# DATA71011: Understanding Data and their Environment

## Coursework Project: ROSSMANN Sales Forecasting

### 1. Introduction

Providing insight into future sales performance allows for the optimisation of inventory management, enhanced financial planning, and improved decision-making and efficiency across an organization. This report presents the process of developing an accurate and reliable sales forecasting model for 1,115 Rossmann drug stores.

It starts by describing and assessing the quality of the data provided. It will then discuss the main findings from data exploration before explaining the steps taken to pre-process the data for machine learning (ML) models. Section 5 will consider the rationale behind choosing the model and describe the training and scoring process. After presenting the results of model performance a discussion will address key findings, assumptions and limitations prior to a conclusion.

### 2. Data

The dataset store.csv consists of 1,115 observations each relating to a unique store. All stores are described by 9 variables, which include assignment into categorical store and assortment types. A binary variable indicates whether stores have adopted ‘Promo2’, a promotion which runs continually and is renewed every three months. Further attributes define the week, and year in which ‘Promo2’ was introduced and detail the monthly renewal cycle. Remaining attributes state the distance of a store’s nearest competitors (in meters), and the month and year in which they opened. For three stores, no values are provided for the distance to the nearest competitor. Moreover, 354 stores are missing information on when the nearest competitor was opened.

Train.csv contains 1,017,209 observations which record the value of daily sales made by stores in the period 01/01/13 – 31/07/15. Along with store number, sales figures, and date, observations detail: the day of the week, whether the store was open, the number of customers, if a store-specific promotion applied, and the holiday status of that day.

Train.csv is missing observations for 180 stores in the 184-day period between 01/07/14 and 31/12/14. Otherwise, store 988 on 01/01/13 is the only other missing instance and there are no missing values.

The test.csv dataset contains the same variables as train.csv over the forecasting period 01/08/15 – 17/09/15. Values are not provided in the sales and customer columns as these are to be predicted. The only other missing values in correspond to store 622 which has no values for the variable 'Open' in the 11, non-Sunday days between 05/09/15 - 17/09/15.

Test.csv is missing observations for 259 stores, but all 48 days of the period are present for the remaining 856 and there are no missing values.

Overall, the provided data is of high quality. Records are largely complete - missing observations in train.csv account for only 3.15% of the historical data. Importantly, there is no detectable pattern to missing values or observations which, if present, would introduce bias into ML models. Data entry is highly consistent, with all values satisfying a high level of validity and uniqueness. Therefore, assuming that the data is of high accuracy, it is in a good condition for further analysis.

## 3. Exploratory Data Analysis (EDA)

Figure 1 plots the change in average monthly sales across stores, according to the different promotions which are ongoing, over the time period of the training dataset. From this visualisation it is evident that the 'Promo' variable has a massive impact on sales whereas stores which participate in Promo2 often have worse sales performance than otherwise. Also evident in this graph is the seasonal trend in December, where sales increase significantly. Interestingly the impact of December on sales amplified when 'Promo' is also applied.

