Demand Forecasting and Inventory Optimization for a Food Processing Company

Karan Joseph, Rohit Deshmukh, Zied Abdelmoula, and Nikhil Bansal

Contents

1. Introduction

The goal of the project is to develop a demand forecasting tool to find the best levels of aggregation and the best forecasting technique that can be used to predict weekly demands for a leading Food Processing Company based in North America. The company has several warehouses across USA and Canada, which they use to source retailers and distribution centers across North America. As a complementary service, we have also developed an inventory optimization tool using Integer Linear Programming. Our aim is to improve their production plan for the coming horizons by giving them better forecasts and streamlining their inventory levels.

The company has given us a partial database, with information concerning the actual demand history (univariate time-series dataset) for all the products manufactured in the year 2019. The data at hand is composed of 2407 SKUs, and each of these items falls into one of 4 unique production division categories (CA-FS, CA-RTL, US-FS, and US-RTL). To decrease the variability in the dataset and computational complexity, the company suggested developing forecast models at aggregated levels and then later disaggregating them to get individual product demands for each SKU.

Division level category assignment is being made by the company based on business intuition and product sense. But since SKUs having different trends and seasonality can get aggregated together under the same division, we believe this will decrease the overall influence of the time series components and thereby reduce its forecasting power. From our research, we learned that time-series clustering can be an alternative solution to this problem because this enabled us to group together items having similar demand patterns. Therefore, we are performing demand forecasting on two different levels of aggregation – **1) Division level and 2) Cluster level** – for building our predictive models.

In order to analyze the forecast results and develop our prescriptive inventory solution, we have used only the top 5 items by quantity from each division (20 items in total). We based this idea on the popular Pareto principle, and we have assumed that this will give us a near accurate representation of the overall data. The forecast models were then developed using 80% of data as the train set and the remaining 20% as the test set. The overall performance of models was compared using the test-set and we have used only this half of the data for the inventory model. The inventory optimization model was developed using an Integer Linear Programming algorithm for the last 8 weeks of 2019 (test set).

2. Methodology

To develop the forecasting tool for the project, we have used 4 time-series-based forecasting techniques; Holt-Winter's, SARIMA, Prophets, and Random Forrest at 2 levels of aggregation – Division and cluster. For comparing the results, we are mainly using Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE). But since different items have different scales of demand, we have used MAPE as our primary metric for comparison.To accommodate the use of RMSE, we would have to scale the quantities for each

SKU and transform it back for the inventory model. This factor of complexity was also a reason why we had decided to avoid relying on only the RMSE values. Other metrics such as Cov can be used here too, but we are using MAPE because it will be easier to explain from a business perspective.

Before developing the forecast models, data pre-processing and time-series clustering was done to group the SKUs into the 2 levels of aggregation which we mentioned earlier. The aggregated forecasts were then broken down into their respective SKUs using a fixed scaling factor, and the best results for each SKU (based on MAPE) were taken as input for the Inventory Optimization model.

Time-series Clustering

The key idea of clustering is to segregate similar data points together to perform further analysis. We are extending the same principle here to time-series data, for grouping SKUs with similar demand patterns into different groups. Since time-series data have a time component, Euclidean distance (ED) might not always be a good measure for calculating similarity because it will only calculate the distance between data points at their respective time phase. Therefore, we are using Dynamic time warping (DTW) as the alternate distance metric. DTW can calculate the similarity between differently aligned time-series patterns because it calculates the distance along the time dimension using a non-linear warping path.

