Coursework 2: Walmart analysis

Module: CIS2144 Data Mining 2021

Module Leader and Lecturer: Dr Huaizhong Zhang

By: Christopher Diaz Montoya
Student ID: 24707686

Contents

Contents of figures

Introduction

This analysis's aim is to see what factors affected 45 of Walmart's stores weekly sales, and to see if future weeks could be predicted accurately within a range. These factors include temperature, fuel prices, unemployment rate, store type and if it was a holiday to name a few. This was completed using the python programming language in Jupyter notebooks, various libraries were used to manipulate and analyse the data set provided. This report will go over the methods used to pre-process the data set and the models used on them.

Data check, input and merge

The first thing done was to upload the data sets provided, these were four CSV files with one of them being saved for last as this was to help make the final predictions and the other three contained historical data. This can be seen done in Figure 1 - Upload CSVs.

```
features_df = pd.read_csv(r"C:\Users\chris\D
stores_df = pd.read_csv(r"C:\Users\chris\Dow
test_df = pd.read_csv(r"C:\Users\chris\Downl
train_df = pd.read_csv(r"C:\Users\chris\Downr
```

Figure 1 - Upload CSVs

The three data sets needed to be uploaded into the program, this was done with the pandas library using the .read_CSV() function which helped upload the file, to do this the file location (path) was inputted into the parenthesis. This was learnt for the last project.

Once that was done the three data sets needed to be merged. This was completed using the .merge() function within the pandas library. As they could not all be merged in one, they were merged one by one. Before merging, the CSV files were checked at the head (top rows of the table) as they needed to be merged on the same columns to maintain the integrity of the data, this was done by ensuring the tables did not have repeated columns. How it was merged can be seen in Figure 2 - Merge files 1 by 1, this was learnt from pandas (n/a).

```
df1 = pd.merge(features_df, train_df, on = ("Store", "Date", "IsHoliday"))
df1
```

Figure 2 - Merge files 1 by 1

Data pre-processing

Data pre-processing is the manipulation of data to remove noise in the data set and for the program to be able to read it (Garcia et al, 2014). This was important as it checks for multiple things. These will be broken down into the subsections within Data pre-processing.

Data completeness

The data needs to be checked for completeness; this was done using .isna().sum() which brings up the amount of missing data in each column, as shown in Figure 3 - Missing data, the markdown columns had a lot of missing data. Due to these markdowns being sales for products, this missing data was filled with a 0, (this was learnt for the last project in lectures), because if the item did not have a markdown, then it should not be given one.

```
df2.isna().sum()
```

```
Store                0
Date                 0
Temperature          0
Fuel_Price           0
MarkDown1       270889
MarkDown2       310322
MarkDown3       284479
MarkDown4       286603
MarkDown5       270138
CPI                  0
Unemployment         0
IsHoliday            0
Dept                 0
Weekly_Sales         0
Type                 0
Size                 0
dtype: int64
```

Figure 3 - Missing data

Feature exploration

Next, the columns were checked against the weekly sales column, this was done with the use of graphs and a heat map. The graphs were created using seaborn and had the weekly sales data on the y axis and the chosen column to it on the x axis, a histogram/pairplot graph hybrid was created for this to help accurately see the correlations, this was learnt from seaborn(n/a).

As shown in Appendix A columns such as fuel price, CPI and unemployment did not seem to have a notable impact, while size, dept, type, temperature, date and holidays seemed to. This was checked with the heat map seen in Figure 7 - Heatmap correlation this showed that temperature, store, and year had a low correlation to the weekly sales that were not known. Columns with low correlations were removed as shown in Figure 8 - Columns dropped, this was learnt in the lecture.