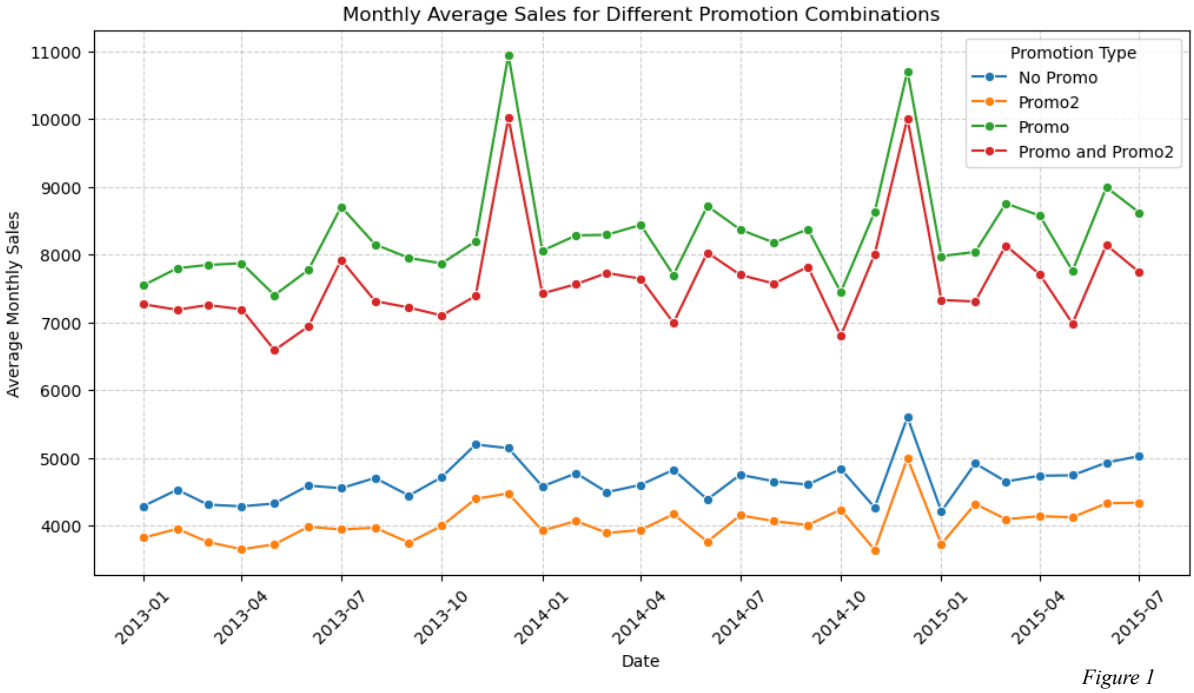


*Figure 1*

Figure 2 illustrates how the distribution of sales vary according to the combination of store type and assortment. Stores of type a, c and d have similar sales distributions irrespective of assortment whereas store type b provides a significantly different distributions across all assortment levels. It implies that store type b could be an influential variable for predicting sales and assortment may have limited explanatory capacity unless it relates to these stores.

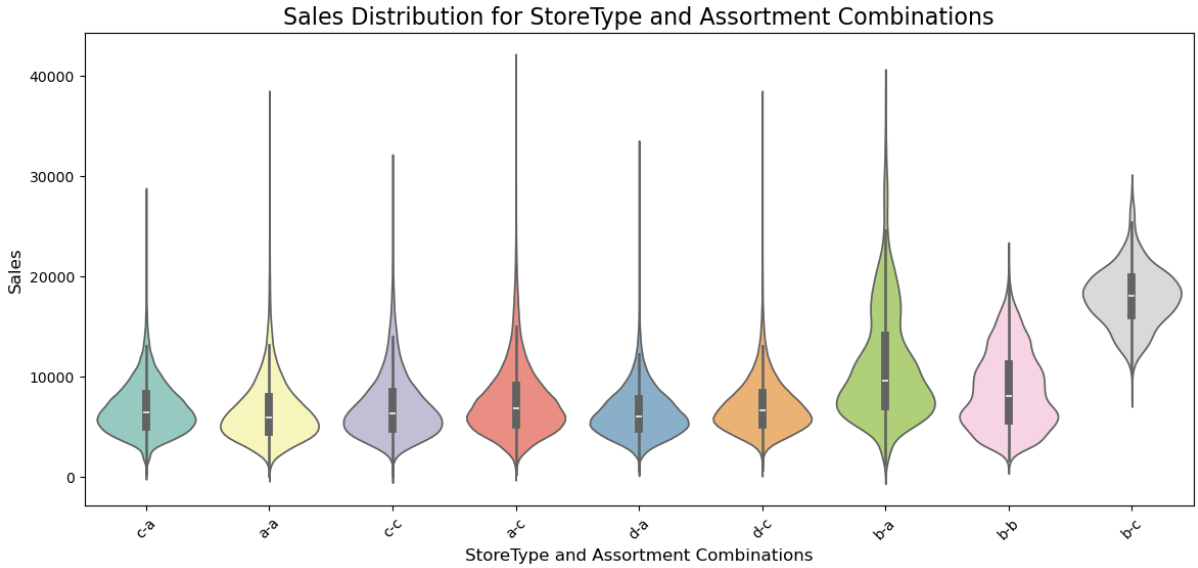


*Figure 2*

## 4. Pre-Processing

### Transformation and Integration

Store.csv is merged with both train.csv and test.csv so that each observation contains the relevant store-specific data. For consistency, all further transformation and integration steps are applied to both the training and test sets.

Upon merging the datasets, 'Promo2' transformed so that it only equals 1 for dates which proceed the week and year in which each store adopted the promotion. This transformation assumes that 'Promo2' is adopted on the first day of the week in which its participation begins.

