National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Sruthi Prabakaran Paruthipattu
Student ID: x19223269

School of Computing
National College of Ireland

Supervisor: Dr. Barry Haycock

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sruthi Prabakaran Paruthipattu
Student ID:	x19223269
Programme:	Data Analytics
Year	2021-2022
Module:	Research Project
Lecturer:	Dr. Barry Haycock
Submission Due Date:	16/12/21
Project Title:	Demand Forecasting based on external factors using clustering and machine learning
Word Count:	1736
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sruthi Prabakaran Paruthipattu

Date: 16/12/21

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sruthi Prabakaran Paruthipattu
Student ID: x19223269

1 Introduction

The main purpose of this manual is to provide all the steps undertaken to perform this research to help replicate the same in the future. It contains the steps performed to study and implement demand forecasting based on external factors.

2 Hardware Requirements

The project was performed in Windows 10 with the 64-bit operating system on Intel® Core(TM) i5 Processor with a RAM of 8GB and other configurations as mentioned in the figure below.

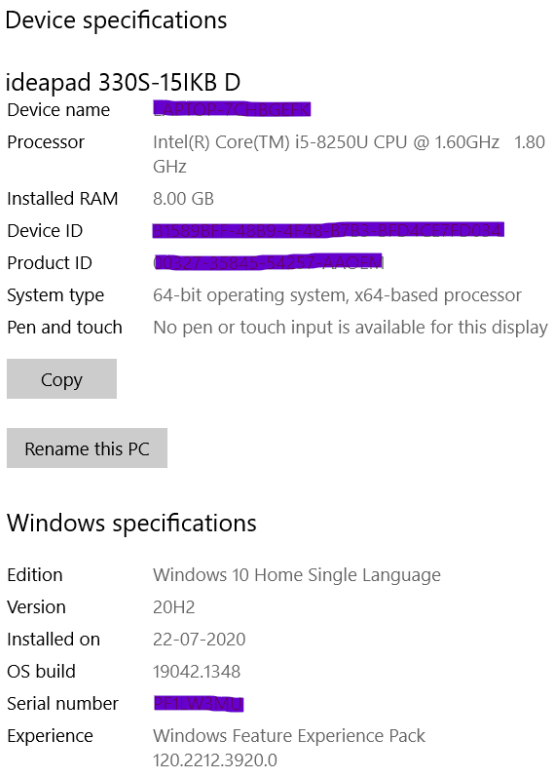


Figure 1: Hardware Configuration

3 Software Requirements

The project was coded in Python3. And Anaconda Navigator was installed and used for this purpose. The Installation can be done for windows from the official website[1]

[1] https://docs.anaconda.com/anaconda/install/windows/

Upon Installation, the Anaconda Navigator looks like figure 2. One can find JupyterLab, Notebook, and other external applications in the navigator. For this research, Jupyter Notebook was used to code and maintain the results obtained.

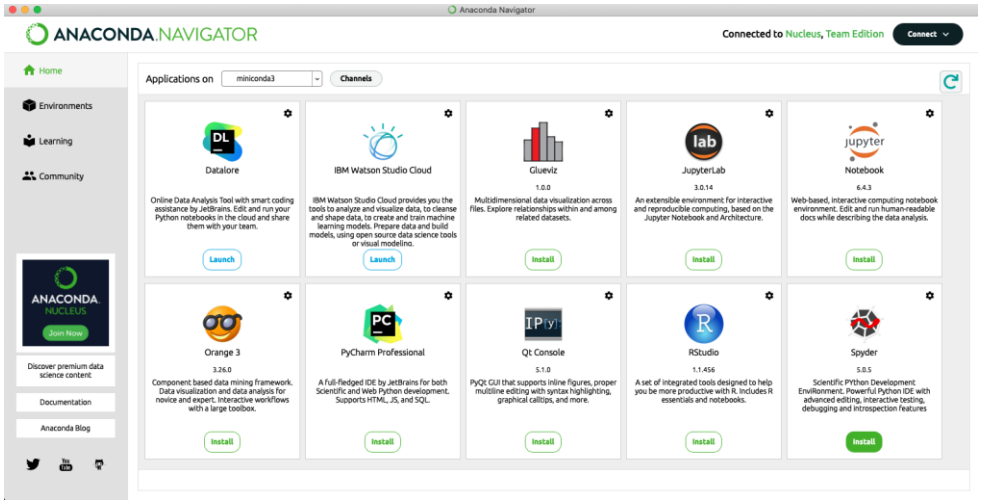


Figure 2: Anaconda Navigator

4 Library Packages

The following Libraries were used to implement different aspects of the project

- NumPy
- SciPy
- MathPlotLib
- Seaborn
- Tensorflow
- Pandas
- Keras
- Sklearn

```
import pandas as pd
import numpy as np
import timeit
import datetime

import seaborn as sns
color = sns.color_palette()
import matplotlib.pyplot as plt
import altair as alt
import plotly.offline as py
import shap

import random
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn import metrics
import math
from sklearn.preprocessing import minmax_scale

import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import tensorflow as tf

from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.layers import LSTM
from keras.layers import BatchNormalization
```

Figure 3: Libraries used

5 Dataset Description

The data, Corporación Favorita[2] used was obtained from Kaggle. It contains sales and transaction information of grocery items. The supermarket chain is located in Ecuador and has multiple branches throughout the country. The data is distributed into 7 files: Train, Test, Transactions, Stores, Holidays, Oil price, and Items.

- **Train and Test:** Contains around 5 GB of sales data every day for 4 years starting from 2013 for a variety of products across multiple stores
- **Items:** Contains description of items sold by the supermarket. The items are described using Item id, family, whether it is perishable or not, and class
- **Store:** Contains store details including store number, city, state, locale, and type of the store and cluster to which the store belongs
- **Holiday events:** Contains holidays between 2013 and 2017, and whether the holiday was transferred from the original day.
- **Oil Prices:** Contains daily oil prices because Ecuador is an oil-dependent country
- **Transactions:** Contains the ay level transaction details of all the stores

6 Dataset Pre-processing

The data pre-processing part involves multiple modules which are explained in detail below:

6.1. Clustering

Clustering was done on the transaction data and unit_ sales data. Due to system constraints, the data was sampled and only the data of the year 2017.

1 Clustering using transaction data

i. Necessary libraries are imported as mentioned in section 4
ii. Required data were imported into the notebook using pd.read() with necessary data types like in figure 4.

```
store = pd.read_csv('stores.csv', dtype={'store_nbr': 'int8', 'cluster': 'int8'})
```

Figure 4: Data import

[2] https://www.kaggle.com/c/favorita-grocery-sales-forecasting/data

iii. The holiday's data have specific days where the holiday was due to an event, transferred from another day, a bridge between two holidays, or an additional holiday. All these days were converted as holidays and days which were originally the holiday were assigned being a workday as in the figure below.

```
holidays = pd.read_csv("holidays_events.csv")
holidays["date"] = pd.to_datetime(holidays["date"])
holidays['type'] = holidays['type'].replace(['Additional','Bridge','Event','Transfer'], 'Holiday')
mask = (holidays['transferred'] == True)
holidays['type'][mask] = 'Work Day'
print(holidays['type'].value_counts())
```

```
holidays['Year'] = pd.DatetimeIndex(holidays['date']).year
holidays['Month'] = pd.DatetimeIndex(holidays['date']).month
holidays['Day'] =pd.DatetimeIndex(holidays['date']).day.astype(np.uint8)
```

```
holidays = holidays.drop(labels=['Year','Month','Day','transferred'],axis=1)
```

Figure 5: Pre-processing holiday data

iv. The date was initially converted to DateTime. Like in the above figure, the year, month, and day were extracted using DatetimeIndex from the date.

