WALMART SALES FORECASTING

INTRODUCTION:

The primary objective of this project is to develop an accurate and reliable sales forecasting model for Walmart, a leading retail giant, to optimize inventory management, improve supply chain efficiency, and enhance overall business operations.Forecasting is the process of estimation of quantity, type and quality of future work e.g. sales. . In this paper, a comparative analysis of some of the Supervised Machine Learning Techniques have been done such as 1. LightGBM, 2. Linear Regression, 3. Extra Tree Method, 4. AdaBoost Classifier, 5. Arima, 6. Random Forest, 7. K nearest neighbors, 8. XGBoost, etc.to build a prediction model and precisely estimate possible sales of 45 retail outlets of Walmart store which are at different geographical locations. Walmart is one of the foremost stores across the world and thus authors would like to predict the sales accurately. Certain events and holidays affect the sales periodically, which sometimes can also be on a daily basis. The forecast of probable sales is based on a combination of features such as previous sales data, promotional events, holiday week, temperature, fuel price, CPI i.e., Consumer Price Index and Unemployment rate in the state. The data is collected from 45 outlets of Walmart and the prediction about the sales of Walmart was done using various Supervised Machine Learning Techniques. The contribution of this paper is to help the business owners decide which approach to follow while trying to predict the sales of their Supermarket taken into account different scenarios including temperature, holidays, fuel price, etc. This will help them in deciding the promotional and marketing strategy for their products. The primary purpose of Walmart's sales forecasting is to optimize inventory management, streamline supply chain operations, implement effective pricing strategies, allocate resources efficiently, and enhance customer satisfaction. It serves as a vital tool for informed decision-making, waste reduction, and maintaining a competitive edge in the dynamic retail industry.

LITERATURE SURVEY:

Existing problem:

Existing problems in Walmart's sales forecasting include data quality issues, the complexity of seasonality and trends, challenges in incorporating external factors, managing a diverse product assortment, adapting to location-specific variations, modeling promotions and pricing impacts, and the need for accurate forecasts in a rapidly changing market. Addressing these challenges requires a multi-faceted approach involving data analytics, technology, and domain expertise.

References:

1. "Walmart Sales Forecasting using XGBoost algorithm and Feature engineering," 2020 International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE), Bangkok, Thailand, 2020, pp. 458-461, doi: 10.1109/ICBASE51474.2020.00103.

2. Raizada, Stuti, and Jatinderkumar R. Saini. "Comparative Analysis of Supervised Machine Learning Techniques for Sales Forecasting." International Journal of Advanced Computer Science and Applications 12.11 (2021).

3. Shilong, Zhang. "Machine learning model for sales forecasting by using XGBoost." 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE). IEEE, 2021.

4. Deepa, K., and G. Raghuram. "Sales Forecasting Using Machine Learning Models." Annals of the Romanian Society for Cell Biology (2021): 3928-3936.

5. Akande, Yetunde Faith, et al. "Application of XGBoost Algorithm for Sales Forecasting Using Walmart Dataset." International Conference on Advances in Electrical and Computer Technologies. Singapore: Springer Nature Singapore, 2021.

6. Yi, Siming. "Walmart Sales Prediction Based on Machine Learning." Highlights in Science, Engineering and Technology 47 (2023): 87-94.

7. Behera, Gopal, Ashutosh Bhoi, and Ashok Kumar Bhoi. "A Comparative Analysis of Weekly Sales Forecasting Using Regression Techniques." Intelligent Systems: Proceedings of ICMIB 2021. Singapore: Springer Nature Singapore, 2022. 31-43.

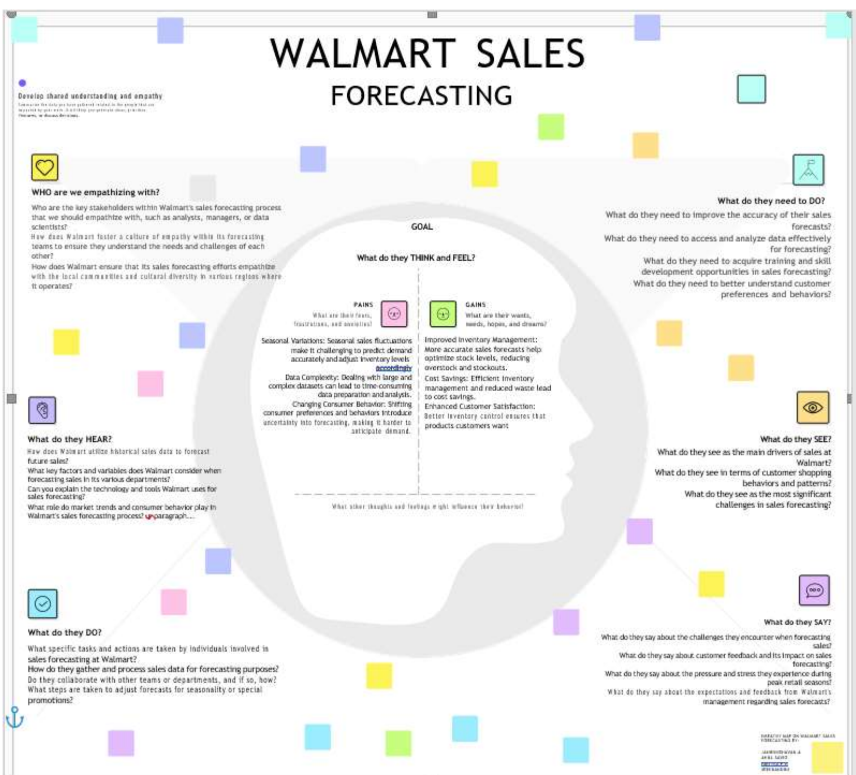
Problem Statement Definition:

The problem statement for Walmart sales forecasting is to accurately predict sales despite data complexity, seasonality, location variability, and the influence of promotions and pricing. The objective is to develop a robust forecasting system to optimize inventory, supply chain, and pricing strategies for improved efficiency and customer satisfaction.

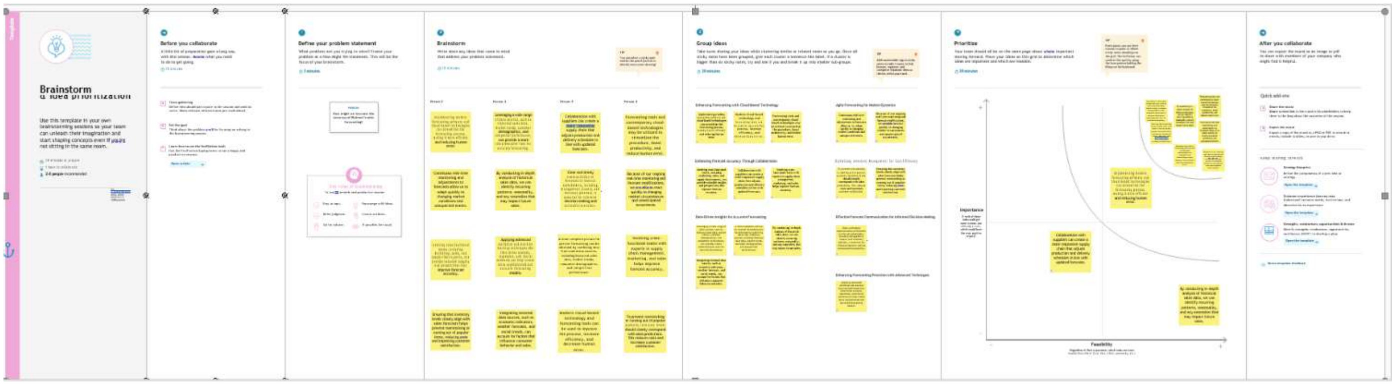
Walmart's sales forecasting system faces challenges such as inaccurate forecasts, data complexity, regional variations, and difficulties in managing promotions and inventory efficiently. They need a user-friendly, accuratesystem that accommodates these complexitiesand provides timely insights.

IDEATION & PROPOSED SOLUTION:

Empathy Map Canvas:



Ideation & Brainstorming:



REQUIREMENT ANALYSIS:

Functional requirement:

Functional requirements for a Walmart sales forecasting system would describe the specific features and capabilities that the system needs to have in order to meet the needs of the business. These requirements are typically written in a clear and detailed manner to guide the development of the system. Here are some functional requirements for a Walmart sales forecasting system:

Data Integration:

> The system should be able to integrate data from various sources, including historical sales data, inventory levels, seasonal trends, and external factors like weather, holidays, and economic indicators.

Forecasting Models:

> The system should support multiple forecasting models, such as time series analysis, regression analysis, and machine learning algorithms, to provide accurate sales forecasts.

historical Data Analysis:

> The system should allow users to analyze historical sales data to identify patterns, trends, and seasonality.

Demand Segmentation:

> The system should be capable of segmenting demand based on different criteria, such as product categories, store locations, and customer demographics.

Scalability:

> The system should be scalable to handle a large volume of data and a growing number of products and stores.

Real-time Data Updates:

The system should provide real-time updates of sales data, allowing for adjustments in forecasts based on changing conditions.

Scenario Analysis:

Users should be able to conduct "what-if" scenario analysis to understand the impact of different variables on sales forecasts.

Non-Functional requirements:

Non-functional requirements for a Walmart sales forecasting system focus on qualities, constraints, and characteristics of the system that are not directly related to its specific features or functionalities. These requirements are essential for ensuring the system's overall performance, reliability, and user experience. Here are some non-functional requirements for a Walmart sales forecasting system:

Performance: Response Time: The system should provide quick responses to user queries and requests, ensuring that forecasts and reports are generated in a timely manner.

Scalability:

The system should be able to handle an increasing volume of data and users without a significant decrease in performance.

Availability:

The system should be available 24/7 with minimal downtime for maintenance or updates.

Reliability:

The system should be highly reliable, with a low likelihood of errors or failures in forecasting and data processing.

Data Security:

Ensure the protection of sensitive sales and customer data, including encryption, access control, and compliance with data privacy regulations.

Authentication and Authorization:

Implement robust authentication mechanisms to verify the identity of users and restrict access to authorized personnel.

Compliance:

The system should comply with relevant industry standards and legal requirements, especially in terms of data privacy and security.

Usability:

The system should have an intuitive and user-friendly interface to ensure that users can easily interact with the forecasting tools and reports.