Categorical variables relating to store types, assortments, the day of the week, and state holidays are transformed using one-hot encoding to create binary variables representing each individual category. To prevent perfect collinearity, the dummy variables referring to: store type 'a', assortment 'a', day of the week '2', and state holiday '0', are dropped and will form a baseline form which output can be compared to.

EDA does not show clear seasonality of sales across years, however, there is a noticeable trend across months. Extracting the month of the year from the date provided in train.csv and performing further one-hot encoding (dropping January) ensures that ML models can capture this variability.

Finally, the binary variables 'Promo' and 'SchoolHoliday' are preserved in their original state as they are suitable for ML modelling.

### Data Reduction

All variables relating to a store's nearest competitor are dropped from the dataset to maintain consistency across observations. Data on competition only relates to current competitors. Given that 48.88% of current competitors opened after the start of the historical sales period, this data cannot accurately be matched to observations which predate their arrival.

After transforming the variable, 'Promo2', attributes referring to the week and year in which participation began, are dropped. The customers variable is removed because it does not have values in the test dataset and observations where the store is closed are removed, as they correspond to zero sales in all instances so do not require prediction.

### Data Cleaning

Whilst there are 33,121 observations missing from the training dataset, no attempt is made to impute them because imputation would be difficult, and the relative size of missing instances is not large enough to significantly affect the statistical power of the ML models.

The only remaining missing values refer to store 622 in the testing set where missing values in the variable 'Open' have been transformed to 1 because this store is typically open from Monday – Saturday.

## 5. Model Fitting

To forecast the sales data, a standard linear regression model is used. Whilst this is a simple model, unlike more complex, deep learning alternatives, it maintains interpretability through coefficient values which provide estimates for the effect of individual variables on the response.

For this project, instead of fitting one model to the whole dataset predictions a unique model is fitted to all 1,115 stores in order to produce more accurate. In this way, forecasts for each store are produced by models which are only trained on its own sales data. This individualised approach allows for variables relating to store type and assortment to be dropped as they no longer have explanatory value.

The linear regression models are trained on the data using a 10-fold cross-validation method. The training set is divided into 10 folds with 9 folds used to train the model and the remaining fold used to test its accuracy. By repeating this procedure 10 times, each fold is used for both training and testing. The final accuracy score for each model is the average of the 5 testing rounds. Using this method prevents overfitting by ensuring the models generalise well to unseen data and reduces the variance of performance estimates.

The performance metric used in this project is the Root Mean Squared Percentage Error (RMSPE) which is defined by the equation below:

$$\text{RMSPE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2},$$

Where a model's RMSPE average error in percentage terms of its estimation in comparison to the real sales value.

## 6. Results

After performing cross-validation on all 1,115 stores, the average RMSPE across all models was 0.1797. This means that on average, predictions made by the linear regression models will deviate from the true value by 17.97%. There was significant variation of scores across store models with a minimum and maximum of 0.0908 and 1.3239. A median value of 0.1682 shows that the mean is raised by a right skew to the distribution.

A key benefit of linear regression models are the interpretable coefficients. The average coefficient values across all models are displayed in figure 3. From these results it is evident that December, Mondays, and store-specific promotions are the main drivers of positive sales value.

