211.242170	8.106	0	151315	2010	2	6
2	211.289143	8.106	0	151315	2010	2	7
3	211.319643	8.106	0	151315	2010	2	8
4	211.350143	8.106	0	151315	2010	3	9
...	...	...	...	...	...	...	...
421565	192.013558	8.684	1	118221	2012	9	39
421566	192.170412	8.667	1	118221	2012	10	40
421567	192.327265	8.667	1	118221	2012	10	41
421568	192.330854	8.667	1	118221	2012	10	42
421569	192.308899	8.667	1	118221	2012	10	43

[420211 rows x 13 columns]

```
x=df4.drop(columns=["Weekly_Sales"],axis=1)
y=df4["Weekly_Sales"]
x.head()
```

	Store	Dept	IsHoliday	Temperature	Fuel_Price	CPI	Unemployment \
0	1	1	0	42.31	2.572	211.096358	8.106
1	1	1	1	38.51	2.548	211.242170	8.106
2	1	1	0	39.93	2.514	211.289143	8.106
3	1	1	0	46.63	2.561	211.319643	8.106
4	1	1	0	46.50	2.625	211.350143	8.106

	Type	Size	year	month	week
0	0	151315	2010	2	5
1	0	151315	2010	2	6
2	0	151315	2010	2	7
3	0	151315	2010	2	8
4	0	151315	2010	3	9

```
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, random_state=47)
```

## ARIMA

ARIMA is a forecasting technique that projects the future values of a series based entirely on its own inertia. The time series models assume that the given time-series data set is stationary meaning it has got the constant mean and constant variance. In order to apply ARIMA, we are required to calculate the triplet value (p,d,q). The value of p is number of Auto regressors, value of d is number of differences required to make series stationary and q is number of moving averages. In Auto-ARIMA, the triplet (p,q,d) values are automatically selected on the basis of least AIC and BIC scores which best fits the model.

```
import pmdarima
from pmdarima.arima import auto_arima

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt

DTRmodel = DecisionTreeRegressor(max_depth=3, random_state=0)
DTRmodel.fit(x_train, y_train)
y_pred = DTRmodel.predict(x_test)

print("R2 score :", r2_score(y_test, y_pred))
print("MSE score :", mean_squared_error(y_test, y_pred))
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

R2 score : 0.3775568770578843
MSE score : 318683407.9587218
RMSE: 17851.706023759238
```

```
rf1 = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1, max_depth=20,
                           max_features='sqrt', min_samples_split=3)
rf1.fit(x_train, y_train)
y_pred1 = rf1.predict(x_test)

print("R2 score :", r2_score(y_test, y_pred))
print("MSE score :", mean_squared_error(y_test, y_pred1))
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred1)))

R2 score : 0.3775568770578843
MSE score : 106584538.10076953
RMSE: 10323.978792150318
```

## **XgBoost**

XGBRegressor algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable.

```
from xgboost import XGBRegressor
model = XGBRegressor(objective='reg:squarederror', nthread=4, n_estimators=500, max_depth=4,
```

```
learning_rate= 0.5)
model.fit(x_train,y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.5, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=500, n_jobs=None, nthread=4,
             num_parallel_tree=None, ...)
```

```
y_pred2 = model.predict(x_test)
```

```
print("R2 score :",r2_score(y_test, y_pred2))
print("MSE score :",mean_squared_error(y_test, y_pred2))
print("RMSE: ",sqrt(mean_squared_error(y_test, y_pred2)))
```

```
R2 score : 0.9482143725653065
MSE score : 26513619.680080418
RMSE: 5149.137760837286
```

```
y_pred2
```

```
array([14768.21 , 362.50912, 18262.584 , ..., 1361.111 ,
       -383.30942, 2940.908 ], dtype=float32)
```

```
from sklearn.linear_model import Ridge
```

```
rr_model = Ridge(alpha=0.5)
rr_model.fit(x_train,y_train)
```

```
Ridge(alpha=0.5)
```

```
y_pred3 = model.predict(x_test)
```

```
42print("R2 score :",r2_score(y_test, y_pred3))
print("MSE score :",mean_squared_error(y_test, y_pred3))
print("RMSE: ",sqrt(mean_squared_error(y_test, y_pred3)))
```

```
R2 score : 0.9482143725653065
MSE score : 26513619.680080418
RMSE: 5149.137760837286
```

```
y_test
```

```
416618 13480.30
289535  44.19
267082 16511.71
277358 28412.57
102078  9113.78
```

```

...
132922    618.00
392742   15343.43
384522    856.38
341550    294.02
374273   2625.17
Name: Weekly_Sales, Length: 84043, dtype: float64