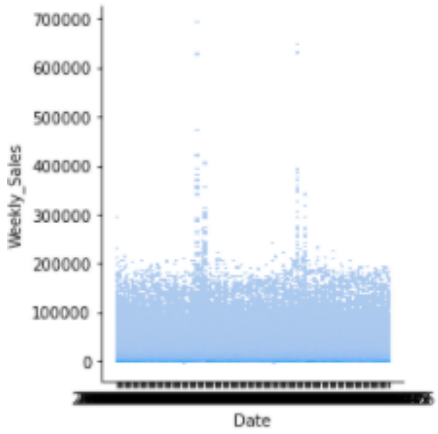


Figure 4 - Data graph

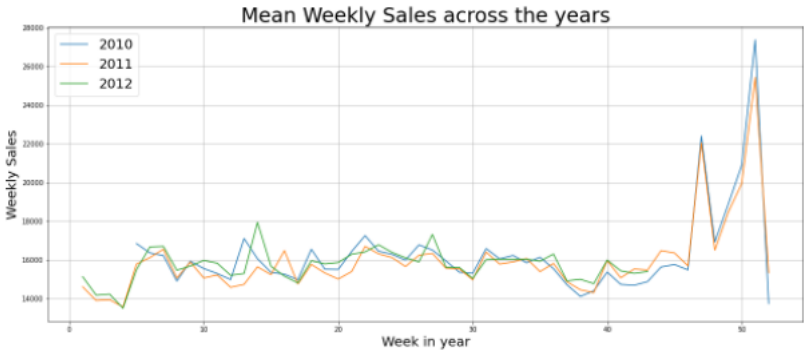


Figure 5 - Weekly Sales graph

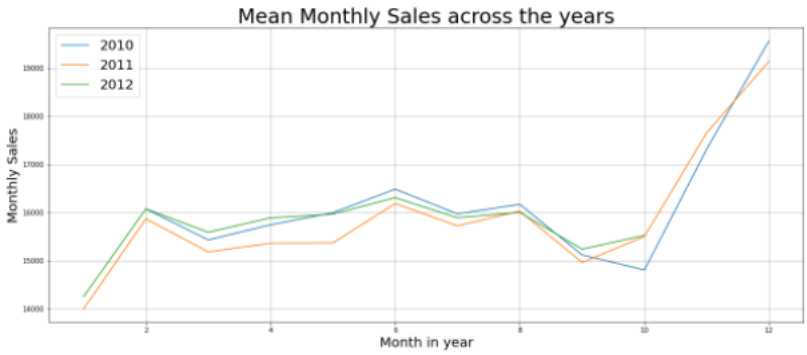


Figure 6 - Monthly sales graph

As the holidays and the date columns were important as based on Figure 4 - Data graph there are two sets of spikes. The date column was split up into the year, month and week column, the year column was dropped as this had a low correlation with weekly sales. The month and week columns were left, as holidays were in certain months and weeks throughout the year. The data was visualized to see if there were any spikes in certain months throughout the years given in the datasets. As shown in Figure 5 - Weekly Sales graph and Figure 6 - Monthly sales graph there are major spikes towards the end of the year, in the week's graphs, a drop occurred after the first spike showing the importance. With more time this would have been analysed deeper and checked if the spikes in weekly sales were on a holiday, after a holiday, or before a holiday to see if there was a correlation.

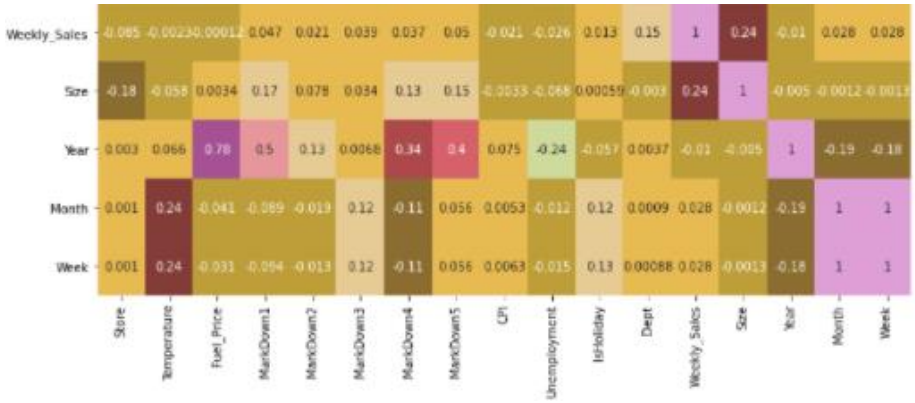


Figure 7 - Heatmap correlation

```
df = df.drop(columns = ["Date", "CPI", "Unemployment", "Fuel_Price", "Temperature", "Year"])

# Check data frame is as expected
df
```