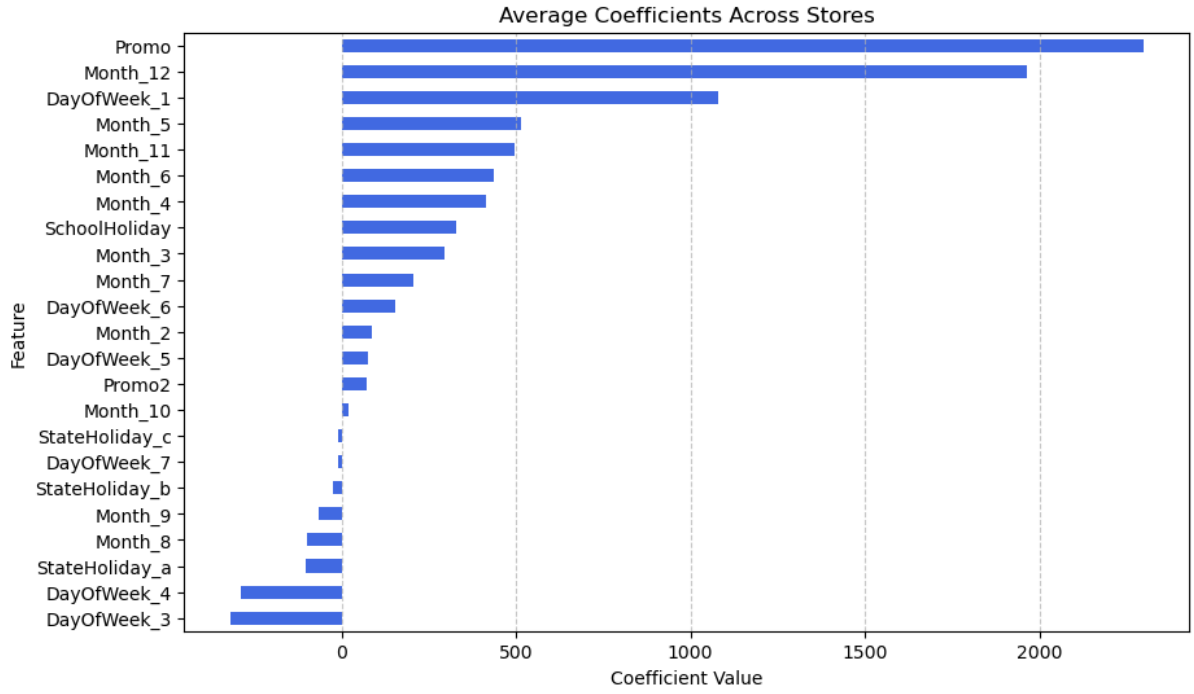


*Figure 3*

## 7. Discussion

The model's RMSPE of 0.1797 indicates a reasonable level of accuracy, yet there is room for improvement, particularly given the large variation in scores across stores. A key strength of these models is interpretability since linear regression allows for clear insight into which variable influence sales. Additionally, fitting separate models for each store boosts accuracy, as stores are only trained on their own data.

However, this approach has several limitations. The assumption of linear relationships between variables may be too simplistic to capture all of the interactions occurring in the data. A non-linear model such as random forest could improve predictive accuracy although this would come at a cost of interpretability. Additionally, the model could be improved by including time-dependent variables such sales lags. Adding autoregressive features to the help the model capture temporal dependencies and adjust to immediate trends.

## 8. Conclusion

Following a structured approach consisting of, EDA, pre-processing and model training, this project develops unique sales forecasting models for each of the 1,115 Rossmann stores in Germany. The average RMSPE of 0.1797 demonstrates moderate accuracy performance with significant variability across stores. Strengths of the model include its interpretability and tailored approach, but it may lack the complexity to increase the precision of its predictions. To support further iterations of this project, Rossmann should consider expanding data collection practices to provide more data of higher resolution and granularity which would aide forecasting efforts.

