Introduction

In this report, I will focus on the critical data pre-processing steps used to achieve the overall goal of predicting 6 weeks of daily sales for 1,115 drug stores in Germany. This report will include some visualisations of exploratory data analysis as well as a discussion of the various techniques and methods used to pre-process the data, such as data cleaning and transformation, data linkage, missing value detection and imputation, outlier detection and treatment, and encoding. The report will also justify the forecasting model chosen and provide insights into the impact of data pre-processing on the performance of the sales forecasting model. In conclusion, the importance of data pre-processing in ensuring forecast accuracy will be demonstrated.

Section 1: Dataset Review

There were three excel datasets provided for this report. The first is the "stores" dataset, which contains additional information for the 1,115 drug stores under consideration. It includes the store number, the store type or model, the level of product assortment, information about the nearest competition, and information about the current promotion that each store is running. I believe two key components of the dataset could have an impact on sales forecasting. The details of the nearest competitor would come first. Understanding the competitive landscape through competition distance can assist in identifying potential market trends and shifts, which can inform sales forecasting. The second category is Promotions. Promotions can increase demand for a product or service, which can lead to increased sales. It is possible to predict the potential impact of future promotions on sales by analysing the effect of previous promotions on sales.
Looking at the data quality of the 'stores' data in general, the dataset was incomplete because about 6 of its variables had missing data, some of which were irrelevant. This will be covered in greater detail in the Data linkage section. The following is the percentage of completeness of the six variables:

Variable	% Completeness
CompetitionDistance	99.7
CompetitionOpenSinceMonth	68.3
CompetitionOpenSinceYear	68.3
Promo2SinceWeek	51.3
Promo2SinceYear	51.3
PromoInterval	51.3

Table 1: % Completeness for 'Stores' Data

In terms of validity, the dataset was valid because all of the variables followed the expected syntax. It was mostly correct, but the 'Promo Interval' column could have been represented better. When comparing the train and test datasets, the main identifier, the 'store' number,' was consistent. The uniqueness dimension of data quality was also met by the store number. According to my knowledge, the data was timely.

The "train" dataset, which is a two-and-a-half-year compilation of each store's historical sales data, is the second dataset. It contains the store number, the day of the week and the date of sale, the total sales for the day, the number of customers for the day, promotional information, and holiday information. Aside from the previously mentioned promotional data relevance, I believe holiday data could be useful because it can provide some insight into changes in consumer behaviour, which

generally have an impact on sales. The 'train' dataset had no violations and was 100% complete in terms of data quality.
The third dataset is the 'test' which has a structure similar to the 'train' except for unknown sales and customer values.

Section 2: Data Integration

It was necessary to combine the data because the information needed to perform additional analyses on the drug stores was stored in two different datasets. This involved combining both datasets into one unified view.

```
[306…
#Merge the datasets
merged_dataset=pd.merge(train,store,on='Store',how='left')
test=pd.merge(test,store,on='Store',how='left')
merged_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1017209 entries, 0 to 1017208
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1017209 non-null  int64
 1   DayOfWeek                  1017209 non-null  int64
 2   Date                       1017209 non-null  datetime64[ns]
 3   Sales                      1017209 non-null  int64
 4   Customers                  1017209 non-null  int64
 5   Open                       1017209 non-null  int64
 6   Promo                      1017209 non-null  int64
 7   StateHoliday               1017209 non-null  object
 8   SchoolHoliday              1017209 non-null  int64
 9   StoreType                  1017209 non-null  object
 10  Assortment                 1017209 non-null  object
 11  CompetitionDistance        1014567 non-null  float64
 12  CompetitionOpenSinceMonth  693861 non-null   float64
 13  CompetitionOpenSinceYear   693861 non-null   float64
 14  Promo2                     1017209 non-null  int64
 15  Promo2SinceWeek            509178 non-null   float64
 16  Promo2SinceYear            509178 non-null   float64
 17  PromoInterval              509178 non-null   object
dtypes: datetime64[ns](1), float64(5), int64(8), object(4)
memory usage: 147.5+ MB
```

Figure 2.1: Merged Dataset information

The two files, train and stores, were linked using the store number as the key as this was consistent between datasets. The integration was done by performing a left merge of the "train" and "stores" datasets because we wanted the resulting dataset to contain all the rows of the "train" dataset and the matching rows from the "stores" dataset.
However, combining these two datasets resulted in some data quality issues. The first issue was incompleteness, as there was more missing data in the newly merged dataset because rows in the "train" dataset that had no matching row in the "stores" dataset became NaN values for the columns in the "stores" dataset. The second was an issue of inconsistency in timelines as the data from "stores" happen to be current information, but the data from "train" is historical data and so merging these two created a mismatch for some variables such as 'Promo2'. These issues reinforced the need to perform data pre-processing before training our model.

As previously stated, there were issues with mismatch due to differences in the dataset's time period measurement after merging the datasets.
The first was that in some cases, the promotion (Promo2) had not yet begun when the sales were measured, so the values of Promo2 in those cases were set to 0 to indicate that there was no possibility of running a promo at that time. There was a similar problem with 'CompetitionDistance' in that the competition store had not yet opened at the time of sales measurement, so these values were also Not Applicable. To address this mismatch, new columns were added to the dataset to aid in comparisons with the previously mentioned columns.
This was accomplished by deriving columns "CurrentWeekOfYear", "CurrentYear", and "CurrentMonth" from the date column, as shown below, and only after the date column was converted to datetime index.

```
18  CurrentWeekOfYear          1017209 non-null  UInt32
19  CurrentYear                1017209 non-null  int64
20  CurrentMonth               1017209 non-null  int64
```

Figure 2.2: New columns created

Following the creation of these new columns, all rows of the "Promo2" column where "CurrentYear" was less than or equal to "Promo2SinceYear" (the year the Promo began) and "CurrentWeekOfYear" was less than "Promo2SinceWeek" (the week the Promo began) were changed to 0 and other details on the promotion were changed to not applicable (NaN)
Similarly, all rows in the "CompetitionDistance" column where "CurrentYear" was less than or equal to "CompetitionOpenSinceYear" (the year the competitor began) and "CurrentMonth" was less than "CompetitionOpenSinceMonth" (the month the competitor began) was changed to NaN, as were the other competition details.