v. The train data was subsampled for the year 2017.

```
1 transaction_2017 = transactions.loc[transactions['Year']==2017]
```

```
1 transaction_2017.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12097 entries, 71391 to 83487
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   date          12097 non-null  datetime64[ns]
 1   store_nbr     12097 non-null  int8          
 2   transactions  12097 non-null  int16         
 3   Year          12097 non-null  int64         
 4   Month         12097 non-null  int64         
 5   Day           12097 non-null  uint64        
dtypes: datetime64[ns](1), int16(1), int64(2), int8(1), uint64(1)
memory usage: 508.0 KB
```

Figure 6: Locating data for the year 2017

vi. Next, the datasets were merged using pd. merge() based on the left join

```
data = pd.merge(transactions, holidays[['date','type','locale']], on = 'date', how = 'left')
```

```
train_data = pd.merge(left=data, right=store[['store_nbr','city','type']], how='left', left_on='store_nbr', right_on='store_
```

Figure 7: Data Merge

vii. Next, the data was cleaned by renaming column names, checking for null values, encoding categorical variables. The null values were replaced using the

dataframe.replace() function. For encoding, the Label encoder from sklearn.preprocessing was used.

```
train_data = train_data.rename(columns={"type_y": "store_type", "type_x": "day_type"})
```

```
train_data.isnull().sum().sort_values(ascending=False)
```

```
locale          71096
day_type        71096
store_type          0
city                0
Day                 0
Month               0
Year                0
transactions        0
store_nbr           0
date                0
dtype: int64
```

```
train_data['locale'] = train_data['locale'].replace(np.nan, 'No Holiday')
train_data['day_type'] = train_data['day_type'].replace(np.nan, 'Work Day')
```

```
train_data
```

```
labelencoder=LabelEncoder()
train_data['city'] = labelencoder.fit_transform(train_data['city'])
train_data['day_type'] = labelencoder.fit_transform(train_data['day_type'])
train_data['locale'] = labelencoder.fit_transform(train_data['locale'])
train_data['store_type'] = labelencoder.fit_transform(train_data['store_type'])
```

Figure 8: Handling null values and categorical values

viii. The transaction values were scaled using np.log to distribute the values around a range.

ix. Since store type is used in the clustering process, the datatype was converted as a categorical variable

x. Using the isocalender() function, the date was decoded into the year, week, and day count for aggregation and visualization

```
train_data['year'],train_data['week'],train_data['day']=list(zip(*train_data.date.apply(lambda x: x.isocalendar())))
```

Figure 9: Extracting year, week, and day from the date

2 Clustering using unit sales

i. The same process as mentioned in the above section was performed for this as well. The only difference was that in place of transaction data, train data was used to perform clustering of unit sales.

3 Grouping of Clusters and EDA on individual clusters

i. The data was visualized to see the distribution of stores by the default clustering provided in the dataset, type of stores, count of stores across the states and cities.

```
#Count of stores in different types and clusters
plt.figure(figsize=(15,12))

plt.subplot(221)
# Count of stores for each type
temp = store['cluster'].value_counts()
#plot
sns.barplot(temp.index,temp.values,color=color[5])
plt.ylabel('Count of stores', fontsize=12)
plt.xlabel('Cluster', fontsize=12)
plt.title('Store distribution across cluster', fontsize=15)
```

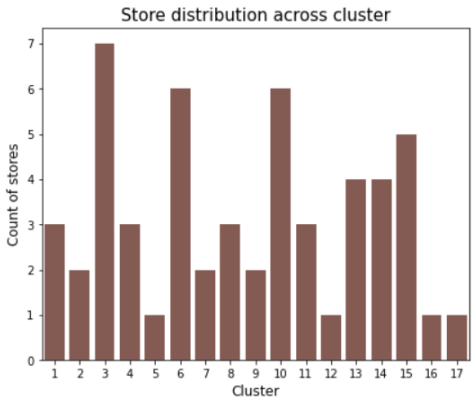


Figure 10: Store distribution across the default clusters

```
plt.subplot(222)
# Count of stores for each type
temp = store['type'].value_counts()
#plot
sns.barplot(temp.index,temp.values,color=color[7])
plt.ylabel('Count of stores', fontsize=12)
plt.xlabel('Type of store', fontsize=12)
plt.title('Store distribution across store types', fontsize=15)
```

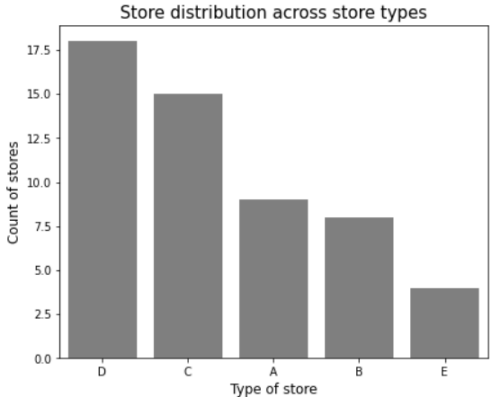


Figure 11: Store distribution across different store types

```
plt.subplot(223)
# Count of stores for each type
temp = store['state'].value_counts()

sns.barplot(temp.index,temp.values,color=color[8])
plt.ylabel('Count of stores', fontsize=12)
plt.xlabel('state', fontsize=12)
plt.title('Store distribution across states', fontsize=15)
plt.xticks(rotation='vertical')
```

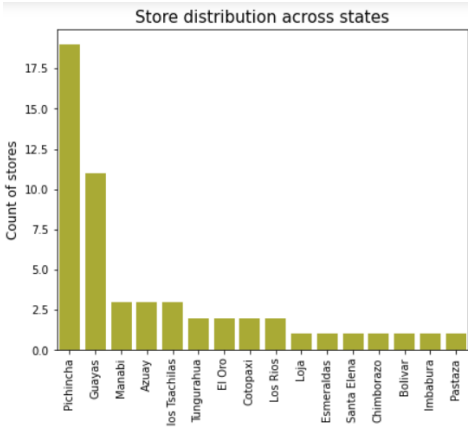


Figure 12: Store distribution across states in Ecuador

```
plt.subplot(224)
# Count of stores for each type
temp = store['city'].value_counts()

sns.barplot(temp.index,temp.values,color=color[9])
plt.ylabel('Count of stores', fontsize=12)
plt.xlabel('City', fontsize=12)
plt.title('Store distribution across cities', fontsize=15)
plt.xticks(rotation='vertical')
plt.show()
```

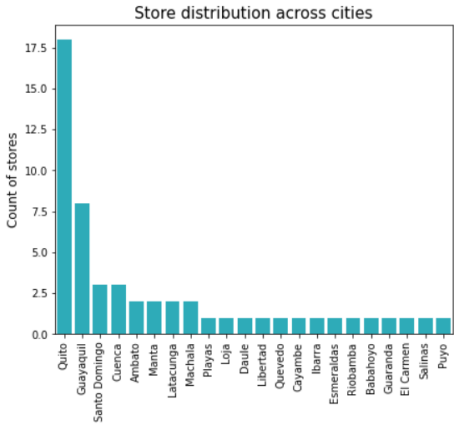


Figure 13: Store distribution across cities in Ecuador

ii. To group the clusters separately, the input from the user was obtained and only stores present in that cluster was read from train data

iii. The date column was converted to DateTime and the data for the year 2017 was extracted

iv. The data were merged using left join.