	Store	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	IsHoliday	Dept	Weekly_Sales	Type	Size	Month	Week
0	1	0.00	0.00	0.0	0.00	0.00	False	1	24924.50	A	151315	2	5
1	1	0.00	0.00	0.0	0.00	0.00	False	2	50605.27	A	151315	2	5

Figure 8 - Columns dropped

Data transformation

The data all needed to be of a numerical type, this was because it helps the program read and understand the data. The Weekly_Sales column was checked and was of the continuous type of data, this did not allow for ranges (categories) which was what was needed, this was implemented with pandas.cut as shown in Figure 9 - Continuous to categorical.

```
df["Weekly_Sales"] = pd.cut(df.Weekly_Sales, bins = [-5000, -1, 5000, 10000, 15000, 20000
                            labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
```

Figure 9 - Continuous to categorical

One hot encoding was used for this as shown in Figure 10 - One hot encoding for one of two reasons, the first to ensure the columns is of numerical type, and the second the ensure columns of numerical type are not weighted differently in the model, instead of just assigning it a number, it would assign it an array of 0s and 1s so everything was weighted equally.

```
# One hot encoding used to convert columns
df["IsHoliday"] = pd.get_dummies(df["IsHoliday"])
df["Type"] = pd.get_dummies(df["Type"])

# Done to make sure store id number does not change outp
df["Store"] = pd.get_dummies(df["Store"])

# Done to make sure Weekly_sales does not have a higher
df["Weekly_Sales"] = pd.get_dummies(df["Weekly_Sales"])
```

Figure 10 - One hot encoding

Data split

The target column needed to be removed from the current data frame, this was so when the data was inputted into the machine learning models the results were hidden, this allowed to check the accuracy of the models created. This was done with .drop().

The data was split into three, training data, validation data and test data, this was so the first large chunk amounted to 60% of the data would be to train the model, to help the model understand what it was looking for and how to find it. The next two sections were to check the accuracy of the model, as both these data frames had the answers to the weekly sales hidden, after the model made its prediction, this could be used to check the accuracy.

```
X = df.drop("Weekly_Sales", 1) # Features assinged to X, output hidden
y = df.Weekly_Sales # Output or result hidden in seperate variable y

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size = 0.25, randor
```

Figure 11 - Split data

Machine Learning models

The oxford dictionary (n/a) defines machine learning as "a type of artificial intelligence in which computers use huge amounts of data to learn how to do tasks rather than being programmed to do them", this means that these different types of models accept an input, that being the data, and be able to create a prediction or output. Each model used will be explained from what was learnt in lectures, Navlani (2018) and from Patel (2018).

Decision tree

A decision tree looks like a tree. It starts with one column to make a choice and then based on the decision it goes down the tree, down its branches until it reaches the end and comes to a decision, this last decision is called the leaf node, like the lead at the end of a tree. This model is good for categorical data as based on the decisions it will flow down the tree until it meets the criteria to be assigned into one of the categories. This was why the weekly_sales columns was changed from a continuous data type to ranges as decision trees do not work on continuous values.

```
dtc = DecisionTreeClassifier().fit(X_train, y_train)

# Predictions of test stored
y_val = dtc.predict(X_valid) # Validation data
y_predic = dtc.predict(X_test) # Test data
```

Figure 12 - Decision tree code

Random forest

This model is an ensemble method, an ensemble method is the use of multiple models to help conclude. The reason this is an ensemble method is that it uses multiple decision trees, hence the name random forest. This takes more time to compute as it is multiple models.

```
rfc = RandomForestClassifier(n_estimators = 40).fit(X_train, y_train)

# Predictions of test stored
y_val = rfc.predict(X_valid) # Validation data
y_predic = rfc.predict(X_test) # Test data
```