```

### Comparing the models

For comparing the models PrettyTable function is defined.

```

from sklearn.model_selection import cross_val_score
rf = RandomForestRegressor(n_estimators=58, max_depth=27,
min_samples_split=3,min_samples_leaf=1)
rf.fit(x_train, y_train.ravel())
y_pred =rf.predict(x_test)

from sklearn.model_selection import cross_val_score
xg_reg =XGBRegressor(objective="reg:squarederror", nthread= 4, n_estimators= 500, max_depth= 4,
learning_rate= 0.5)
xg_reg.fit(x_train, y_train)
pred=xg_reg.predict(x_train)
y_pred=xg_reg.predict(x_test)

cv=cross_val_score(xg_reg,x,y,cv=10)
cv

array([0.70955904, 0.57916837, 0.54206571, 0.75134278, 0.84435843,
       0.7636532 , 0.82873503, 0.80243917, 0.71913666, 0.8266943 ])

np.mean(cv)

0.7367152697407299

pickle.dump(rf,open('final_model.pkl','wb'))

import sklearn
print(sklearn.__version__)

1.2.2

pred1=xg_reg.predict(x_train)
pred2=xg_reg.predict(x_test)

```

### Application Building

In this section, we will be building a web application that is integrated to the model we built.

A UI is provided for the uses where they have to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

Building serverside script

Building HTML Pages:

### Building HTML Pages:


For this project, an HTML file called 'index.html' is created and saved in templates folder.

The index.html page looks like:

# Sales Prediction

Store:

Department:

Date:  

Is Holiday

Size:

Temperature:

Fule Prices:

CPI:

unemployment:

Type:

### Building the python code in flask:

Import the libraries:

```
from flask import Flask,render_template,request
app = Flask(__name__)
import pickle
import numpy as np
from datetime import datetime
import pandas as pd
```

Then load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (\_\_name\_\_) as argument.

```
model = pickle.load(open('final_model.pkl','rb'))
@app.route('/')
def start():
    return render_template('index.html')
```

Render the HTML page:

```
@app.route('/predict',methods=['POST'])
```

Here we will be using declared constructor to route to the HTML page which we have created earlier and stores in the templates folder.

In the above example, '/' URL is bound with index.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:



```
@app.route('/predict',methods =['POST'])
```

```
def predict():  
    s=request.form["store"]  
    d=request.form["dept"]  
    d1=request.form["date"]
```

```
date_obj = datetime.strptime(d1, '%Y-%m-%d')  
year = date_obj.year  
month = date_obj.month  
day = date_obj.day  
s1=request.form["size"]  
h=request.form["h"]  
if (h=="yes"):  
    h=1  
else:  
    h=0  
f=request.form["fuel"]  
c=request.form["CPI"]  
t1=request.form["temp"]  
u=request.form["unemployment"]  
t=request.form["type"]
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the y\_pred() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
if (t=="A"):  
    t=0  
else:  
    t=1  
f=[[float(s),float(d),float(s1),float(h),float(t1),float(year), float(month), float(day)  
    ,float(c),float(u),float(t),float(f)]]  
  
prediction = model.predict(f)  
print(prediction)  
  
return render_template("index.html",y="The predicted profit is "+str(np.round(prediction[0])))  
  
if __name__ == '__main__':  
    app.run(debug=True)
```