v. The weather data was not available in the dataset and was downloaded for each city in Ecuador separately for the year 2017 from Meteostat[3]

vi. Merging weather data was a challenge since it involved multiple cities. So, this was performed using a regular column formation as in the figure below

```
# city = ['Ambato','Babahoyo','Cayambe','Esmeraldas','Guaranda'
if (train_data['city'] == 'Ambato').any():
    train_data['tavg'] = Ambato['tavg']
if (train_data['city'] == 'Babahoyo').any():
    train_data['tavg'] = Babahoyo['tavg']
if (train_data['city'] == 'Cayambe').any():
    train_data['tavg'] = Cayambe['tavg']
if (train_data['city'] == 'Esmeraldas').any():
    train_data['tavg'] = Esmeraldas['tavg']
if (train_data['city'] == 'Guaranda').any():
    train_data['tavg'] = Guaranda['tavg']
if (train_data['city'] == 'Guyaquil').any():
    train_data['tavg'] = Guyaquil['tavg']
if (train_data['city'] == 'Latacunga').any():
    train_data['tavg'] = Latacunga['tavg']
if (train_data['city'] == 'Libertad').any():
    train_data['tavg'] = Libertad['tavg']
if (train_data['city'] == 'Loja').any():
    train_data['tavg'] = Loja['tavg']
if (train_data['city'] == 'Machala').any():
    train_data['tavg'] = Machala['tavg']
if (train_data['city'] == 'Quito').any():
    train_data['tavg'] = Quito['tavg']
if (train_data['city'] == 'Quevedo').any():
    train_data['tavg'] = Quevedo['tavg']
if (train_data['city'] == 'Riobamba').any():
    train_data['tavg'] = Riobamba['tavg']
if (train_data['city'] == 'Salinas').any():
    train_data['tavg'] = Salinas['tavg']
if (train_data['city'] == 'Santo Domingo').any():
    train_data['tavg'] = Santo_domingo['tavg']
```

Figure 14: Merging weather data

[3] https://meteostat.net/en/

vii. Once all the merging process was done, the same data cleaning process as mentioned in the above section was performed.

viii. Based on the cluster number, the pre-processed data was saved for future use.

7 Modeling

7.1. Clustering on transaction data

i. Different aggregation values were computed based on different combinations of features using the group_by function

```
#creating a table with means and std of transaction volume per type of day per store
Means1=train_data.groupby(['store_nbr','day_type']).transactions.agg(['mean','std']).unstack(level=1)

#creating a table with means and std of transaction volume per day of the week per store
Means2=train_data.groupby(['store_nbr','day']).transactions.agg(['mean','std']).unstack(level=1)

# Creating a table  with the daily average of transaction volume per store
sales_by_store=train_data.groupby(['store_nbr']).transactions.sum()/train_data.groupby(['store_nbr']).transactions.count()
# Creating a new columns with ratio of transactions of the day / daily average
train_data['normalized']=[v/sales_by_store[s] for (s,v) in zip(train_data.store_nbr,train_data.transactions)]

#creating a table with means and std of normalized transaction volume per type of day per store
Means1_norm=train_data.groupby(['store_nbr','day_type']).normalized.agg(['mean','std']).unstack(level=1)
#creating a table with means and std of normalized transaction volume per day of the week per store for working day and hol
Means2_norm=train_data.groupby(['store_nbr','day'])['day_type','normalized'].agg(['mean','std']).unstack(level=1)
#creating a table with means of normalized transaction volume per type of day per store
Means3_norm=train_data.groupby(['store_nbr','day_type'])['normalized'].agg(['mean']).unstack(level=1)
#creating a table with means and std of normalized transaction volume per day of the week per store
Means4_norm=train_data.groupby(['store_nbr','day'])['normalized'].agg(['mean','std']).unstack(level=1)
```

Figure 15: Aggregation of transaction data

ii. Dendrograms were constructed for each aggregated value and were visualized. Out of all the four dendrograms, the aggregation based on day type for each store on normalized transaction values seemed to be more clear and the clustering was done based on this. The visual representation of each cluster is given in the Annex.

```
clustering=AggClust(n_clusters=6)
cluster=clustering.fit_predict(Means1_norm)
store['new_cluster']=cluster
plt.figure(figsize=(10, 7))
plt.scatter(store['store_nbr'], store['new_cluster'], c=cluster)
```

Figure 16: Creating clusters

7.2. Clustering based on unit sales

i. Two types of aggregation were performed. The first one was based on the store number for every item with unit sales. The second one was based on store number, item number, and day of the week for unit sales. For both cases, only the mean value was used.

```
Means2_norm=train_data.groupby(['store_nbr','item_nbr'])['unit_sales'].agg(['mean']).unstack(level=1)
```

```
Means1_norm=train_data.groupby(['store_nbr','item_nbr','day'])['unit_sales'].agg(['mean']).unstack(level=1)
```

Figure 17: Aggregation based on unit sales

ii. Since the results contained infinite and null values, they had to be handled.

iii. The aggregation based on store and item number was considered by analyzing the results from the dendrogram. It seemed more clear and there was more separation between the connecting tree lines. All the visualizations are provided in the Annex.

7.3. Random Forest for cluster 3

i. The random forest regressor was initialized with the below parameters

```
rf = RandomForestRegressor(n_jobs = -1, n_estimators = 15,min_samples_split = 10,random_state=0)
y = rf.fit(Xg_train, Yg_train)
print('Train accuracy',rf.score(Xg_train,Yg_train))
```

Figure 18: Random Forest regressor

7.4. XGBoost for cluster 3

i. The XGBoost model was run for 20 rounds with the following set of parameters.

```
def rmspe(y, yhat):
    try:
        rmspe = np.sqrt(np.mean((yhat / y-1) ** 2))
    except ZeroDivisionError:
        print("Oh, no! You tried to divide by zero!")
    return rmspe
```

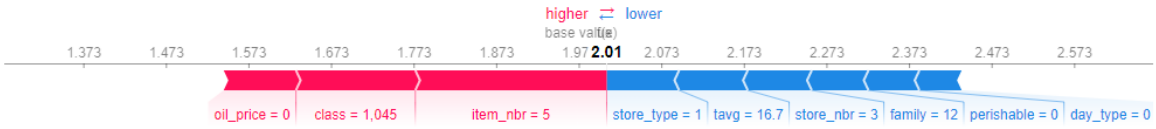
```
def rmspe_xg(yhat, y):
    y = np.expm1(y.get_label())
    yhat = np.expm1(yhat)
    return "rmspe", rmspe(y, yhat)
```

```
params = {"objective": "reg:linear",
          "booster" : "gbtree",
          "eta": 0.3,
          "max_depth": 25,
          "subsample": 0.9,
          "colsample_bytree": 0.7,
          "silent": 1,
          "seed": 1301
          }
num_boost_round = 20
watchlist = [(dtrain, 'train'), (dvalid, 'eval')]
```

```
gbm = xgb.train(params, dtrain, num_boost_round, evals = watchlist,
  early_stopping_rounds = 20, feval = rmspe_xg, verbose_eval = True)
```

Figure 19: XGBoost regressor

ii. The results were visualized using SHAP.




```
shap.force_plot(explainer.expected_value,shap_values[1,:],X_train.iloc[1,:])
```

Figure 20: SHAP plot for each row

iii. The overall SHAP visualization is provided in the Annex.

7.5. Univariate LSTM & RF for cluster 3

i. Firstly, the data were scaled using Standard Scaler

```
scaler = StandardScaler()
# transform data
train_scaled = scaler.fit_transform(train)
test_scaled = scaler.fit_transform(test)

print(train_scaled)
print(train_scaled.shape)
print(test_scaled.shape)
```

Figure 21: Scaling data using Standard Scaler

ii. The unit sales data and date from cluster 3 data was converted into time-series data and reshaped to be processed by the LSTM

```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

look_back = 7
trainX, trainY = create_dataset(train_scaled, look_back)
testX, testY = create_dataset(test_scaled, look_back)
print(trainX.shape)
print(testX.shape)


trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Figure 22: Creating time series data for LSTM

iii. LSTM network has the LSTM layer with 16 neurons and consecutive dense and dropout of 0.2. The output layer consists of one neuron.