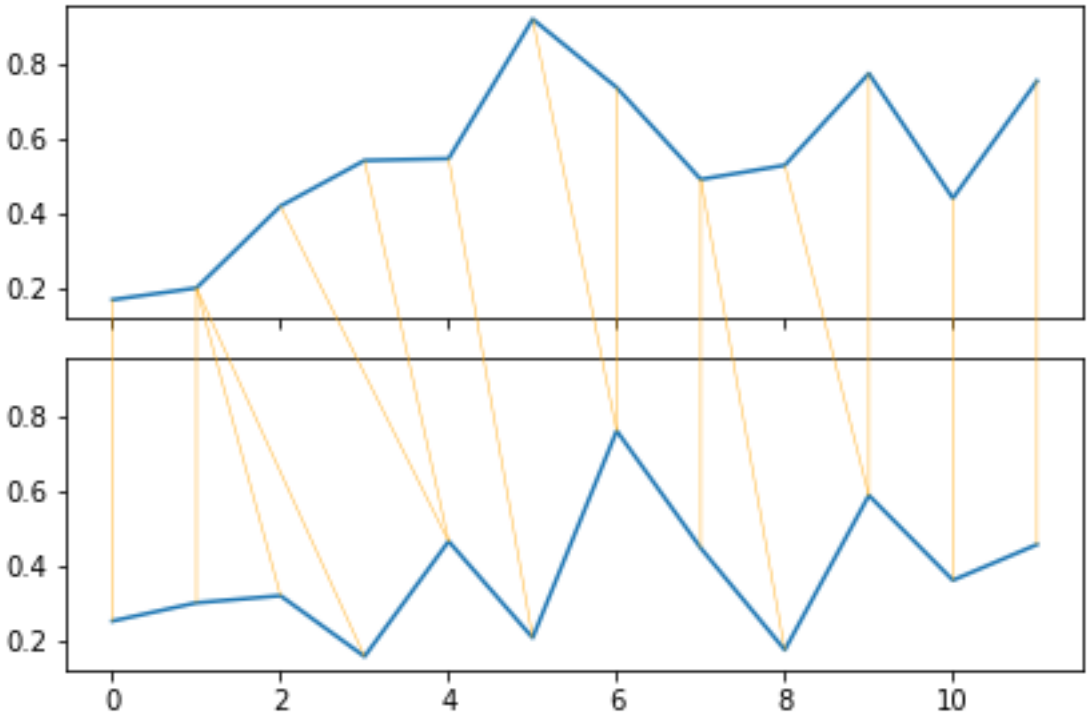


Figure 1: Dynamic time warping: non-linear warping path

Based on the DTW distance metric, we performed time-series clustering using different techniques: agglomerative clustering using single linkage, agglomerative clustering using ward linkage, and K-Means clustering. Silhouette scores and cluster plots were then drawn to evaluate these clusters.