Section 3: Missing Values

Missing values were introduced into the dataset to ensure that the reality of those observations was appropriately represented when dealing with the difference in timing of the merged datasets.

```
[259]: merged_dataset.isna().sum()

[259]: Store                          0
       DayOfWeek                      0
       Date                           0
       Sales                          0
       Customers                      0
       Open                           0
       Promo                          0
       StateHoliday                   0
       SchoolHoliday                  0
       StoreType                      0
       Assortment                     0
       CompetitionDistance         2642
       CompetitionOpenSinceMonth  323348
       CompetitionOpenSinceYear   323348
       Promo2                         0
       Promo2SinceWeek            508031
       Promo2SinceYear            508031
       PromoInterval              508031
       dtype: int64
```

Figure 3.1: Missing values before handling timing

```
[262]: merged_dataset.isna().sum()

[262]: Store                          0
       DayOfWeek                      0
       Date                           0
       Sales                          0
       Customers                      0
       Open                           0
       Promo                          0
       StateHoliday                   0
       SchoolHoliday                  0
       StoreType                      0
       Assortment                     0
       CompetitionDistance        86275
       CompetitionOpenSinceMonth  406981
       CompetitionOpenSinceYear   406981
       Promo2                         0
       Promo2SinceWeek            576915
       Promo2SinceYear            576915
       PromoInterval              576915
       CurrentWeekOfYear              0
       CurrentYear                    0
       CurrentMonth                   0
       dtype: int64
```

Figure 3.2: Missing values after handling timing

The 'PromoInterval' missing values were replaced with 'NotApplicable' in the same string format as the other attributes in the column. The missing values in the 'CompetitionDistance' column were replaced by the mean 'CompetitionDistance' of the corresponding 'StoreType' of the missing value. The use of the distance-based KNNimputer was considered, but it was decided against because it is computationally expensive. All other timing-related columns, such as 'Promo2SinceYear', 'Promo2SinceWeek', 'CompetitionOpenSinceMonth', and 'CompetitionOpenSinceYear', had their missing values replaced with the current times. This was done because the imputation of these columns with zero values would imply that the Promo has been running for 2023 years or that the Competition has been open for 2023x12 months. Using the current timing ensures that any calculation to determine the length of time a promotion has been running or competition has been open returns zero.

```
[266]: merged_dataset.isna().sum()

[266]: Store                        0
       DayOfWeek                    0
       Date                         0
       Sales                        0
       Customers                    0
       Open                         0
       Promo                        0
       StateHoliday                 0
       SchoolHoliday                0
       StoreType                    0
       Assortment                   0
       CompetitionDistance          0
       CompetitionOpenSinceMonth    0
       CompetitionOpenSinceYear     0
       Promo2                       0
       Promo2SinceWeek              0
       Promo2SinceYear              0
       PromoInterval                0
       CurrentWeekOfYear            0
       CurrentYear                  0
       CurrentMonth                 0
       dtype: int64
```

Figure 3.3: Dataset after handling missing values

Before moving on to the Exploratory Data Analysis, it was noticed that the ‘StateHoliday’ column had zero values in number and string format so these were replaced with ‘None’, to be encoded later.

Section 4: Exploratory Data Analysis

In terms of outliers, one value in each of the customer and sales columns was significantly higher than all of the others. Because it is unclear whether this is a correct value, it will be included in the machine learning model.

```
[271]: merged_dataset.plot(y=['Sales', 'Customers'], kind='box', subplots=True, layout=(2,2), figsize=(10,10))

[271]: Sales           AxesSubplot(0.125,0.53;0.352273x0.35)
       Customers    AxesSubplot(0.547727,0.53;0.352273x0.35)
       dtype: object
```

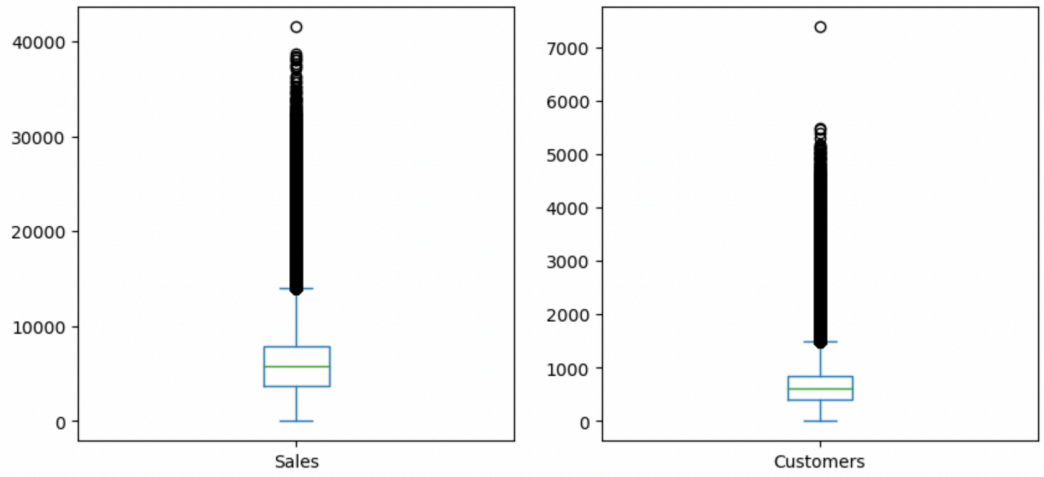

Figure 4.1: Boxplot of Sales and Customers column

Next, taking a look at the composition of the categorical variables in the dataset;

```
[272]: sns.countplot(data=merged_dataset,x='Assortment')

[272]: <AxesSubplot:xlabel='Assortment', ylabel='count'>
```