```
model = Sequential()
model.add(LSTM(16, input_shape=(1, look_back)))
model.add(Dense(8))
model.add(Dropout(0.2))

model.add(Dense(8))
model.add(Dropout(0.2))

model.add(Dense(4))
model.add(Dropout(0.1))

model.add(Dense(2))
model.add(Dropout(0.1))

model.add(Dense(1))


model.compile(loss = 'mse', optimizer='adam', metrics=['mse','mae'])
```

Figure 23: LSTM network

iv. The predicted values were changed back to the original by retracing the lookback

```
trainPredictPlot = np.empty_like(training_data)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

testPredictPlot = np.empty_like(testing_data)
testPredictPlot[:, :] = np.nan
testPredictPlot[look_back:len(testPredict)+look_back, :] = testPredict
```

Figure 24: Retracing to original data

v. The predicted values were passed to a dataframe and the residuals were calculated by subtracting the original target from the predicted value.

```
results_train = pd.DataFrame(trainPredict)
results_test = pd.DataFrame(testPredict)

results_train["trainY"]=trainY
results_test["testY"]=testY
```

```
results_train=results_train.rename(columns={0: "Predicted"})
results_test=results_test.rename(columns={0: "Predicted"})
```

```
train_res = results_train["Predicted"]-results_train["trainY"]
test_res = results_test["Predicted"]-results_test["testY"]
```

Figure 25: Calculating residuals

vi. The Random forest model was regressed with the residual value from the previous step

```
rf = RandomForestRegressor(n_jobs = 1, n_estimators = 10,min_samples_split = 10,random_state=0)
y = rf.fit(X_train, Y_train)
```

```
Y_train=Y_train.values.reshape(2539480,1)
Y_train.shape
```

```
(2539480, 1)
```

Figure 26: Random forest

vii. The results from the random forest were added to the prediction made from the LSTM network

```
results_train["RandomForest"]=pd.DataFrame(y_pred_train)
results_test["RandomForest"]=pd.DataFrame(y_pred_test)
```

```
results_train["final_prediction"] = results_train["RandomForest"]+results_train["Predicted"]
results_test["final_prediction"] = results_test["RandomForest"]+results_test["Predicted"]
```