Figure 13 - Random forest code

K Nearest Neighbour

This was the only used unsupervised model used. It works by mapping out everything as dots and then based on how many neighbour dots the next inputted data has depended on which group the data is classified to. To find the perfect number of neighbours the elbow method was used which will be discussed alongside the Kmeans model. This model can be used for clustering.

```
knn = KNeighborsClassifier(n_neighbors = 3).fit(X_train, y_train)

# Predictions of test stored
y_val = knn.predict(X_valid) # Validation data
y_predic = knn.predict(X_test) # Test data
```

Figure 14 - KNN code

Kmeans

This model is good for clustering. This works by making its own clusters or weekly_sales groups, adding a centre dot and allocating an area to that group and then using the Euclidean distance to calculate which group the prediction belongs to.

```
X, y = make_blobs(n_samples= 115064, random_state = 0, cluster_std = .2)
plt.scatter(X[:, 0], X[:, 1], s = 10) # S = size, rest makes the graph
```

```
<matplotlib.collections.PathCollection at 0x1c371e6fbe0>
```

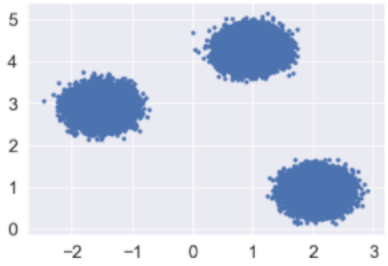


Figure 15 - Kmeans make blobs

As shown in Figure 16 - Elbow methods, optimal no. of clusters, the elbow method was used to identify the best number of clusters for the K-means and KNN models. Alade (2018) explains that this was to find the best value for k, it gives centres and calculates the distance for data from those centres and the graph shows the best number of clusters or centroids for that data.

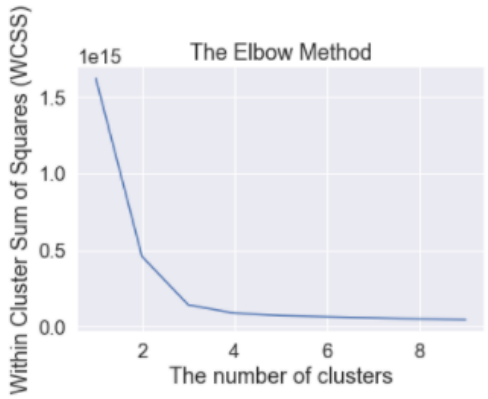


Figure 16 - Elbow methods, optimal no. of clusters

Teamwork

This project was initially started a team project, but due to the other member leaving without communicating or contributing anything, the project was behind schedule. This forced the project to not be completed in the desired manner, this being the decision tree not being manually created along with not having enough time to see why the model precision, recall and f1 score are low but the accuracy was high.

This should not have been an issue as the project should have been continued but was paused for two weeks awaiting the response of the other team member to meet and work. Although this time was used to work on another project.

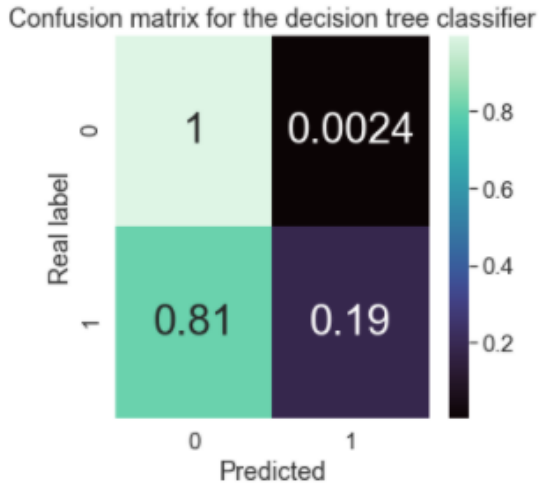
Results

The supervised models are evaluated in a few ways, these are the understood meanings according to Agarwal (2019) and lectures:

1. Accuracy: Amount predicted accurately.
2. Precision: What was true and predicted true.
3. Recall: What proportion of true was predicted as true.
4. F1 – score: Balance between precision and recall.