Scalability:

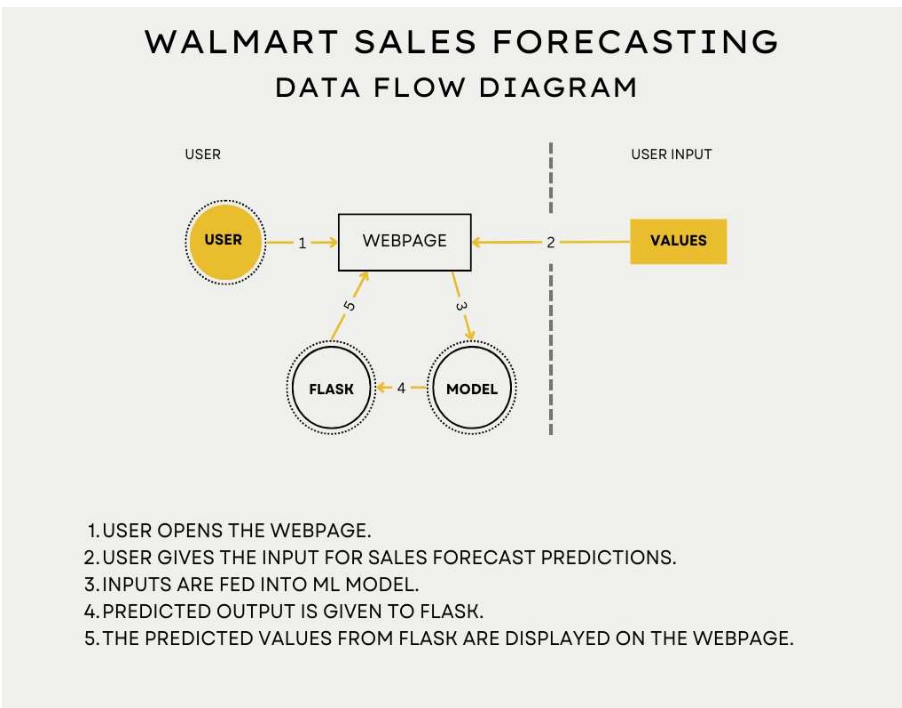
The system should be designed to handle increasing data volumes and user loads, ensuring that it can grow with the business.

Interoperability:

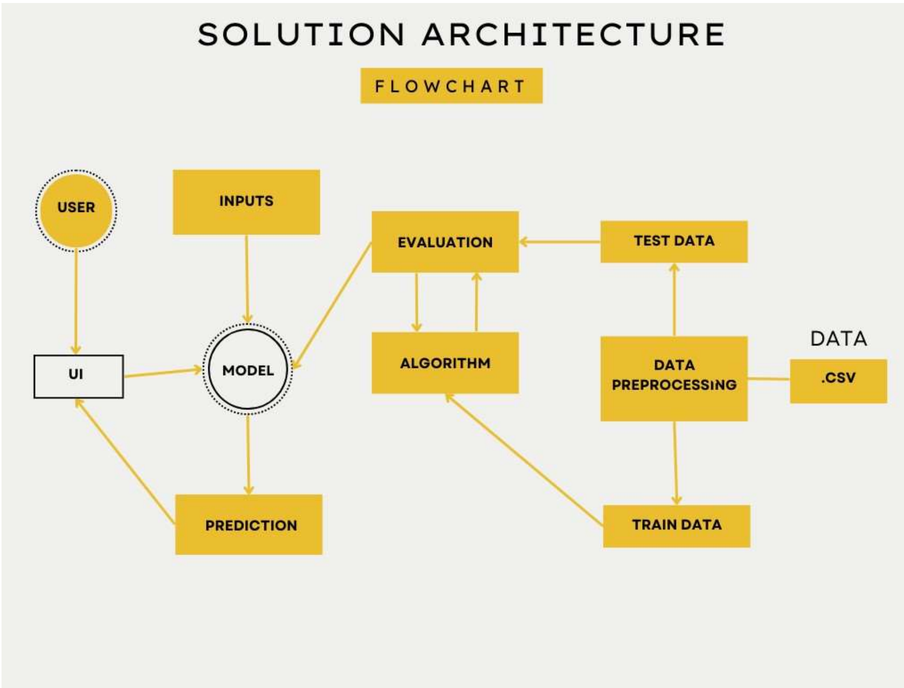
The system should be able to integrate with other systems and data sources within the Walmart ecosystem, such as inventory management and point-of-sale systems.

PROJECT DESIGN:

Data Flow Diagrams & User Stories:



Solution Architecture:

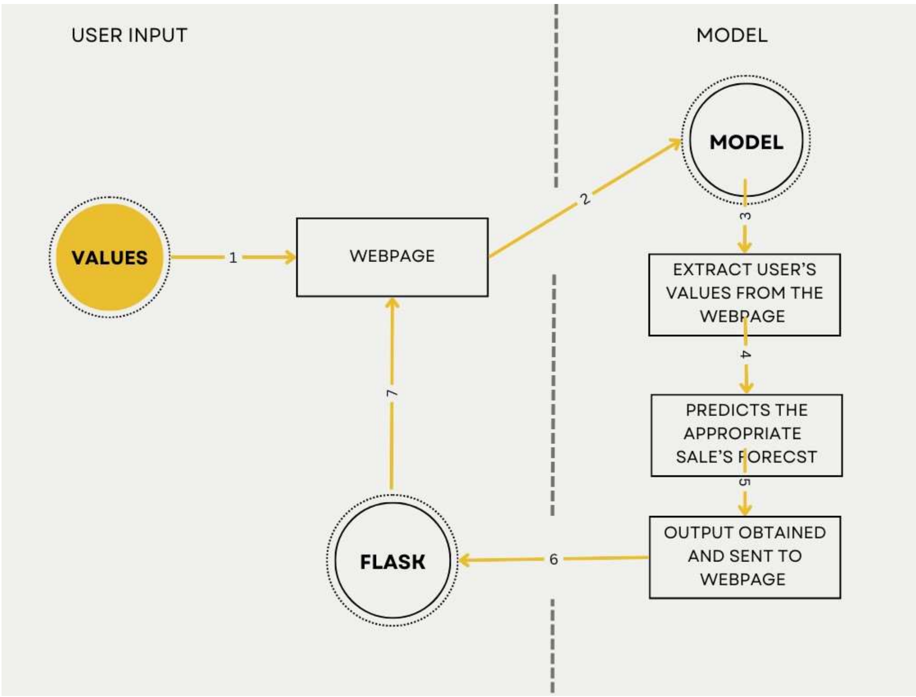


The structure of the software is as follows:

- **User Inputs:** This component provides a user interface for the user to input data intothe system.
- **Data Preprocessing:** This component cleans and prepares the data for training and evaluation. This may include tasks such as removing outliers, scaling the data, and converting the data to a format that is compatible with the chosen machine learning algorithm.
- **Model:** This component represents the machine learning model. The model is trained on a set of data and then used to make predictions on new data. Evaluation: This component evaluates the performance of the model on a held-out test set. This helps to ensure that the model is able to generalize to new data.
- **Prediction:** This component uses the trained model to make predictions on new data

PROJECT PLANNING & SCHEDULING:

Technical Architecture:



1. User provides input to the webpage.
2. The webpage is linked with the trained model.
3. These inputs are extracted and feeded into the trained model.
4. Model predicts the appropriate forecast.
5. Now the outputs are obtained and sent to the webpage again.
6. Here Flask used to connect the trained model and the webpage.
7. Final obtained result is displayed in the webpage.

Sprint Planning & Estimation:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Dashboard	USN-1	As a user, I have the capability to furnish the following information: Store, Size, Department, Temperature, Date, and IsHoliday.	I can access my dashboard	High	Sprint-1
		USN-2	As a user, I can ask the website to predict the result using various input values.	I can access my dashboard	High	Sprint-1
		USN-3	As a user, I can receive the predicted result after giving inputs and clicking predict in the webpage.	I can access my predicted result in dashboard	High	Sprint-2
Administrator						

Sprint Delivery Schedule:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date	Sprint Release Date (Actual)
Sprint-1	20	14 Days	15 Oct 2023	28 Oct 2023	20	28 Oct 2023
Sprint-2	20	12 Days	29 Oct 2023	09 Nov 2023	20	09 Nov 2023

CODING & SOLUTIONING:

Importing Libraries:

Import the necessary libraries as shown below

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
import pickle
from os import path
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasRegressor
```

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 used in the same way.

```
data = pd.read_csv('train.csv')
```

```
stores = pd.read_csv('stores.csv')

features = pd.read_csv('features.csv')
```

After reading the datasets we will be viewing them

Training Dataset

```
data.shape

(421570, 5)

data.tail()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday
421565	45	93	10/26/2012	2487.80	False
421566	45	94	10/26/2012	5203.31	False
421567	45	95	10/26/2012	56017.47	False
421568	45	97	10/26/2012	6817.48	False
421569	45	98	10/26/2012	1076.80	False

```
data.info()

<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  object
```

```
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     421570 non-null  bool
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 13.3+ MB
```

Dataset containing info of Stores:

```
stores.shape

(45, 3)

stores.tail()
```

	Store	Type	Size
40	41	A	196321
41	42	C	39690
42	43	C	41062
43	44	C	39910
44	45	B	118221

```
stores.info()

<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
```

Dataset containing additional data of Stores:

```
features.shape

(8190, 12)
features.tail()
```

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
8185	41	7/26/2013	67.56	3.582	497.670000	1454.290000	6.30	4.000000	2418.00	172.460809	7.826821	False
8186	42	7/26/2013	83.32	3.865	7032.371786	3384.176594	0.17	3292.935886	756.79	172.460809	7.826821	False
8187	43	7/26/2013	79.13	3.620	43.370000	3384.176594	1.18	3292.935886	531.35	172.460809	7.826821	False
8188	44	7/26/2013	83.62	3.669	134.310000	3384.176594	1.00	3292.935886	199.75	172.460809	7.826821	False
8189	45	7/26/2013	76.06	3.804	212.020000	851.730000	2.06	10.880000	1864.57	172.460809	7.826821	False

```
features.info()

<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   object
```

```
 2   Temperature   8190 non-null   float64
 3   Fuel_Price    8190 non-null   float64
 4   MarkDown1     8190 non-null   float64
 5   MarkDown2     8190 non-null   float64
 6   MarkDown3     8190 non-null   float64
 7   MarkDown4     8190 non-null   float64
 8   MarkDown5     8190 non-null   float64
 9   CPI           8190 non-null   float64
 10  Unemployment  8190 non-null   float64
 11  IsHoliday     8190 non-null   bool
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 712.0+ KB
```

Handling missing values of features dataset:

```
features["CPI"].fillna(features["CPI"].median(),inplace=True)

features["Unemployment"].fillna(features["Unemployment"].median(),inplace=True)

 for i in range(1,6):

features["MarkDown"+str(i)] = features["MarkDown"+str(i)].apply(lambda x: 0 if x < 0 else x)

features["MarkDown"+str(i)].fillna(value=0,inplace=True)
```