```
prediction = results_train["final_prediction"]
test_pred = results_test["final_prediction"]
```

Figure 27: Calculating the final prediction

viii. The same process was repeated for cluster 5

Annex

i. Results of Clustering using transaction data

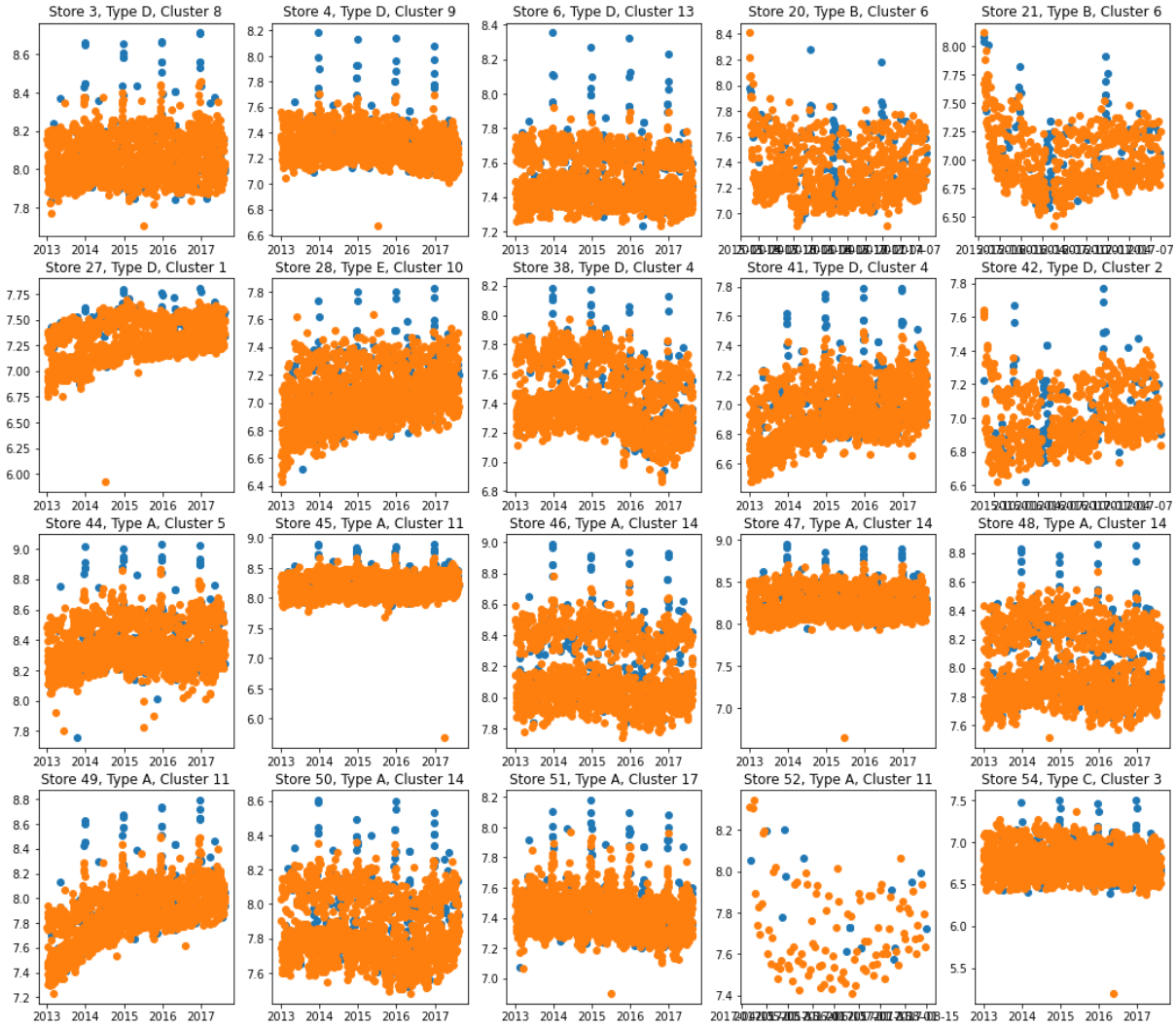


Figure 28a: Transaction data: Cluster 0

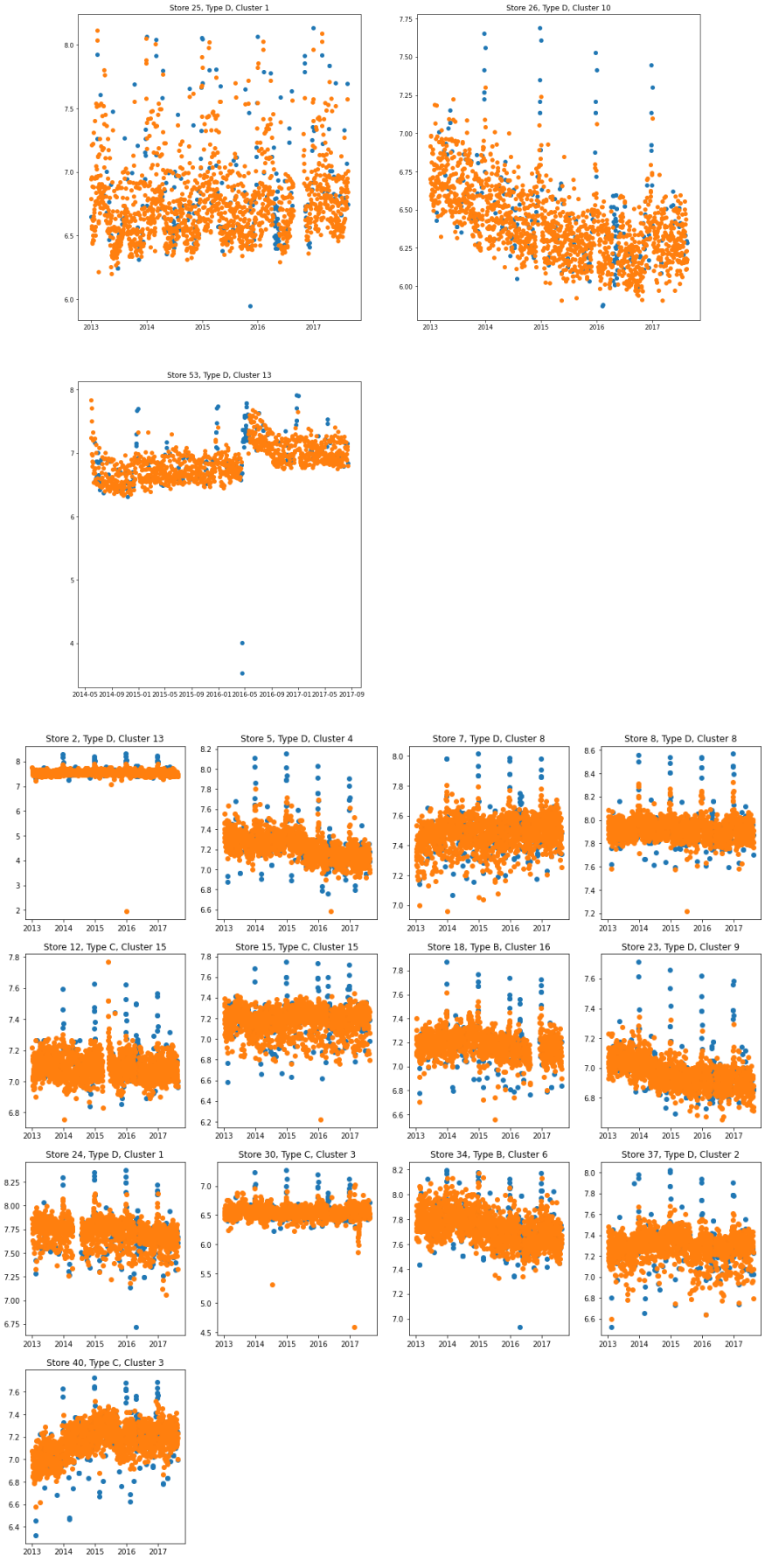


Figure 29: Transaction data: Cluster 1 and Cluster 2

