Capstone Project – Project Walmart

-by Maroju Shiva Acharya

Table of Contents:

Problem Statement:

A retail store that has multiple outlets across the country are facing issues in managing the inventory - to match the demand with respect to supply. You are a data scientist, who has to come up with useful insights using the data and make prediction models to forecast the sales for X number of months/years.

Project Objective:

1. Using the above data, come up with useful insights that can be used by each of the stores to improve in various areas.

2. Forecast the sales for each store for the next 12 weeks.

Data Description:

One of the leading retail stores in the US, Walmart, would like to predict the sales and demand accurately. There are certain events and holidays which impact sales on each day. There are sales data available for 45 stores of Walmart. The business is facing a challenge due to unforeseen demands and runs out of stock some times, due to the inappropriate machine learning algorithm. An ideal ML algorithm will predict demand accurately and ingest factors like economic conditions including CPI, Unemployment Index, etc.

Walmart runs several promotional markdown events throughout the year. These markdowns precede prominent holidays, the four largest of all, which are the Super Bowl, Labour Day, Thanksgiving, and Christmas. The weeks including these holidays are weighted five times higher in the evaluation than non-holiday weeks. Part of the challenge presented by this competition is modeling the effects of markdowns on these holiday weeks in the absence of complete/ideal historical data. Historical sales data for 45 Walmart stores located in different regions are available.

This is the historical data that covers sales from 2010-02-05 to 2012-11-01, in the file Walmart_Store_sales. Within this file you will find the following fields:

Dataset Information:

The walmart.csv contains 6435 rows and 8 columns.

Feature Name	Description
Store	Store number
Date	Week of Sales
Weekly_Sales	Sales for the given store in that week
Holiday_Flag	If it is a holiday week
Temperature	Temperature on the day of the sale
Fuel_Price	Cost of the fuel in the region
CPI	Consumer Price Index
Unemployment	Unemployment Rate

Data Pre-processing Steps and Inspiration:

1. **Checking the Null Values:**

   The current data has no Null Values present and it is necessary to deal with null values as it can have an impact on machine learning models. The presence of null values can introduce challenges during the training and prediction phases, and it is important to handle them appropriately.

```
In [8]: 1 df.isnull().sum()
Out[8]: Store           0
        Date            0
        Weekly_Sales    0
        Holiday_Flag    0
        Temperature     0
        Fuel_Price      0
        CPI             0
        Unemployment    0
        dtype: int64
```

2. **Checking Duplicated Values:**

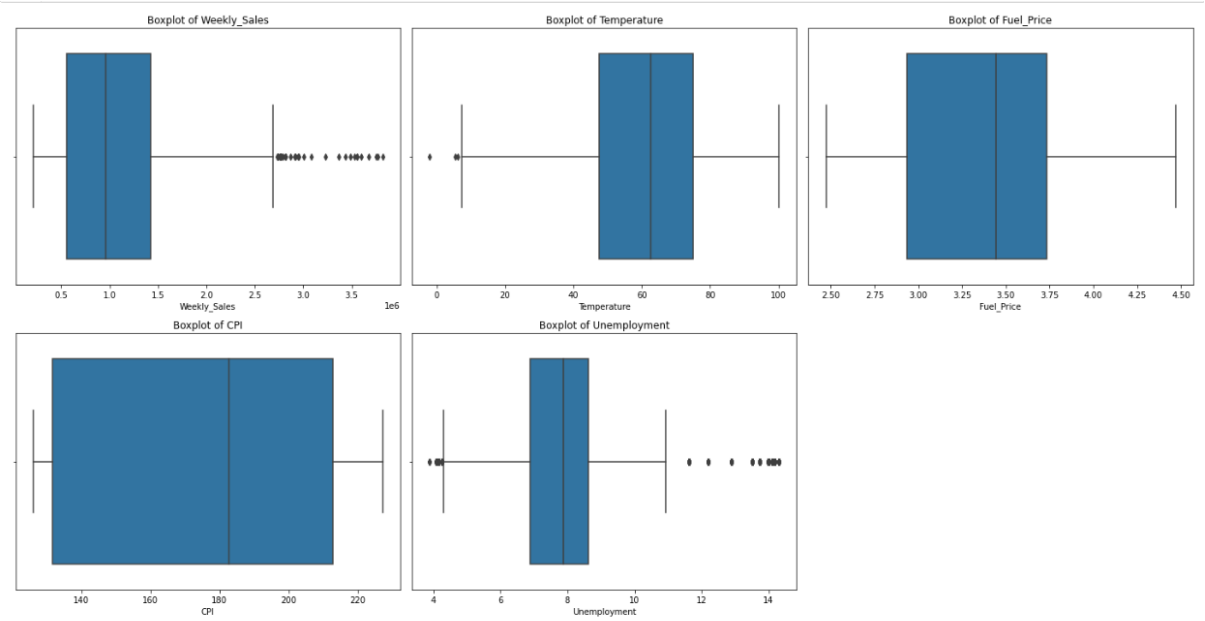
   There are no duplicated values present in this data set. Treating duplicated values in a dataset is important for several reasons like Duplicated values can introduce inaccuracies in the dataset and compromise data integrity. If duplicate values are not addressed, they can bias the results and lead to misleading conclusions.

```
In [9]: 1 df.duplicated().sum()
Out[9]: 0
```

3. Checking for Outliers and Treating the Outliers:

Outliers can significantly impact data integrity by introducing noise or errors in the dataset. Outliers may arise due to various reasons such as measurement errors, data entry mistakes, or genuinely unusual observations and they can negatively affect the statistical analysis and the training process of a machine learning algorithm resulting in lower accuracy.

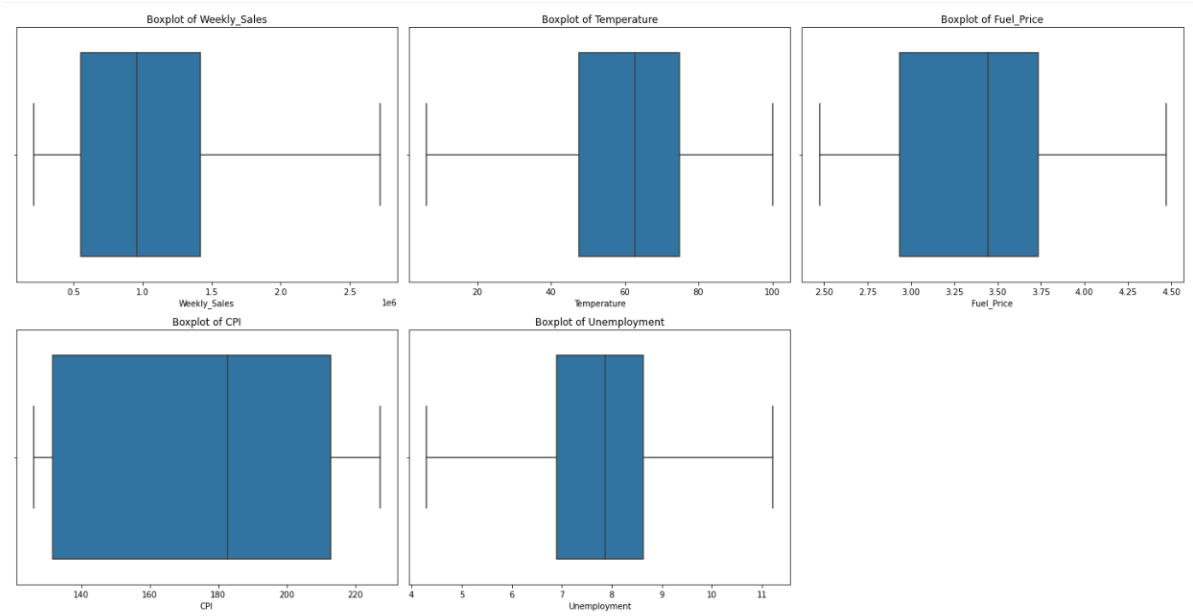
Here We checked for the outliers with the Box plot