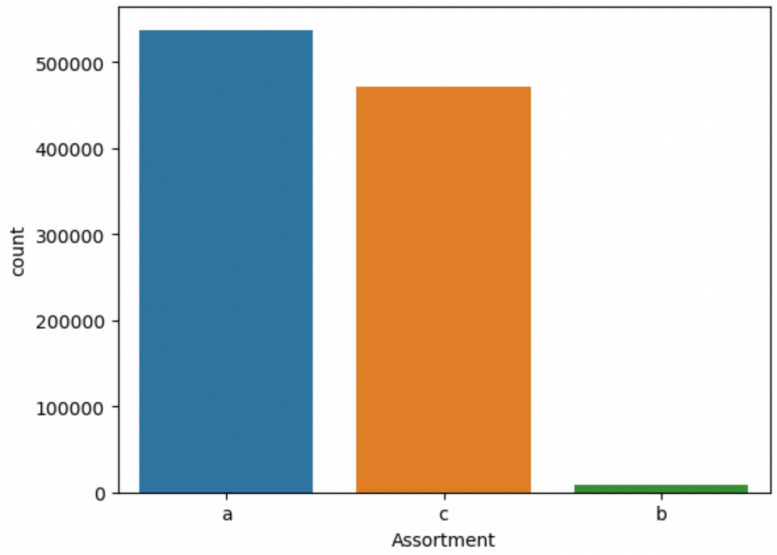

Figure 4.2: Countplot of Assortment column

The countplot shows there's more of the basic level assortment (a) in the stores.

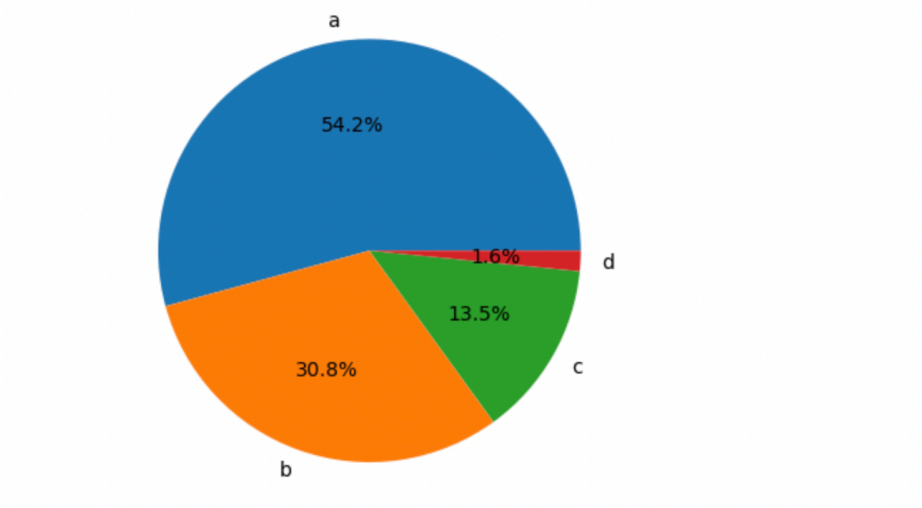

Figure 4.3: Pieplot of StoreType

Figure 4.3 shows the percentage distribution of the store types. Over half of the stores are of StoreType a.

More insight into the relationship between Sales and the other variables is as follows;

```
[268]: Text(0, 0.5, 'Sales')
```

Figure 4.4: Scatterplot of Sales vs Customers

The scatterplot shows that there's a positive linear relationship between Sales and the number of customers on any given day.

```
[269]: Text(0, 0.5, 'Sales')
```

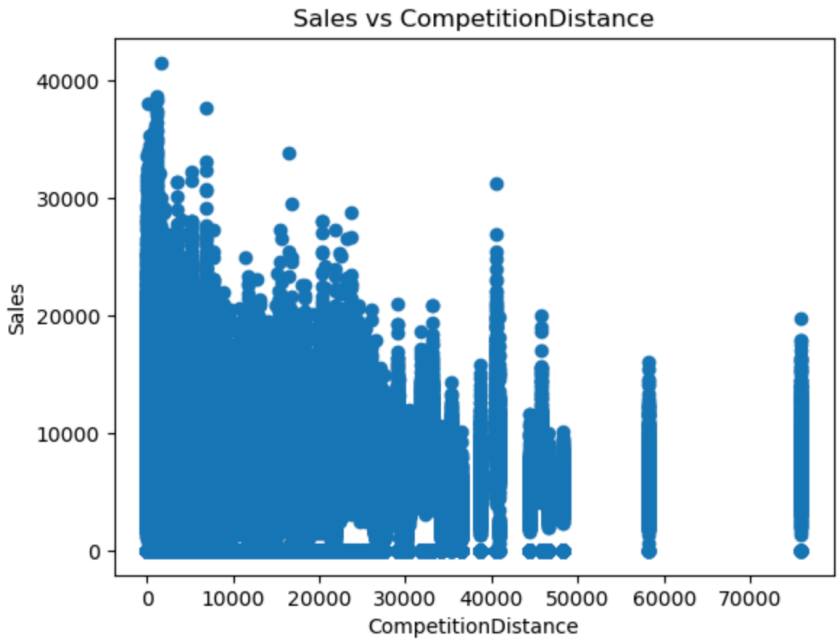

Figure 4.5: Scatterplot of Sales vs CompetitionDistance

There doesn't seem to be any strong linear relationship between Sales and the distance of the competition according to the scatterplot in Figure 4.5.

```
[278]: sns.barplot(data=merged_dataset, x="StateHoliday", y="Sales", hue="Promo2")
[278]: <AxesSubplot:xlabel='StateHoliday', ylabel='Sales'>
```