As shown from Figure 17 - Decision tree evaluation, Figure 18 - KNN evaluation and Figure 19 - Random forest evaluation, the models have good accuracy, but the other evaluation metrics are low. A confusion matrix was used to help visualize this. This meant the model or data still had some tuning to do as the models were misclassifying. This is like an earth asteroid impact model predicting no, over, and over, this could be 99% accurate, but this is not a valuable evaluation metric in cases like these, thus the other metrics could have been explored.

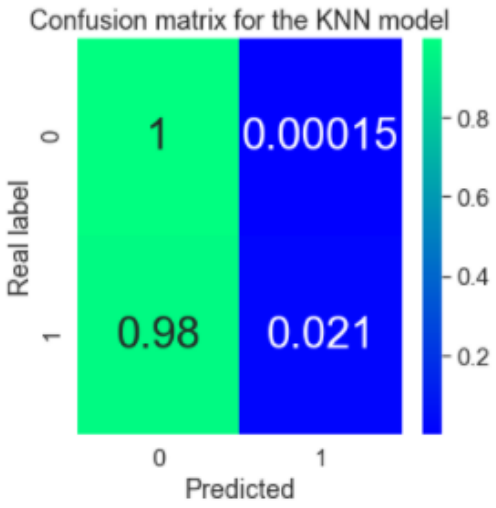
```
The decision tree models accuracy is
99.48288540455916 % Validation data
99.5303271105629 % Test data
Confusion matrix
[[83873   200]
 [  196    45]]
```




```
Precision:  0.1836734693877551
Recall:  0.18672199170124482
F1 score:  0.18518518518518517
```

Figure 17 - Decision tree evaluation

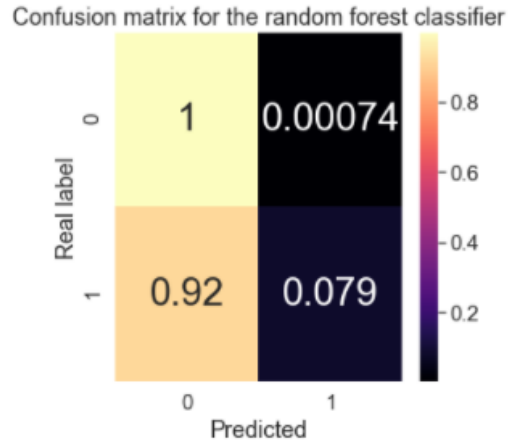
```
The knn models accuracy is
99.68214056977489 % Validation data
99.70467538012667 % Test data
[[84060    13]
 [  236     5]]
```




```
Precision:  0.2777777777777778
Recall:  0.02074688796680498
F1 score:  0.038610038610038616
```

Figure 18 - KNN evaluation

```
The random forest models accuracy is
99.6252105225704 % Validation data
99.66316388737339 % Test data
[[84011    62]
 [  222    19]]
```




```
Precision:  0.2345679012345679
Recall:  0.07883817427385892
F1 score:  0.11801242236024845
```

Figure 19 - Random forest evaluation

As shown in Figure 20 - Kmeans output the kmeans model needs improvement or something may have been missed as the groups overlap and should be separate.

```
est = KMeans(n_clusters = 3)  # 3 clusters identifi
est.fit(test_df)
y_kmeans = est.predict(test_df)
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 20,
plt.show()
```

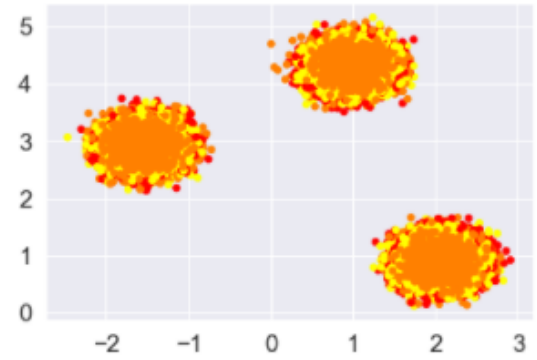


Figure 20 - Kmeans output

Reflection

Although his was a group project completed individually more could have been done, the expectation maximization for the Kmeans model could have been completed and if there was more time the models f1 score could be improved with models done manually to do so. The model could have also used k-folds instead of the train, validation, and test split used to prevent overfitting. To go the extra mile an interactive interface could have been created allowing the choice of models and the choice for custom columns in the data frame.