Merging Training Dataset and merged stores-features Dataset:

```
data = pd.merge(data,stores,on='Store',how='left')
data = pd.merge(data,features,on=['Store','Date'],how='left')
data['Date'] = pd.to_datetime(data['Date'])
data.sort_values(by=['Date'],inplace=True)
data.set_index(data.Date, inplace=True)
data['IsHoliday_x'].isin(data['IsHoliday_y']).all()
True

data.drop(columns='IsHoliday_x',inplace=True)

data.rename(columns={"IsHoliday_y" : "IsHoliday"}, inplace=True)

data.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 421570 entries, 2010-02-05 to 2012-10-26
Data columns (total 16 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   Type          421570 non-null  object
 5   Size          421570 non-null  int64
 6   Temperature   421570 non-null  float64
 7   Fuel_Price    421570 non-null  float64
 8   MarkDown1     421570 non-null  float64
 9   MarkDown2     421570 non-null  float64
 10  MarkDown3     421570 non-null  float64
 11  MarkDown4     421570 non-null  float64
 12  MarkDown5     421570 non-null  float64
 13  CPI           421570 non-null  float64
 14  Unemployment  421570 non-null  float64
 15  IsHoliday     421570 non-null  bool
dtypes: bool(1), datetime64[ns](1), float64(10), int64(3), object(1)
memory usage: 51.9+ MB
```

Splitting Date Column:

```
data['Year'] = data['Date'].dt.year

data['Month'] = data['Date'].dt.month

data['Week'] = data['Date'].dt.week
```

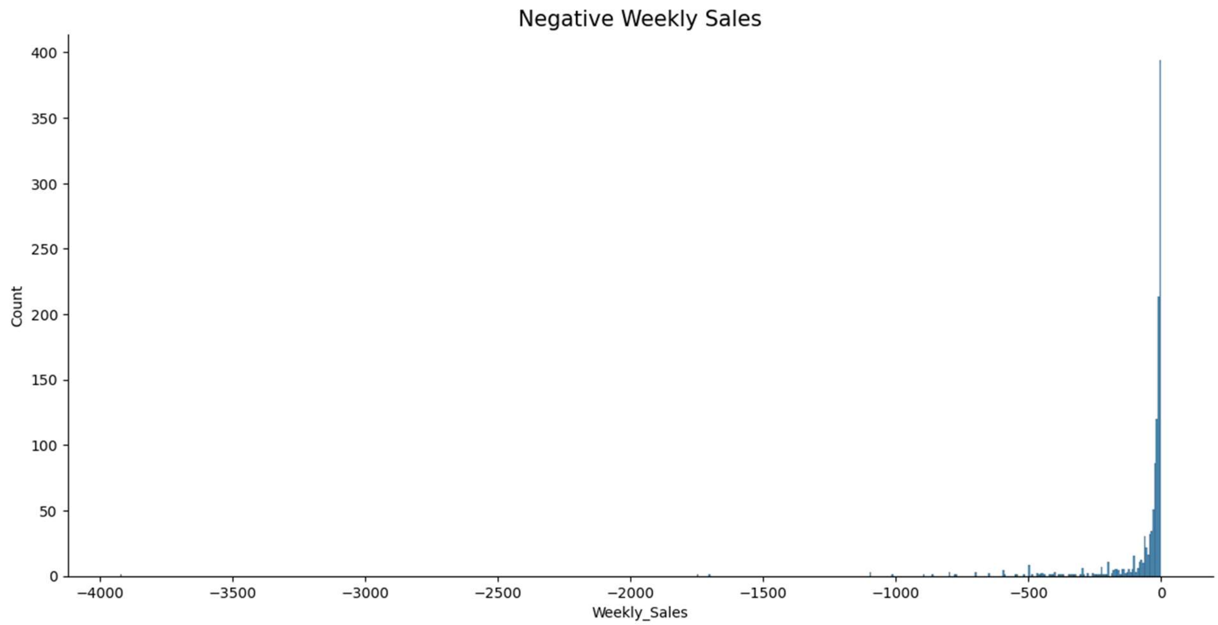
Outlier Detection and Abnormalities:

```
agg_data = data.groupby(['Store', 'Dept']).Weekly_Sales.agg(['max', 'min', 'mean', 'median',
'std']).reset_index()

agg_data.isnull().sum()
Store      0
Dept       0
max        0
min        0
mean       0
median     0
std       37
dtype: int64

store_data = pd.merge(left=data,right=agg_data,on=['Store', 'Dept'],how ='left')

store_data.dropna(inplace=True)
```

```
data = store_data.copy()

del store_data

data['Date'] = pd.to_datetime(data['Date'])

data.sort_values(by=['Date'],inplace=True)

data.set_index(data.Date, inplace=True)

data['Total_MarkDown'] =
data['MarkDown1']+data['MarkDown2']+data['MarkDown3']+data['MarkDown4']+data['MarkD
own5']

data.drop(['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5'], axis =
1,inplace=True)

numeric_col =
['Weekly_Sales','Size','Temperature','Fuel_Price','CPI','Unemployment','Total_MarkDown']

data_numeric = data[numeric_col].copy()

data.shape
(421533, 20)

data = data[(np.abs(stats.zscore(data_numeric)) < 2.5).all(axis = 1)]

data.shape
(377857, 20)
```

Negative Weekly Sales:

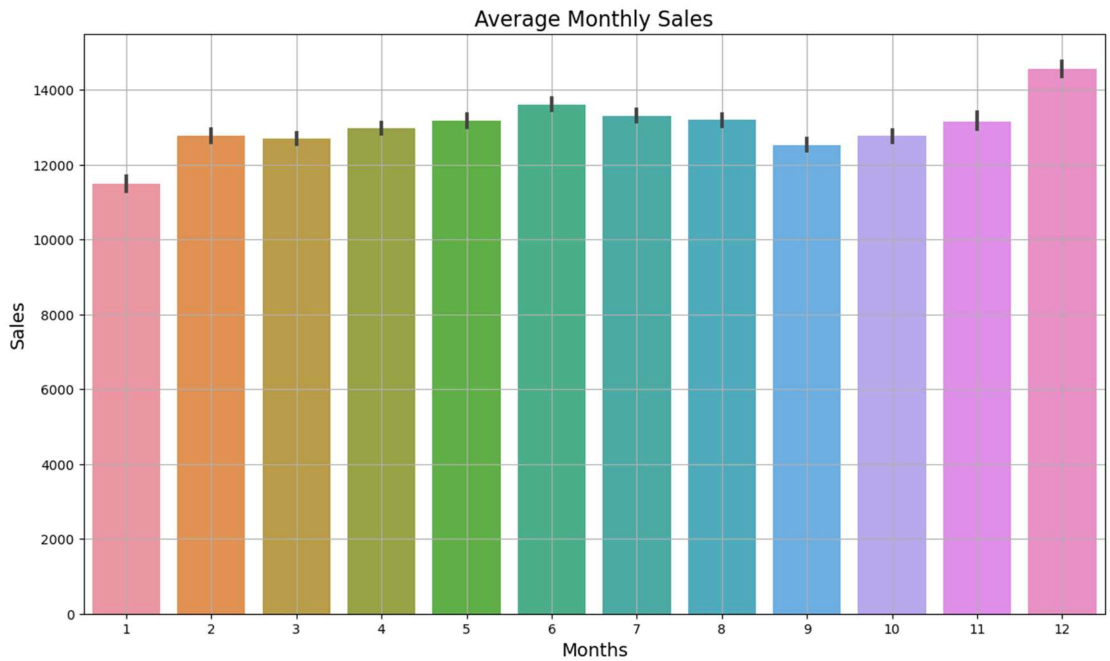
```
y = data["Weekly_Sales"][data.Weekly_Sales < 0]

sns.displot(y,height=6,aspect=2)

plt.title("Negative Weekly Sales", fontsize=15)

plt.savefig('negative_weekly_sales.png')

plt.show()
```




```
data=data[data['Weekly_Sales']>=0]

data.shape

(376657, 20)
data['IsHoliday'] = data['IsHoliday'].astype('int')

data

data.to_csv('preprocessed_walmart_dataset.csv')
```

Data Visuallizations
Average Monthly Sales

```
plt.figure(figsize=(14,8))

sns.barplot(x='Month',y='Weekly_Sales',data=data)

plt.ylabel('Sales',fontsize=14)

plt.xlabel('Months',fontsize=14)

plt.title('Average Monthly Sales',fontsize=16)

plt.savefig('avg_monthly_sales.png')

plt.grid()
```

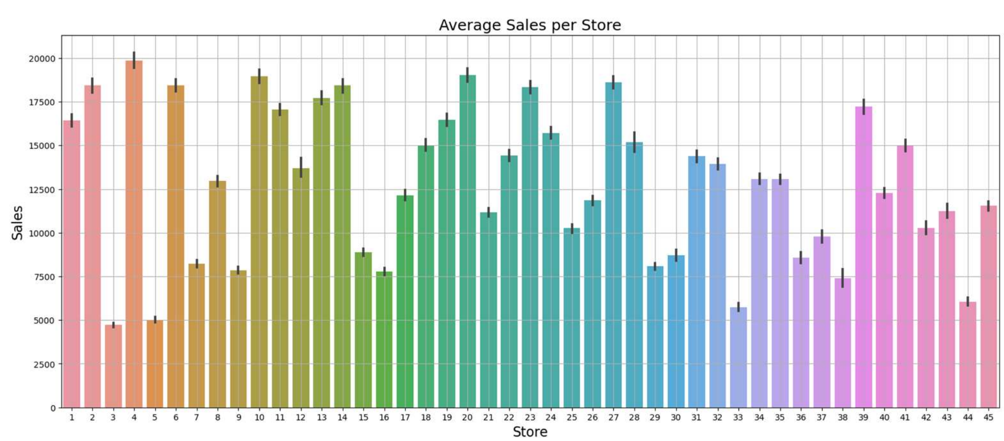


Monthly Sales for Each Year:

```
data_monthly = pd.crosstab(data["Year"], data["Month"],
values=data["Weekly_Sales"],aggfunc='sum')

data_monthly

fig, axes = plt.subplots(3,4,figsize=(16,8))

plt.suptitle('Monthly Sales for each Year', fontsize=18)

k=1

for i in range(3):

    for j in range(4):

      sns.lineplot(ax=axes[i,j],data=data_monthly[k])

      plt.subplots_adjust(wspace=0.4,hspace=0.32)

      plt.ylabel(k,fontsize=12)

      plt.xlabel('Years',fontsize=12)

      k+=1


plt.savefig('monthly_sales_every_year.png')

plt.show()
```