from the above plots we can observe that Weekely_sales, Tempearture and Unemployment has Outliers. So we will going to treat these 3 features

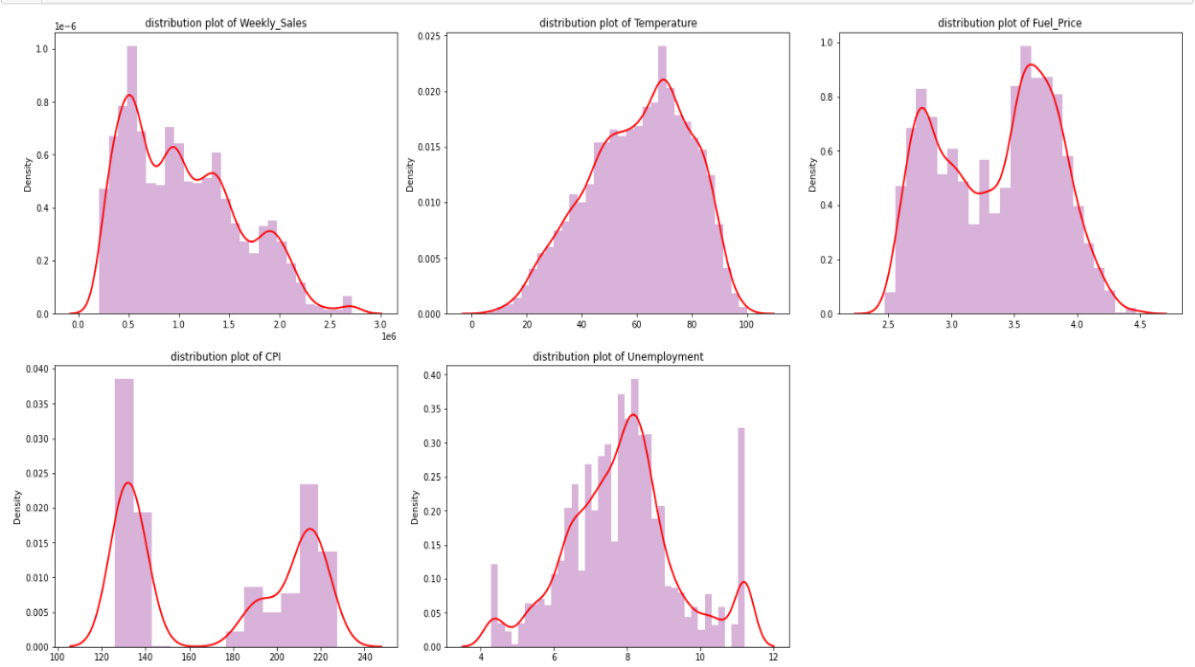
```
def treat_outliers():
    l_o = ['Weekly_Sales','Temperature','Unemployment']
    for i in l_o:
        Q1, Q3 = np.quantile(df[i], [0.25,0.75])
        IQR = Q3-Q1
        ll = Q1-1.5*IQR
        ul = Q3+1.5*IQR
        df[i] = np.where(df[i]>ul, ul, np.where(df[i]<ll, ll, df[i]))
```

We plot the Datasets after treating the outliers

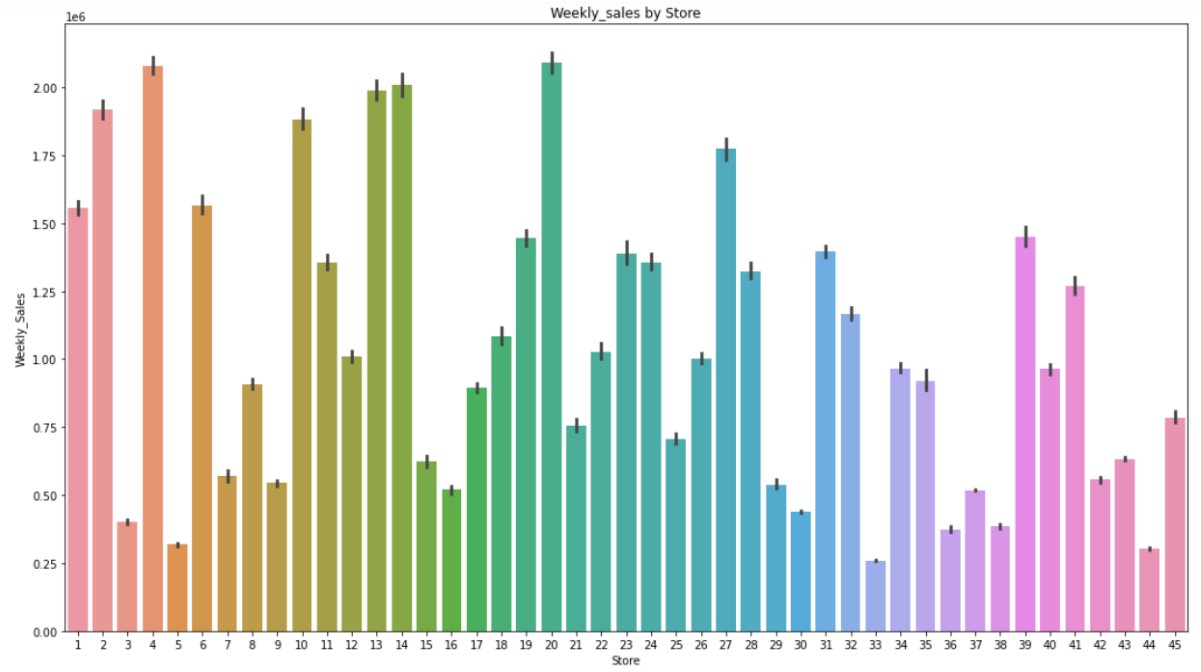


now we can observer that we have successfully treated all the outliers in the data