# Appendix

## Python code:

```
### UNDERSTANDING DATA AND THEIR ENVIRONEMNT - COURSEWORK ###


#Installing packages

import os

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import datetime

from sklearn.model_selection import KFold

from sklearn.metrics import mean_squared_error

from sklearn.linear_model import LinearRegression


#Reading the data

os.chdir('/Users/luke/Documents/University/MSc Data Science/Data and Environment/Coursework')

store = pd.read_csv('Store.csv')

train = pd.read_csv('Train.csv')

test = pd.read_csv('Test.csv')




## DESCRIBINBG THE DATA ##


#Store - missing observations

store.value_counts()


#Store - missing values

store_missing_values = store[store.isnull().any(axis=1)]

store_missing_values.isnull().sum()


#Train - missing observations

train['Date'] = pd.to_datetime(train['Date'])

test['Date'] = pd.to_datetime(test['Date'])

stores = train['Store'].unique()

start_date = '2013-01-01'
```

```
end_date = '2015-07-31'
date_range = pd.date_range(start=start_date, end=end_date)
full_index = pd.MultiIndex.from_product([stores, date_range], names=['Store', 'Date'])
full_df = pd.DataFrame(index=full_index).reset_index()
full_df['Date'] = pd.to_datetime(full_df['Date'])
merged_df = full_df.merge(train, on=['Store', 'Date'], how='left')
missing_data = merged_df[merged_df.isnull().any(axis=1)]
missing_by_store = missing_data['Store'].value_counts()
missing_by_date = missing_data['Date'].value_counts()
#percentage of missing values
len(missing_data)/(len(train)+len(missing_data))

#Train - missing values
train_missing_values = train[train.isnull().any(axis=1)] #empty

#Test - missing observations
list1 = store['Store'].unique()
list2 = test['Store'].unique()
values_not_in_list2 = [x for x in list1 if x not in list2]

#Test - missing values
test.isnull().sum() #11 missing values on Open
test_missing_values = test[test['Open'].isnull()] #store 622

#Checking for consistent data types
store.apply(lambda col: col.map(type).nunique() if col.dtype == 'O' else 1)
train.apply(lambda col: col.map(type).nunique() if col.dtype == 'O' else 1)
test.apply(lambda col: col.map(type).nunique() if col.dtype == 'O' else 1)
store['PromoInterval'].unique()
train['StateHoliday'].unique()



## PRE-PROCESSING ##

## Transformation and Integration

#Merging the store data with the training and test datsets
```

```
attributes_train = pd.merge(train, store, on='Store', how='inner')

attributes_test = pd.merge(test, store, on='Store', how='inner')


#Promo2 is integrated into a new variable (train set)

attributes_train['Promo2SinceWeek'] = pd.to_numeric(attributes_train['Promo2SinceWeek'],
errors='coerce').fillna(-1).astype(int)

attributes_train['Promo2SinceYear'] = pd.to_numeric(attributes_train['Promo2SinceYear'],
errors='coerce').fillna(-1).astype(int)

def calculate_first_day_of_week(year, week):

    try:

        # Use the year and week number to calculate the first day of the week

        return datetime.datetime.strptime(f'{year}-W{int(week)}-1', "%Y-W%U-%w").date()

    except (ValueError, TypeError):

        return None  # Handle invalid or NaN values by returning None

attributes_train['Promo2Since'] = attributes_train.apply(

    lambda row: calculate_first_day_of_week(row['Promo2SinceYear'], row['Promo2SinceWeek']), axis=1)

attributes_train['Promo2Since'] = pd.to_datetime(attributes_train['Promo2Since'], errors='coerce')

attributes_train['Date'] = pd.to_datetime(attributes_train['Date'], errors='coerce')

attributes_train['Promo2'] = attributes_train.apply(

    lambda row: 1 if pd.notnull(row['Promo2Since']) and row['Date'] >= row['Promo2Since'] else 0,

    axis=1)


#Promo2 is integrated into a new variable (test set)

attributes_test['Promo2SinceWeek'] = pd.to_numeric(attributes_test['Promo2SinceWeek'],
errors='coerce').fillna(-1).astype(int)

attributes_test['Promo2SinceYear'] = pd.to_numeric(attributes_test['Promo2SinceYear'],
errors='coerce').fillna(-1).astype(int)

attributes_test['Promo2Since'] = attributes_test.apply(

    lambda row: calculate_first_day_of_week(row['Promo2SinceYear'], row['Promo2SinceWeek']), axis=1)

attributes_test['Promo2Since'] = pd.to_datetime(attributes_test['Promo2Since'], errors='coerce')

attributes_test['Date'] = pd.to_datetime(attributes_test['Date'], errors='coerce')

attributes_test['Promo2'] = attributes_test.apply(

    lambda row: 1 if pd.notnull(row['Promo2Since']) and row['Date'] >= row['Promo2Since'] else 0,