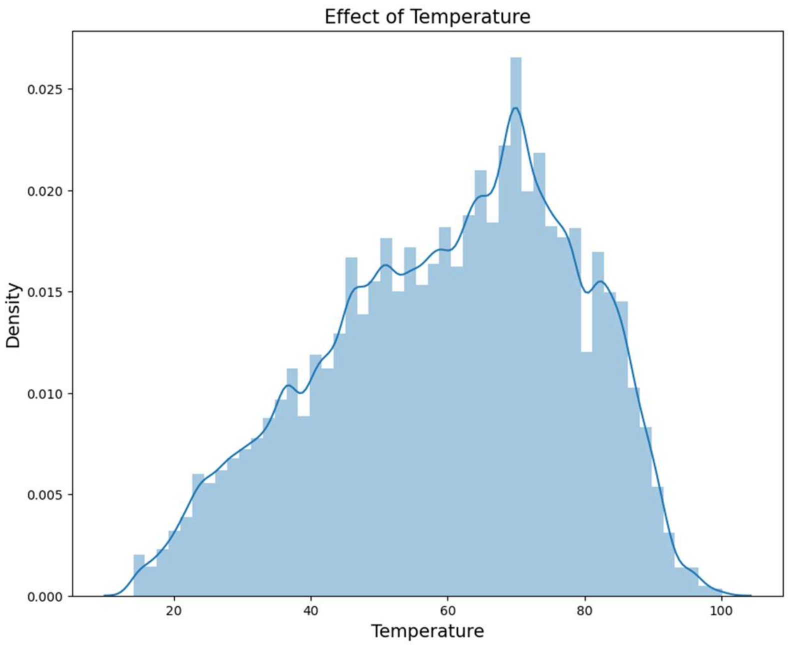
Average Weekly Sales Store wise:

```
plt.figure(figsize=(20,8))
sns.barplot(x='Store',y='Weekly_Sales',data=data)
plt.grid()
plt.title('Average Sales per Store', fontsize=18)
plt.ylabel('Sales', fontsize=16)
plt.xlabel('Store', fontsize=16)
plt.savefig('avg_sales_store.png')
plt.show()
```



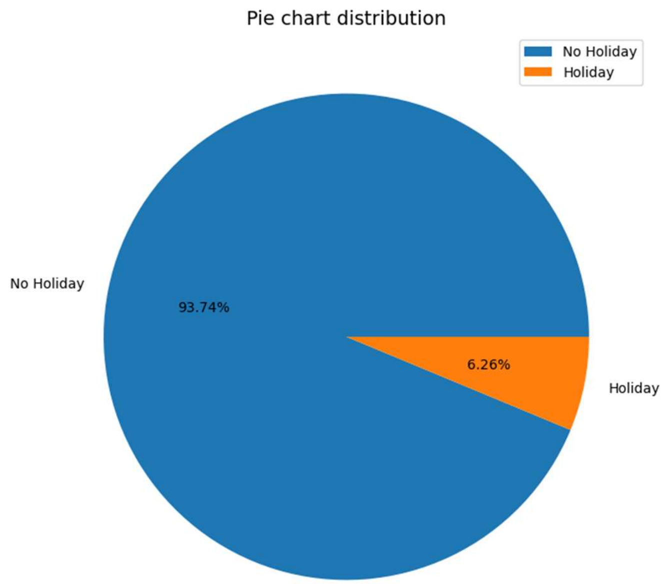
Sales Vs Temperature:

```
plt.figure(figsize=(10,8))
sns.distplot(data['Temperature'])
plt.title('Effect of Temperature',fontsize=15)
plt.xlabel('Temperature',fontsize=14)
plt.ylabel('Density',fontsize=14)
plt.savefig('effect_of_temp.png')
plt.show()
```



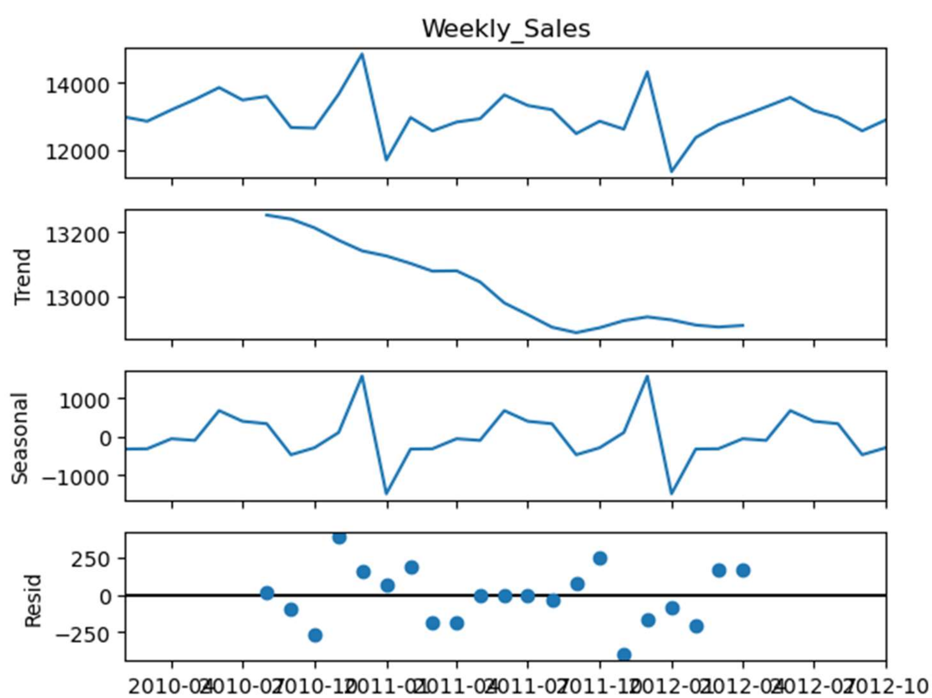
Holiday Distribution:

```
plt.figure(figsize=(8,8))
plt.pie(data['IsHoliday'].value_counts(),labels=['No Holiday','Holiday'],autopct='%0.2f%%')
plt.title("Pie chart distribution",fontsize=14)
plt.legend()
plt.savefig('holiday_distribution.png')
plt.show()
```



Time Series Decompose:

```
sm.tsa.seasonal_decompose(data['Weekly_Sales'].resample('MS').mean(),
model='additive').plot()

plt.savefig('seasonal_decompose.png')

plt.show()
```



One-hot-encoding:

```
cat_col = ['Store','Dept','Type']

data_cat = data[cat_col].copy()

data_cat.tail()
```

	Store	Dept	Type
Date			
2012-10-26	45	95	B
2012-10-26	45	58	B
2012-10-26	45	23	B
2012-10-26	45	85	B
2012-10-26	45	98	B

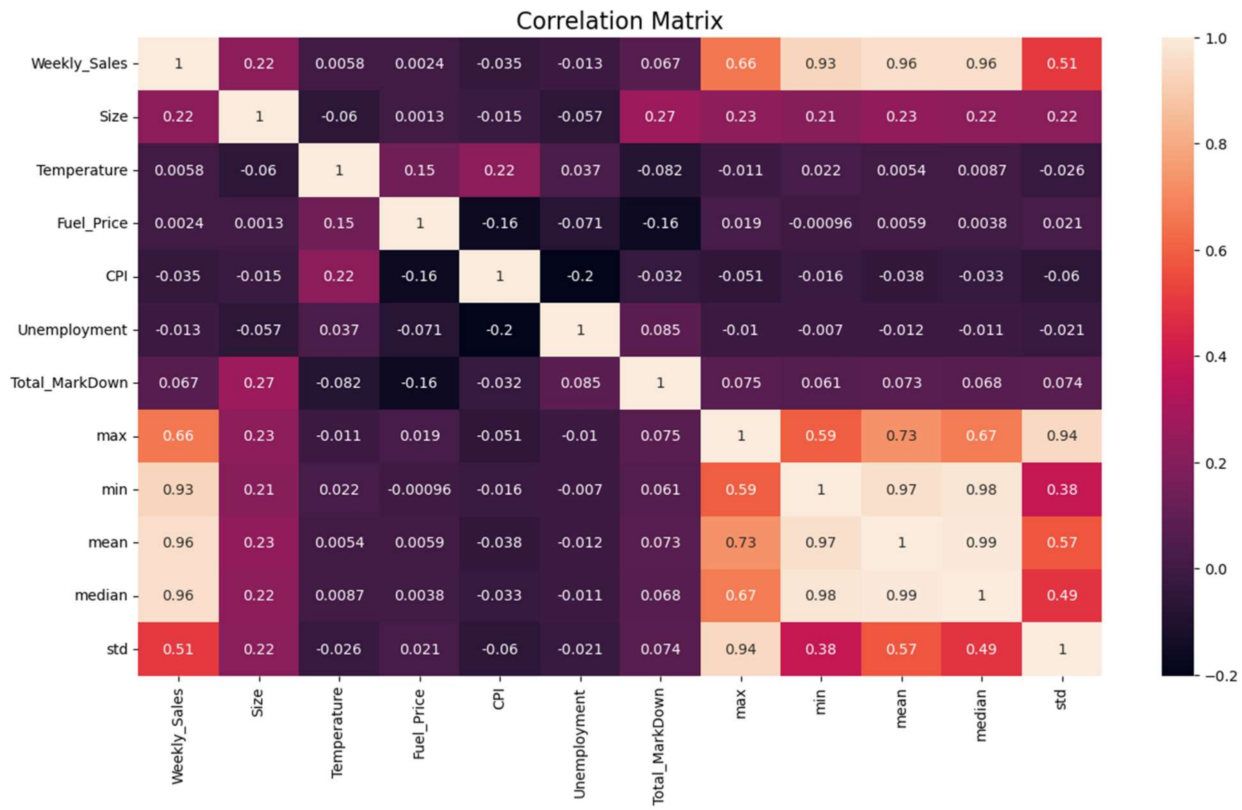
```
data_cat = pd.get_dummies(data_cat,columns=cat_col)

data_cat.head()

data.shape

(376657, 20)
data = pd.concat([data, data_cat],axis=1)

data.shape

(376657, 149)
data.drop(columns=cat_col,inplace=True)
data.drop(columns=['Date'],inplace=True)
data.shape

(376657, 145)
```

Data Normalization:

```
num_col
['Weekly_Sales','Size','Temperature','Fuel_Price','CPI','Unemployment','Total_MarkDown','max','min','mean','median','std']

minmax_scale = MinMaxScaler(feature_range=(0, 1))

def normalization(df,col):

  for i in col:

    arr = df[i]

    arr = np.array(arr)

    df[i] = minmax_scale.fit_transform(arr.reshape(len(arr),1))

  return df

data.head()

data = normalization(data.copy(),num_col)

data.head()
```

Correlation between features of dataset:

```
plt.figure(figsize=(15,8))

corr = data[num_col].corr()

sns.heatmap(corr,vmax=1.0,annot=True)

plt.title('Correlation Matrix',fontsize=16)

plt.savefig('correlation_matrix.png')

plt.show()
```