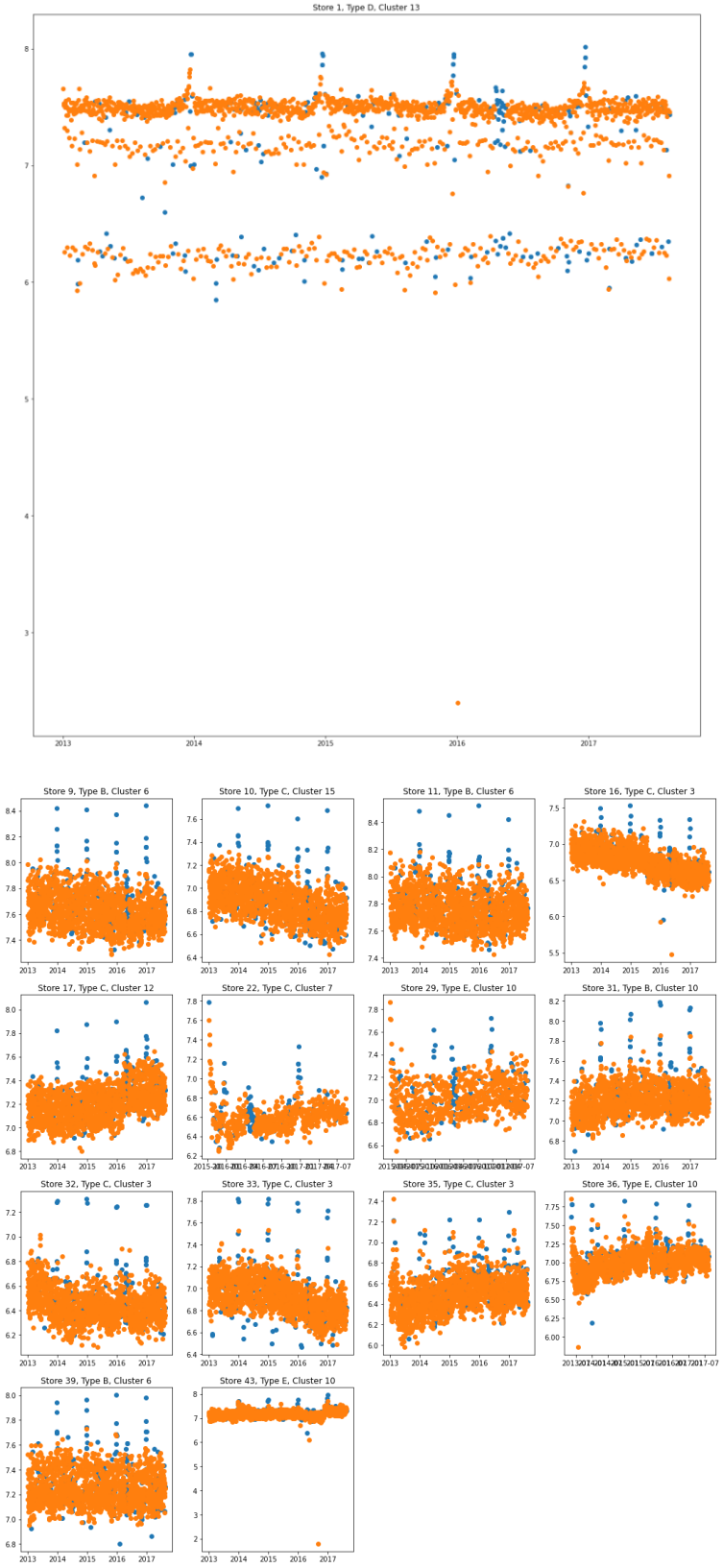


Figure 30: Transaction data: Cluster 3 and Cluster 4

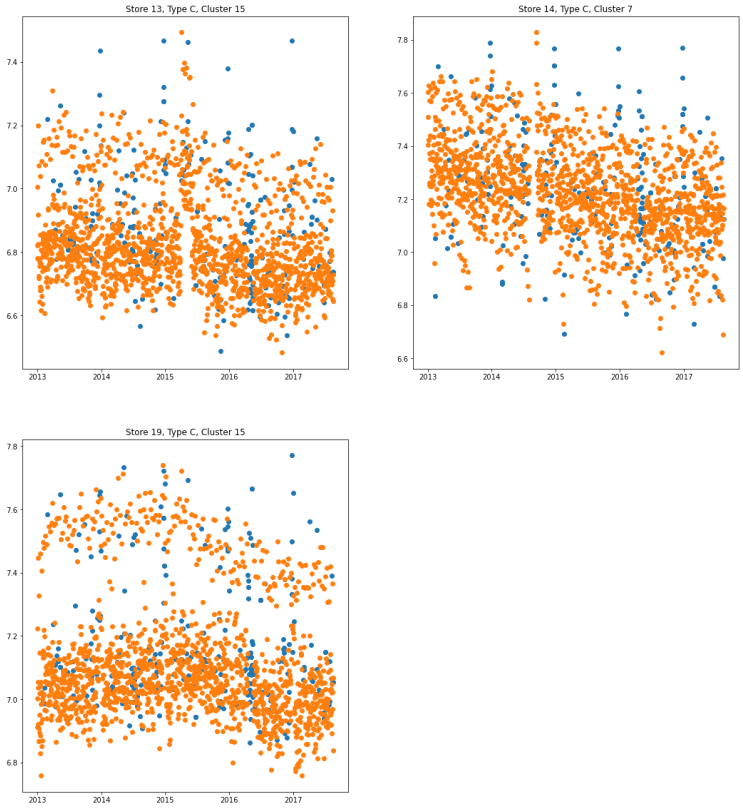


Figure 31: Transaction data: Cluster 5

ii. Results of Clustering using unit_sales

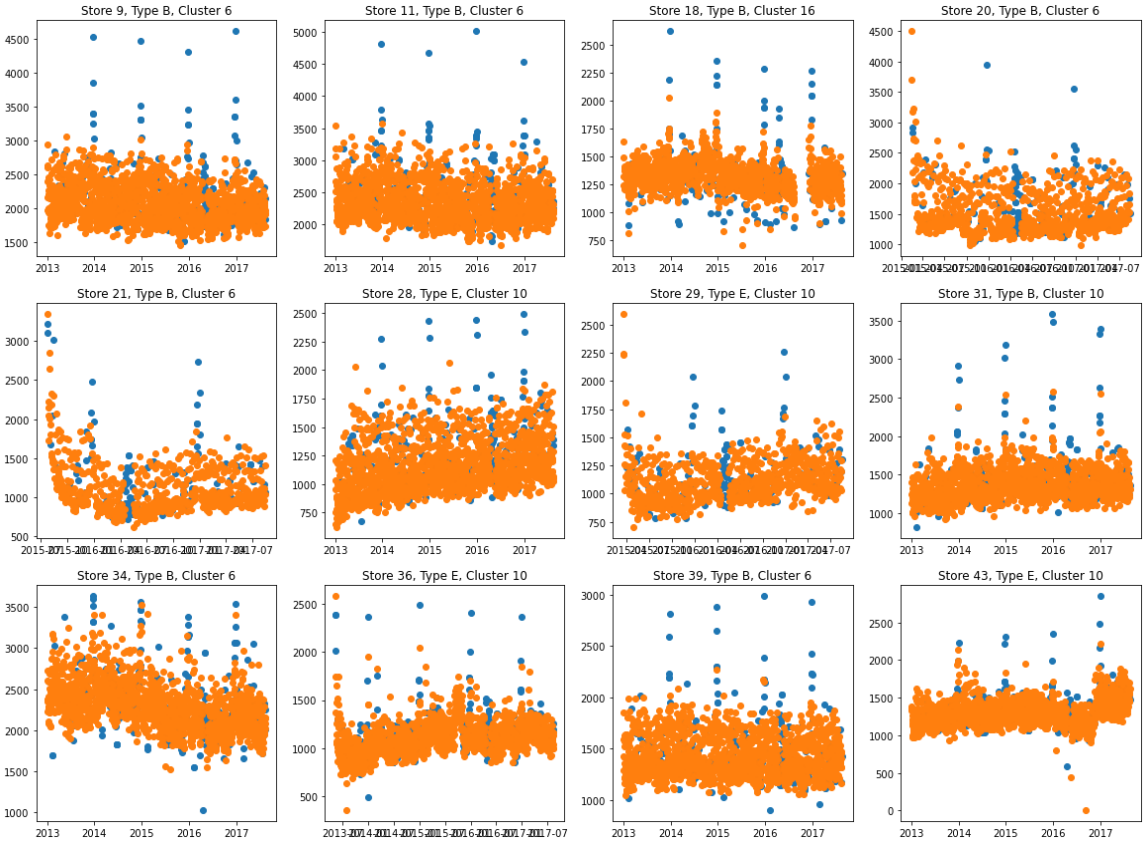


Figure 32: Unit_sales data: Cluster 0

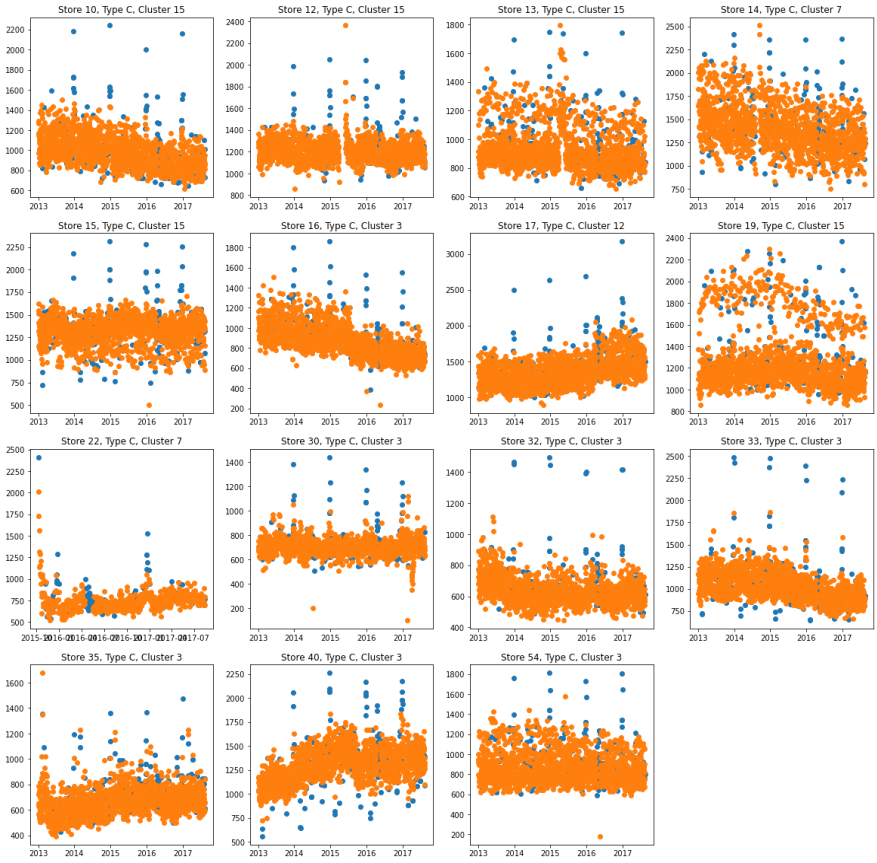


Figure 33: Unit_sales data: Cluster 1