    axis=1)


#Extracting month from the date

attributes_train['Month'] = attributes_train['Date'].dt.month

attributes_test['Month'] = attributes_test['Date'].dt.month
```

```
## Plotting time series grpah to show seasonal trends and differences between promo statuses


attributes_train['YearMonth'] = attributes_train['Date'].dt.to_period('M')

monthly_avg_sales = attributes_train.groupby(['YearMonth', 'Promo',
'Promo2'])['Sales'].mean().reset_index()

monthly_avg_sales['YearMonth'] = monthly_avg_sales['YearMonth'].astype(str)

monthly_avg_sales['YearMonth'] = pd.to_datetime(monthly_avg_sales['YearMonth'])

monthly_avg_sales['Promo_Combination'] = monthly_avg_sales.apply(

    lambda row: 'No Promo' if row['Promo'] == 0 and row['Promo2'] == 0 else

                'Promo' if row['Promo'] == 1 and row['Promo2'] == 0 else

                'Promo2' if row['Promo'] == 0 and row['Promo2'] == 1 else

                'Promo and Promo2', axis=1)


plt.figure(figsize=(12, 6))

sns.lineplot(data=monthly_avg_sales, x='YearMonth', y='Sales', hue='Promo_Combination', marker='o')

plt.xlabel('Date')

plt.ylabel('Average Monthly Sales')

plt.title('Monthly Average Sales for Different Promotion Combinations')

plt.legend(title='Promotion Type')

plt.xticks(rotation=45)

plt.grid(True, linestyle='--', alpha=0.6)

plt.show()



## Data Reduction


#Variables relating to competition distance are removed

attributes_train.drop(columns=['CompetitionDistance', 'CompetitionOpenSinceMonth',
'CompetitionOpenSinceYear'], inplace=True)

attributes_test.drop(columns=['CompetitionDistance', 'CompetitionOpenSinceMonth',
'CompetitionOpenSinceYear'], inplace=True)


#Extra variables relating to 'Promo 2' dropped

attributes_train.drop(columns=['Promo2Since', 'Promo2SinceWeek', 'Promo2SinceYear', 'PromoInterval'],
inplace=True)

attributes_test.drop(columns=['Promo2Since', 'Promo2SinceWeek', 'Promo2SinceYear', 'PromoInterval'],
inplace=True)
```

```
#Customers are removed as a variable

attributes_train = attributes_train.drop(columns=['Customers'])

attributes_test = attributes_test.drop(columns=['Customers'])


#Removing cases where the store is closed and then dropping open as a variable

attributes_train = attributes_train[attributes_train['Open'] != 0]

attributes_test = attributes_test[attributes_test['Open'] != 0]


#Removing sales from attributes_test

attributes_test.drop(columns=['Sales'], inplace=True)



## Data Cleaning


#Removing cases of zero sales

zero_sales = attributes_train[attributes_train['Sales'] == 0]

attributes_train = attributes_train[attributes_train['Sales'] != 0]


#Filling missing values for store 622

attributes_test.fillna(0, inplace=True)


#Removing Open as a variable

attributes_train.drop(columns=['Open'], inplace=True)

attributes_test.drop(columns=['Open'], inplace=True)



#Type-Assortment combination on sales

attributes_train['Store_Assortment_Combo'] = attributes_train['StoreType'] + '-' +
attributes_train['Assortment']

plt.figure(figsize=(14, 6))

sns.violinplot(x='Store_Assortment_Combo', y='Sales', data=attributes_train, palette='Set3')

plt.title("Sales Distribution for StoreType and Assortment Combinations", fontsize=16)

plt.xlabel("StoreType and Assortment Combinations", fontsize=12)

plt.ylabel("Sales", fontsize=12)

plt.xticks(rotation=45)

plt.show()
```

```
#One-hot ecnoding explanatory variables

attributes_train = pd.get_dummies(attributes_train, columns=['Month'], prefix='Month',
drop_first=True)

attributes_train = pd.get_dummies(attributes_train, columns=['StoreType'], prefix='StoreType',
drop_first=True)

attributes_train = pd.get_dummies(attributes_train, columns=['Assortment'], prefix='Assortment',
drop_first=True)

attributes_train = pd.get_dummies(attributes_train, columns=['DayOfWeek'], prefix='DayOfWeek',
drop_first=False)

attributes_train = attributes_train.drop(columns=['DayOfWeek_2'])

attributes_train['StateHoliday'] = attributes_train['StateHoliday'].astype(str)

attributes_train = pd.get_dummies(attributes_train, columns=['StateHoliday'], prefix='StateHoliday',
drop_first=True)


attributes_test = pd.get_dummies(attributes_test, columns=['Month'], prefix='Month', drop_first=True)

attributes_test = pd.get_dummies(attributes_test, columns=['StoreType'], prefix='StoreType',
drop_first=True)

attributes_test = pd.get_dummies(attributes_test, columns=['Assortment'], prefix='Assortment',
drop_first=True)

attributes_test = pd.get_dummies(attributes_test, columns=['DayOfWeek'], prefix='DayOfWeek',
drop_first=False)

attributes_test = attributes_test.drop(columns=['DayOfWeek_2'])

attributes_test['StateHoliday'] = attributes_test['StateHoliday'].astype(str)

attributes_test = pd.get_dummies(attributes_test, columns=['StateHoliday'], prefix='StateHoliday',
drop_first=True)




## Matching columns between train and test

missing_cols = set(attributes_train.columns) - set(attributes_test.columns)

for col in missing_cols:

    attributes_test[col] = False

attributes_test = attributes_test[attributes_train.columns]

attributes_test = attributes_test.drop(columns=['Sales'])




## MODEL FITTING ##
```

```
#Creating lists for linear regression outputs
rmse_list = []
rmspe_list = []
coefficients_list = []
predictions_list = []

#Creating a dictionary to store individual models
store_models = {}

#Making a list of feature columns
feature_columns = ['Promo', 'SchoolHoliday',
                   'Promo2', 'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6', 'Month_7',
                   'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Month_12',
                   'DayOfWeek_1', 'DayOfWeek_3', 'DayOfWeek_4', 'DayOfWeek_5',
                   'DayOfWeek_6', 'DayOfWeek_7', 'StateHoliday_a',
                   'StateHoliday_b', 'StateHoliday_c']

#Function to calculate RMSPE
def rmspe(y_true, y_pred):
    return np.sqrt(np.mean(((np.abs(y_true - y_pred)) / y_true) ** 2))

# Train Linear Regression models for each store
for store_id in attributes_train['Store'].unique():
    store_data = attributes_train[attributes_train['Store'] == store_id].copy()
    X_store = store_data[feature_columns]
    y_store = store_data['Sales']
    model = LinearRegression()
    kf = KFold(n_splits=10, shuffle=False)

    rmse_scores = []
    rmspe_scores = []

    #Perform cross-validation
    for train_index, val_index in kf.split(X_store):
        X_train, X_val = X_store.iloc[train_index], X_store.iloc[val_index]
        y_train, y_val = y_store.iloc[train_index], y_store.iloc[val_index]
```

```
        #Model fit and prediction
        model.fit(X_train, y_train)
        y_val_pred = model.predict(X_val)

        #Calculkate RMSE and RMSPE for the fold
        rmse_fold = np.sqrt(mean_squared_error(y_val, y_val_pred))
        rmse_scores.append(rmse_fold)
        rmspe_fold = rmspe(y_val, y_val_pred)
        rmspe_scores.append(rmspe_fold)

    #Average RMSE and RMSPE across folds and storing values
    avg_rmse = np.mean(rmse_scores)
    avg_rmspe = np.mean(rmspe_scores)
    rmse_list.append({'Store': store_id, 'RMSE': avg_rmse})
    rmspe_list.append({'Store': store_id, 'RMSPE': avg_rmspe})

    #Model is refit to all of a store's data
    model.fit(X_store, y_store)
    store_models[store_id] = model #stored to allow for future prediction

    #Storing coefficiecnts
    coefficients_dict = {'Store': store_id}
    for feature, coef in zip(feature_columns, model.coef_):
        coefficients_dict[feature] = coef
    coefficients_list.append(coefficients_dict)


#Create dataframes
rmse_df = pd.DataFrame(rmse_list)
rmspe_df = pd.DataFrame(rmspe_list)
coefficients_df = pd.DataFrame(coefficients_list)

#Get the average rmse and rmspe across all models
average_rmse = rmse_df['RMSE'].mean()
average_rmspe = rmspe_df['RMSPE'].mean()

#Average coefficients
```

```
average_coefficients = coefficients_df.drop(columns=['Store']).mean().sort_values(ascending=False)


#Looking at distirbution of RMSPE
rmspe_df.describe()



#Plotting coefficients
plt.figure(figsize=(10, 6))
average_coefficients.plot(kind='barh', color='royalblue')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.title('Average Coefficients Across Stores')
plt.gca().invert_yaxis()  # Largest bars at the top
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.show()




## MAKING PREDICTIONS ##


predictions_list = []


#Make predicition on each store
for store_id in attributes_test['Store'].unique():
    store_data = attributes_test[attributes_test['Store'] == store_id].copy()
    X_store = store_data[feature_columns]
    model = store_models.get(store_id)
    if model:
        store_data['Predicted_Sales'] = model.predict(X_store)
        predictions_list.append(store_data[['Store', 'Date', 'Predicted_Sales']])


#Create dataframe of predictions
predictions = pd.concat(predictions_list, ignore_index=True)


# Display sample predictions
print(predictions.head())
```