Recursive Feature Elimination

```
feature_col = data.columns.difference(['Weekly_Sales'])

feature_col

Index(['CPI', 'Dept_1', 'Dept_10', 'Dept_11', 'Dept_12', 'Dept_13', 'Dept_14',
       'Dept_16', 'Dept_17', 'Dept_18',
       ...
       'Type_B', 'Type_C', 'Unemployment', 'Week', 'Year', 'max', 'mean',
       'median', 'min', 'std'],
      dtype='object', length=144)

'''
param_grid={'n_estimators':np.arange(10,25)}
tree=GridSearchCV(RandomForestRegressor(oob_score=False,warm_start=True),param_grid,cv
=5)
tree.fit(data_train[feature_col],data_train['Weekly_Sales'])
'''
```

```
\nparam_grid={'n_estimators':np.arange(10,25)}\ntree=GridSearchCV(RandomForestRegressor(
oob_score=False,warm_start=True),param_grid,cv=5)\ntree.fit(data_train[feature_col],data_train
['Weekly_Sales'])\n"

#tree.best_params_

radm_clf = RandomForestRegressor(oob_score=True,n_estimators=23)
radm_clf.fit(data[feature_col], data['Weekly_Sales'])

pkl_filename = "feature_elim_regressor.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(radm_clf, file)
  print("Saved model to disk")
else:
  print("Model already saved")
Saved model to disk
indices = np.argsort(radm_clf.feature_importances_)[::-1]
feature_rank = pd.DataFrame(columns = ['rank', 'feature', 'importance'])

for f in range(data[feature_col].shape[1]):
    feature_rank.loc[f] = [f+1,
                           data[feature_col].columns[indices[f]],
                           radm_clf.feature_importances_[indices[f]]]

feature_rank
```

	rank	feature	importance
0	1	mean	4.899501e-01
1	2	median	4.380400e-01
2	3	Week	1.967489e-02
3	4	Temperature	8.771803e-03
4	5	CPI	5.885204e-03
...	...	...	...

	rank	feature	importance
139	140	Dept_51	2.360081e-10
140	141	Dept_45	2.115956e-10
141	142	Dept_78	4.336395e-12
142	143	Dept_39	8.461573e-15
143	144	Dept_43	2.175517e-15

144 rows × 3 columns

```
x=feature_rank.loc[0:22,['feature']]
x=x['feature'].tolist()
print(x)
```

```
['mean', 'median', 'Week', 'Temperature', 'CPI', 'max', 'Fuel_Price', 'min', 'Unemployment', 'std', '
Month', 'Total_MarkDown', 'Dept_16', 'Dept_18', 'IsHoliday', 'Size', 'Dept_3', 'Year', 'Dept_1', 'D
ept_9', 'Dept_11', 'Dept_5', 'Dept_7']
X = data[x]
Y = data['Weekly_Sales']
data = pd.concat([X,Y],axis=1)
data
data.to_csv('final_data.csv')
```

Data Splitted into Training, Validation, Test

```
X = data.drop(['Weekly_Sales'],axis=1)

Y = data.Weekly_Sales

X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.20, random_state=50)
```

EXTRA TREES METHOD:

```
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np


# Initialize the Extra Trees regressor
et_regressor = ExtraTreesRegressor(n_estimators=100)

# Fit the regressor on the training data
et_regressor.fit(X_train, y_train)

# Make predictions on the test set
y_pred = et_regressor.predict(X_test)

# Calculate regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2) Value: {r2}")
```

Mean Squared Error (MSE): 0.0008816651590708112

Mean Absolute Error (MAE): 0.014999600508864733

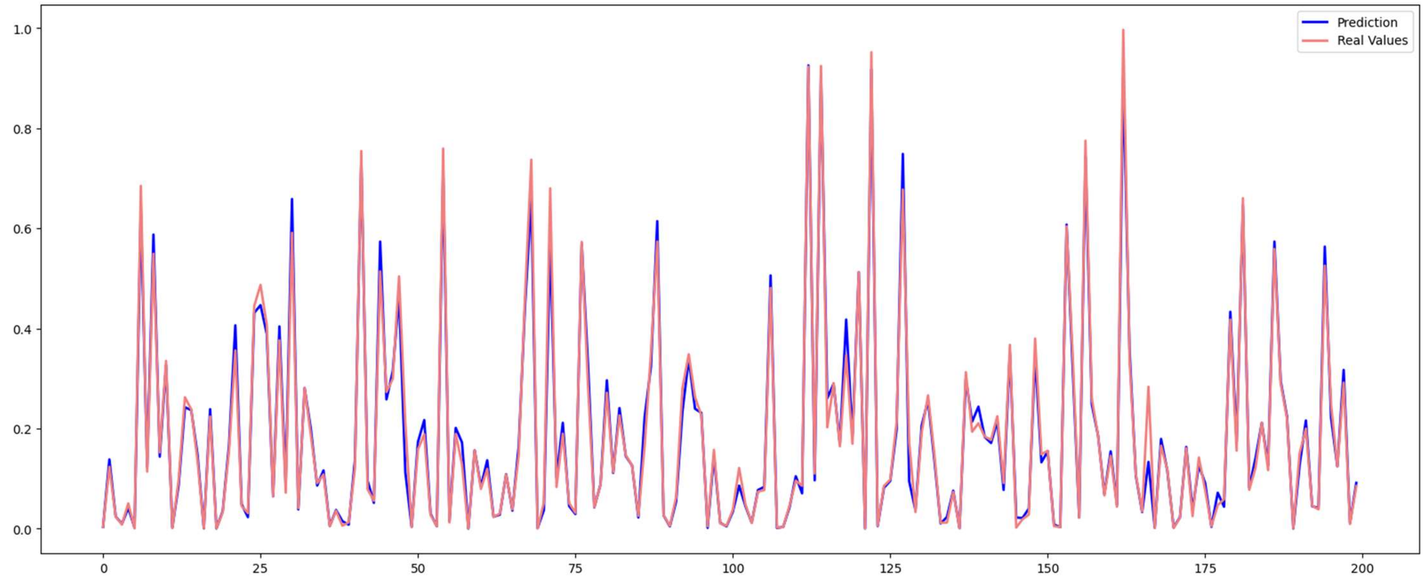
Root Mean Squared Error (RMSE): 0.029692846934418586

R-squared (R2) Value: 0.9803685869647454

```
er = ExtraTreesRegressor()

er.fit(X_train, y_train)

er_acc = er.score(X_test,y_test)*100

print("Accuracy - ",er_acc)
```

Accuracy - 98.04031174361452

```
import matplotlib.pyplot as plt

# Create a figure with a larger size
plt.figure(figsize=(20, 8))

# Plot the first 200 predicted values in blue
plt.plot(y_pred[:200], label="Prediction", linewidth=2.0, color='blue')

# Plot the first 200 actual values in light coral
plt.plot(y_test[:200].values, label="Real Values", linewidth=2.0, color='lightcoral')

# Add a legend to the plot
plt.legend(loc="best")

# Save the plot as an image
plt.savefig('extra_trees_real_pred.png')

# Display the plot
plt.show()
```



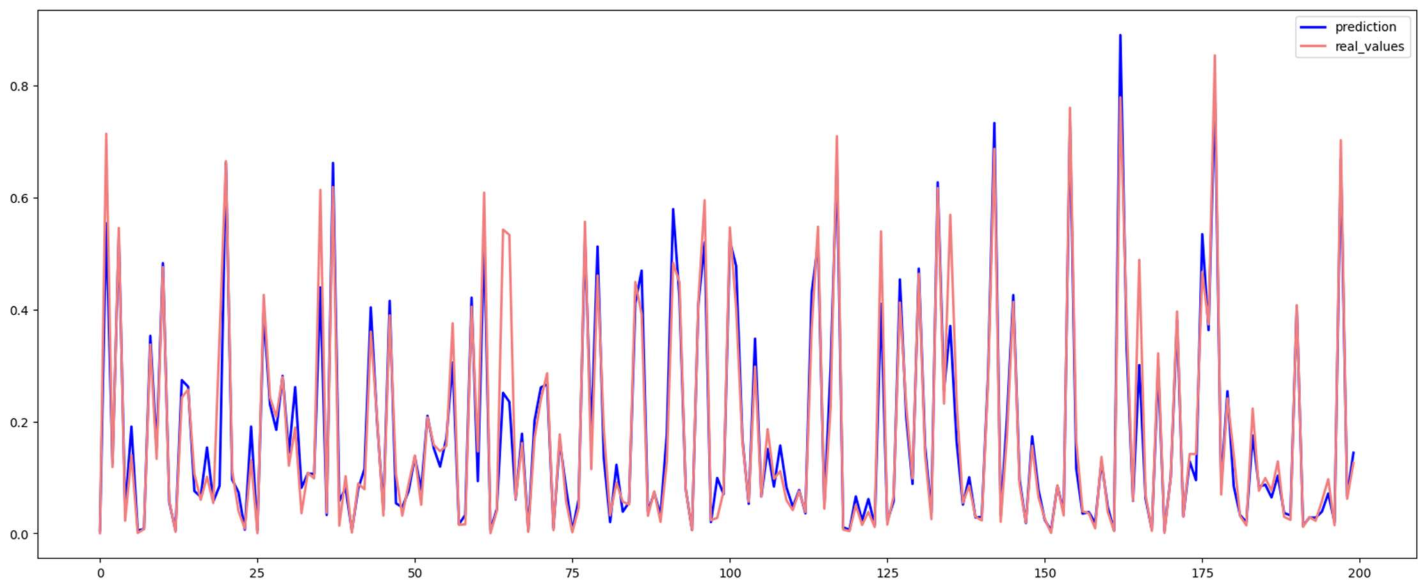
LightGBM (Light Gradient Boosting Machine):

```
import lightgbm as lgb

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, mean_absolute_error,
explained_variance_score,r2_score

import matplotlib.pyplot as plt

train_data = lgb.Dataset(X_train, label=y_train)

params = {

    'objective': 'regression',  # for regression task

    'metric': 'mse',  # mean squared error as the evaluation metric

    'boosting_type': 'gbdt',  # gradient boosting decision tree

    'num_leaves': 31,  # number of leaves in each tree

    'learning_rate': 0.05,

    'feature_fraction': 0.9,
```

```
}

# Train the LightGBM model
num_round = 100  # Number of boosting rounds (you can adjust this)
bst = lgb.train(params, train_data, num_round)

# Make predictions on the test set
y_pred = bst.predict(X_test, num_iteration=bst.best_iteration)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
explained_var = explained_variance_score(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"R-squared (R2) Value: {r2}")
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
print(f"Explained Variance: {explained_var}")

# Create a DataFrame to compare actual and predicted values
lgbm_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lgbm_df.to_csv('lgbm_real_pred.csv')

# Create a plot to visualize the results (first 200 data points)
plt.figure(figsize=(20, 8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0, color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0, color='lightcoral')
```

```
plt.legend(loc="best")

plt.savefig('lgbm_real_pred.png')

plt.show()

bst.save_model('lgbm_model.txt')

print("Saved model to disk")
```

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.06798
8 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2668
[LightGBM] [Info] Number of data points in the train set: 301325, number of used features: 23
[LightGBM] [Info] Start training from score 0.179686
R-squared (R2) Value: 0.9551490411617666
Mean Squared Error: 0.002024916012821558
Mean Absolute Error: 0.024645265935532106
Explained Variance: 0.955149091340892