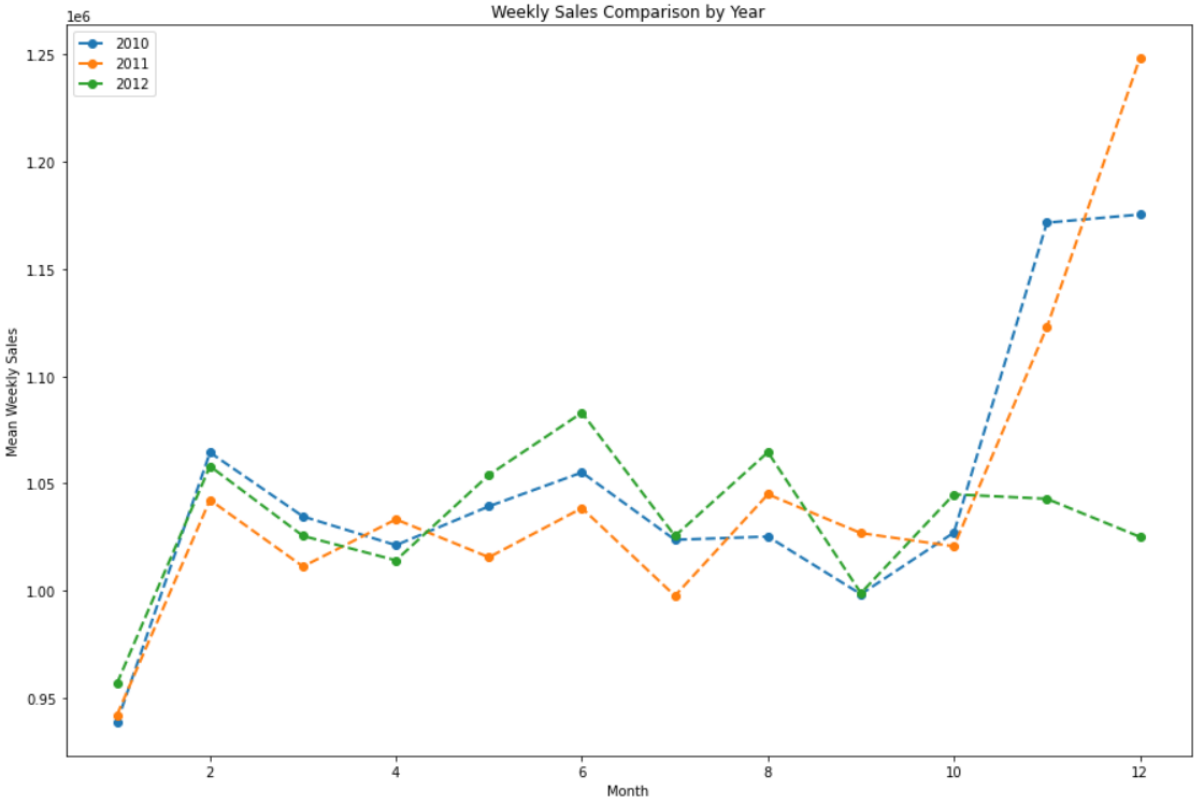
4. Checking the Distribution of data and insights in it:



From the above Plots we can infer that Weekely sales has right Skewed Data and remaing all features are closer to zero skewed data. The Feature Tempeature, Fuel_price and Unemployment are closely to normal Distributed Data, where Mean, median and model are lies at the same point.

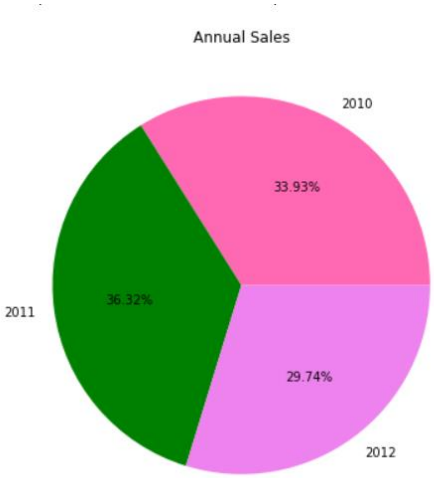


From the above graph, we can say that on average sales in store 20 is the best and 4 and 14 following it and store 33 has very less sales followed by 5 and 44

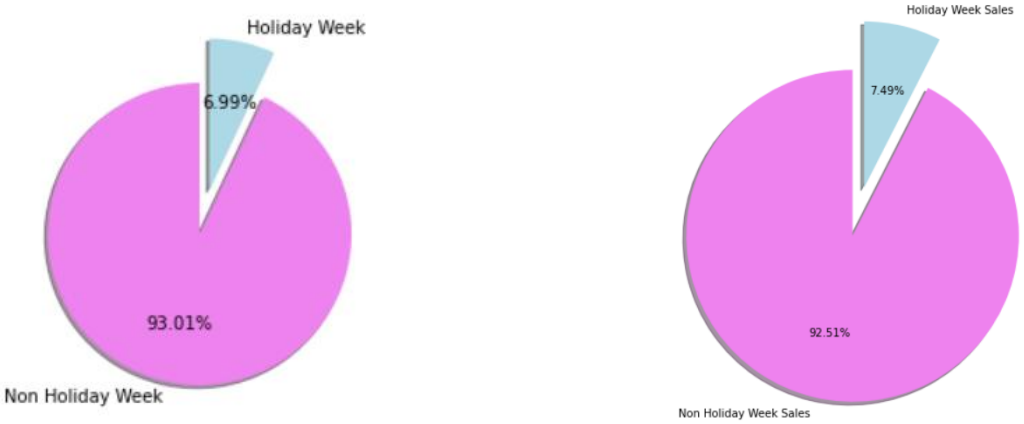


From the above graph we can say that in year 2010 and 2011 the december month has highest sales in that 2011 has very high comparatively. so for years 2010 and 2011 January

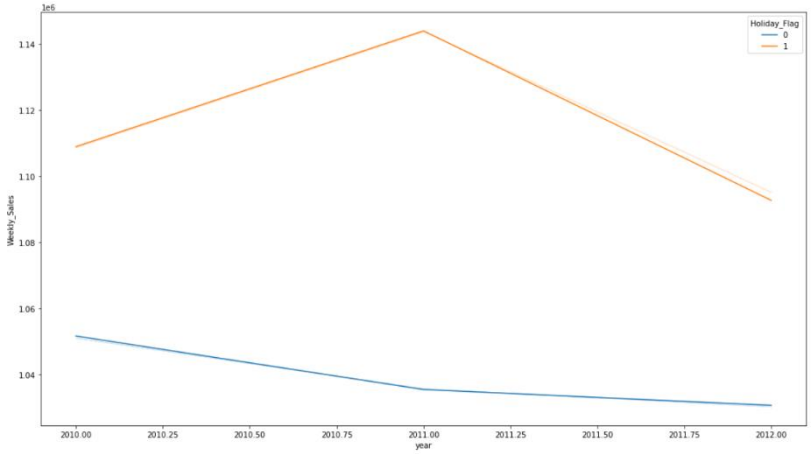
month has very low saleas and december has high where as for year 2012 in january very less sales and in june high sales.