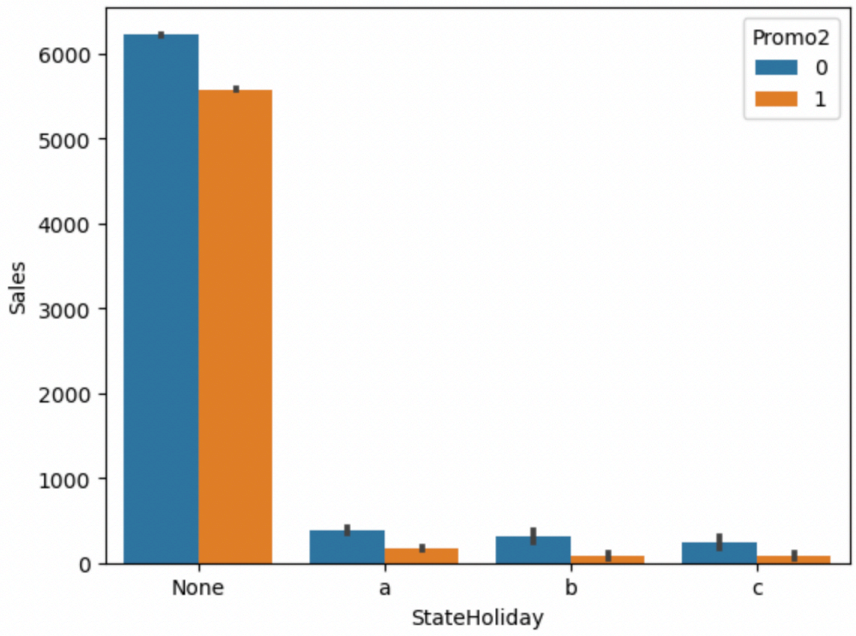

Figure 4.6: Barplot of Sales vs StateHoliday

The barplot shows that sales tend to be higher when there are no state holidays and when there are no ongoing promos.

```
[279]: sns.barplot(data=merged_dataset, x="SchoolHoliday", y="Sales")
[279]: <AxesSubplot:xlabel='SchoolHoliday', ylabel='Sales'>
```

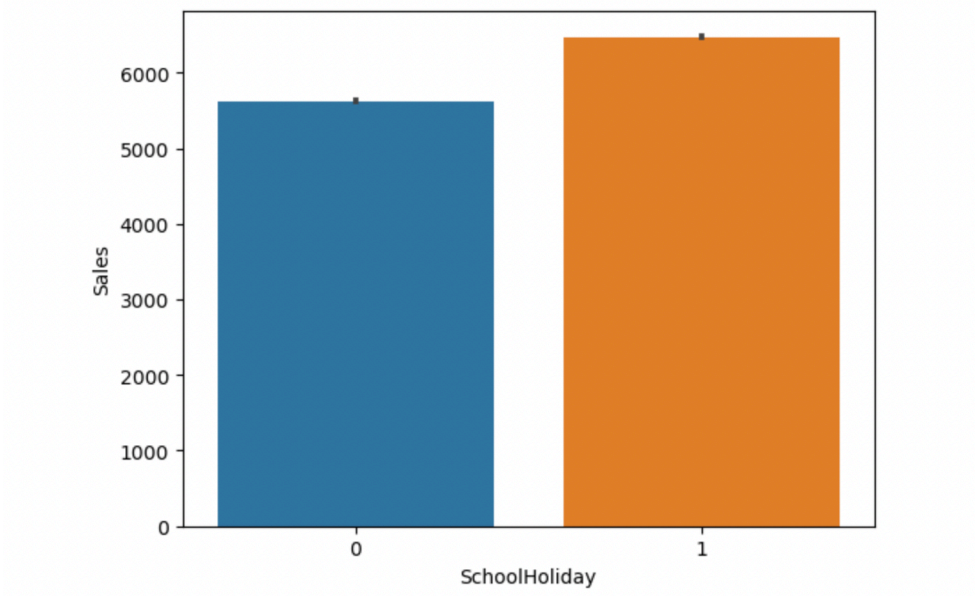

Figure 4.7: Barplot of Sales vs SchoolHoliday

On the contrary, Sales tend to be higher when there are school holidays.

Aggregation of Sales over time could also give another angle to the dynamics of sales;

[270]:

CurrentYear	2013	2014	2015
CurrentMonth			
1	4553.557545	5588.635568	5480.437841
2	5980.127675	5881.260352	6607.806746
3	5488.029510	6068.519247	5888.175647
4	6017.810224	6265.350757	6364.706404
5	6023.188485	5917.941586	5517.233667
6	5346.181375	5926.648316	6052.264610
7	5938.847447	6064.435556	6330.641428
8	5898.974772	5656.245867	4842.378219
9	5357.333124	5018.884655	6240.784241
10	5097.289975	5757.458773	5040.930301
11	5922.300329	5808.671198	4976.568610
12	6310.159815	5970.384467	4635.836771

Figure 4.8: Pivot table of Sales by Month and Year

Figure 4.8 shows that the year 2015 generally had higher sales, especially in the months of February, April & July.