Linear Regression Model:

```
from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import StandardScaler


# Initialize the linear regression model

lr = LinearRegression()


# Initialize the StandardScaler

scaler = StandardScaler()


# Fit and transform the training data with the scaler

X_train_scaled = scaler.fit_transform(X_train)

X_train=X_train_scaled

# Fit the linear regression model with the scaled data
```

```
lr.fit(X_train_scaled, y_train)
```

```
lr_acc = lr.score(X_test,y_test)*100
```

```
print("Linear Regressor Accuracy - ",lr_acc)

y_pred = lr.predict(X_test)
```

```
print("MAE" , metrics.mean_absolute_error(y_test, y_pred))

print("MSE" , metrics.mean_squared_error(y_test, y_pred))

print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
print("R2" , metrics.explained_variance_score(y_test, y_pred)) lr_df =
pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

lr_df.to_csv('lr_real_pred.csv')
```

```
lr_df
```

```
plt.figure(figsize=(20,8))

plt.plot(lr.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')

plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')

plt.legend(loc="best")

plt.savefig('lr_real_pred.png')
```

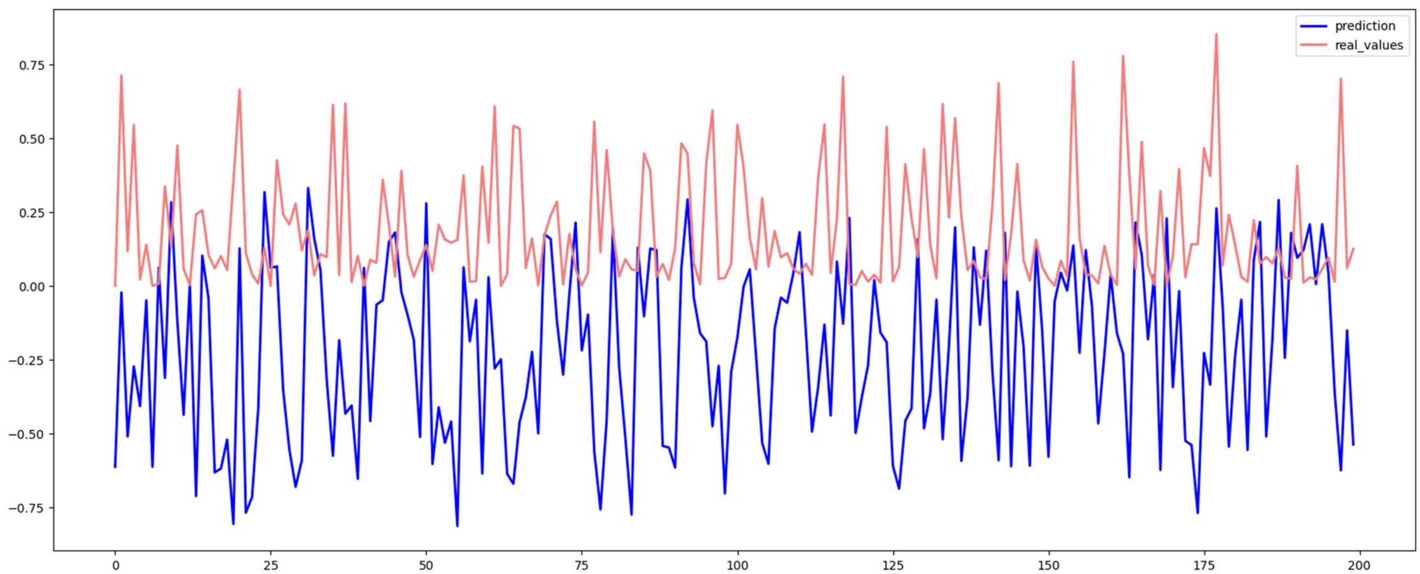
```
plt.show()
```

```
Linear Regressor Accuracy -  -550.4425523482787
MAE 0.024645265935532106
MSE 0.002024916012821558
RMSE 0.04499906679945216
R2 0.955149091340892
```

	Actual	Predicted
Date		
2010-10-22	0.000404	-0.612121
2010-05-07	0.713607	-0.021834
2011-09-23	0.118297	-0.509250
2010-08-06	0.545391	-0.271982
2012-08-17	0.022539	-0.406744
...	...	...

	Actual	Predicted
Date		
2011-09-30	0.137852	-0.534517
2011-10-07	0.055045	-0.544479
2011-02-18	0.338530	0.196864
2011-10-28	0.631689	-0.546636
2011-05-20	0.037323	-0.130799

75332 rows × 2 columns



Saving trained model:

```
pkl_filename = "linear_regressor.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(lr, file)
  print("Saved model to disk")
```

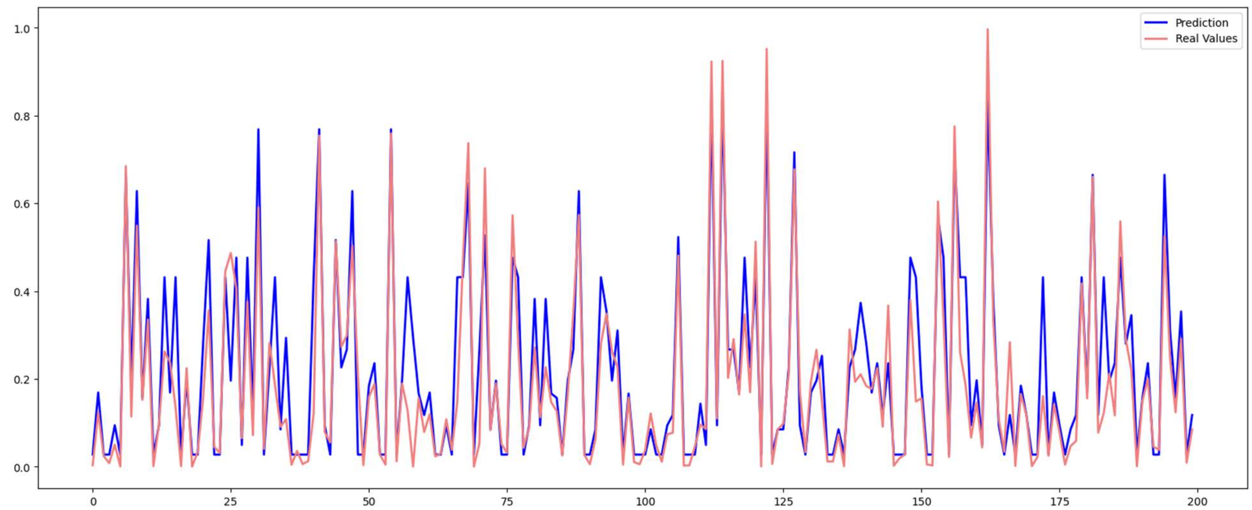
```
else:
  print("Model already saved")
```

Saved model to disk

ADABOOST CLASSIFIER:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import AdaBoostRegressor  # Import AdaBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Fit the classifier on the training data
ada_regressor = AdaBoostRegressor(n_estimators=100)  # You can choose a different number of
estimators

# Fit the regressor on the training data
ada_regressor.fit(X_train, y_train)

# Make predictions on the test set
y_pred = ada_regressor.predict(X_test)

# Calculate regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2) Value: {r2}")

# Create a scatter plot of predicted vs. actual values
plt.figure(figsize=(20, 8))
plt.plot(y_pred[:200], label="Prediction", linewidth=2.0, color='blue')
plt.plot(y_test[:200].values, label="Real Values", linewidth=2.0, color='lightcoral')
plt.legend(loc="best")
plt.savefig('ada_real_pred.png')
plt.show()
```

Mean Squared Error (MSE): 0.005677102799603772
Mean Absolute Error (MAE): 0.04545054862894153
Root Mean Squared Error (RMSE): 0.07534655134512643
R-squared (R2) Value: 0.8735919767771264



Random Forest Regressor Model:

```
rf = RandomForestRegressor()

rf.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

```
rf_acc = rf.score(X_test,y_test)*100

print("Random Forest Regressor Accuracy - ",rf_acc)
```

Random Forest Regressor Accuracy - 97.88907135637824

```
y_pred = rf.predict(X_test)

from sklearn.ensemble import RandomForestRegressor

n_estimators = 100  # can change this value to your desired number of trees

rf_model = RandomForestRegressor(n_estimators=n_estimators)

number_of_trees = rf_model.n_estimators

print("MAE" , metrics.mean_absolute_error(y_test, y_pred))

print("MSE" , metrics.mean_squared_error(y_test, y_pred))
```

```
print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print("R2" , metrics.explained_variance_score(y_test, y_pred))
```

```
Number of trees in the Random Forest: 100
MAE 0.015440129853658986
MSE 0.000934034751993661
RMSE 0.03056198213456812
R2 0.9787502531437008
```

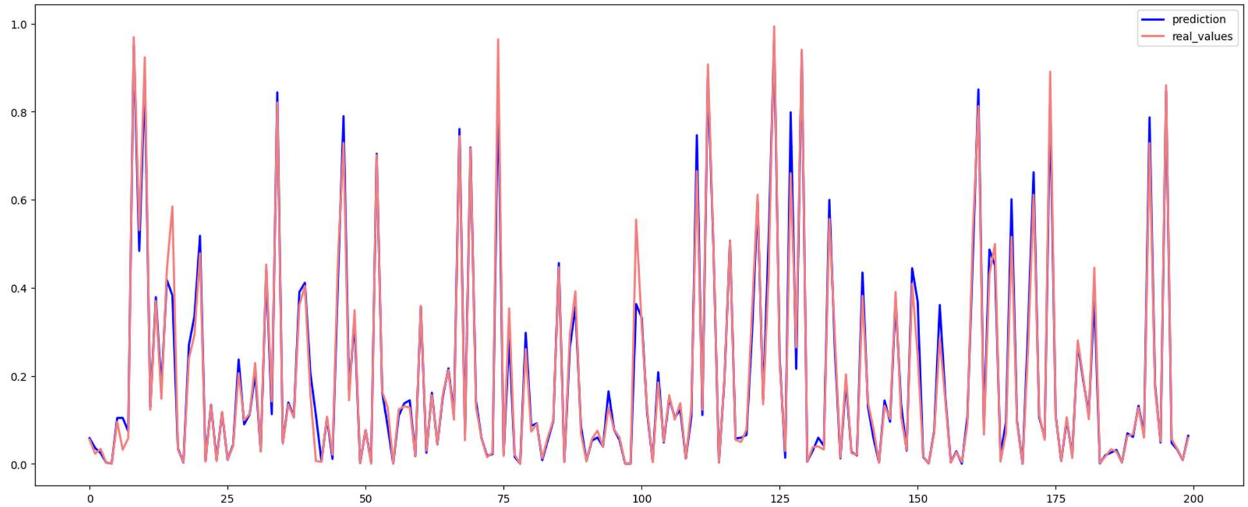
```
rf_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

rf_df.to_csv('./predictions/rf_real_pred.csv')

rf_df
```