When we check overall sales Year wise from the above graph we can say that 2011 has highest sales then followed by 2010 and 2012



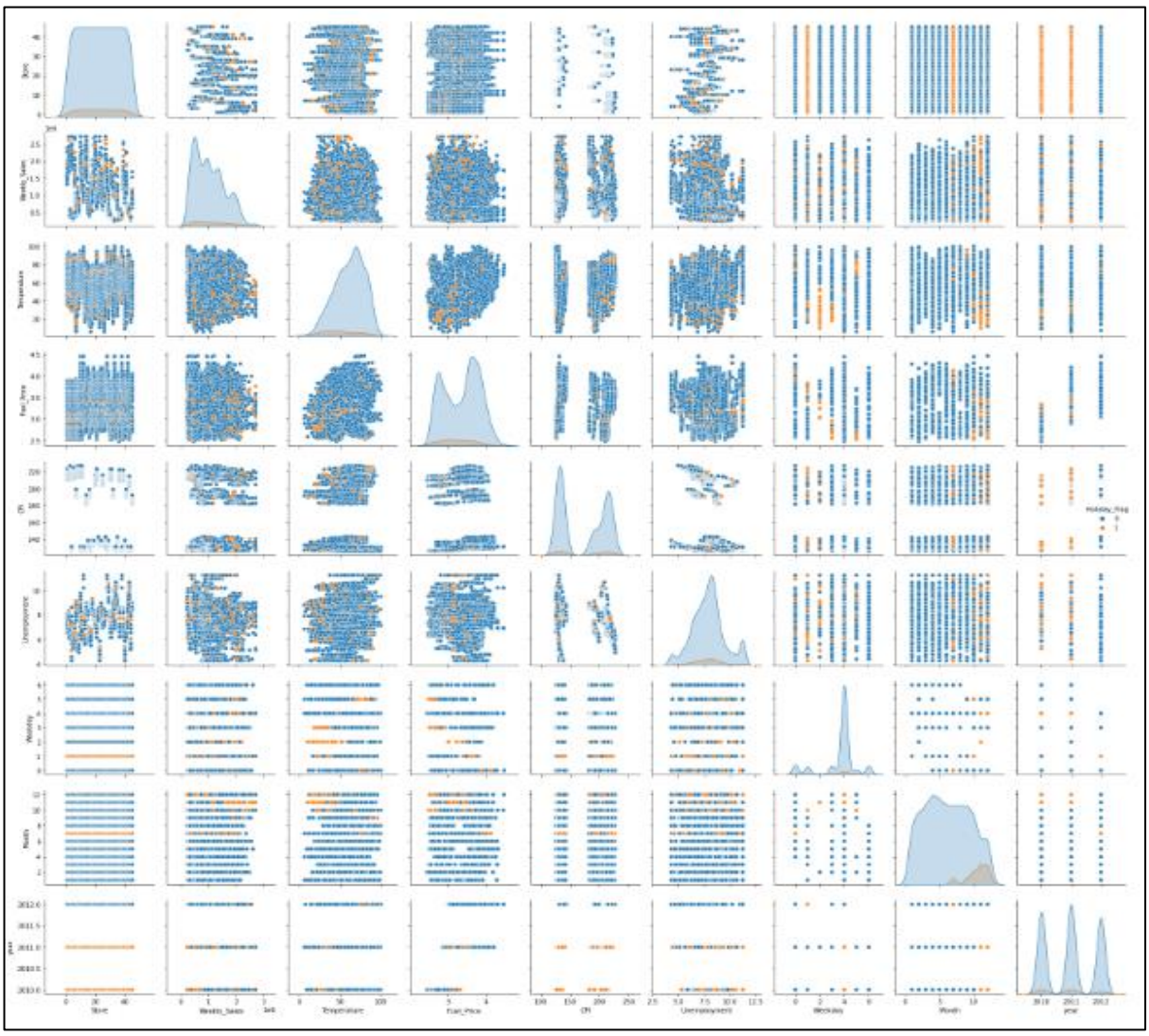
From the above plots we can observe that Holiday weeks are approx. 7 percent of the whole data and the weekely sales are also has relatively same percentage.



From the above graph the Sales are high in Holiday week compared to non holiday week for all the data.

Pair Plot:

To quickly examine the correlations, distributions, and potential patterns between different variables in your dataset.

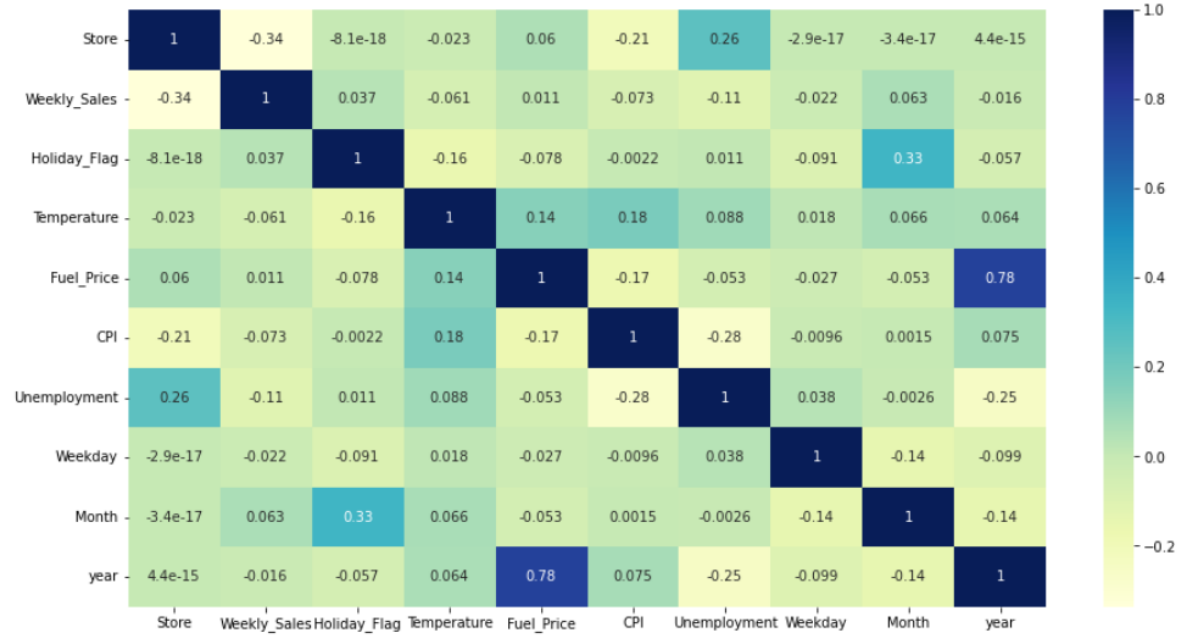


Correlation Matrix Using Heat Map:

Here, we will create a Correlation Matrix to identify the correlation between various fields.

```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),cmap="YlGnBu", annot=True)
```

```
<AxesSubplot:>
```



From the above plot we can observe that

year and Fuel_price has max Correlation with each Other and followed by 'Month and Holiday_flag', 'Store and Unenployment'

Fuel Price and Unemployment has the least correlation with the Weekly Sales comparatively.