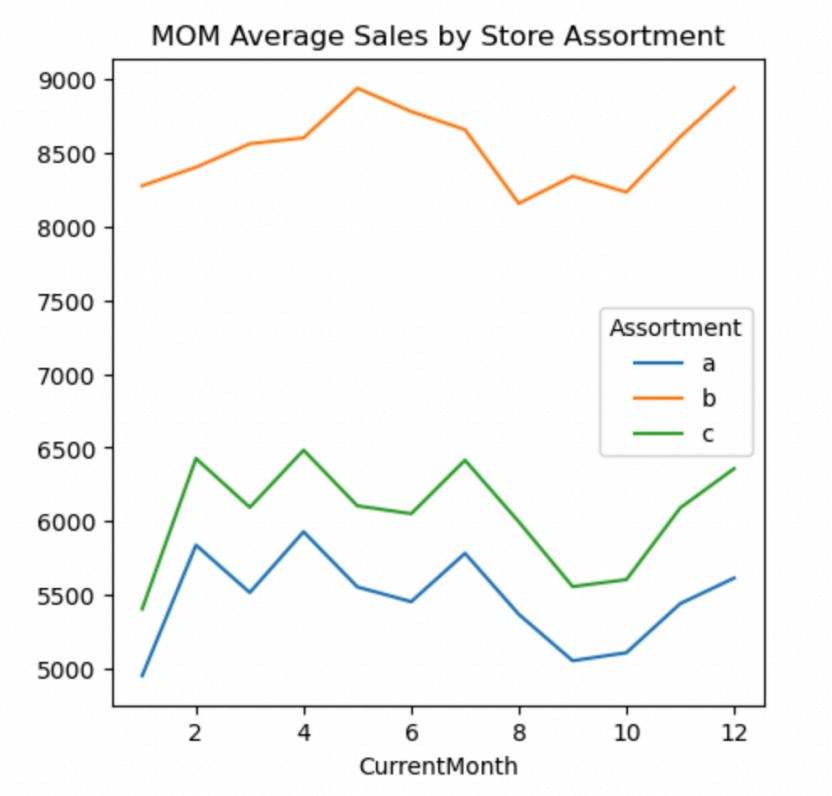

Figure 4.9: Month over month Average Sales by Assortment

Stores with the extra assortment (b) consistently had higher sales than stores with the other assortment levels. There was also a different Month over Month trend between assortment level b stores and the others, especially in the first 7 months of the year

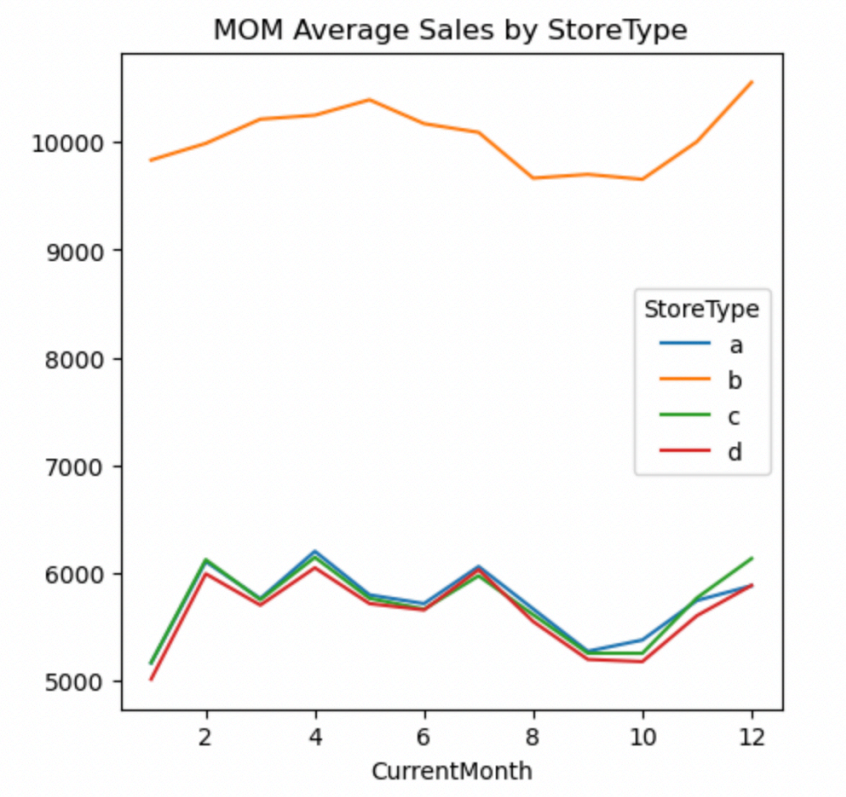

Figure 5.0: Month over month Average Sales by StoreType

Figure 5.0 shows that in terms of store types, StoreType b had significantly higher sales than the other store types

Finally, it's a good idea to see if some of the independent variables in the dataset are related.

```
[280]: sns.barplot(data=merged_dataset, x="StoreType", y="CompetitionDistance")
```

```
[280]: <AxesSubplot:xlabel='StoreType', ylabel='CompetitionDistance'>
```