### Activity 3: Run the application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

## 8. PERFORMANCE TESTING:

### 8.1 Performance Metrics:

S.No.	Parameter	Values	Screenshot
1.	Model Summary	<p>Objective: The primary objective of this project is to develop a predictive model for accurate sales forecasting across various product categories in different Walmart stores. This will facilitate optimized inventory management, efficient promotion planning, and overall revenue enhancement.</p> <p>Dataset: The dataset provided by Walmart encompasses historical sales data, encompassing product details, store locations, promotional events, and relevant features.</p> <p>Data Preprocessing:</p> <p>Data Cleaning: We commenced by addressing missing values, outliers, and data inconsistencies for a cleaner dataset.</p> <p>Feature Engineering: New features were created, including seasonality, holiday indicators, and trend components, to augment the predictive capacity of the model.</p>	

	<p>Encoding: Categorical variables were encoded into numerical representations, employing one-hot encoding and label encoding techniques.</p> <p>Scaling: Numerical features were standardized to ensure uniform scaling.</p> <p>Exploratory Data Analysis (EDA): In-depth analysis of the dataset provided valuable insights into sales trends, correlations, and patterns, steering our feature selection and engineering process.</p> <p>Model Selection: We experimented with several regression models:</p> <p>Linear Regression: Utilized as a baseline model for establishing initial performance benchmarks.</p> <p>Decision Trees and Random Forests: Employed to capture non-linear relationships between features and sales, potentially revealing complex patterns.</p> <p>Gradient Boosting (e.g., XGBoost, LightGBM): Ensemble methods chosen for their ability to often yield high predictive accuracy.</p> <p>Model Training and Validation:</p> <p>Train-Test Split: The dataset was partitioned into training and testing sets to facilitate model evaluation.</p> <p>Cross-Validation: Implemented k-fold cross-validation to ascertain the model's generalization capability.</p> <p>Hyperparameter Tuning: Hyperparameters were fine-tuned using grid search and random search methods, optimizing model performance.</p> <p>Evaluation Metric: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (<math>R^2</math>) were employed to quantitatively measure the model's accuracy in predicting sales.</p> <p>Final Model Deployment: The trained model can be deployed using frameworks like Flask.</p>	
--	--	--

2.	Accuracy	<b>Regression Model:</b> R2 score : 0.9482143725653065 MSE score : 26513619.680080418 RMSE: 5149.137760837286-	<pre>print("R2 score :",r2_score(y_test, y_pred3)) print("MSE score :",mean_squared_error(y_test, y_pred3)) print("RMSE: ",sqrt(mean_squared_error(y_test, y_pred3)))</pre> <pre>R2 score : 0.9482143725653065 MSE score : 26513619.680080418 RMSE: 5149.137760837286</pre>
----	----------	---	---

## 9. ADVANTAGES & DISADVANTAGES

### Advantages:

- **Improved Business Decision Making:** Accurate sales forecasts enable Walmart to make informed decisions regarding inventory management, staffing levels, and marketing strategies.
- **Enhanced Customer Experience:** By optimizing inventory levels, Walmart can ensure that products are consistently available to customers, leading to higher customer satisfaction.
- **Optimized Promotional Events:** The system's ability to analyze the impact of holidays on sales allows Walmart to plan and execute promotional events more effectively, maximizing revenue during peak periods.
- **Data-Driven Insights:** The system provides valuable insights into sales trends, allowing Walmart to adapt quickly to changing market conditions and customer preferences.
- **Efficient Resource Allocation:** With accurate sales forecasts, Walmart can allocate resources more efficiently, optimizing staffing levels, inventory stocking, and marketing efforts.
- **Improved Profitability:** By minimizing overstocking and understocking, Walmart can reduce carrying costs and increase profitability.
- **Competitive Advantage:** A sophisticated sales forecasting system can give Walmart a competitive edge by allowing them to respond more quickly and effectively to market dynamics.