Feature Engineering:

Feature engineering, also known as feature creation, is the process of constructing new features from existing data to train a machine learning model. This step can be more important than the actual model fitting step because a machine learning algorithm only learns from the data we give it and creating features that are relevant to a task is absolutely crucial.

Datetime Features:

Date fields are rich sources of information that can be used with machine learning models. However, these date-time variables do require some feature engineering to convert them into numerical data. We extract new features from “Date” feature such as year, month, weekday.

```
df.Date=pd.to_datetime(df.Date,errors='coerce')
df['Weekday']=df.Date.dt.weekday
df['Month']=df.Date.dt.month
df['year']=df.Date.dt.year
df.head(2)
```

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Weekday	Month	year
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	6	5	2010
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	3	12	2010

Scaling the Data before training the Model:

Now as the problem statement is to forecast the sales, So here Weekely_sales is the Target Column. So here we will divide our data in X and y and once we divide the data we will perform the Scaling on the data before we train.

- Scaling ensures that all features are on a similar scale. When features have different scales, it can lead to biased model training, as features with larger scales may dominate the learning process. Scaling helps to prevent this issue and allows each feature to contribute proportionately to the model's training.
- Scaling ensures that the importance or weights assigned to features are comparable. Without scaling, features with larger scales may be perceived as more important by the model solely due to their larger numerical values. Scaling enables a fair comparison of feature importance.
- It's important to note that not all models require scaling. For example, tree-based models like decision trees and random forests are not sensitive to feature scales. However, many other models, such as linear regression, logistic regression, neural networks, and support vector machines, can benefit from feature scaling to varying degrees.
- Common scaling techniques include standardization (subtracting mean and dividing by standard deviation), min-max scaling (scaling to a specific range, e.g., [0, 1]), and normalization (scaling to unit norm). Here We used the standardization technique.

```
#importing Standard Scaler

from sklearn.preprocessing import StandardScaler
```

```
st = StandardScaler()
```

```
X = st.fit_transform(X)
```

Splitting Data in Train Test:

The train-test split is a common practice in machine learning to evaluate the performance of a model on unseen data. It involves splitting our dataset into two subsets: one for training the model and the other for evaluating its performance. The training set is used to fit the model's parameters, while the test set is used to assess the model's generalization ability.

```
#importing train test split to split data

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y , test_size=0.3 , random_state = 0)
```

```
print(X_train.shape, y_train.shape)
```

```
(4504, 9) (4504,)
```

```
print(X_test.shape, y_test.shape)
```

```
(1931, 9) (1931,)
```

Here we split the data as test_size=0.3 which indicates that 70% of the data is used for training the Model and 30% is used for testing the Model.

Choosing the Algorithm for the Project:

Here Target column here is Weekely_sales. We can observe that the Values are Continuous in nature, So We need to apply Regression Model on this particular dataset to forecast the sales.

I Have experimented and compared different Regression algorithms to determine the best fit for this particular problem. The list of Models I have performed are:

- **Linear Regression:**

  Linear regression is a popular supervised machine learning algorithm used for predicting a continuous target variable based on one or more input features. It assumes a linear relationship between the input features and the target variable. The goal of linear regression is to find the best-fit line that minimizes the difference between the predicted values and the actual target values.

- **Decision Tree:**

  A decision tree is a popular machine learning algorithm used for both classification and regression tasks. It is a flowchart-like model that uses a tree-like structure of decisions and their possible consequences. The algorithm builds a predictive model by recursively partitioning the data based on feature values, optimizing for reduced impurity or increased information gain.

  In a decision tree, each internal node represents a feature or attribute, each branch represents a decision based on that feature, and each leaf node represents a class label (in classification) or a predicted value (in regression). The goal is to create a tree that predicts the target variable accurately while keeping the tree as small and simple as possible.

- **Random Forest:**

  The Random Forest model is an ensemble learning method that combines multiple decision trees to create a more accurate and robust predictive model. Each decision tree in the ensemble is trained on a different subset of the data, using a random subset of features at each split. The final prediction is made by aggregating the predictions of all the individual trees. It is known for its high accuracy and robustness. By combining multiple decision trees and averaging their predictions, Random Forest reduces overfitting and improves generalization. It can handle complex datasets with a high number of features and instances.

- **K-Nearest Neighbors (KNN) :**

  The K-Nearest Neighbors (KNN) algorithm is a simple yet powerful machine learning model used for both classification and regression tasks. It is a non-parametric algorithm that makes predictions based on the similarity between a new sample and the labeled samples in the training data.

  In this algorithm, the number "K" refers to the number of nearest neighbors to consider when making a prediction. So the he performance of the KNN algorithm depends on choosing the appropriate value for K. A small K may lead to overfitting, while a large K may lead to underfitting.

- **Support Vector Machines (SVM):**

  Support Vector Machines (SVM) is a powerful and versatile machine learning algorithm used for both classification and regression tasks. SVM aims to find an optimal hyperplane that separates different classes or predicts continuous values, while maximizing the margin between the classes or minimizing the error.

  In SVM, the algorithm maps the input data into a high-dimensional feature space and finds the best hyperplane that separates the classes or predicts the values. It is particularly useful in high-dimensional spaces, can handle limited data, and offers flexibility with different kernel functions. However, it is important to consider computational complexity, feature scaling, and parameter tuning for optimal performance.

- **Gradient Boosting:**

  Gradient Boosting is a powerful ensemble learning method that combines multiple weak prediction models (typically decision trees) to create a strong predictive model. It is a boosting algorithm that builds the model in a stage-wise manner, where each new model in the ensemble focuses on correcting the mistakes made by the previous models.

  It is a powerful algorithm that can achieve high accuracy and handle various data types. It is suitable for complex tasks and can effectively handle noisy data. However, it is important to consider computational complexity and hyperparameter tuning when using Gradient Boosting models.

- **XG Boost:**

  XGBoost, short for Extreme Gradient Boosting, is a popular and highly efficient implementation of the Gradient Boosting algorithm. It is designed to optimize the performance and speed of gradient boosting models and it has gained popularity in various machine learning competitions.

It is known for its high performance and efficiency. It is designed to optimize the speed and scalability of gradient boosting models. XGBoost implements parallel processing techniques, tree pruning, and other optimizations that make it faster than traditional gradient boosting implementations. It also provides a feature importance analysis that helps you understand the contribution of each feature in the model's predictions. This analysis can guide feature selection, identify irrelevant or redundant features, and provide insights into the underlying data patterns.

