Walmart Sales Forecasting Analysis

An Overview of The Dataset

The dataset I was working with is Walmart.csv downloaded from Kaggle. This dataset is a 6435 (rows) x 8 (columns) dataset. The input attributes are : Store, Date, Weekly_Sales, Holiday_Flag, Temperature, Fuel_Price, CPI, Unemployment

Approach

First, I read in the Walmart.csv dataset using the parse_dates field. This helps me to see the 'Date' column in YYYY-MM-DD format as opposed to its original MM-DD-YYYY presented in the csv file. I found that there are 45 unique values for Store, 892 unique fuel prices, and 349 unique values of unemployment. Further, the data types are as follows:

```
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         6435 non-null   int64  
 1   Date          6435 non-null   object 
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64  
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2), object(1)
memory usage: 402.3+ KB
```

I did **Exploratory Data Analysis (EDA**) first to correlate the temperature with the weekly sales. I found that when the temperature is lowest (rainy season?) and when the temperature is highest (inching towards 100 ° F, and few shoppers want to venture out to the stores?), the weekly sales are the lowest. The highest weekly sales of over $300M was achieved when the temperature was between 70 - 71.99 °F.

Next, I found the correlation between unemployment and weekly sales. When unemployment was between 8 - 8.199, the sum of weekly_sales was the highest at $576.7 M. Interestingly, sum of weekly_sales was lowest when unemployment was lowest. Using a box plot, I correlated how unemployment affects weekly sales. Here are my findings:

The Q1 (or 25th percentile) of unemployment is at 6.891. The Q3 (or 75th percentile) is at 8.622. The minimum is at 3.874, the median is at 7.874, and the maximum is at 14.313. The lower fence (Q1 - 1.5 IQR*[1]) is at 4.308 and the upper fence (Q3 + 1.5 IQR) is at 10.926. A data point that is below the lower fence is a low outlier. A data point that is above the upper fence is a high outlier.

For the DataFrame df that has been read in, my code creates a large figure with a histogram for each numerical column in the DataFrame. Each histogram is looking at how store, holiday_flag, temperature, fuel prices, CPI and unemployment are affecting weekly sales.

Seaborn plots are used to visualize the distribution of data such as weekly sales, temperature, fuel price and unemployment.

Doing Analysis using TIME SERIES factor

First, I converted valid dates to datetime objects, and replaced the NaT (Not a Time) values - missing or null values in the datetime64[ns] column with a specific date, the minimum date value (or the earliest date) in 'Date' column. Then I extracted the 'Year', 'Month', 'Day' and 'WeekOfYear' columns. The numeric days were mapped to Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday. The numeric months were mapped respectively from January to December.

Using Seaborn countplot, I examined the counts of the observation of the years — 2010, 2011, and 2012 being counted. Similarly, I counted the observation of the days — Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday — and also the observation of the months — January, February, March, April, May, June, July, August, September, October, November and December.

The pie chart I generated using matplotlib library helped me to visualize the annual sales data for the years 2010, 2011, and 2012. I grouped the DataFrame df by the 'Year' column, and summed up the 'Weekly_Sales' column for that year. The pie-chart showed 2010 having the largest percentage (73.87%) for weekly sales, followed by 2011 (14.72%) and 2012 (11.41%).

The pie chart that help visualize the weekly sales data by days of the week shows that Sunday is the day with the largest amount of sales (at 65.98%), while Wednesday has the smallest amount of sales (at 1.38%).

[1] IQR refers to the interquartile range (Q3 - Q1).

Interestingly, January (surprise: not December!) takes the prize for the month that chalks up the largest amount of weekly sales. We see that non-special holiday week accounts for 92.5 % of the weekly sales, and special holiday week actually accounts for just 7.5% of the weekly sales. In plotting the histogram, we see that indeed the first weeks of January account for more weekly sales in the year than all the other weeks of the year.

Data Processing for Model Fit

Next, I generated a correlation matrix to visualize the correlation between the different input features.

There is strong correlation between the store # and unemployment; strong correlation between weekly sales and holiday flag; strong correlation between temperature and CPI; strong correlation between fuel price and year; strong correlation between unemployment and holiday flag; between unemployment and month of the year; strong correlation between the year and fuel price; strong correlation between the month and holiday flag; strong correlation between the week of the year and holiday flag.

Conversely, the lower the fuel price, the more we see the increase in the weekly sales. The higher the temperature, the lower would be the weekly sales (could be that customers do not want to venture out to shop in the physical store when the weather is warm?).

One-Hot Encoding For Encoding Categorical Variable as Ordinal Variables

Using the OrdinalEncoder, I transformed categorical variables 'Month' and 'Day' into ordinal variables and printed out a concise summary using X.info().

Splitting The Data Into Training and Testing Sets

Split the available dataset into 40% testing and 60% training using a random seed of 42. We will use the resulting X_train, X_test, Y_train and Y_test variables to evaluate the performance of the model.

One-Hot Encoding For Encoding the 'Store' Variable

Here, we use sklearn's ColumnTransformer to encode the 'Store' variable. The OneHotEncoder will be configured with sparse=False to return a dense array. The 'Store' column in both the X_train and X_test is one-hot encoded and the rest of the columns remain unchanged (the remainder ='passthrough' means columns not specified in the transformer will be passed without any transformation).

Note: Although sparse array takes up less memory usage, the processing time may be longer. Thus I have opted to return the result as a dense array which takes less processing time.

Model Training

Next, we scale the data for better fitting and less time consumption. I used the RobustScaler, and have a pred_model() function to print out the mean squared error (MSE), the mean absolute error (MAE) and r2 score. Here, we will use various regression models such as Linear Regression, Lasso Regression, Ridge Regression, Random Forest Regression and ElasticNet model, and observe the MSE, MAE and r2 scores.

We find that Ridge Regression model and Lasso Regression model presented the highest MAE, while Linear Regression model and Ridge Regression model presented the highest MSE with almost similar R2 score. This means the 3 models — Linear Regression, Lasso Regression and Ridge Regression are performing well with high accuracy, and their almost similar R2 scores indicate that in general these 3 models explain the dataset quite well in that there isn't too much of overfitting.

On the other hand, the Random Forest Regression model, as an ensemble model, exhibits lower MSE and MAE but with a higher R2 score than the other 3 previous models. The slightly higher R2 score indicates that the Random Forest Regression model is better for capturing the complex relationships. The ElasticNet model has a much lower R2 score, indicating that this model isn't a great fit and would likely not be capable of capturing the complex relationships. It gives higher MSE value compared to Lasso, Ridge and Linear Regression models. But it also gives lower MAE value compared to the other 3 models.

Stacked Version of Regression Model

Next, I defined K-fold cross-validator to provide train/test indices to split the data in train/test sets. I selected to use XGB Regressor , LGBM Regressor , Gradient Boosting Regressor, Random Forest Regressor, Support Vector Regression (SVR), LassoCV, and StackingCV Regression. My code then outputs the mean squared error score (MSE) —`3564343152.1680093` on the full training set, and the mean absolute error (MAE) — `63625.65450878982` of the test set.

Using the Deep Learning Network For The Same Prediction

I created a sequential model which has a linear stack of layers. The layers consist of BatchNormalization, Dense (fully connected) layer with 8 units and ReLU activation, Dropout 30%

of the neurons during training to prevent overfitting. Then BatchNormalization normalizes the data again. Another layer with 24 units and ReLU activation, and a final dense layer with 10 units.
The mean squared error score (MSE) and the mean absolute error (MAE) is 71291904000.0 on the test set.