### Disadvantages:

- **Initial Implementation Complexity:** Setting up the system requires significant initial effort, including data integration, model training, and application development.
- **Data Quality Dependency:** The accuracy of sales forecasts is heavily reliant on the quality and completeness of the underlying data. Inaccurate or incomplete data can lead to less reliable predictions.
- **Model Training and Maintenance:** Machine learning models require ongoing monitoring and periodic retraining to remain accurate and relevant. This can be resource-intensive.
- **Cost of Technology Integration:** Implementing and maintaining the technology stack, including cloud services and machine learning tools, may involve ongoing costs.
- **Sensitivity to External Factors:** While the system accounts for holidays, it may still be sensitive to unexpected external factors (e.g., economic events, natural disasters) that can impact sales.

- User Adoption and Training: Walmart employees may need training to effectively use and interpret the forecasts generated by the system.
- Security and Privacy Concerns: Storing and processing sensitive sales data requires robust security measures to protect against unauthorized access or data breaches.

## 10. CONCLUSION

In conclusion, the proposed sales forecasting system for Walmart presents a valuable opportunity to enhance business operations and decision-making processes. By leveraging advanced analytics, machine learning algorithms, and a user-friendly web application, Walmart stands to benefit from improved inventory management, optimized promotional events, and data-driven insights into sales trends. This system empowers Walmart to adapt swiftly to changing market conditions, ultimately leading to increased customer satisfaction and higher profitability.

While the implementation of the system involves initial complexities and ongoing maintenance efforts, the long-term advantages far outweigh these challenges. The potential for more efficient resource allocation, reduced carrying costs, and a competitive edge in the retail industry make this initiative a worthwhile investment.

To ensure the success of the project, it is essential to prioritize data quality and security, as well as provide adequate training and support to Walmart employees using the system. Regular monitoring and periodic model retraining will also be crucial to maintain the accuracy and relevance of sales forecasts.

Overall, the sales forecasting system has the potential to revolutionize Walmart's retail operations, driving better outcomes for both the company and its customers. With careful planning and execution, this initiative can contribute significantly to Walmart's continued success in the competitive retail market.

## 11. APPENDIX

### 11.1 Source code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

!pip install pmdarima

import pickle


df_train=pd.read_csv("/content/train.csv")

df_features=pd.read_csv("/content/features.csv")

df_stores=pd.read_csv("/content/stores.csv")

df_test=pd.read_csv("/content/test.csv")


df_train.head()


df_features.head()


df_stores.head()


df=df_train.merge(df_features,how='left',indicator=True).merge(df_stores,how='left')
```

```
df.head()
```

```
df.isnull().sum()
```

```
df2=df.drop(['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5'],axis=1)
```

```
df2.isnull().sum()
```

```
df2[df2.Weekly_Sales<0]
```

```
df3=pd.DataFrame(df2[df2.Weekly_Sales>0])
```

```
df3.head()
```

```
df3[df3.Weekly_Sales<0].sum()
```

```
df3['Type'].unique()
```

```
df3['Type'].value_counts()
```

```
stores = ['Type A','Type B']

data = df3['Type'].value_counts()

# Creating plot

fig, ax = plt.subplots()

plt.pie(data,labels=stores,autopct='%.0f%%')

ax.set_title('Which Type of stores has more sales')

# show plot

plt.show()


df3['year'] = pd.DatetimeIndex(df3['Date']).year

df3['month'] = pd.DatetimeIndex(df3['Date']).month

df3['week'] = pd.DatetimeIndex(df3['Date']).week

df3.head()


import matplotlib.pyplot as mp

import pandas as pd

import seaborn as sns
```



```
# import file with data

data = df3

# prints data that will be plotted

# columns shown here are selected by corr() since

# they are ideal for the plot

print(data.corr())

sns.set_theme(style="whitegrid")

# plotting correlation heatmap

dataplot = sns.heatmap(data.corr(), cmap="YlGnBu", annot=True)

sns.set(rc = {'figure.figsize':(40,12)})