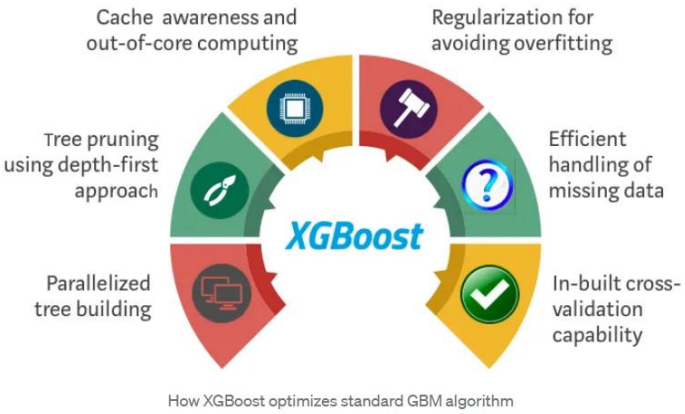
Motivation and Reasons for Choosing the Algorithm:

XGBoost is ensemble tree method that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements. It is a popular machine learning algorithm known for its high performance and effectiveness in various predictive modeling tasks. When combined with GridSearchCV, which performs hyperparameter tuning, XGBoost becomes an even more powerful tool for model optimization.

There are several key motivations and reasons for choosing the XGBoost algorithm with GridSearchCV:

- **High predictive performance:** XGBoost is designed to deliver high predictive accuracy by utilizing an ensemble of weak prediction models, typically decision trees. It employs a gradient boosting framework that sequentially adds new models to correct the mistakes made by previous models, thereby reducing the overall prediction error. This technique often leads to better performance compared to other algorithms, making it a compelling choice for many applications.
- **Speed and scalability:** XGBoost is optimized for speed and efficiency, allowing it to handle large datasets with millions of instances and thousands of features. The algorithm employs parallel processing and tree pruning techniques, which significantly enhance its scalability and computational efficiency. This makes XGBoost an attractive option for real-world applications that involve big data.
- **Flexibility and customization:** XGBoost provides a wide range of hyperparameters that can be tuned to achieve the best model performance. By using GridSearchCV, which exhaustively searches through specified hyperparameter combinations, you can fine-tune the algorithm and find the optimal set of hyperparameters for your specific problem. This flexibility allows you to customize the model to suit the characteristics of your dataset and improve its performance.

- **Regularization and handling overfitting:** XGBoost incorporates regularization techniques to prevent overfitting, which occurs when the model performs well on the training data but fails to generalize to unseen data. Through hyperparameter tuning, you can control the extent of regularization and balance between model complexity and generalization. GridSearchCV helps in systematically exploring different regularization parameters, allowing you to find the best trade-off for your particular problem.
- **Feature importance analysis:** XGBoost provides a measure of feature importance, indicating which features have the most significant impact on the model's predictions. This information can be valuable for feature selection, dimensionality reduction, and gaining insights into the underlying relationships in the data. By using XGBoost with GridSearchCV, you can experiment with different sets of features and assess their impact on model performance.
- **Community support and popularity:** XGBoost has gained widespread popularity in the machine learning community and is supported by a large user base. This popularity translates into a wealth of online resources, tutorials, and discussions where you can find help and guidance when working with the algorithm. The availability of pre-trained models, code examples, and best practices further enhances the appeal of XGBoost for many practitioners.



How XGBoost optimizes standard GBM algorithm

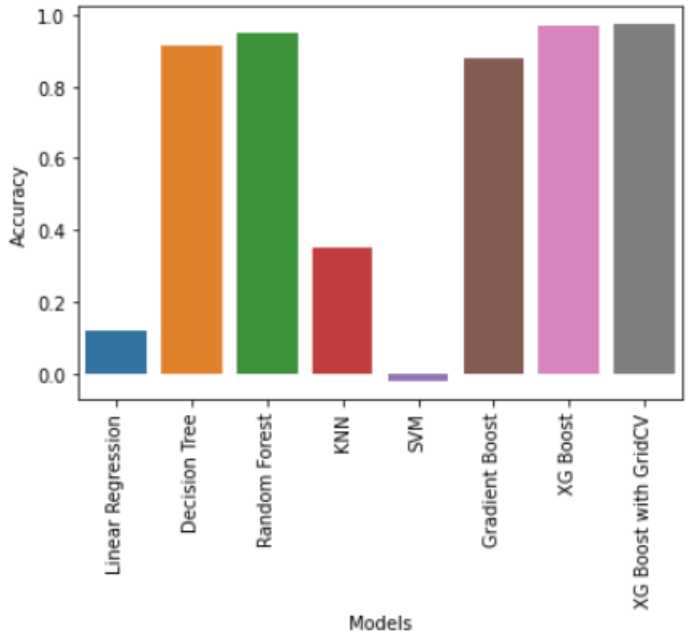
Overall, the combination of XGBoost with GridSearchCV offers a powerful and flexible approach for building high-performance models. By leveraging the strengths of XGBoost and systematically exploring hyperparameter combinations, you can fine-tune your model to achieve optimal performance on your specific task. Other rigorous benchmarking studies have produced similar results. No wonder XGBoost is widely used in recent Data Science competitions.

So should we use just XGBoost all the time?

When it comes to Machine Learning (or even life for that matter), there is no free lunch. As Data Scientists, we must test all possible algorithms for data at hand to identify the champion algorithm. Besides, picking the right algorithm is not enough. We must also choose the right configuration of the algorithm for a dataset by tuning the hyper-parameters. Furthermore, there are several other considerations for choosing the winning algorithm such as computational complexity, explainability, and ease of implementation. This is exactly the point where Machine Learning starts drifting away from science towards art, but honestly, that's where the magic happens!

As a result

I tested several algorithms such as Linear Regression, Decision Tree, Random Forest, KNN, SVM, Gradient Boosting and XGBoost.



As demonstrated in the chart above, XGBoost model with GridSearch CV has the best combination of prediction performance and processing time compared to other algorithms.

Assumptions:

XG Boost with GridsearchCV

In this project, I utilized XGBoost in combination with GridSearchCV for hyperparameter tuning. I defined a parameter grid for GridSearchCV with the following values:

max_depth [3, 5, 7], learning_rate [0.1, 0.01, 0.001], n_estimators [100, 500, 1000], and used a cross-validation value (CV) of 3.