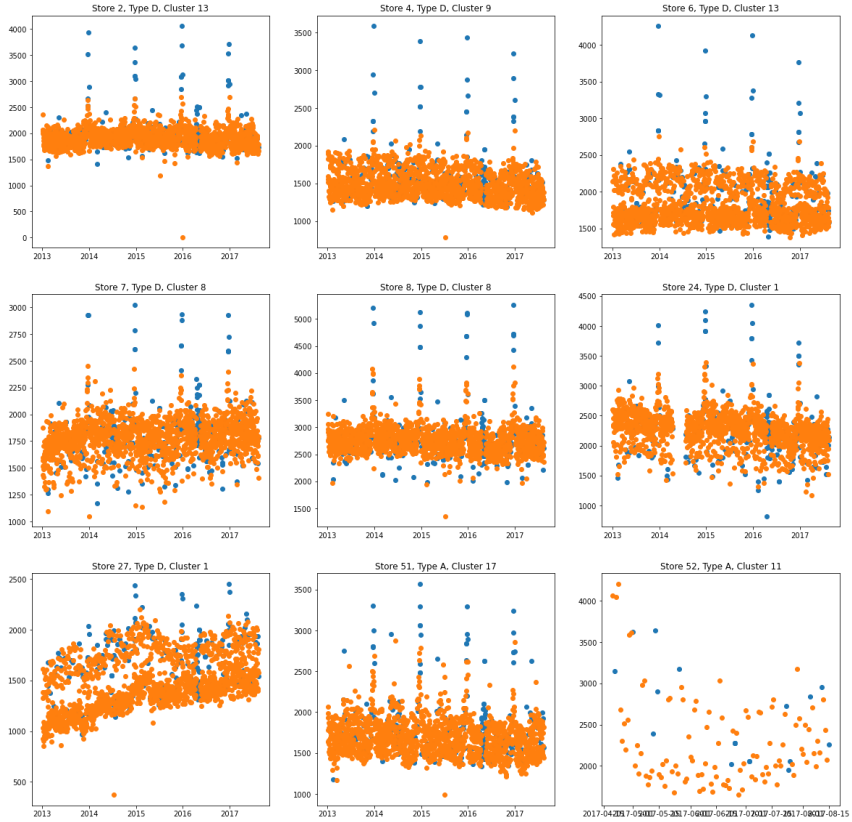


Figure 34: Unit_sales data: Cluster 2

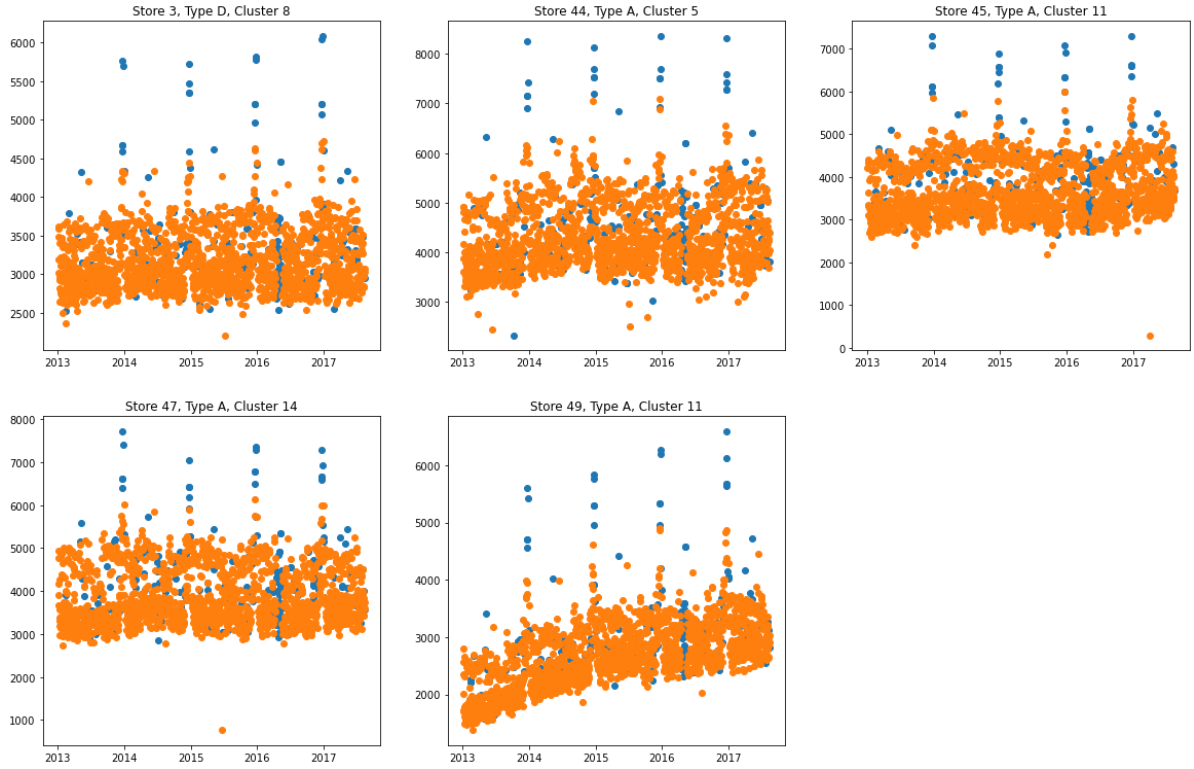


Figure 35: Unit_sales data: Cluster 3

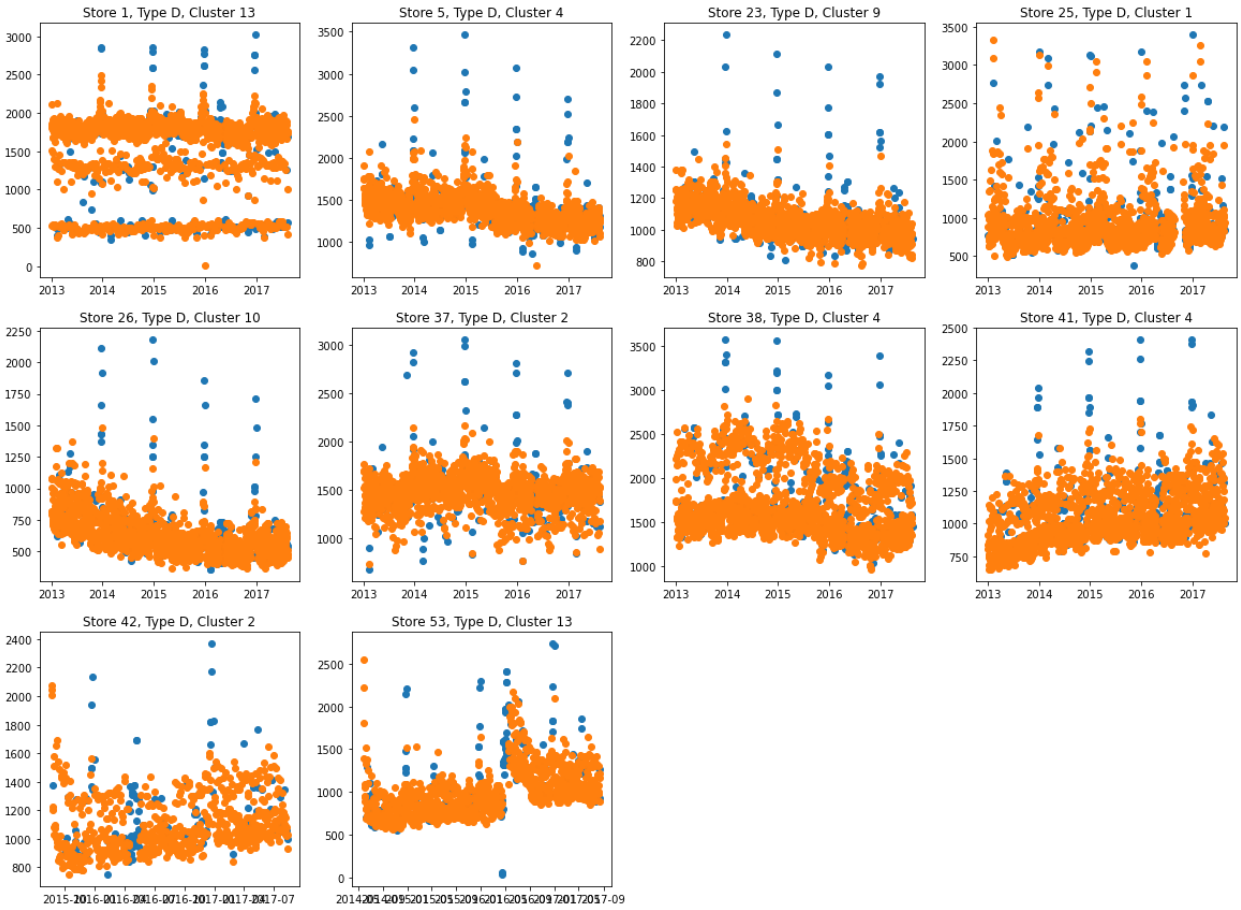


Figure 36: Unit_sales data: Cluster 4

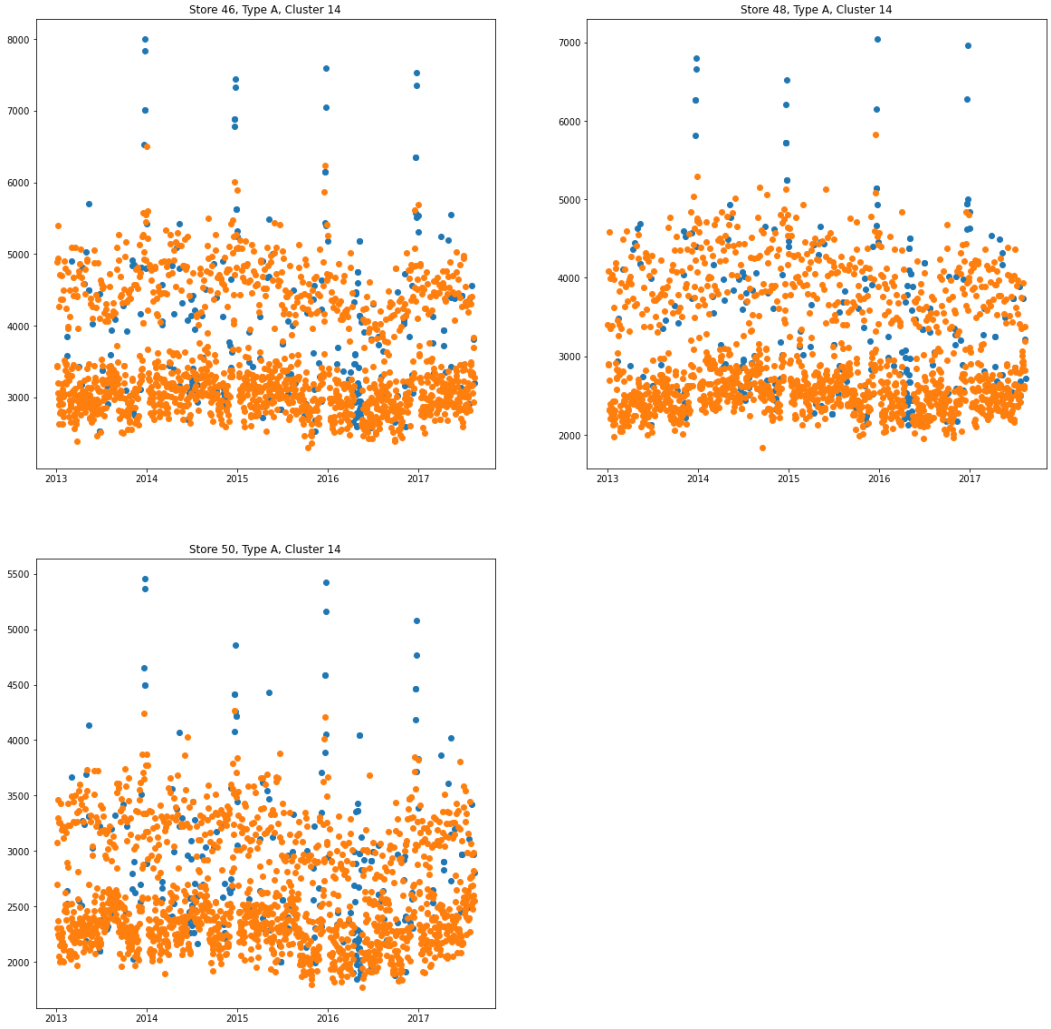
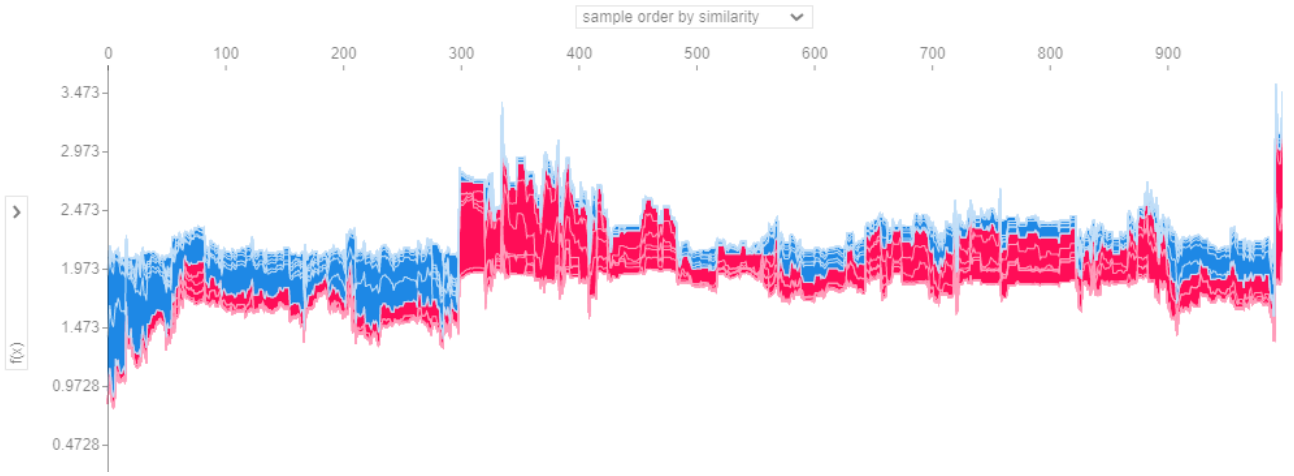


Figure 37: Unit_sales data: Cluster 5

iii. SHAP results for Cluster 3 and Cluster 5

```
shap.force_plot(explainer.expected_value,shap_values[1:1000,:],X_train.iloc[1:1000,:])
```



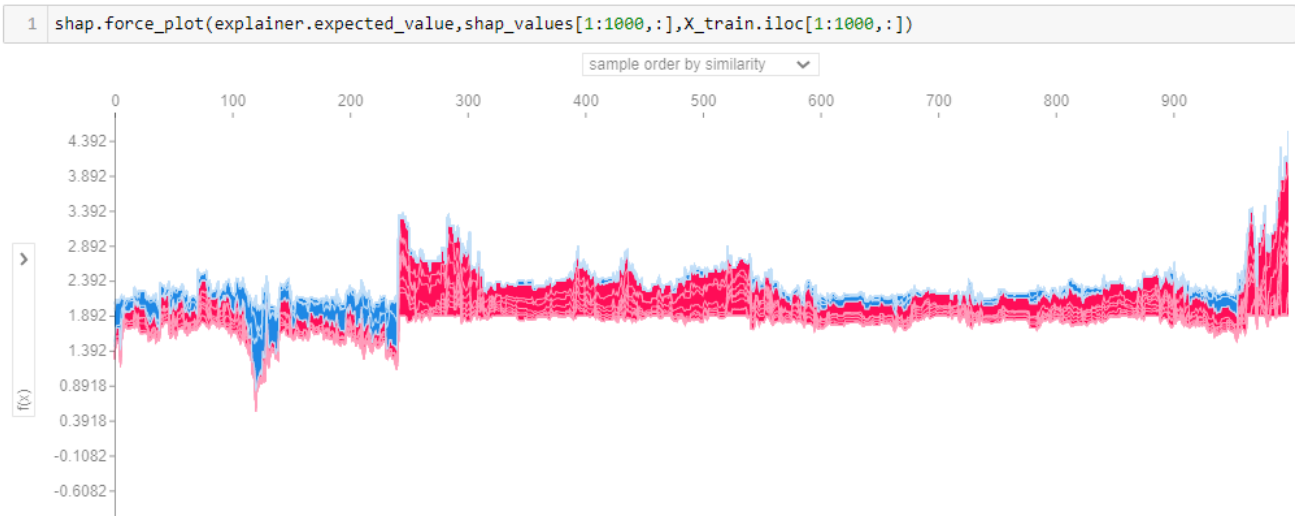


Figure 38: SHAP force plot for Cluster 3 and Cluster 5