# displaying heatmap

mp.show()

sns.barplot(x='year', y="Fuel_Price", data=df3)

sns.set(rc = {'figure.figsize':(4,1)})

sns.barplot(x='Store', y="Weekly_Sales", data=df3)

sns.set(rc = {'figure.figsize':(12,8)})
```

```
sns.lineplot(x="Store",y="Unemployment",data=data)
```

```
df3['Dept'].unique()
```

```
sns.pointplot(x="Dept",y="Weekly_Sales",data=df3)
```

```
sns.set(rc = {'figure.figsize':(40,10)})
```

```
df4=df3.drop(columns=["Date"],axis=1)
```

```
df4.head()
```

```
month_wise_sales = pd.pivot_table(df4, values = "Weekly_Sales", columns = "year", index = "month")
```

```
month_wise_sales.plot()
```

```
# Import label encoder
```

```
from sklearn import preprocessing
```

```
# label_encoder object knows how to understand word labels.
```

```
le= preprocessing.LabelEncoder()
```

```
# Encode labels in column 'species'.

df4['IsHoliday']= le.fit_transform(df4['IsHoliday'])

df4['Type']= le.fit_transform(df4['Type'])


df4.isnull().sum()


df4=df4.drop(columns=['_merge'],axis=1)


# df4=df4.drop([309678])

# df4


x=df4.drop(columns=['Weekly_Sales'],axis=1)

y=df4['Weekly_Sales']

x.head()


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, random_state=47)


import pmdarima

from pmdarima.arima import auto_arima
```

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import r2_score, mean_squared_error

from math import sqrt


DTRmodel = DecisionTreeRegressor(max_depth=3, random_state=0)

DTRmodel.fit(x_train, y_train)

y_pred = DTRmodel.predict(x_test)


print("R2 score :", r2_score(y_test, y_pred))

print("MSE score :", mean_squared_error(y_test, y_pred))

print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))


rf1 = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1, max_depth=20,

                           max_features='sqrt', min_samples_split=3)

rf1.fit(x_train, y_train)

y_pred1 = rf1.predict(x_test)


print("R2 score :", r2_score(y_test, y_pred))
```

```
print("MSE score :",mean_squared_error(y_test, y_pred1))

print("RMSE: ",sqrt(mean_squared_error(y_test, y_pred1)))


from xgboost import XGBRegressor

model = XGBRegressor(objective='reg:squarederror', nthread= 4, n_estimators= 500, max_depth= 4,
learning_rate= 0.5)

model.fit(x_train,y_train)


y_pred2 = model.predict(x_test)


print("R2 score :",r2_score(y_test, y_pred2))

print("MSE score :",mean_squared_error(y_test, y_pred2))

print("RMSE: ",sqrt(mean_squared_error(y_test, y_pred2)))


y_pred2


from sklearn.linear_model import Ridge

rr_model = Ridge(alpha=0.5)

rr_model.fit(x_train,y_train)


y_pred3 = model.predict(x_test)
```

```
print("R2 score :",r2_score(y_test, y_pred3))

print("MSE score :",mean_squared_error(y_test, y_pred3))

print("RMSE: ",sqrt(mean_squared_error(y_test, y_pred3)))
```

```
y_test
```

```
from sklearn.model_selection import cross_val_score
```

```
rf = RandomForestRegressor(n_estimators=58, max_depth=27,
min_samples_split=3,min_samples_leaf=1)
```

```
rf.fit(x_train, y_train.ravel())
```

```
y_pred =rf.predict(x_test)
```

```
from sklearn.model_selection import cross_val_score
```

```
xg_reg =XGBRegressor(objective="reg:squarederror", nthread= 4, n_estimators= 500, max_depth= 4,
learning_rate= 0.5)
```

```
xg_reg.fit(x_train, y_train)
```

```
pred=xg_reg.predict(x_train)
```

```
y_pred=xg_reg.predict(x_test)
```

```
cv=cross_val_score(xg_reg,x,y,cv=10)
```

```
cv

np.mean(cv)

pickle.dump(rf,open('final_model.pkl','wb'))

import sklearn

print(sklearn.__version__)

pred1=xg_reg.predict(x_train)

pred2=xg_reg.predict(x_test)

print(pred1,pred2)
```

## 11.2 GitHub & Project Demo Link

GitHub:

Project Demo Link: <https://projectdemo.z30.web.core.windows.net/>