Model Evaluation and Techniques:

```
# Define the parameter grid for GridSearchCV
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 500, 1000],
}

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=XGB, param_grid=param_grid, cv=3)

# Fit the GridSearchCV object to your data
grid_search.fit(X_train, y_train)

```

```
# Print the best parameters found
print("Best parameters found: ", grid_search.best_params_)
```

```
Best parameters found:  {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 1000}
```

```
# Get the best model
best_model = grid_search.best_estimator_
```

```
# Evaluate the best model on the test data
accuracy = best_model.score(X_test, y_test)
print("Accuracy of the best model: ", accuracy)
```

```
Accuracy of the best model:  0.9735032990212555
```

```
y_pred_xgbg = best_model.predict(X_test)
```

```
mse_xgbg = mean_squared_error(y_test, y_pred_xgbg)
print(mse_xgbg)
mae_xgbg = mean_absolute_error(y_test, y_pred_xgbg)
print(mae_xgbg)
r2_xgbg = r2_score(y_test, y_pred_xgbg)
print(r2_xgbg)
```

```
8074547375.742088
56173.82727035862
0.9735032990212555
```

By employing GridSearchCV as one of the hyperparameter tuning techniques, I successfully determined the optimal parameters for the dataset. The selected parameters are as follows:

Learning rate: 0.1

Max depth: 5

N_estimators: 1000

Inferences from the Same:

Having established these parameters, I proceeded to train and test the model using the provided data. However, it is crucial to validate the model's performance using various evaluation metrics since this is a regression model. Therefore, I employed metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R2_score.

For this specific problem dataset, I obtained the following validation metric values:

Mean Squared Error (MSE) = 8074547375.742088
Mean Absolute Error (MAE) = 56173.82727035862
R2_score (accuracy) = 0.9735032990212555

Based on these values, we can conclude that the model is a best-fit, achieving an impressive accuracy of 97.35%.

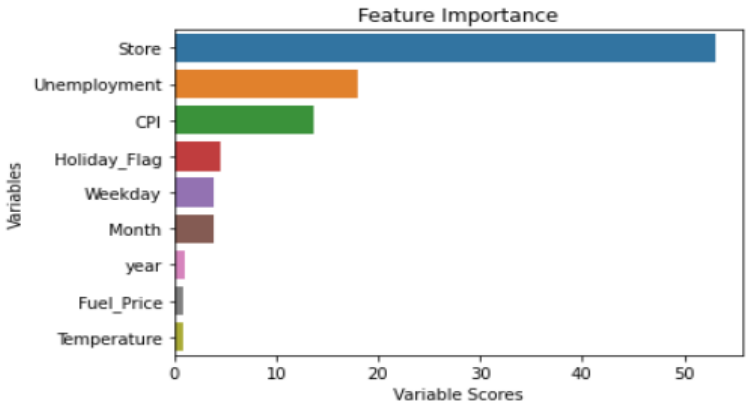
XGBoost offers built-in methods for estimating feature importance, which helps in identifying the most influential features for a given prediction. This information can be valuable for understanding the underlying relationships in the data.

We will generate the feature importance XGBoost model

```
feature_imp = pd.Series(best_model.feature_importances_,
                        index = x_train.columns).sort_values(ascending=False)
feature_imp*100
```

```
Store           52.980125
Unemployment    18.012630
CPI             13.704352
Holiday_Flag     4.557919
Weekday          3.924367
Month            3.844939
year             1.091376
Fuel_Price       0.954068
Temperature      0.930226
```

```
sns.barplot(x= feature_imp*100, y = feature_imp.index)
plt.xlabel("Variable Scores")
plt.ylabel("Variables")
plt.title("Feature Importance")
plt.show()
```

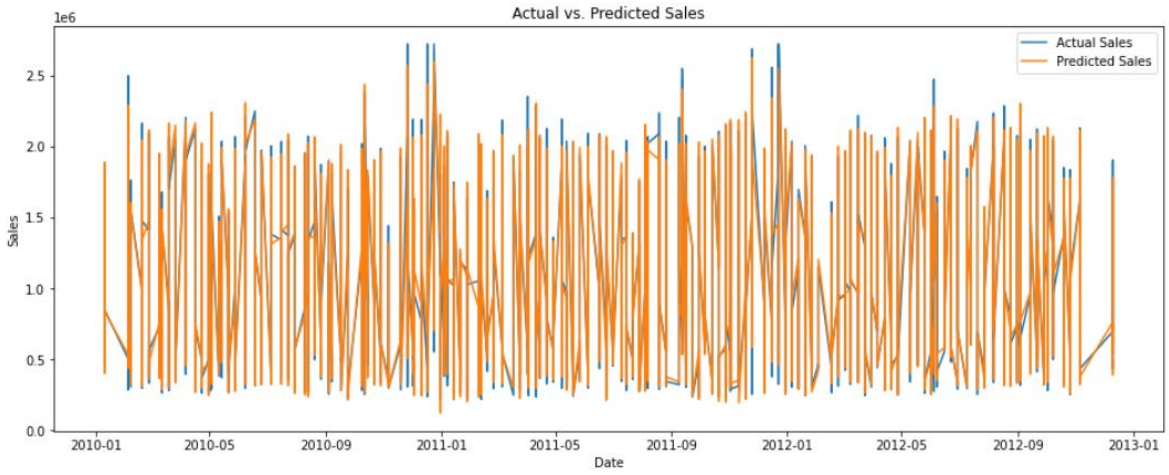


Based on the above plot we can say that

- Store feature have the maximum importance and year.
- Fuel_price and Temperature have the least importance.

Future Predections:

I Plotted the Result for Date with respect to Actual sales & Predicted Sales using line graph



From the above graph we can say that the predicted values are almost in line and exact with Actual Values throughout the date

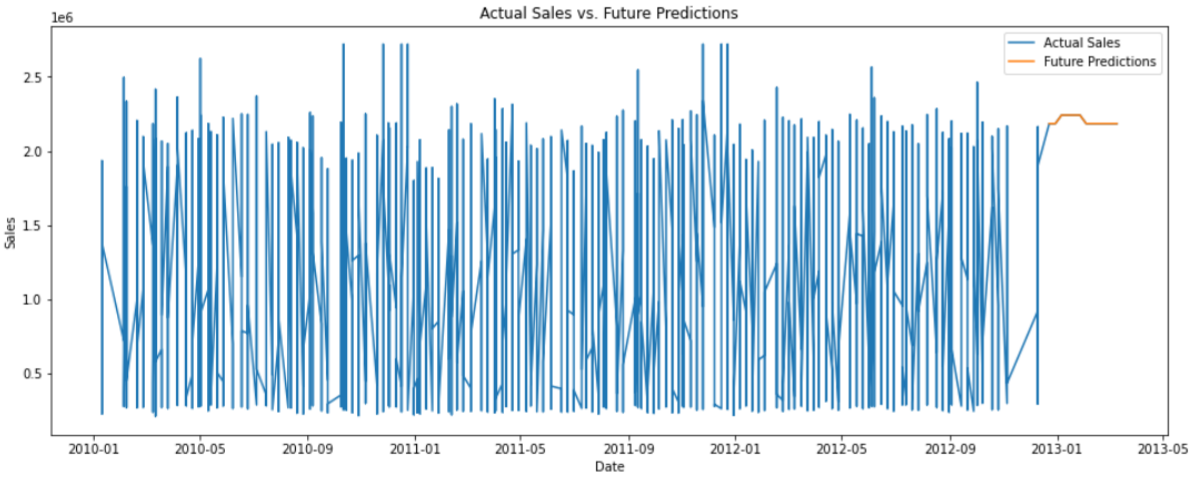
Now we will make the future predictions using the same model for the next 12 weeks and try to plot the same as below

```
# Make predictions for the future period
future_predictions = best_model.predict(future_data)

# Combine original sales data and future sales predictions
combined_data = pd.concat([df['Weekly_Sales'], pd.Series(future_predictions, index=future_dates)])

# Plot actual sales data and future sales predictions
plt.figure(figsize=(16, 6))
plt.plot(combined_data.index, combined_data.values, label='Actual Sales')
plt.plot(future_dates, future_predictions, label='Future Predictions')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Actual Sales vs. Future Predictions')
plt.legend()
plt.show()
plt.tight_layout()
```