The project also had a hidden aim, to learn as much as possible about data mining, although the Kmeans model is incomplete and models are not the most accurate, this aim has been achieved as a general knowledge of many techniques such as data visualization, machine learning models and data pre-processing were learnt.

Bibliography

AGARWAL, R., 2019. *The 5 classification evaluation metrics every data scientist must know* [online]. Available from: https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226 [Accessed 2 January 2022].

ALADE, T., 2018. *Tutorial: how to determine the optimal number of cluster for k-means clustering* [online]. Available from: https://blog.cambridgespark.com/how-to-determine-the-optimal-number-of-clusters-for-k-means-clustering-14f27070048f [Accessed 26 December 2022].

GARCIA, S., LUENGO, J., HERRERA, F., 2014. *Data preprocessing in Data Mining* [ebook]. New York: Springer international publishing. Available from: https://www.google.co.uk/books/edition/Data_Preprocessing_in_Data_Mining/SbFkBAAAQBAJ?hl=en&gbpv=0 [Accessed 17 November 2021].

NAVLANI, A., 2018. *KNN Classification using Scikit-learn* [online]. Available from: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn [Accessed 20 December 2022].

NEXTDAYVIDEO., 2016. *Pre-Modelling: Data preprocessing and feature exploration in Python* [online video]. Available from: https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226 [Accessed 23 November 2021].

OXFORDLEARNER'SDICTIONARY., N/A. *machine learning* [online dictionary]. Available from: https://www.oxfordlearnersdictionaries.com/definition/english/machine-learning [Accessed 1 December 2021].

PANDAS ., N/A. *pandas.DataFrame.merge* [online]. Available from: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html [Accessed 16 November 2021].

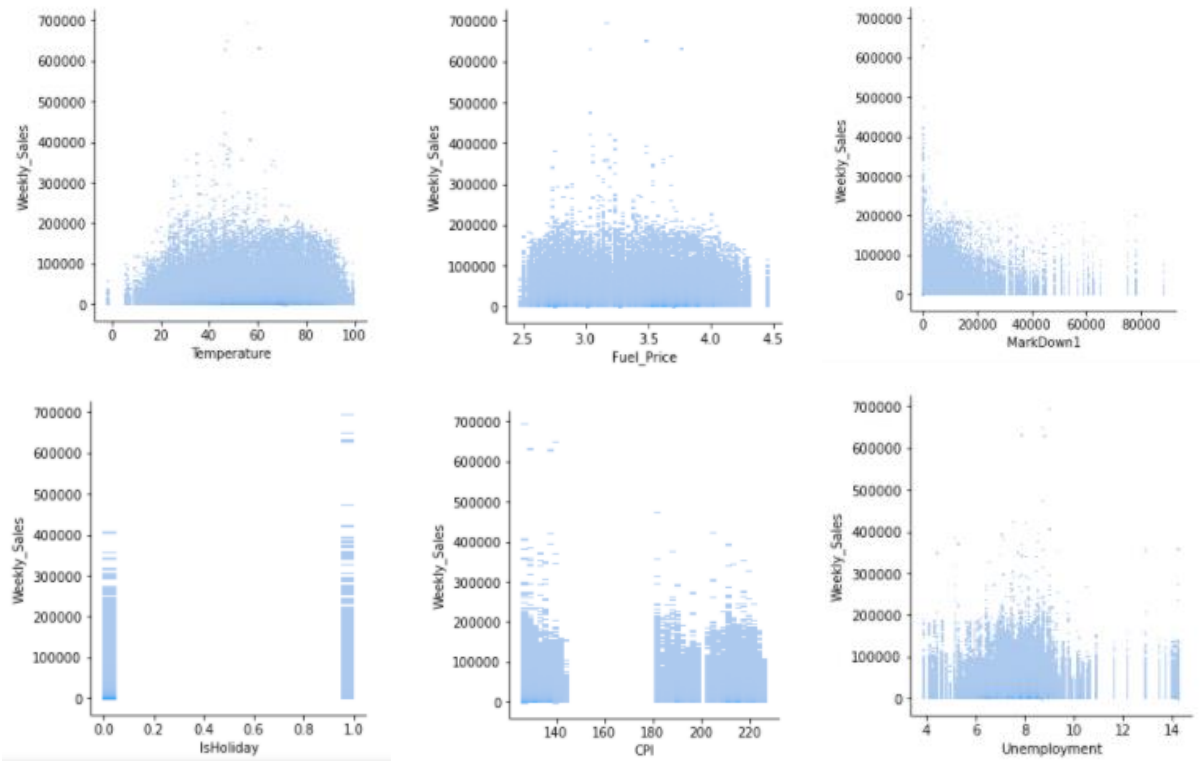
PATEL, A., 2018. *Machine Learning Algorithm Overview* [online]. Available from: https://medium.com/ml-research-lab/machine-learning-algorithm-overview-5816a2e6303 [Accessed 5 December 2022].