	Actual	Predicted
Date		
2011-08-05	0.161661	0.124485
2010-07-09	0.364278	0.320277
2011-07-01	0.005003	0.012285
2012-01-06	0.015856	0.020360
2011-08-26	0.000318	0.000566
...	...	...
2011-01-28	0.169068	0.176886
2010-08-20	0.252860	0.272780
2010-11-26	0.265617	0.393226
2010-03-12	0.008865	0.015019
2010-02-12	0.230510	0.258844

74850 rows × 2 columns

```
plt.figure(figsize=(20,8))

plt.plot(rf.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')

plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
```

```
plt.legend(loc="best")

plt.savefig('plots/rf_real_pred.png')

plt.show()
```



Saving trained model:

```
pkl_filename = "./models/randomforest_regressor.pkl"

if (not path.isfile(pkl_filename)):

  # saving the trained model to disk

  with open(pkl_filename, 'wb') as file:

    pickle.dump(rf, file)

  print("Saved model to disk")

else:

  print("Model already saved")
```

Saved model to disk

K Neighbors Regressor Model:

```
knn = KNeighborsRegressor(n_neighbors = 1,weights = 'uniform')

knn.fit(X_train,y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
```

```
weights='uniform')
```

```
knn_acc = knn.score(X_test, y_test)*100
print("KNeigbhbors Regressor Accuracy - ",knn_acc)
```

KNeigbhbors Regressor Accuracy - 91.97260309962996

```
y_pred = knn.predict(X_test)
print("MAE" , metrics.mean_absolute_error(y_test, y_pred))
print("MSE" , metrics.mean_squared_error(y_test, y_pred))
print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2" , metrics.explained_variance_score(y_test, y_pred))
```

MAE 0.033122163743083126
MSE 0.003624289656000884
RMSE 0.060202073519114635
R2 0.9199211034808975

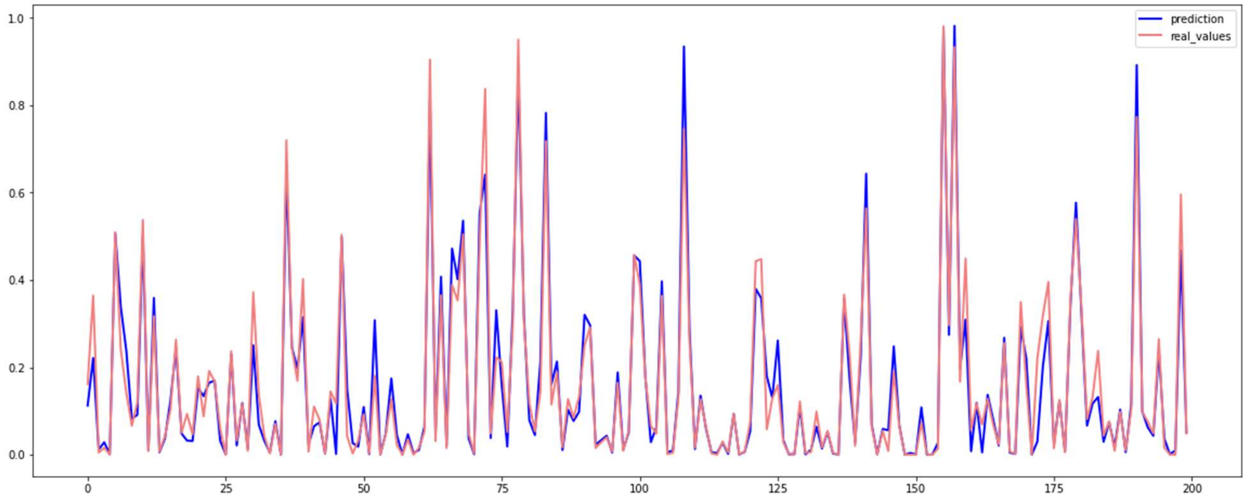
```
knn_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
knn_df.to_csv('./predictions/knn_real_pred.csv')
knn_df
```

	Actual	**Predicted**
Date		
2011-08-05	0.161661	0.112559
2010-07-09	0.364278	0.221307
2011-07-01	0.005003	0.011921
2012-01-06	0.015856	0.028551
2011-08-26	0.000318	0.001063
...	...	...
2011-01-28	0.169068	0.229475
2010-08-20	0.252860	0.262688
2010-11-26	0.265617	0.203904
2010-03-12	0.008865	0.001663
2010-02-12	0.230510	0.287258

74850 rows × 2 columns

```
plt.figure(figsize=(20,8))
plt.plot(knn.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
```

```
plt.legend(loc="best")
plt.savefig('plots/knn_real_pred.png')
plt.show()
```



Saving trained model:

```
pkl_filename = "./models/knn_regressor.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(knn, file)
  print("Saved model to disk")
else:
  print("Model already saved")
Saved model to disk
```

XGboost Model:

```
xgbr = XGBRegressor()

xgbr.fit(X_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

```
xgb_acc = xgbr.score(X_test,y_test)*100

print("XGBoost Regressor Accuracy - ",xgb_acc)
```

```
XGBoost Regressor Accuracy -  94.21152336133142
```

```
y_pred = xgbr.predict(X_test)

print("MAE" , metrics.mean_absolute_error(y_test, y_pred))

print("MSE" , metrics.mean_squared_error(y_test, y_pred))

print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print("R2" , metrics.explained_variance_score(y_test, y_pred))
```

```
MAE 0.026771808878560288
MSE 0.0026134394830486384
RMSE 0.051121810248157665
R2 0.9421152350249367
```

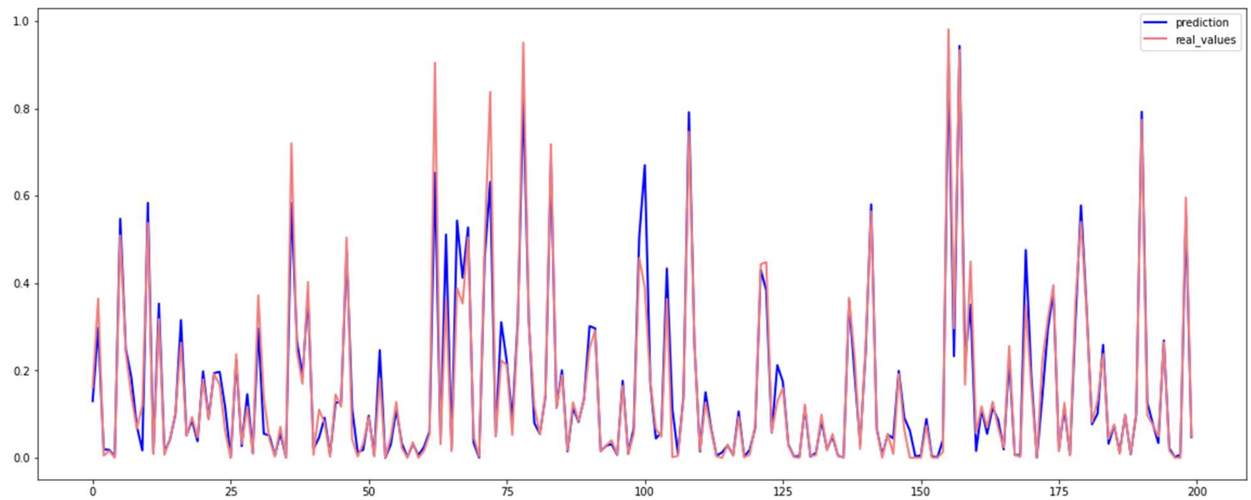
```
xgb_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

xgb_df.to_csv('./predictions/xgb_real_pred.csv')

xgb_df
```

	Actual	Predicted
Date		
2011-08-05	0.161661	0.129809
2010-07-09	0.364278	0.297181
2011-07-01	0.005003	0.019209
2012-01-06	0.015856	0.018191
2011-08-26	0.000318	0.002950
...	...	...
2011-01-28	0.169068	0.228197
2010-08-20	0.252860	0.234475
2010-11-26	0.265617	0.404794
2010-03-12	0.008865	0.011655

	Actual	Predicted
Date		
2010-02-12	0.230510	0.241285

74850 rows × 2 columns

```
plt.figure(figsize=(20,8))

plt.plot(xgbr.predict(X_test[:200]), label="prediction", linewidth=2.0,color='blue')

plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')

plt.legend(loc="best")

plt.savefig('plots/xgb_real_pred.png')

plt.show()
```



Saving trained model:

```
pkl_filename = "./models/xgboost_regressor.pkl"

if (not path.isfile(pkl_filename)):

  # saving the trained model to disk

  with open(pkl_filename, 'wb') as file:

    pickle.dump(xgbr, file)
```

```
  print("Saved model to disk")

else:

  print("Model already saved")
```

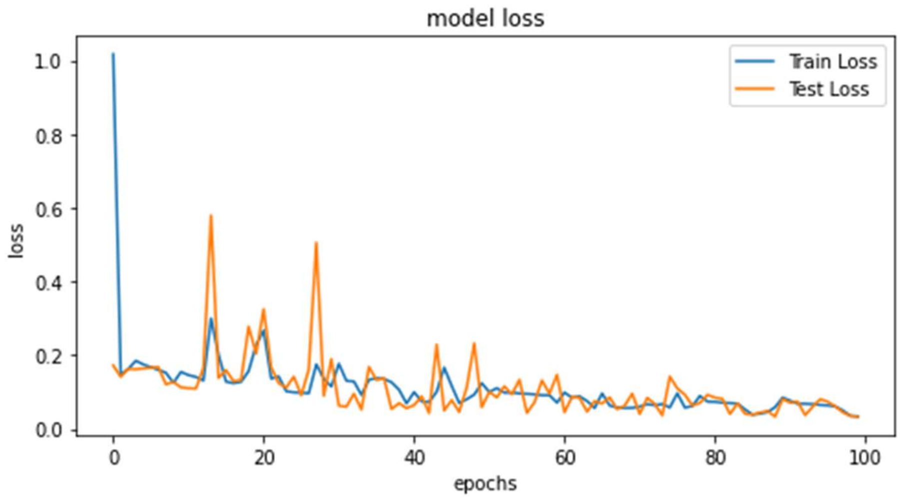
Saved model to disk

Custom Deep Learning Neural Network:

```
def create_model():
  model = Sequential()
  model.add(Dense(64, input_dim=X_train.shape[1], kernel_initializer='normal',activation='relu')
)
  model.add(Dense(32, kernel_initializer='normal'))
  model.add(Dense(1, kernel_initializer='normal'))
  model.compile(loss='mean_absolute_error', optimizer='adam')
  return model
estimator_model = KerasRegressor(build_fn=create_model, verbose=1)
history = estimator_model.fit(X_train, y_train, validation_split=0.2, epochs=100, batch_size=50
00)
```