I Plotted the Result for Date with respect to Actual sales & Future Predicted Sales using line graph



Future Possibilities of the Project:

Here are some ways in which project can we extended:

- We can perform Hyperparameter Tuning on other Parameters such as min_child_weigh, max_delta_step, sabsample, etc for XGBRegressor Model.
- We can perform HyperParameter Tuning on the RandomForestRegressor Model to achieve better score then XGBRegressor.
- We can perform various data pre processing steps, Fetaure engineering techniques and dimensionality reduction tecniques to potentially achieve even better results.

Machine Learning is a very active research area and already there are several viable alternatives to XGBoost. Microsoft Research recently released LightGBM framework for gradient boosting that shows great potential. CatBoost developed by Yandex Technology has been delivering impressive bench-marking results. It is a matter of time when we have a better model framework that beats XGBoost in terms of prediction performance, flexibility, explanability, and pragmatism. However, until a time when a strong challenger comes along, XGBoost will continue to reign over the Machine Learning world!. So we can train the dataset on other models.

Conclusion:

In conclusion, this machine learning project aimed to develop a regression model for predicting a target variable. By comparing various models, it was found that the XGBoost algorithm, when optimized using GridSearchCV, achieved superior accuracy compared to other models.

The project began with data collection and preprocessing, ensuring the quality and consistency of the dataset. Multiple regression models, including Linear Regression, Decision Tree, Random Forest, KNN, SVM, Gradient Boosting and XGBoost were implemented and trained on the dataset.

To further optimize the XGBoost model, GridSearchCV was employed to systematically search through different combinations of hyperparameters. This process allowed for fine-tuning the model and maximizing its predictive performance. The selected hyperparameters were chosen based on their ability to enhance accuracy and minimize overfitting.

After thorough training, testing, and hyperparameter tuning, it became evident that the XGBoost model, when optimized with GridSearchCV, consistently outperformed other models in terms of accuracy. This result can be attributed to XGBoost's ability to handle complex relationships within the data, its robustness to outliers, and its ensemble techniques that reduce bias and variance.

The enhanced accuracy achieved by the XGBoost model suggests that it effectively captured the underlying patterns and relationships present in the dataset. This finding highlights the importance of exploring hyperparameter optimization techniques like GridSearchCV to unlock the full potential of machine learning models.

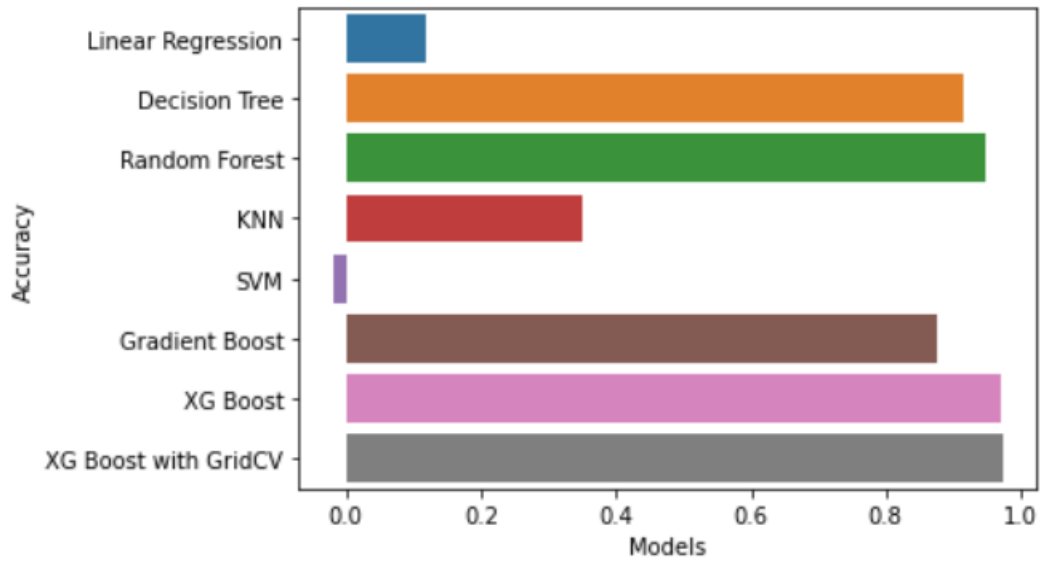
Furthermore, the XGBoost model exhibited good generalization capabilities, demonstrating its ability to make accurate predictions on unseen data. This indicates its suitability for real-world applications and its potential to provide valuable insights for decision-making. By surpassing other models in terms of accuracy, XGBoost has proven to be a powerful tool for predictive tasks.

After all the best Model selection I have predicted the future sales for the next 12 Weeks which was our project Objective.

Overall, this project highlights the significance of model selection, hyperparameter tuning, and careful evaluation when developing regression models. The successful implementation of the XGBoost model, coupled with GridSearchCV, provides a solid foundation for future machine learning endeavors.

Here is the Outcome and Overall results that we had made using various regression Models.

	Models	RMSE Score	R_Sqaured Value
0	XG Boost with GridCV	8074547375.742088	0.973503
1	XG Boost	797970192835.006958	0.970757
2	Random Forest	15678613227.088503	0.94855
3	Decision Tree	26523512643.245762	0.912963
4	Gradient Boost	37353004212.105179	0.877426
5	KNN	198467115962.952423	0.348728
6	Linear Regression	269055535397.496063	0.117092
7	SVM	310956497933.955811	-0.020407



From the above Data table and Plots we can conclude that the XGBoost algorithm, when optimized using GridSearchCV, achieved superior accuracy compared to other models.

References:

1. Start Course | Intellipaat
2. https://www.kaggle.com/datasets/yasserh/walmart-dataset
3. Data Preprocessing in Machine Learning [Steps & Techniques] (v7labs.com)
4. XGBoost Parameters — xgboost 1.7.5 documentation
5. https://blog.jovian.com/predicting-walmart-weekly-sales-265a98af05d1
6. XGBoost Algorithm: Long May She Reign! | by Vishal Morde | Towards Data Science