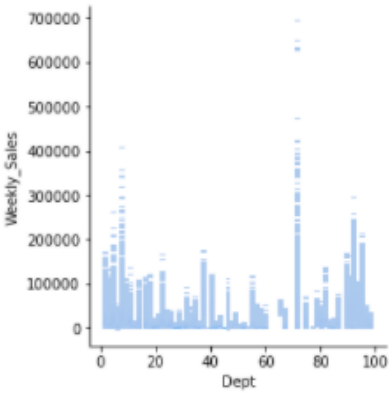
SEABORN., N/A. *seaborn.pairplot* [online]. Available from: https://seaborn.pydata.org/generated/seaborn.pairplot.html [Accessed 19 November 2021].

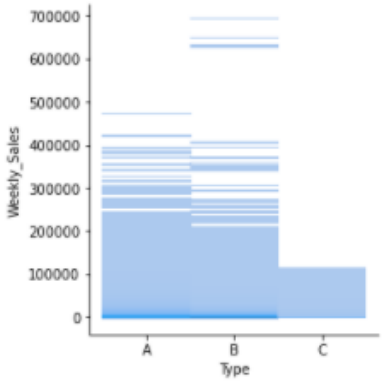
SCIKIT-LEARN., N/A. *sklearn.metrics.confusion_matrix* [online]. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html [Accessed 1 January 2022].

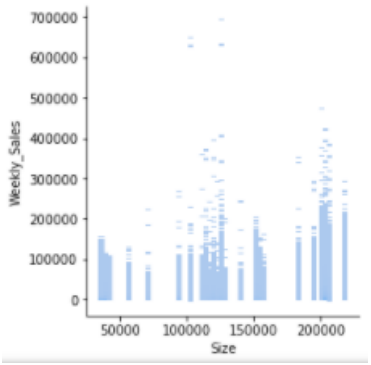
Appendices

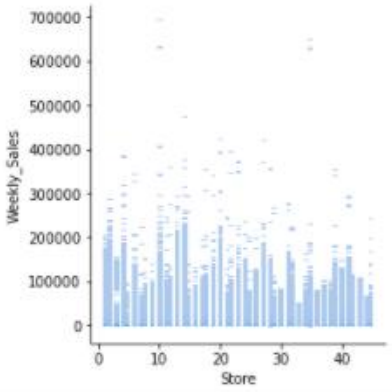
Appendix A

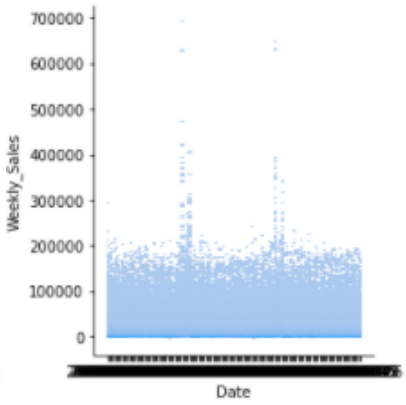



Weekly_Sales
Dept


Weekly_Sales
Type
A
B
C


Weekly_Sales
Size


Weekly_Sales
Store


Weekly_Sales
Date