```
48/48 [==============================] - 0s 10ms/step - loss: 0.0771 - val_loss: 0.0714
Epoch 92/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0677 - val_loss: 0.0742
Epoch 93/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0686 - val_loss: 0.0377
Epoch 94/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0668 - val_loss: 0.0606
Epoch 95/100
48/48 [==============================] - 0s 8ms/step - loss: 0.0663 - val_loss: 0.0809
Epoch 96/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0655 - val_loss: 0.0740
Epoch 97/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0629 - val_loss: 0.0616
Epoch 98/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0539 - val_loss: 0.0453
Epoch 99/100
48/48 [==============================] - 0s 10ms/step - loss: 0.0373 - val_loss: 0.0359
Epoch 100/100
48/48 [==============================] - 0s 9ms/step - loss: 0.0346 - val_loss: 0.0335
```

```
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.savefig('plots/dnn_loss.png')
plt.show()
```




```
dnn_acc = metrics.r2_score(y_pred, y_test)*100
print("Deep Neural Network accuracy - ",dnn_acc)
```

Deep Neural Network accuracy - 90.50328742871066

```
y_pred = estimator_model.predict(X_test)
```

2340/2340 [==============================] - 3s 977us/step

```
print("MAE" , metrics.mean_absolute_error(y_test, y_pred))
print("MSE" , metrics.mean_squared_error(y_test, y_pred))
print("RMSE" , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2" , metrics.explained_variance_score(y_test, y_pred))
```

MAE 0.033255980538121045
MSE 0.0038670810150368187
RMSE 0.062185858641951856
R2 0.9144106847304281

```
dnn_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
dnn_df.to_csv('./predictions/dnn_real_pred.csv')
dnn_df
```

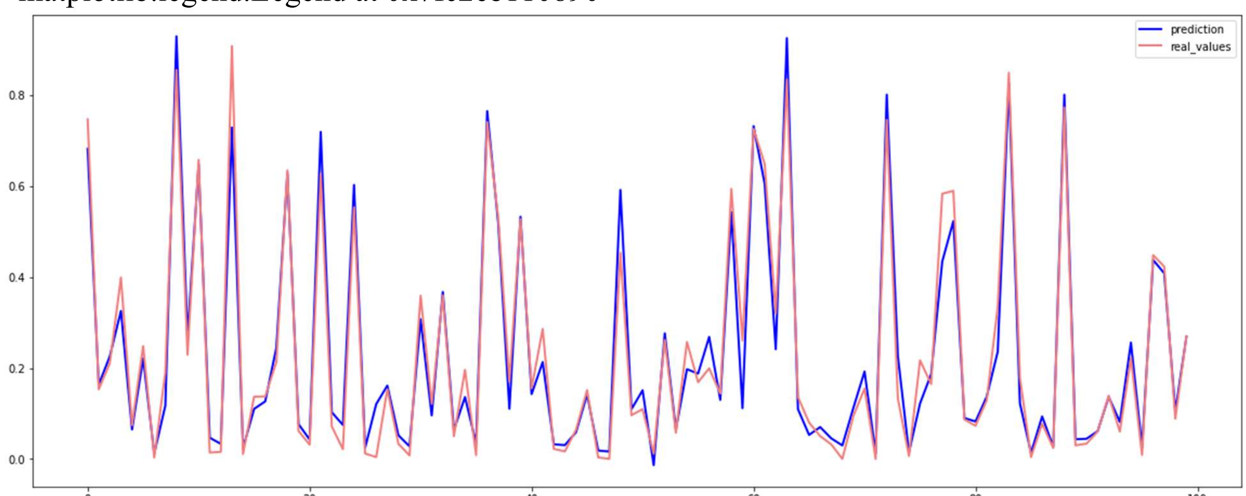
	Actual	**Predicted**
Date		
2011-08-05	0.161661	0.124761
2010-07-09	0.364278	0.289382
2011-07-01	0.005003	0.034531
2012-01-06	0.015856	0.024284
2011-08-26	0.000318	0.015496

	Actual	Predicted
Date		
...	...	...
2011-01-28	0.169068	0.233344
2010-08-20	0.252860	0.236093
2010-11-26	0.265617	0.342386
2010-03-12	0.008865	0.023427
2010-02-12	0.230510	0.242022

74850 rows × 2 columns

```
plt.figure(figsize=(20,8))
plt.plot(estimator_model.predict(X_test[200:300]), label="prediction", linewidth=2.0,color='blue
')
plt.plot(y_test[200:300].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.savefig('plots/dnn_real_pred.png')
plt.legend(loc="best")
```

4/4 [==============================] - 0s 5ms/step
<matplotlib.legend.Legend at 0x7fc2eb110890>




```
filepath = './models/dnn_regressor.json'
weightspath = './models/dnn_regressor.h5'
if (not path.isfile(filepath)):
  # serialize model to JSON
  model_json = estimator_model.model.to_json()
  with open(filepath, "w") as json_file:
```

```
    json_file.write(model_json)
  print("Saved model to disk")
else:
  print("Model already saved")
```

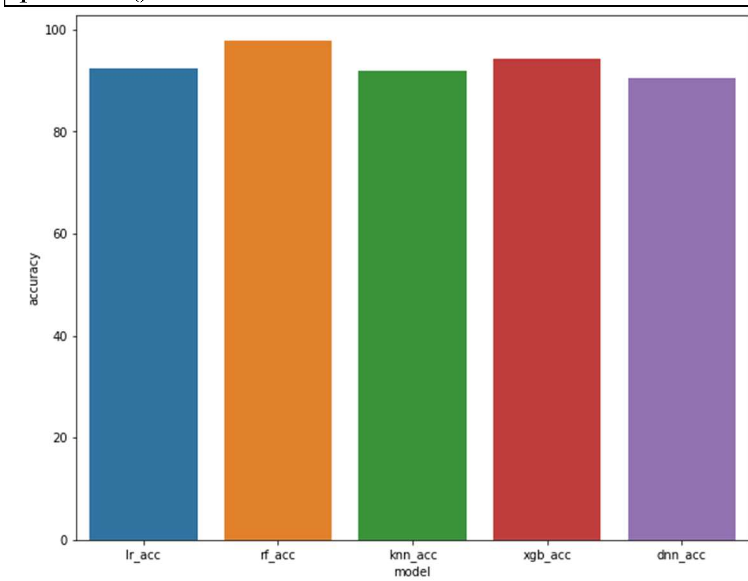
Saved model to disk

Comparing Models:

```
acc = {'model':['lr_acc','rf_acc','knn_acc','xgb_acc','dnn_acc'],'accuracy':[lr_acc,rf_acc,knn_acc,xgb_acc,dnn_acc]}
acc_df = pd.DataFrame(acc)
acc_df
```

	model	accuracy
0	lr_acc	92.280797
1	rf_acc	97.889071
2	knn_acc	91.972603
3	xgb_acc	94.211523
4	dnn_acc	90.503287

```
plt.figure(figsize=(10,8))
sns.barplot(x='model',y='accuracy',data=acc_df)
plt.savefig('plots/compared_models.png')
plt.show()
```



PERFORMANCE TESTING:

	MAE	MSE	RMSE	R2
Light GBM	0.024645265935532106	0.002024916012821558	0.378992469344156	0.9551490411617666
Linear Regression	0.0246452659355321	0.00202491601282156	0.449990667994522	0.955149091340892
Extra Tree Method	0.0149996005088647	0.000881665159070811	0.296928469344186	0.980368586964745
Ada Boost	0.0454505486289415	0.00567710279960377	0.0753465513451264	0.873591976777126
Random Forest	0.155225368975386	0.000953062336469744	0.0308717222332505	0.978890990012565
K nearest neighbour	0.0331221637430831	0.00362428965600088	0.0602020735191146	0.919921103480875
XG Boost	0.0267718088785603	0.00261343948304864	0.0511218102481577	0.942115235024937

RESULTS:

The Extra Trees Regression algorithm outperformed other regression models with an exceptional accuracy of 98% and an R-squared value of 0.980368586964745. This indicates a high level of precision in predicting walmart store sales forecasting .

ADVANTAGES & DISADVANTAGES:

Advantages of Walmart Sales Forecasting:

1. Efficient Inventory: Accurate forecasting helps Walmart maintain the right amount of products, reducing costs and ensuring items are available when customers need them.
2. Streamlined Supply Chain: Sales forecasts guide supply chain operations, making sourcing, distribution, and transportation more efficient.
3. Effective Pricing: Walmart can adjust prices based on forecasts to match market demand and competition, improving sales strategies.

Disadvantages and Challenges:

1. Data Quality: Inaccurate or incomplete data can lead to less reliable forecasts.
2. Model Complexity: Advanced models can be complex to develop and maintain, requiring specialized expertise.
3. Model Accuracy: Forecasting models can be affected by unpredictable market conditions.
4. Cost: Developing and maintaining advanced models can be expensive in terms of technology and personnel.
5. External Factors: Unforeseen events like economic shocks can disrupt forecasts.
6. Model Interpretability: Complex models may lack transparency, making it challenging to understand their predictions.
7. Continuous Monitoring: Regular updates are needed to maintain forecasting accuracy.
8. Forecast Uncertainty: Forecasts can't eliminate uncertainty, and decisions based solely on them carry risks.

CONCLUSION:

Based on the dataset used, it can be said that Extra Tree Regression Technique is the best to predict the sales of Walmart Store in future followed by Random Forest Regression Technique. This result could be useful for other retail store owners as well in order to determine their sales and they could directly opt for Sales Prediction using Extra Tree Regression Technique or Random Forest Approach rather than spending time in doing analysis using other Supervised Machine Learning Algorithms. The other retailers could also be benefitted by doing the demand analysis on the similar grounds. This study contributed in understanding the fact that external factors, such as Unemployment rate, Holiday Week, CPI, etc. also plays a vital role while predicting the sales of any retail store.

FUTURE SCOPE:

Based on the above experimentation, it has been observed that Simple Regression techniques for building the prediction models may not be the best choice for sales prediction if the management is trying to predict the sales for lesser duration and have historical data only for few years. This is because the accuracy is good only for ensemble learning techniques which involves averaging of results obtained from multiple decision trees. Therefore, the business owner should choose Ensemble Learning Models. One limitation of this study is that based on the variance in training data, the predictions obtained from a specific algorithm may vary. So, the owner has to decide the algorithm effectively given his requirements.

The observation that ensemble learning techniques outperform simple regression for sales prediction in scenarios with limited historical data suggests a promising future scope for Walmart's sales forecasting. To enhance accuracy, Walmart can further explore ensemble learning models, develop adaptable algorithms, and invest in data augmentation and feature engineering. Integrating external data sources, continuous model monitoring, interpretability, and providing forecast confidence intervals can also contribute to more reliable sales predictions, ensuring Walmart's competitive edge in the dynamic retail market.

APPENDIX: