Walmart Store Sales Forecasting

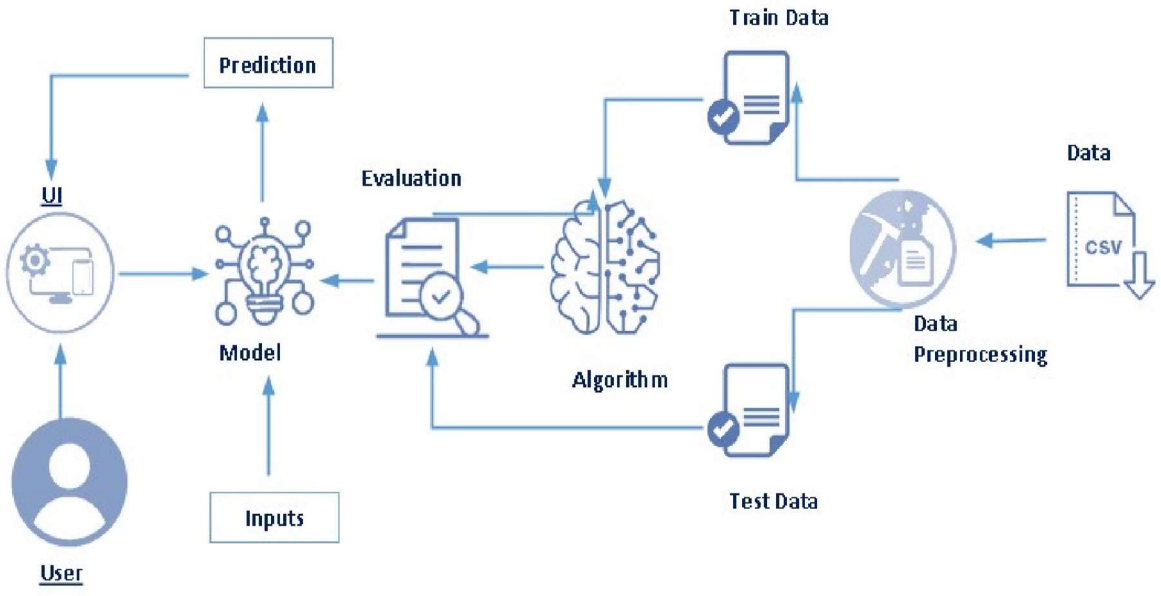
Sales forecasting is a critical process integral to anticipating future sales accurately, empowering companies like Walmart to make well-informed business decisions and project both short-term and long-term performance. The methodology employed for such forecasts typically involves leveraging historical sales data, industry-wide benchmarks, and prevailing economic trends. In the context of this analysis, Walmart, a prominent retail corporation operating a chain of hypermarkets, has provided comprehensive data encompassing 45 stores, inclusive of store details and monthly sales information.

Walmart strategically orchestrates promotional markdown events throughout the year, notably preceding significant holidays such as the Super Bowl, Labor Day, Thanksgiving, and Christmas. It is noteworthy that during the evaluation of sales data, the weeks coinciding with these major holidays carry a weight five times higher than non-holiday weeks. The dataset is structured on a weekly basis.

The primary objective of this analysis is to discern the impact of holidays, specifically Christmas, Thanksgiving, Super Bowl, and Labor Day, on store sales. To achieve this, advanced algorithms, including ARIMA, Random Forest, and XgBoost, will be employed. The data will be systematically split into training and testing sets to assess the performance of each algorithm.

Furthermore, the integration of Flask for streamlined web application development and the deployment of the model using IBM infrastructure will be seamlessly executed. This comprehensive approach ensures a robust analysis of holiday impacts on Walmart store sales, coupled with a sophisticated and user-friendly interface for potential applications in real-world scenarios.

Architecture



Prerequisites:

The successful completion of this project requires a foundational understanding of the following software, concepts, and packages:

1. **Anaconda Navigator:**

Download and install Anaconda Navigator by referring to the link provided below:

https://www.youtube.com/watch?v=5mDYijMfSzs

2. **Python Packages:**

Open the Anaconda Prompt as an administrator.

Install the necessary Python packages by executing the following commands:

- pip install numpy and press enter.
- pip install pandas and press enter.
- pip install scikit-learn and press enter.
- pip install matplotlib and press enter.
- pip install seaborn and press enter.
- pip install pmdarima and press enter.

These packages are essential for data manipulation, analysis, and visualization, as well as for implementing machine learning algorithms in the subsequent phases of the project. Ensuring the availability of these tools and packages will contribute to a seamless execution of the analysis and model training processes.

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:

By the culmination of this project, participants will achieve the following objectives:

1. **Fundamental Understanding of Machine Learning:**

Attain knowledge of fundamental concepts and techniques employed in machine learning.

2. **Comprehensive Data Insight:**

Develop a broad understanding of data, encompassing its nuances and significance.

3. **Data Pre-processing Proficiency:**

Acquire expertise in pre-processing data, including transformation techniques for handling outliers, and gain familiarity with visualization concepts.

Project Flow:

1. **Data Loading and Cleaning:**

Load the dataset and perform cleaning operations, including feature extraction.

2. **Model Training:**

Utilize machine learning models to train on the prepared dataset.

3. **Testing and Evaluation:**

Test the trained model and conduct a comprehensive evaluation of its performance.

Activities to Accomplish:

1. **Data Collection:**

Collect the dataset or create a dataset for analysis.

2. **Visualizing and Analyzing Data:**

Perform descriptive analysis on the dataset.

3. **Data Pre-processing:**

- Check for null values within the dataset.
- Implement null value filling strategies.
- Split the data into training and testing sets.

4. **Model Building:**

- Import the necessary libraries for building machine learning models.
- Initialize the selected model for the analysis.
- Train and test the model using the prepared dataset.
- Evaluate the performance of the model.

5. **Model Persistence:**

Save the trained model for potential future use.

This structured approach ensures a step-by-step progression through the essential phases of a machine learning project, fostering a comprehensive understanding of data manipulation, model building, and performance evaluation.

Project Structure :

Create the Project folder which contains files as shown below

For the development of a Flask application, it is imperative to organize HTML pages within the designated "templates" folder and script the application using a Python script, denoted as "app.py." The saved model, denoted as "Final.model.pkl," will be integrated into the Flask application to facilitate predictive functionalities.

The dataset folder comprises several CSV files, each serving a distinct purpose:

1. **train.csv:**

This dataset encapsulates historical training data, spanning from February 5, 2010, to November 1, 2012. It includes the following fields:

- Store: Store number
- Dept: Department number
- Date: Week
- Weekly_Sales: Sales for the specified department in the designated store
- IsHoliday: Indicator of whether the week is a special holiday week

2. **test.csv:**

This file mirrors the structure of train.csv, excluding the weekly sales column. Importantly, this dataset will not be utilized in this project. Instead, all other datasets will be merged to create a consolidated dataset for both training and testing purposes.

3. **features.csv:**

This dataset incorporates additional information pertaining to the store, department, and regional activity for specified dates. Key fields within this dataset include:

- Store: Store number
- Date: Week
- Temperature: Average temperature in the region

- Fuel_Price: Cost of fuel in the region
- MarkDown1–5: Anonymized data associated with promotional markdowns executed by Walmart. MarkDown data is available only post-November 2011 and may not be universally accessible for all stores at all times. Missing values are denoted as "NA."
- CPI: Consumer Price Index
- Unemployment: Unemployment rate
- IsHoliday: Indication of whether the week is a special holiday week

Notably, the considered holidays encompass Super Bowl, Labor Day, Thanksgiving, and Christmas, with corresponding dates specified.

The structural organization of the project involves the creation of HTML pages stored in the "templates" folder, complemented by the scripting within "app.py." Integration of the pre-trained model, denoted as "Final.model.pkl," will be implemented for seamless incorporation into the Flask application. The dataset, comprising various CSV files, will be strategically merged to form a comprehensive dataset for subsequent training and testing procedures. This meticulous approach ensures a formal and organized framework for the development of the Flask application.

Data Collection:

In machine learning, the significance of data cannot be overstated, as it forms the cornerstone for training algorithms effectively. This section is dedicated to facilitating the download of the requisite dataset, a pivotal step enabling the training of machine learning algorithms.

To procure the dataset, various reputable open sources are available, such as Kaggle.com, the UCI repository, among others. For this specific project, the dataset is encapsulated within the "Walmart-dataset.zip" archive, containing essential files such as "features.csv," "stores.csv," "train.csv," and "test.csv." This dataset has been sourced from Kaggle.com, a renowned platform for data science competitions and resources. Interested individuals are encouraged to utilize the provided link for dataset acquisition.

Link: https://www.kaggle.com/code/caesarlupum/walmart-store-sales-forecasting-anomaly-analysis/input

Ensuring access to a reliable and comprehensive dataset is paramount for the successful implementation of machine learning algorithms. Therefore, interested parties are advised to follow the provided link for a seamless download of the Walmart dataset, which is integral to the subsequent phases of this project.

Visualizing and Analyzing the Data:

Following the successful download of the dataset, the next imperative step involves a comprehensive exploration and understanding of the data through the application of visualization and analysis techniques. This phase is instrumental in unveiling insights and patterns inherent in the dataset.

Various techniques exist for gaining a nuanced understanding of data, and the selected ones for this project are outlined below. It is important to note that while these techniques are effective, the application of additional and diverse methods is encouraged for a thorough exploration:

1. **Visualization Techniques**:

- Utilize graphical representations to enhance comprehension.
- Employ techniques such as histograms, box plots, and scatter plots for univariate and multivariate exploration.
- Leverage seaborn and matplotlib libraries for creating visually informative charts.

2. **Analysis Techniques:**

- Conduct descriptive analysis to unveil fundamental statistics and properties of the dataset.
- Explore the distribution of key variables and assess their central tendencies.
- Identify potential outliers and anomalies that may influence subsequent analyses.

This phase serves as a critical foundation for the ensuing steps in the project. By employing these visualization and analysis techniques, a nuanced understanding of the dataset is achieved, facilitating informed decisions regarding data pre-processing and subsequent modeling. It is important to acknowledge that the exploration process is dynamic, and additional techniques can be incorporated based on the specific characteristics and requirements of the dataset.

Importing the libraries

Import the necessary libraries as shown in the image.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import pyplot

import plotly.graph_objs as go
import plotly as py
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

```
import warnings
import pandas as pd

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', 150)
pd.set_option('display.max_rows', 150)
```

Reviewing the Dataset

The dataset may be presented in various formats such as .csv, Excel files, .txt, .json, and so forth. Consequently, the dataset can be effectively parsed through the utilization of the pandas library.

Within the pandas library, a dedicated function, namely read_csv(), serves the purpose of reading datasets. As a requisite parameter, the function necessitates the specification of the directory wherein the CSV file is stored.

It is noteworthy that irrespective of the dataset format, a uniform approach is employed for reading all datasets.

```
[ ]  features = pd.read_csv('/content/drive/MyDrive/features.csv')
     train = pd.read_csv('/content/drive/MyDrive/train.csv')
     test = pd.read_csv('/content/drive/MyDrive/test.csv')
     stores = pd.read_csv('/content/drive/MyDrive/stores.csv')
     sample_submission = pd.read_csv('/content/drive/MyDrive/sample_submission.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

```
features.head()
```

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

```
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

```
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

Descriptive analysis

Descriptive analysis entails a comprehensive examination of fundamental data attributes through statistical procedures. In this context, the pandas library encompasses a valuable function known as describe(). By employing the describe() function, a profound understanding of categorical features is attainable, elucidating unique, predominant, and frequently occurring values. Furthermore, for continuous features, this function facilitates the derivation of essential statistical metrics such as mean, standard deviation, minimum, maximum, and percentile values.

```
%%time
features.describe()
```

```
CPU times: user 27.9 ms, sys: 0 ns, total: 27.9 ms
Wall time: 34.7 ms
```

	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
count	8190.000000	8190.000000	8190.000000	4032.000000	2921.000000	3613.000000	3464.000000	4050.000000	7605.000000	7605.000000
mean	23.000000	59.356198	3.405992	7032.371786	3384.176594	1760.100180	3292.935886	4132.216422	172.460809	7.826821
std	12.987966	18.678607	0.431337	9262.747448	8793.583016	11276.462208	6792.329861	13086.690278	39.738346	1.877259
min	1.000000	-7.290000	2.472000	-2781.450000	-265.760000	-179.260000	0.220000	-185.170000	126.064000	3.684000
25%	12.000000	45.902500	3.041000	1577.532500	68.880000	6.600000	304.687500	1440.827500	132.364839	6.634000
50%	23.000000	60.710000	3.513000	4743.580000	364.570000	36.260000	1176.425000	2727.135000	182.764003	7.806000
75%	34.000000	73.880000	3.743000	8923.310000	2153.350000	163.150000	3310.007500	4832.555000	213.932412	8.567000
max	45.000000	101.950000	4.468000	103184.980000	104519.540000	149483.310000	67474.850000	771448.100000	228.976456	14.313000

```
[ ] %%time
    train.describe()
```

```
CPU times: user 41.3 ms, sys: 8.01 ms, total: 49.3 ms
Wall time: 54.2 ms
```

	Store	Dept	Weekly_Sales
count	421570.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123
std	12.785297	30.492054	22711.183519
min	1.000000	1.000000	-4988.940000
25%	11.000000	18.000000	2079.650000
50%	22.000000	37.000000	7612.030000
75%	33.000000	74.000000	20205.852500
max	45.000000	99.000000	693099.360000

```
[ ] %%time
    stores.describe()
```

```
CPU times: user 4.83 ms, sys: 4.2 ms, total: 9.03 ms
Wall time: 8.65 ms
```

	Store	Size
count	45.000000	45.000000
mean	23.000000	130287.600000
std	13.133926	63825.271991
min	1.000000	34875.000000
25%	12.000000	70713.000000
50%	23.000000	126512.000000
75%	34.000000	202307.000000
max	45.000000	219622.000000

Data Pre-processing

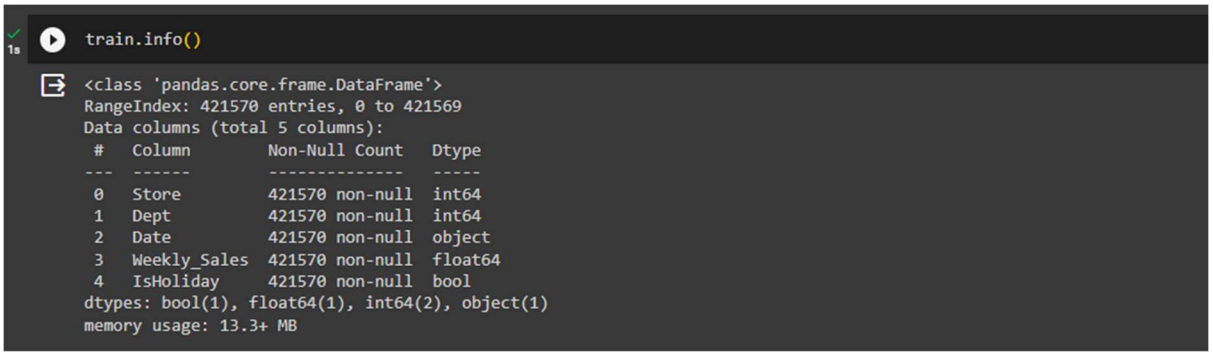
Upon comprehending the data description, it is imperative to engage in the pre-processing of the acquired dataset. The downloaded dataset, in its current form, may exhibit considerable randomness, rendering it unsuitable for training a machine learning model. Consequently, meticulous cleaning of the dataset is imperative to yield optimal results. This undertaking encompasses a series of systematic steps:

- Handling Missing Values
- Managing Categorical Data
- Addressing Outliers
- Implementing Scaling Techniques
- Segregating the Dataset into Training and Test Sets

It is essential to acknowledge that these outlined steps represent general pre-processing procedures for data preceding its utilization in machine learning endeavors. Depending on the specific characteristics of the dataset in question, it may be necessary to selectively apply these steps in accordance with the unique conditions of the data.

Checking for null values

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



```
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     4032 non-null   float64
 5   MarkDown2     2921 non-null   float64
 6   MarkDown3     3613 non-null   float64
 7   MarkDown4     3464 non-null   float64
 8   MarkDown5     4050 non-null   float64
 9   CPI           7605 non-null   float64
 10  Unemployment  7605 non-null   float64
 11  IsHoliday     8190 non-null   bool   
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 712.0+ KB
```

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115064 entries, 0 to 115063
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   Store      115064 non-null  int64 
 1   Dept       115064 non-null  int64 
 2   Date       115064 non-null  object
 3   IsHoliday  115064 non-null  bool  
dtypes: bool(1), int64(2), object(1)
memory usage: 2.7+ MB
```

For checking the null values, .isnull() function is used. To sum those null values.sum() function is used with it

```
[14] train.isnull().sum()
```

```
Store           0
Dept            0
Date            0
Weekly_Sales    0
IsHoliday       0
dtype: int64
```

```
[15] features.isnull().sum()
```

```
Store             0
Date              0
Temperature       0
Fuel_Price        0
MarkDown1      4158
MarkDown2      5269
MarkDown3      4577
MarkDown4      4726
MarkDown5      4140
CPI             585
Unemployment    585
IsHoliday         0
dtype: int64
```

```
test.isnull().sum()
```

```
Store        0
Dept         0
Date         0
IsHoliday    0
dtype: int64
```

Since the data is divided into separate files, these files will first be merged into a single file and then further analysis will be performed on it.

We'll merge data: features and stores have a commom key 'Stores' we can create a new data using the keys 'Store', 'Dept' and 'IsHoliday'. The data will be loaded into 'dataset, dataset_test'.

```
[18] dataset = train.merge(stores, how='left').merge(features, how='left')
     dataset.shape

     (421570, 16)
```

Missing data is managed through the strategic application of zero values, accomplished by utilizing either the .fillna() or the .replace() function. This approach ensures a systematic handling of absent data points within the dataset, contributing to a comprehensive and robust pre-processing methodology.

Handling Negative Values

```
dataset.describe()
```

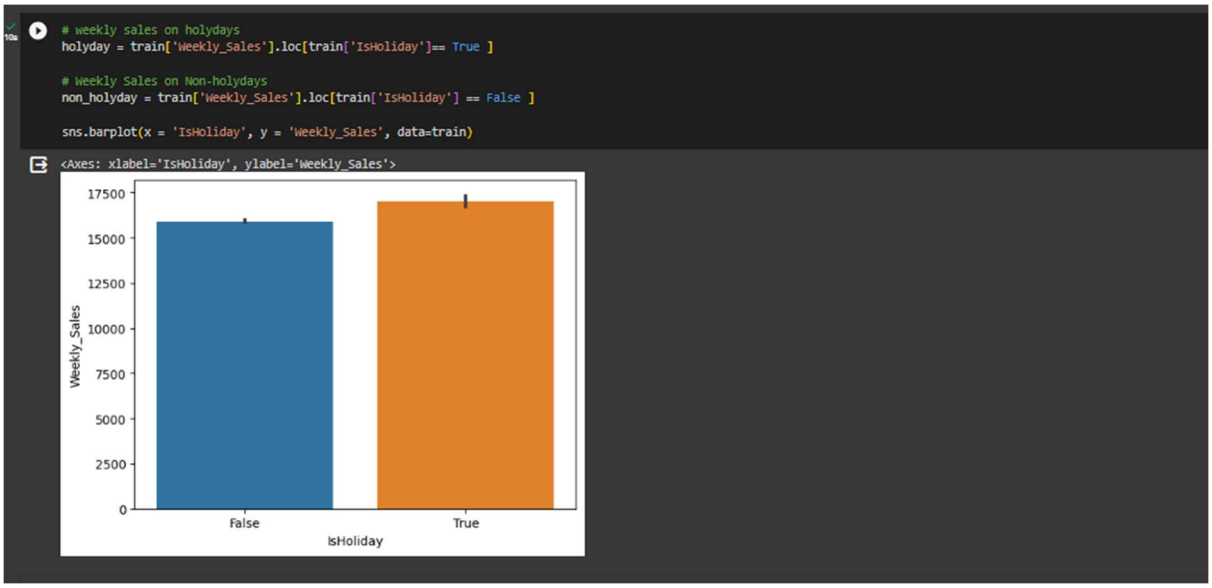
	Store	Dept	Weekly_Sales	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	150681.000000	111248.000000	137091.000000	134967.000000	151432.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123	136727.915739	60.090059	3.361027	7246.420196	3334.628621	1439.421384	3383.168256	4628.975079	171.201947	7.960289
std	12.785297	30.492054	22711.183519	60980.583328	18.447931	0.458515	8291.221345	9475.357325	9623.078290	6292.384031	5962.887455	39.159276	1.863296
min	1.000000	1.000000	-4988.940000	34875.000000	-2.060000	2.472000	0.270000	-265.760000	-29.100000	0.220000	135.160000	126.064000	3.879000
25%	11.000000	18.000000	2079.650000	93638.000000	46.680000	2.933000	2240.270000	41.600000	5.080000	504.220000	1878.440000	132.022667	6.891000
50%	22.000000	37.000000	7612.030000	140167.000000	62.090000	3.452000	5347.450000	192.000000	24.600000	1481.310000	3359.450000	182.318780	7.866000
75%	33.000000	74.000000	20205.852500	202505.000000	74.280000	3.738000	9210.900000	1926.940000	103.990000	3595.040000	5563.800000	212.416993	8.572000
max	45.000000	99.000000	693099.360000	219622.000000	100.140000	4.468000	88646.760000	104519.540000	141630.610000	67474.850000	108519.280000	227.232807	14.313000

Upon inspection of the merged dataset, it has come to attention that within the "Weekly_Sales" column, the minimum sales value is recorded as negative. Given the inherent nature of sales data, which inherently cannot be negative, it is imperative to rectify this discrepancy. Consequently, a corrective measure has been implemented to exclusively consider positive values within the "Weekly_Sales" column, as delineated below.

Exploratory Data Analysis

Upon reviewing the description of the stores dataset, it is evident that there exist three distinct types of stores: Type A, Type B, and Type C. With a total of 45 stores under consideration, a visual representation in the form of a pie chart will be employed to delineate the proportional distribution of each store type. This graphical representation will facilitate a clearer understanding of the percentage distribution across the various store types within the dataset.

check the weekly sales on holidays and non-holidays



Additionally, we will generate a correlation matrix to elucidate the relationships that exist among the diverse features. Correlation, as a bivariate analysis, quantifies the strength (ranging from +1 to –1) of the association between two variables, along with its direction. The direction of the relationship is denoted by the presence of a positive or negative sign; a positive sign signifies a positive relationship, while a negative sign implies a negative relationship. This correlation matrix will serve as a valuable analytical tool to discern the interdependencies among the variables, providing insights into the nature and magnitude of their associations.

Correlation

Now that we have dealt with the categorical variables and the outliers, let's continue with the EDA. One way to try and understand the data is by looking for correlations between the features and the target. We can calculate the Pearson correlation coefficient between every variable and the target using the .corr dataframe method.

The correlation coefficient is not the greatest method to represent "relevance" of a feature, but it does give us an idea of possible relationships within the data. Some general interpretations of the absolute value of the correlation coefficent are:

.00-.19 "very weak"

.20-.39 "weak"

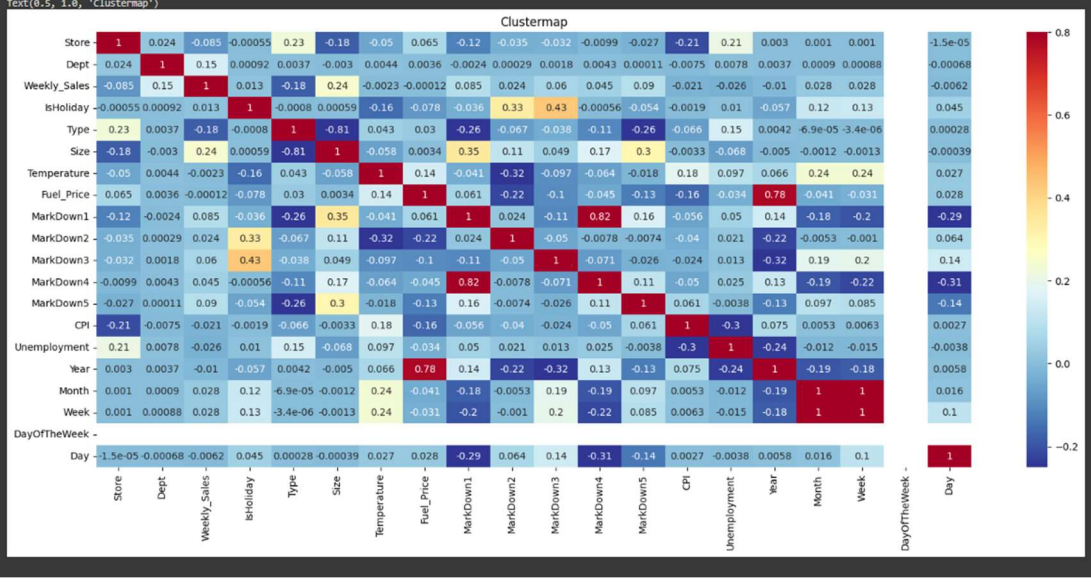
.40-.59 "moderate"

.60-.79 "strong"

.80-1.0 "very strong"

```
%%time
corrs = dataset.corr()
plt.figure(figsize = (20, 8))
# Heatmap of correlations
sns.heatmap(corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = True, vmax = 0.8)
plt.title('Clustermap');
```

```
CPU times: user 1.16 s, sys: 31.2 ms, total: 1.19 s
Wall time: 3.16 s
Text(0.5, 1.0, 'Clustermap')
```



We calculate now the correlations between the features in dataset. The following table shows the first 15 the least correlated features.

```
%%time
features = dataset.columns.values
corrs_ = dataset[features].corr().abs().unstack().sort_values(kind="quicksort").reset_index()
corrs_ = corrs_[corrs_['level_0'] != corrs_['level_1']]
corrs_.head(15).T
```

```
CPU times: user 458 ms, sys: 28.1 ms, total: 486 ms
Wall time: 494 ms
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
level_0	Type	Week	Day	Store	Type	Month	MarkDown5	Dept	Fuel_Price	Weekly_Sales	Type	Day	MarkDown2	Dept	Day
level_1	Week	Type	Store	Day	Month	Type	Dept	MarkDown5	Weekly_Sales	Fuel_Price	Day	Type	Dept	MarkDown2	Size
0	0.000003	0.000003	0.000015	0.000015	0.000069	0.000069	0.000109	0.000109	0.00012	0.00012	0.000279	0.000279	0.00029	0.00029	0.000386

Let's look to the top 15 most correlated features, besides the same feature pairs.

```
corrs_.tail(15).T
```

	384	385	386	387	388	389	390	391	392	393	394	395	396	398	399
level_0	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	DayOfTheWeek	Day
level_1	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Year	Month	Week	Day	DayOfTheWeek
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Handling Categorical Values

Evidently, the dataset incorporates categorical data necessitating transformation from categorical representations to either integer encoding or binary encoding. Initially, the categorical values within the "Type" column, representing store types, are converted into indicator variables through the utilization of the get_dummies() function. This procedure facilitates a structured and numerical representation of categorical information, laying the groundwork for further analytical processes.

```
[29] dataset = pd.get_dummies (dataset, columns = ['Type' ] )
```

Subsequently, the "date" column undergoes a dissection into distinct components, namely date, month, year, and day of the week. This segmentation serves to extract valuable temporal information, enabling a more nuanced exploration of the dataset with a finer granularity in terms of time-related features.

```
dataset['Date'] = pd.to_datetime(dataset['Date'])
dataset['month'] = dataset['Date'].dt.month
dataset['Year'] = dataset['Date'].dt.year
```

```
[31] dataset[['Date','month', 'Year']].head()
```

	Date	month	Year
0	2010-02-05	2	2010
1	2010-02-12	2	2010
2	2010-02-19	2	2010
3	2010-02-26	2	2010
4	2010-03-05	3	2010

```
dataset['dayofweek_name'] = dataset['Date'].dt.day_name()
dataset[['Date', 'dayofweek_name']].head()
```

	Date	dayofweek_name
0	2010-02-05	Friday
1	2010-02-12	Friday
2	2010-02-19	Friday
3	2010-02-26	Friday
4	2010-03-05	Friday

```
dataset['is_weekend'] = np.where(dataset['dayofweek_name'].isin(['Sunday', 'Saturday']),1,0)
dataset[['Date', 'is_weekend']].head()
```

	Date	is_weekend
0	2010-02-05	0
1	2010-02-12	0
2	2010-02-19	0
3	2010-02-26	0
4	2010-03-05	0

Lastly, the IsHoliday values are changed from Boolean True and False to integer 1 and 0

```
dataset['IsHoliday'] = dataset['IsHoliday'].astype(int)
del dataset['dayofweek_name']
#del df['Date']
```

The initial five rows of the transformed dataset are displayed, followed by the preservation of the modified data in a new CSV format through the utilization of the .to_csv() function.

```
print(dataset.head())
```

```
   Store  Dept       Date  Weekly_Sales  IsHoliday    Size  Temperature  \
0      1     1 2010-02-05      24924.50          0  151315        42.31
1      1     1 2010-02-12      46039.49          1  151315        38.51
2      1     1 2010-02-19      41595.55          0  151315        39.93
3      1     1 2010-02-26      19403.54          0  151315        46.63
4      1     1 2010-03-05      21827.90          0  151315        46.50

   Fuel_Price  MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5  \
0       2.572        NaN        NaN        NaN        NaN        NaN
1       2.548        NaN        NaN        NaN        NaN        NaN
2       2.514        NaN        NaN        NaN        NaN        NaN
3       2.561        NaN        NaN        NaN        NaN        NaN
4       2.625        NaN        NaN        NaN        NaN        NaN

          CPI  Unemployment  Year  Month  Week  DayOfTheWeek  Day  Type_1  \
0  211.096358         8.106  2010      2     5             4    5       1
1  211.242170         8.106  2010      2     6             4   12       1
2  211.289143         8.106  2010      2     7             4   19       1
3  211.319643         8.106  2010      2     8             4   26       1
4  211.350143         8.106  2010      3     9             4    5       1

   Type_2  Type_3  month  year  is_weekend
0       0       0      2  2010           0
1       0       0      2  2010           0
2       0       0      2  2010           0
3       0       0      2  2010           0
4       0       0      3  2010           0
```

Splitting data into train and test

In order to segregate the dataset into distinct training and testing sets, the initially read file, encompassing the transformed data, serves as the precursor to this division.

```
[45] df = dataset

[46] print(df.columns)

Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday', 'Size',
       'Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3',
       'MarkDown4', 'MarkDown5', 'CPI', 'Unemployment', 'Year', 'Month',
       'Week', 'DayOfTheWeek', 'Day', 'Type_1', 'Type_2', 'Type_3', 'month',
       'year', 'is_weekend'],
      dtype='object')
```

Subsequently, the variables X and y are instantiated. Within the input variable, X is assigned by excluding the target variable. Correspondingly, the target variable, y, is assigned to encapsulate the specific feature of interest. The division of training and testing data is orchestrated through the application of the train_test_split() function, sourced from the sklearn library. The essential parameters encompass X, y, test_size, and random_state, thereby facilitating a systematic partitioning of the dataset.

```
[49] from sklearn.model_selection import train_test_split

[50] X = df.loc[: , df.columns != 'Weekly_Sales']
     y = df.loc[: , df.columns == 'Weekly_Sales']
     X = X[["Store", "Dept", "Size", "IsHoliday", "CPI", "Temperature", "Type_2", "Type_3", "MarkDown4", "month", "year"]]

     y = y.values.reshape(-1,1)
     print(X.head())

     X_train , X_test, y_train, y_test  = train_test_split(X, y, test_size = 0.2, random_state = 4)

        Store  Dept    Size  IsHoliday         CPI  Temperature  Type_2  Type_3  \
     0      1     1  151315          0  211.096358        42.31       0       0
     1      1     1  151315          1  211.242170        38.51       0       0
     2      1     1  151315          0  211.289143        39.93       0       0
     3      1     1  151315          0  211.319643        46.63       0       0
     4      1     1  151315          0  211.350143        46.50       0       0

        MarkDown4  month  year
     0        NaN      2  2010
     1        NaN      2  2010
     2        NaN      2  2010
     3        NaN      2  2010
     4        NaN      3  2010
```

It is specified that within the merged data file, the column denoted as "Weekly_Sales" serves as the target variable upon which predictions are to be made.

Model Building

With the completion of data cleaning, the subsequent phase involves model construction. The training of data is executed employing diverse algorithms. In the context of this project, the ARIMA time series forecasting algorithm is applied in conjunction with Random Forest and XGBoost methodologies

ARIMA Modeling

ARIMA, an acronym for AutoRegressive Integrated Moving Average, stands as a forecasting technique that predicts future values of a time series based solely on its inherent inertia. Notably, time series models, such as ARIMA, presume that the given dataset is stationary, indicating a constant mean and variance.

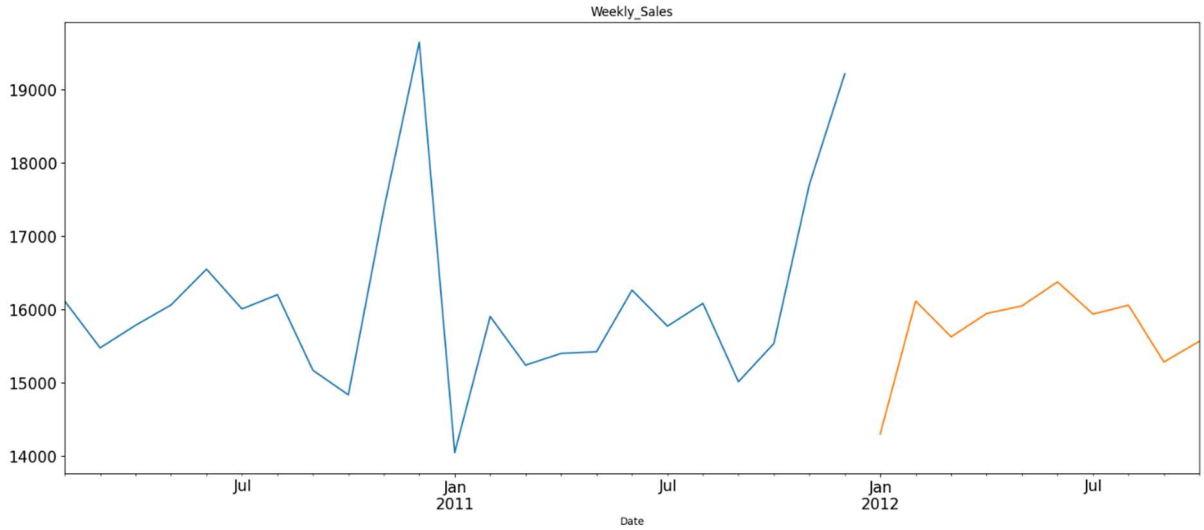
For the application of ARIMA, the determination of the triplet value (p, d, q) becomes imperative. Here, 'p' represents the number of auto-regressors, 'd' signifies the number of differences required to render the series stationary, and 'q' denotes the number of moving averages. Auto-ARIMA, a variant of ARIMA, automatically selects the optimal triplet (p, q, d) values based on the minimal Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) scores, ensuring the selection of the most fitting model.

```
[51] import pmdarima
     from pmdarima.arima import auto_arima
```

```
df.Date = pd.to_datetime(df.Date,format = '%Y-%m-%d')
df.index = df.Date
df = df.drop('Date', axis = 1)
df = df.resample('MS').mean()
# Resampling the time series data with month starting first
# Train-Test splitting of time series data
train_data = df[: int(0.7*(len(df)))]
test_data = df[int(0.7*(len(df))):]

train_data = train_data['Weekly_Sales']
test_data = test_data['Weekly_Sales']

# Plot the Weekly_Sales with respect to years in train and test
train_data.plot(figsize=(20,8), title = 'Weekly_Sales', fontsize = 15)
test_data.plot(figsize=(20,8), title = 'Weekly_Sales', fontsize = 15)
plt.show()
```




```
model_auto_arima = auto_arima(train_data, trace = True , error_action= 'ignore', suppress_warnings=True  )
model_auto_arima = auto_arima(train_data, trace = True , start_p = 0 , start_q = 0 , start_P =0 , start_Q = 0 , max_p = 10, max_q = 10, max_P = 10 , max_Q = 10 , seasonal = True , stepwise = False , suppress_warnings=True , D = 1 ,

model_auto_arima.fit(train_data)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=405.580, Time=0.32 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=398.029, Time=0.06 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=399.847, Time=0.10 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=399.604, Time=0.18 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=513.059, Time=0.02 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=401.739, Time=0.07 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0] intercept
Total fit time: 0.824 seconds
 ARIMA(0,0,0)(0,0,0)[1] intercept   : AIC=398.029, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[1] intercept   : AIC=399.604, Time=0.12 sec
 ARIMA(0,0,2)(0,0,0)[1] intercept   : AIC=inf, Time=0.42 sec
 ARIMA(0,0,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.46 sec
 ARIMA(0,0,4)(0,0,0)[1] intercept   : AIC=399.327, Time=0.48 sec
 ARIMA(0,0,5)(0,0,0)[1] intercept   : AIC=inf, Time=0.56 sec
 ARIMA(1,0,0)(0,0,0)[1] intercept   : AIC=399.847, Time=0.08 sec
 ARIMA(1,0,1)(0,0,0)[1] intercept   : AIC=401.739, Time=0.10 sec
 ARIMA(1,0,2)(0,0,0)[1] intercept   : AIC=404.410, Time=0.15 sec
 ARIMA(1,0,3)(0,0,0)[1] intercept   : AIC=400.484, Time=0.29 sec
 ARIMA(1,0,4)(0,0,0)[1] intercept   : AIC=399.861, Time=0.49 sec
 ARIMA(2,0,0)(0,0,0)[1] intercept   : AIC=399.812, Time=0.07 sec
 ARIMA(2,0,1)(0,0,0)[1] intercept   : AIC=403.489, Time=0.22 sec
 ARIMA(2,0,2)(0,0,0)[1] intercept   : AIC=405.580, Time=0.10 sec
 ARIMA(2,0,3)(0,0,0)[1] intercept   : AIC=402.551, Time=0.29 sec
 ARIMA(3,0,0)(0,0,0)[1] intercept   : AIC=401.000, Time=0.16 sec
 ARIMA(3,0,1)(0,0,0)[1] intercept   : AIC=403.156, Time=0.19 sec
 ARIMA(3,0,2)(0,0,0)[1] intercept   : AIC=405.238, Time=0.30 sec
 ARIMA(4,0,0)(0,0,0)[1] intercept   : AIC=402.909, Time=0.22 sec
 ARIMA(4,0,1)(0,0,0)[1] intercept   : AIC=405.118, Time=0.27 sec
 ARIMA(5,0,0)(0,0,0)[1] intercept   : AIC=404.374, Time=0.18 sec

Best model:  ARIMA(0,0,0)(0,0,0)[1] intercept
Total fit time: 5.207 seconds
```

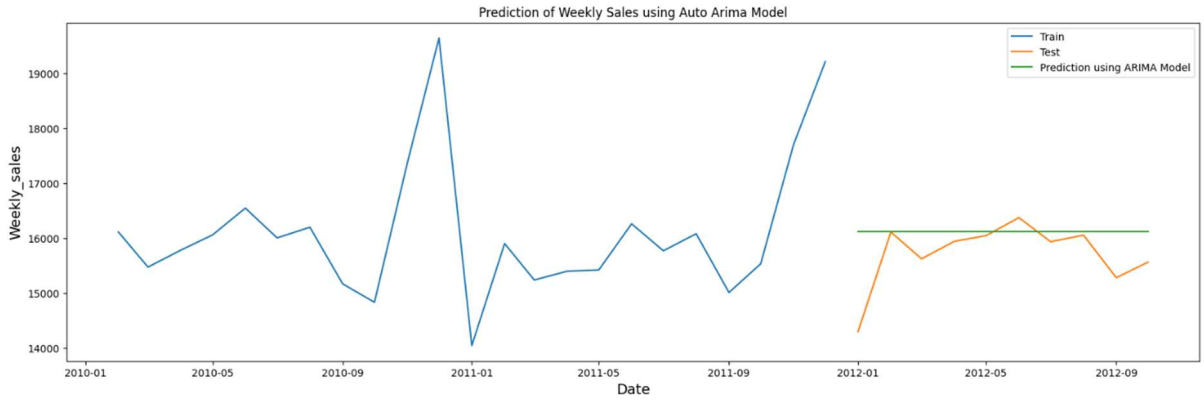
ARIMA
ARIMA(0,0,0)(0,0,0)[1] intercept

```
# Predicting the test values using predict function

forecast = model_auto_arima.predict(n_periods = len(test_data))
forecast = pd.DataFrame(forecast,index = test_data.index, columns = ['Prediction'])
plt.figure(figsize = (20,6))
plt.title('Prediction of Weekly Sales using Auto Arima Model')

plt.plot(train_data, label = 'Train')
plt.plot(test_data, label = 'Test')
plt.plot(forecast, label = 'Prediction using ARIMA Model')

plt.legend(loc = 'best')
plt.xlabel('Date', fontsize=14 )
plt.ylabel('Weekly_sales', fontsize = 14)
plt.show()
```




```
[58] from sklearn.metrics import mean_squared_error, mean_absolute_error
     import math

     # Performance metric for ARIMA model
     print( 'Mean Squared Error (MSE) of ARIMA:' , mean_squared_error(test_data, forecast))
     print( 'Root Mean Squared Error (RMSE) of ARIMA:' , math.sqrt(mean_squared_error(test_data,forecast)))
     print( 'Mean Absolute Deviation (MAD) of ARIMA:', mean_absolute_error(test_data, forecast))

Mean Squared Error (MSE) of ARIMA: 469971.08318421885
Root Mean Squared Error (RMSE) of ARIMA: 685.5443699602665
Mean Absolute Deviation (MAD) of ARIMA: 446.9938500521428
```

Random Forest

The RandomForestRegressor algorithm is instantiated, and the training data is conveyed to the model through the utilization of the .fit() function. Subsequently, predictions on the test data are generated using the .predict() function and stored in a new variable. This systematic procedure forms an integral part of the modeling process, contributing to the evaluation of the Random Forest model's predictive capabilities.

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 50, max_depth= 20 , min_samples_split= 3, min_samples_leaf= 1)
rf.fit (X_train, y_train.ravel())
print( 'Accuracy: ' ,rf.score(X_test,y_test.ravel())*100,'%')

y_pred = rf.predict(X_test)
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

rms = mean_squared_error(y_test, y_pred, squared = False )
print('RMSE:' , rms )
print('MAE:', mean_absolute_error(y_test, y_pred))
```

```
Accuracy : 99.779483752483
RMSE : 4055.838423845328
MAE : 1651.2174847928749
```

```
print('Training Accuracy: ' , rf.score(X_train, y_train.ravel())*100,'%' )
```

```
Training Accuracy:  99.11549542409416 %
```

XgBoost

The XGBoost algorithm is initialized by creating an instance of the XGBRegressor. Subsequently, the model is trained using the training data provided through the .fit() function. Following the training phase, predictions are generated for the test data using the .predict() function, and the results are stored in a new variable. This approach facilitates the evaluation of the model's performance on unseen data.

```
[ ]  import xgboost as xgb
     import warnings
```

```
xg_reg = xgb.XGBRegressor(objective = 'reg:squarederror', nthreads = 4, n_estimators = 500 , max_depth = 4 , learning_rate = 0.5 )
xg_reg.fit(X_train , y_train )
```

outp

XGBRegressor

```
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, nthreads=4,
             num_parallel_tree=None, ...)
```

```
[ ]  pred = xg_reg.predict(X_train)
     y_pred = xg_reg.predict(X_test)
     print( 'Accuracy:', xg_reg.score(X_test, y_test)*100, '%')

     rms = mean_squared_error(y_test, y_pred, squared = False )
     print('RMSE:', rms)

     print('MAE:',mean_absolute_error(y_test, y_pred))
```

```
Accuracy: 93.4370126737608 %
RMSE: 5836.7018351496745
MAE: 3044.4736011197256
```

```
[ ]  print('Training Accuracy :' , xg_reg.score(X_train, y_train )*100, '%')
```

```
Training Accuracy : 94.3954041409121 %
```

Comparing the models

For comparing the models PrettyTable function is defined.

```
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names = ["Model" ,"Training Accuracy","Testing Accuracy","RMSE","MAE"]
tb.add_row(["Random Forest", 99.11, 96.77, 4055.83, 1651.13])
tb.add_row(["XgBoost", 94.23, 93.72, 5660.68, 3129.36])

print(tb)
```

```
+---------------+-------------------+------------------+---------+---------+
|     Model     | Training Accuracy | Testing Accuracy |   RMSE  |   MAE   |
+---------------+-------------------+------------------+---------+---------+
| Random Forest |       99.11       |      96.77       | 4055.83 | 1651.13 |
|    XgBoost    |       94.23       |      93.72       | 5660.68 | 3129.36 |
+---------------+-------------------+------------------+---------+---------+
```

After calling the function, the results of models are displayed as output. From all the models, Random forest is performing well. The XgBoost model is evaluated with cross validation.

```
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 = xgb.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=6)
```

```
np.mean(cv)
```

```
0.7339361048014904
```

The model is saved as

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

● 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.

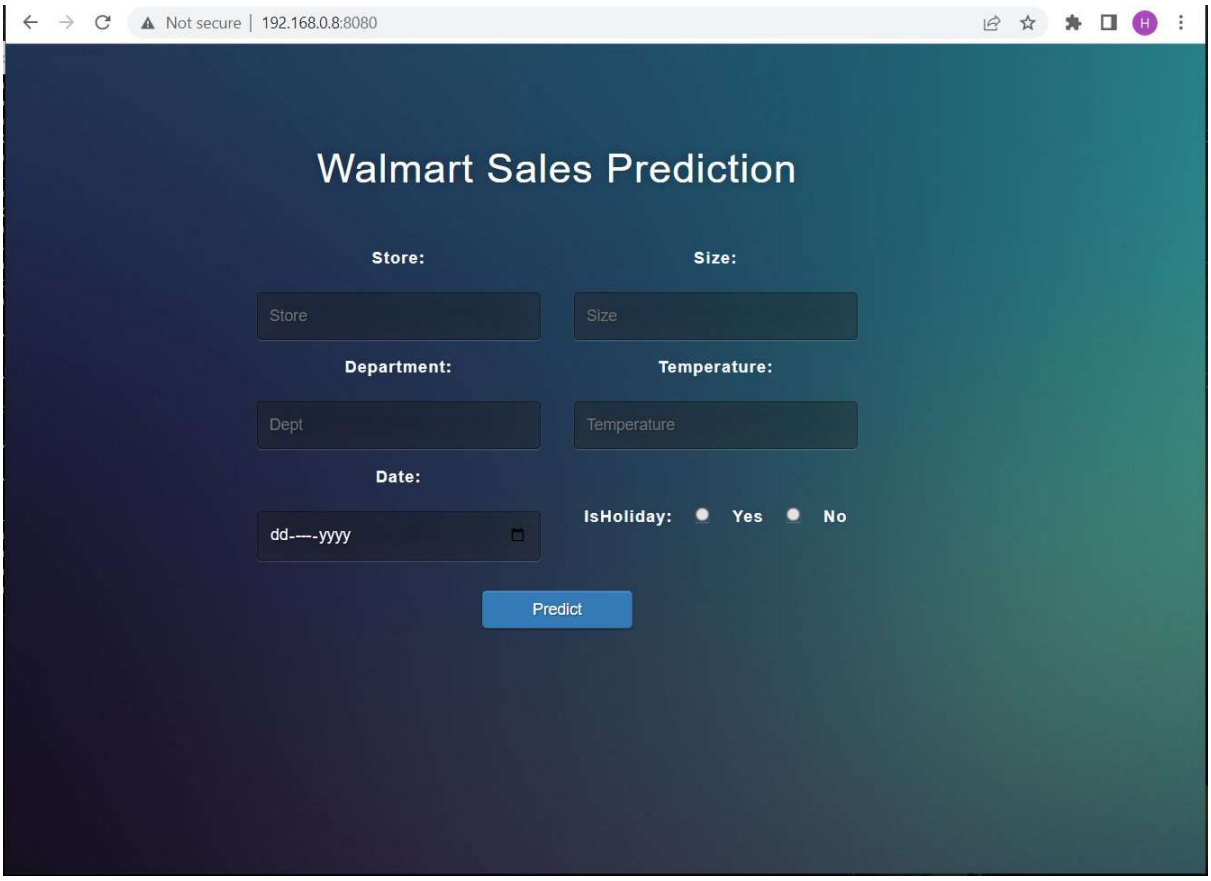
This section has the following tasks

- Building HTML Pages
- Building serverside script

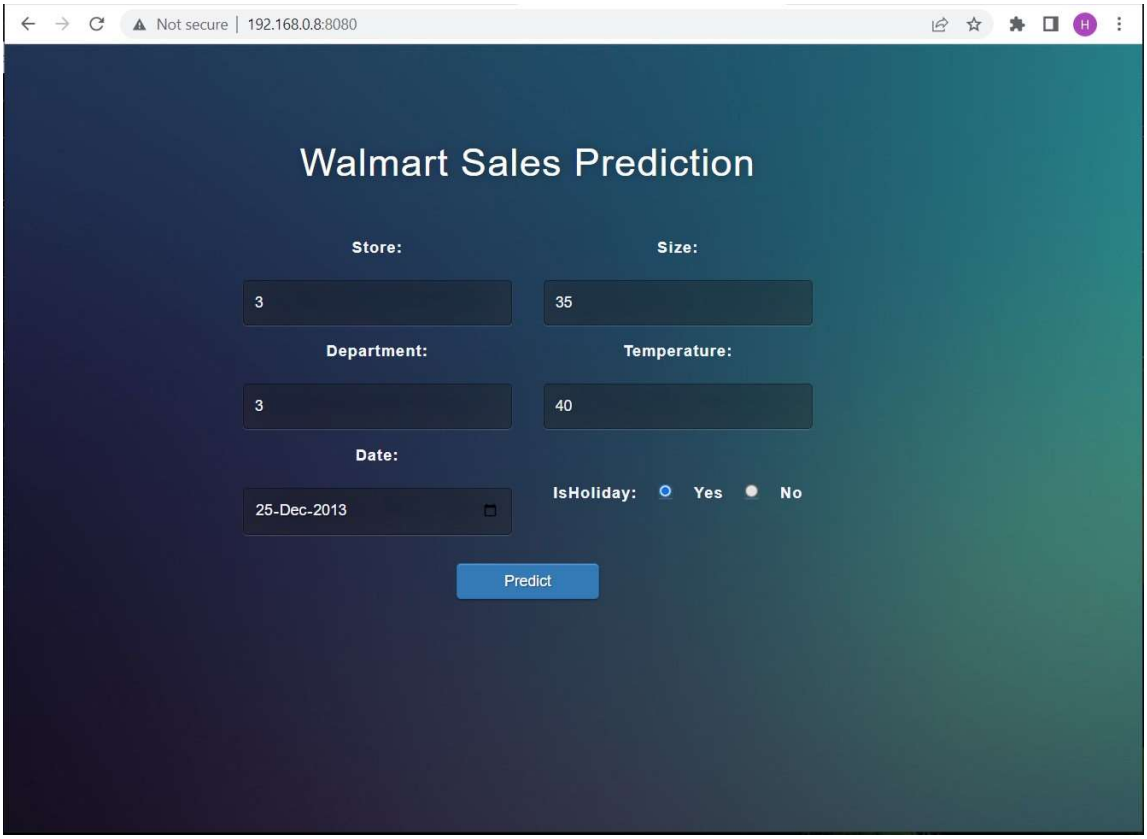
Building HTML Pages:

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

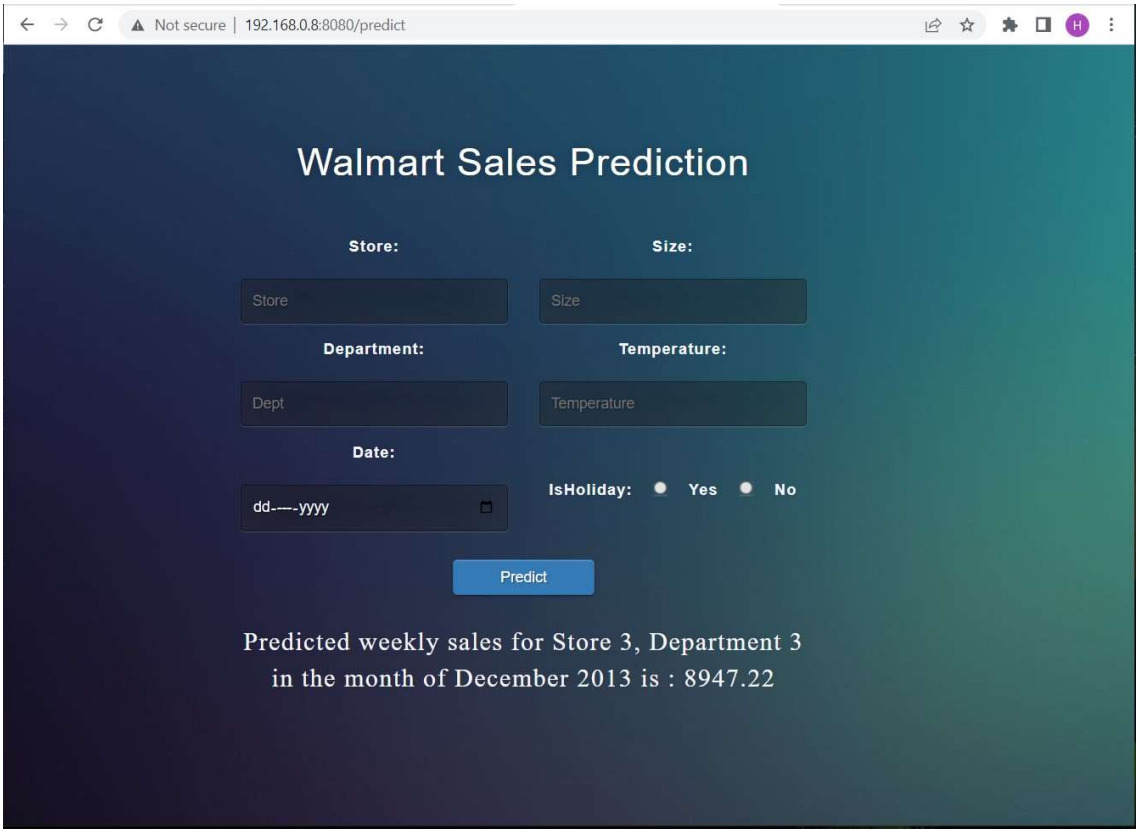
The index.html page looks like:



The index page or the main page is displayed and it has fields for entering certain values to predict the sale of a department and store on a particular and whether that day is a holiday or not.



After entering the details, we can perform the prediction by clicking on the predict button.



After the prediction is done, the predicted sales are displayed.

Building the python code in flask:

Import the libraries

```
import numpy as np
from numpy.core.fromnumeric import size
import pandas as pd
from sklearn.model_selection import train_test_split
from flask import Flask, render_template, request
import pickle
import datetime as dt
import calendar
import os
```

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.

```
app = Flask(__name__)

loaded_model = pickle.load(open('final_model.pkl', 'rb'))
fet = pd.read_csv('merged_data.csv')
```

Render the HTML page:

```
@app.route('/')
def home():
    return render_template('index.html')
```

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():
    store = request.form.get('store')
    dept = request.form.get('dept')
    date = request.form.get('date')
    isHoliday = request.form['isHolidayRadio']
    size=request.form.get('size')
    temp=request.form.get('temp')
```

```
    d=dt.datetime.strptime(date, '%Y-%m-%d')
    year = (d.year)
    month = d.month
    month_name=calendar.month_name[month]
    print("year = ", type(year))
    print("year val = ", year, type(year), month)
    X_test = pd.DataFrame({'Store': [store], 'Dept': [dept],'Size':[size],
                           'Temperature':[temp], 'CPI':[212], 'MarkDown4':[2050],
                           'IsHoliday':[isHoliday], 'Type_B':[0], 'Type_C':[1],
                           'month':[month], 'year':[year]})
    print("X_test = ", X_test.head())
    print("type of X_test = ", type(X_test))
    print("predict = ", store, dept, date, isHoliday)

    y_pred = loaded_model.predict(X_test)
    output=round(y_pred[0],2)
    print("predicted = ", output)
    return render_template('index.html', output=output, store=store, dept=dept,
                           month_name=month_name, year=year)

port = os.getenv('VCAP_APP_PORT','8080')
```

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.

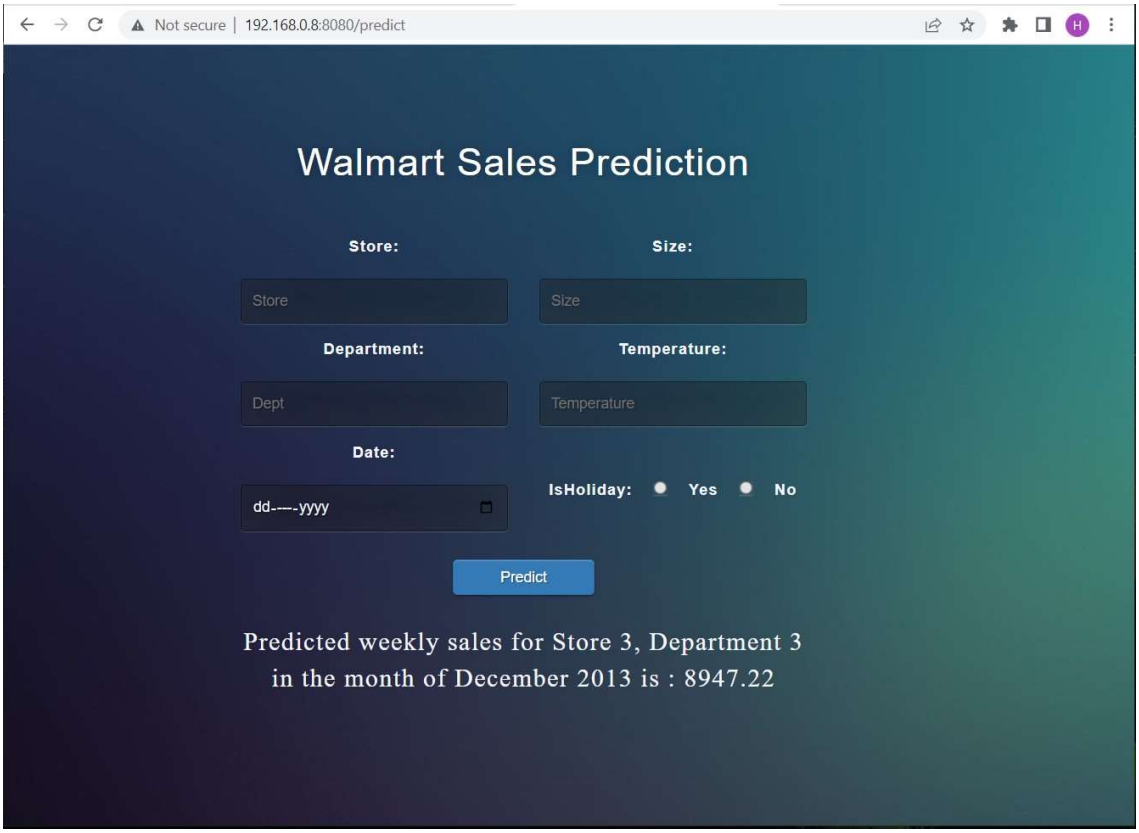
Main Function:

```
if __name__ == "__main__":
    app.secret_key=os.urandom(12)
    app.run(debug=False, host='0.0.0.0', port=port)
```

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.

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.0.8:8080/ (Press CTRL+C to quit)
```



The output can be seen on the HTML page in the browser as well as on the anaconda prompt.

```
192.168.0.8 - - [12/Jun/2022 22:55:34] "GET / HTTP/1.1" 200 -
192.168.0.8 - - [12/Jun/2022 22:55:39] "GET /static/css/style.css HTTP/1.1" 200 -
192.168.0.8 - - [12/Jun/2022 22:55:40] "GET /static/images/BackgroundImg.jpg HTTP/1.1" 200 -
192.168.0.8 - - [12/Jun/2022 22:55:46] "GET /static/images/walmartIcon.png HTTP/1.1" 200 -
year =  <class 'int'>
year val =  2013 <class 'int'> 12
X_test =    Store Dept Size Temperature  CPI  ...  IsHoliday Type_B  Type_C  month  year
0     3    3   35          40  212  ...          1      0       1     12  2013

[1 rows x 11 columns]
type of X_test =  <class 'pandas.core.frame.DataFrame'>
predict =  3 3 2013-12-25 1
192.168.0.8 - - [12/Jun/2022 22:56:13] "POST /predict HTTP/1.1" 200 -
predicted =  8947.22
```