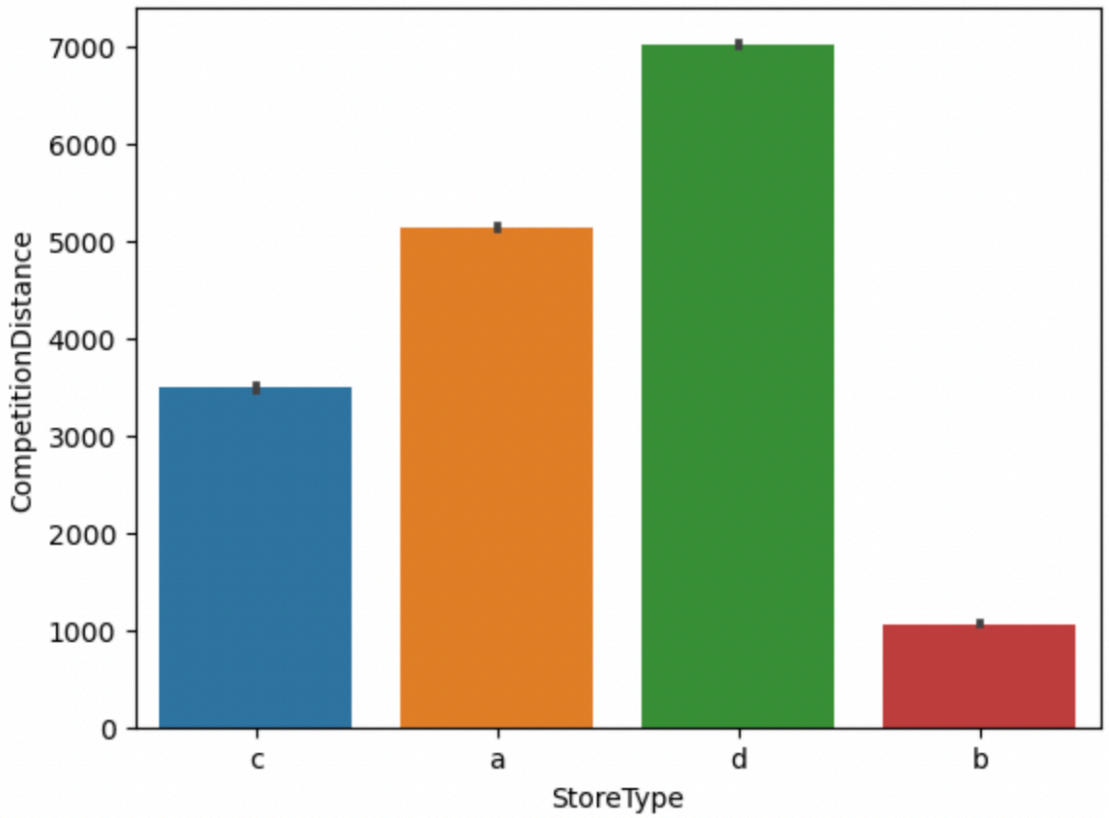

Figure 5.1: Barplot of StoreType vs CompetitionDistance

Stores of StoreType b seem to have competitors much closer to them than the others StoreTypes.

Section 5: Encoding

All categorical variables in the dataset were converted to numerical data because they could be useful inputs in the model. In the "PromoInterval" column, there were three possible combinations of promo interval months, each of which was replaced with the corresponding numerical value of the starting month, e.g. '**Jan**,Apr,Jul,Oct' = **1** and the fourth category 'NotApplicable' = 0. Because there was no need to impose an order on the categories, the OneHotEncoder was used to transform the values from categorical to numerical (binary) values for the "StoreType" and "StateHoliday" variables. Finally, the ordinal encoder was used for the "Assortment" variable because the assortment levels are ordered from basic to extended.

[320]:

	NoStateHoliday	PublicHoliday	EasterHoliday	ChristmasHoliday	StoreA	StoreB	StoreC	StoreD	Assortment
0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

Figure 6.1: Encoded variables added to the dataset

After encoding, the correlation between all the variables is shown in Figure 7 below;

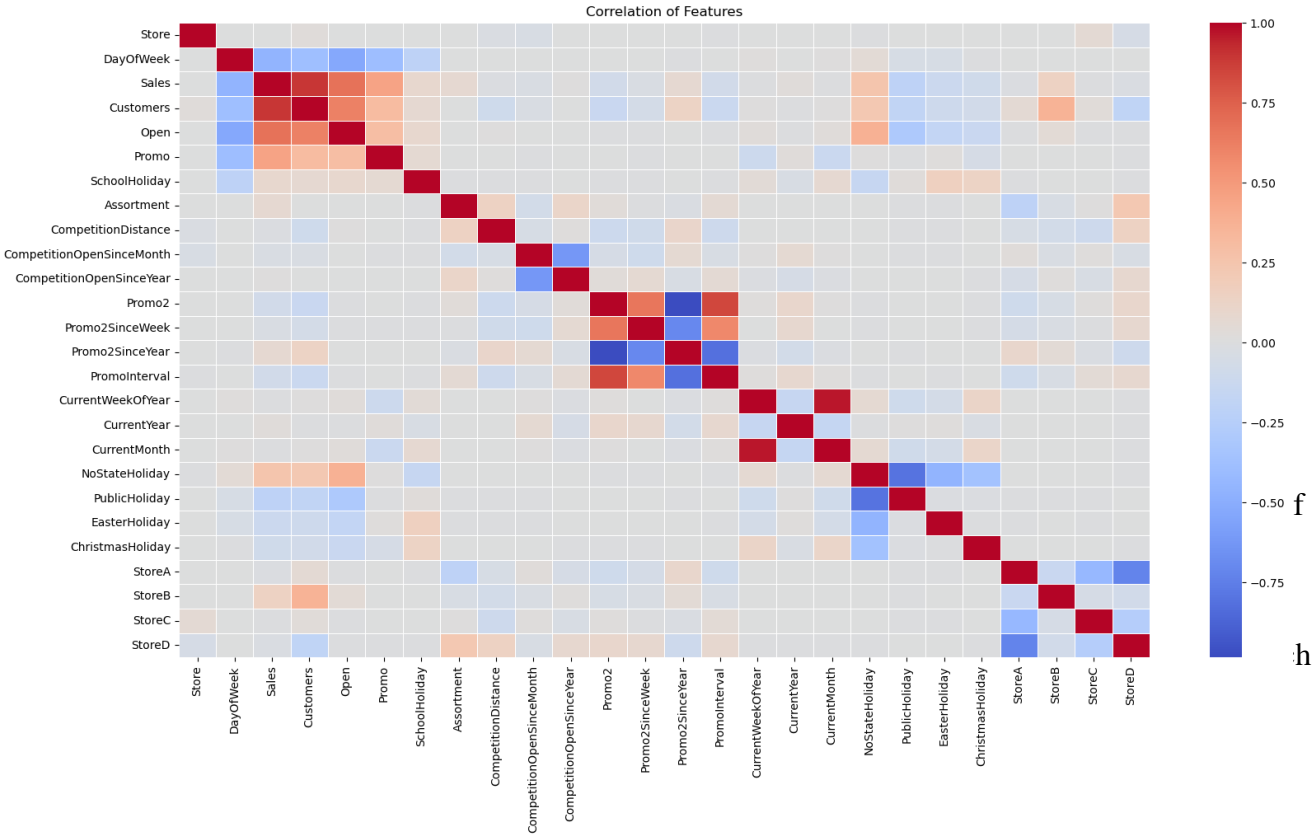

Figure 7: Correlation Matrix of all features

This demonstrates that sales are strongly related to the daily number of customers, the day of the week, whether the store is open or closed, and promotions.
Amongst the independent variables, the encoded sets of promotion, time, holiday, and store type variables have a high correlation. This was anticipated and is due to encoding.
To avoid redundancy, one would like to avoid multicollinearity between predictor variables, which will be accomplished through feature selection.

Section 8: Model Building

To begin model-building, the fully processed dataset was split into training and testing data with a ratio of 70:30. Then, for an initial understanding of the mutual information between each feature and the target, the SelectKbest method was used for feature selection, with the 'score function = mutual_info_regression' as its statistical test.

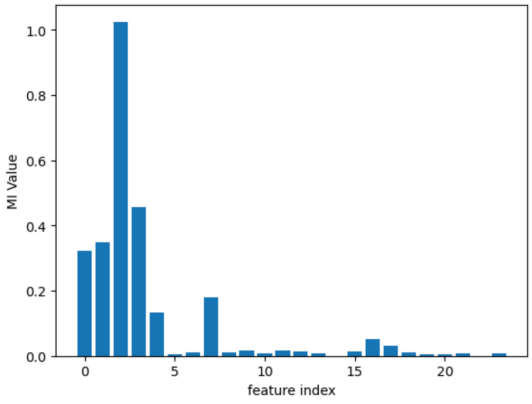

Figure 8.1: Barplot of MI Values for all features

The data was then trained with the XGB Regressor and the forward feature selection method was used to confirm the model with the best combination of the features.

```
[292]: #forward subset selection
model = xgb.XGBRegressor()
sfs1 = sfs(model, k_features=8, forward=True, verbose=2, scoring='r2')
sfs1 = sfs1.fit(X_train, y_train)
feat_names = list(sfs1.k_feature_names_)
print(feat_names)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   34.7s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:  3.1min finished

[2023-02-10 01:13:43] Features: 1/8 -- score: 0.8437539982098798[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   40.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:  4.6min finished

[2023-02-10 01:18:22] Features: 2/8 -- score: 0.9459845062063647[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   44.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed: 20.2min finished

[2023-02-10 01:38:35] Features: 3/8 -- score: 0.9588766842087197[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 31.3min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 52.1min finished

[2023-02-10 02:30:40] Features: 4/8 -- score: 0.9668205360492909[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   53.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:  3.6min finished

[2023-02-10 02:34:17] Features: 5/8 -- score: 0.969046385738559[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   56.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.8min finished

[2023-02-10 02:37:07] Features: 6/8 -- score: 0.969158652650896[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   58.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  2.0min finished

[2023-02-10 02:39:06] Features: 7/8 -- score: 0.9690795894370196[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
['Store', 'DayOfWeek', 'Customers', 'Open', 'Promo', 'CompetitionDistance', 'NoStateHoliday', 'PublicHoliday']
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.1min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.1min finished

[2023-02-10 02:40:11] Features: 8/8 -- score: 0.96888570469341
```

Figure 8.2: Selected features by Forward Subset Feature Selection

Figure 8.2 shows that the combination of between 4 and 8 features has around the same range of R2, which indicates how well these features explain the variation in sales, to be about 0.96. It also shows the retrieved features.

As a result, the final model was built using the eight features 'Store', 'DayOfWeek', 'Customers', 'Open', 'Promo', 'CompetitionDistance', 'NoStateHoliday', and 'PublicHoliday' and achieved a train score of 0.970 and a test score of 0.969.

```
• [295…  #Build final model
xgb = xgb.XGBRegressor()
xgb.fit(X_train, y_train)
score = xgb.score(X_train, y_train)
print("Training R-Squared score: ", score)

y_test_pred = xgb.predict(X_test)
score = metrics.r2_score(y_test, y_test_pred)
print("Test R-Squared score: ", score)
```

```
Training R-Squared score:  0.9706341157146571
Test R-Squared score:  0.9692678316597463
```

Figure 8.3: Final Model Training and Testing scores

One significant limitation of this model was its reliance on the 'Customers' predictor, which resulted in information leakage. Because this information was contained in the training data, which would not usually be available when making predictions on new data, the model appeared to perform well on training data in Figure 8.3 above but underperformed when applied to new data in Figure 8.4.

[298]:
```
#Show predicted values on unseen data
y_test_new = xgb.predict(test)
test['Predicted Sales'] = y_test_new
test.head()
```

[298]:

	Store	DayOfWeek	Customers	Open	Promo	CompetitionDistance	NoStateHoliday	PublicHoliday	Predicted Sales
0	1	4	NaN	1.0	1	1270.0	NaN	NaN	-406.467133
1	3	4	NaN	1.0	1	14130.0	NaN	NaN	-97.032486
2	7	4	NaN	1.0	1	24000.0	NaN	NaN	-140.904129
3	8	4	NaN	1.0	1	7520.0	NaN	NaN	-233.556870
4	9	4	NaN	1.0	1	2030.0	NaN	NaN	168.377991

Figure 8.4: Predicted Sales on test.csv dataset

***NB**: The features in the test.csv were engineered to match the number of features used in the model (Appendix 1.1 & 1.2).*

A recommended way to handle this limitation of information leakage is to re-train the model without the 'Customers' variable as input. This produces the following outcome:

[300]:
```
#Retrain model
xgb_retrained = xgb.XGBRegressor()
xgb_retrained.fit(X_train_retrained, y_train)
score = xgb_retrained.score(X_train_retrained, y_train)
print("Training R-Squared score: ", score)

y_test_pred = xgb_retrained.predict(X_test_retrained)
score = metrics.r2_score(y_test, y_test_pred)
print("Test R-Squared score: ", score)
```

```
Training R-Squared score:  0.8533478899150325
Test R-Squared score:  0.8527154807508938
```

Figure 8.5: Re-trained Model Training and Testing scores

The train score and test score are now 0.853 and 0.852 respectively. This means that, despite having a lower prediction accuracy, the re-trained model will be able to make more realistic predictions using the same features specified above but excluding the 'Customers' feature.

[324]:
```
#Show predicted values on unseen data - retrained
y_test_retrained = xgb_retrained.predict(test_retrain)
test_retrain['Predicted Sales'] = y_test_retrained
test_retrain.head()
```

[324]:

	Store	DayOfWeek	Open	Promo	CompetitionDistance	NoStateHoliday	PublicHoliday	Predicted Sales
0	1	4	1.0	1	1270.0	NaN	NaN	5169.819824
1	3	4	1.0	1	14130.0	NaN	NaN	7874.095215
2	7	4	1.0	1	24000.0	NaN	NaN	9648.726562
3	8	4	1.0	1	7520.0	NaN	NaN	7052.882812
4	9	4	1.0	1	2030.0	NaN	NaN	6963.276855

Figure 8.6: Predicted Sales on test.csv dataset using retrained model

Conclusion

To summarise, data preprocessing is necessary to ensure the quality and reliability of the data used to train a model. This report has covered a variety of data preprocessing techniques, such as creating new variables, dealing with missing values, and encoding categorical variables.

Furthermore, the report addressed the issue of information leakage, which occurs when training data contains information that would not have been available at the time predictions are made, resulting in overfitting and unreliable model performance.

To mitigate this problem, the report retrained a model, excluding the source of the information leakage.

By carefully preprocessing the data and avoiding information leakage, the retrained model will likely improve performance and generalisation ability.

REFERENCES

kaggle.com. (n.d.). *rossmann*. [online] Available at: https://www.kaggle.com/code/kariim/rossmann#gradient-boosting [Accessed 10 Feb. 2023].

kaggle.com. (n.d.). *Rossman Store Sales Exploratory Analysis*. [online] Available at: https://www.kaggle.com/code/jeannemolyneux/rossman-store-sales-exploratory-analysis [Accessed 10 Feb. 2023].

Singh, H. (2021). *Forward Feature Selection | Implementation of Forward Feature Selection*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/04/forward-feature-selection-and-its-implementation/.

Appendix

[327]:
```
test=pd.merge(test,store,on='Store',how='left')
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41088 entries, 0 to 41087
Data columns (total 27 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Store                         41088 non-null  int64
 1   DayOfWeek                     41088 non-null  int64
 2   Date                          41088 non-null  datetime64[ns]
 3   Sales                         0 non-null      float64
 4   Customers                     0 non-null      float64
 5   Open                          41077 non-null  float64
 6   Promo                         41088 non-null  int64
 7   StateHoliday                  41088 non-null  object
 8   SchoolHoliday                 41088 non-null  int64
 9   StoreType_x                   41088 non-null  object
 10  Assortment_x                  41088 non-null  object
 11  CompetitionDistance_x         40992 non-null  float64
 12  CompetitionOpenSinceMonth_x   25872 non-null  float64
 13  CompetitionOpenSinceYear_x    25872 non-null  float64
 14  Promo2_x                      41088 non-null  int64
 15  Promo2SinceWeek_x             23856 non-null  float64
 16  Promo2SinceYear_x             23856 non-null  float64
 17  PromoInterval_x               23856 non-null  object
 18  StoreType_y                   41088 non-null  object
 19  Assortment_y                  41088 non-null  object
 20  CompetitionDistance_y         40992 non-null  float64
 21  CompetitionOpenSinceMonth_y   25872 non-null  float64
 22  CompetitionOpenSinceYear_y    25872 non-null  float64
 23  Promo2_y                      41088 non-null  int64
 24  Promo2SinceWeek_y             23856 non-null  float64
 25  Promo2SinceYear_y             23856 non-null  float64
 26  PromoInterval_y               23856 non-null  object
dtypes: datetime64[ns](1), float64(13), int64(6), object(7)
memory usage: 8.8+ MB
```

Appendix 1.1: Merged test dataset

[321…
```
#Add columns to unseen dataset to match model
feat_names = ['Store' ,'DayOfWeek' ,'Customers' ,'Open' ,'Promo' ,'CompetitionDistance','NoStateHoliday' ,'PublicHoliday']
test[['NoStateHoliday','PublicHoliday']] = np.nan
test = test.loc[:,feat_names]
test.head()
```

[321]:

	Store	DayOfWeek	Customers	Open	Promo	CompetitionDistance	NoStateHoliday	PublicHoliday
0	1	4	NaN	1.0	1	1270.0	NaN	NaN
1	3	4	NaN	1.0	1	14130.0	NaN	NaN
2	7	4	NaN	1.0	1	24000.0	NaN	NaN
3	8	4	NaN	1.0	1	7520.0	NaN	NaN
4	9	4	NaN	1.0	1	2030.0	NaN	NaN

Appendix 1.2: Engineered test data