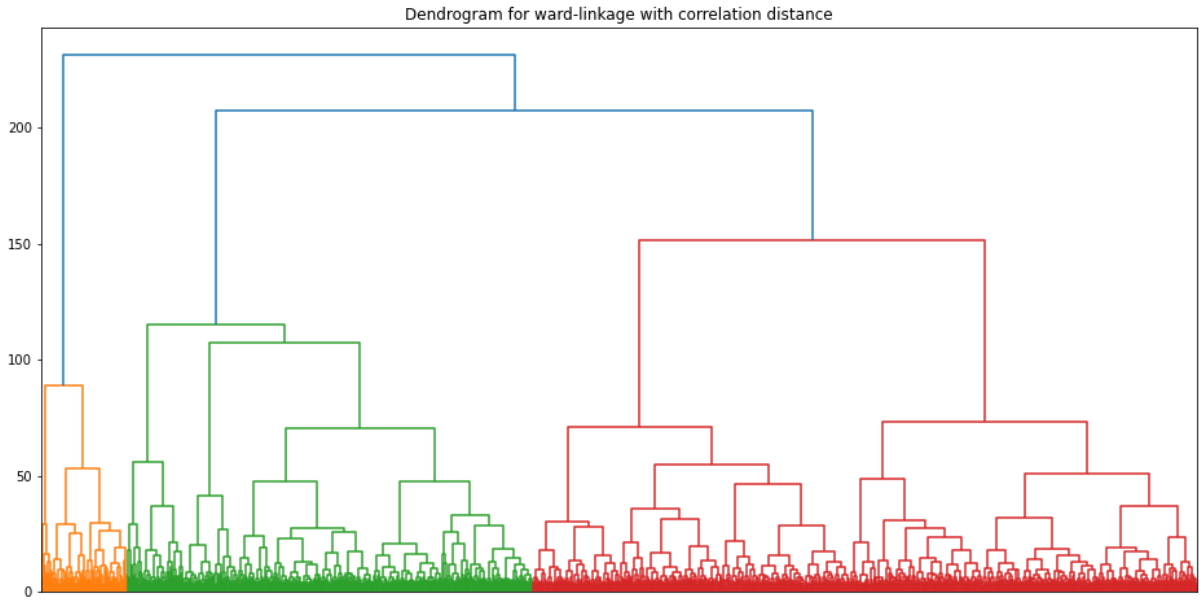


Figure 2: Dendrogram output from Agglomerative Clustering – ward linkage

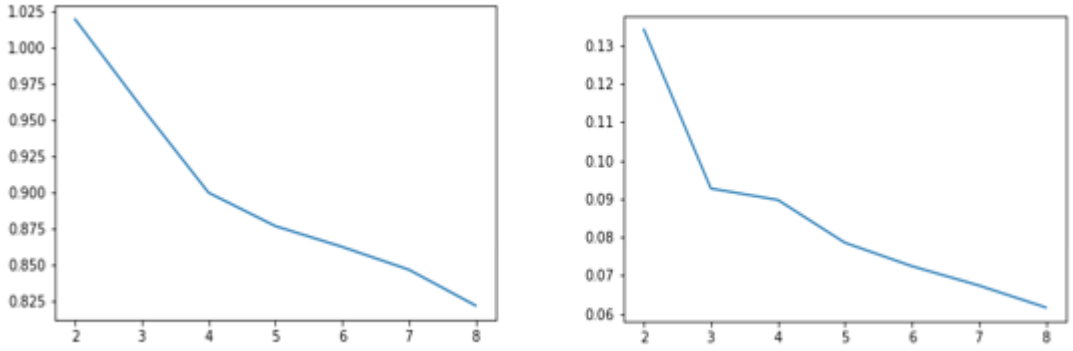


Figure 3: a)Elbow Curve- Inertia, b) Silhouette Score for K-Means Clustering

From the elbow curves in Figure 3, we chose 4 as the number of clusters for K-Means. The number of clusters for the other clustering techniques was chosen according to their respective Dendrograms. Upon comparison, K-means gave us the best results and we decided to use these Cluster IDs as the second level of aggregation for use in the forecast models.

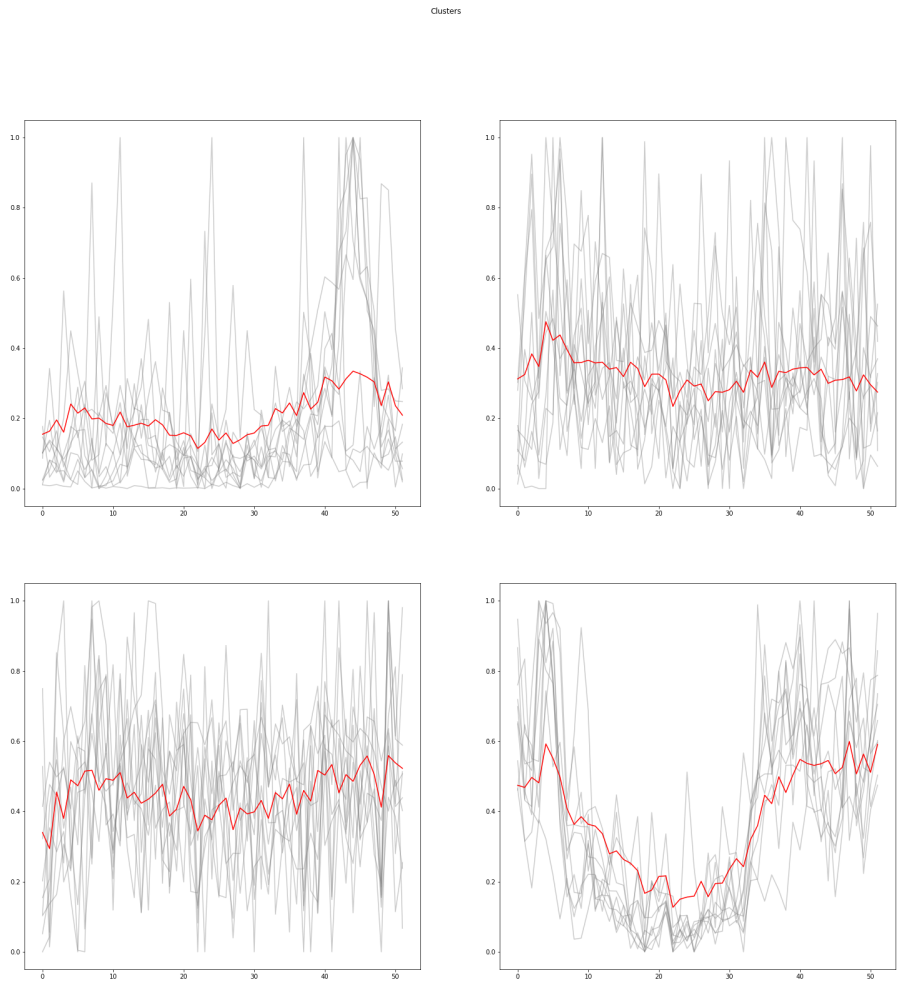


Figure 4: Output of Time-Series Clustering (K-Means)

Demand Forecasting

Before developing the demand forecast models, our first step was to study the different time-series decomposition plots. The decomposition plots helped us understand how the data is represented by its time-series components (seasonality and trend). Since we had limited availability of data, the seasonal patterns were difficult to interpret. However, we have decided to add a seasonal period of 13weeks(1 quarter) in our time-series models. This was chosen according to the results obtained from GridSearch - when optimizing the parameters for SARIMA, and also based on business logic. This seasonal period may be prominent only because we have just one year of data, and the yearly seasonal components might just override this effect if we consider more data.

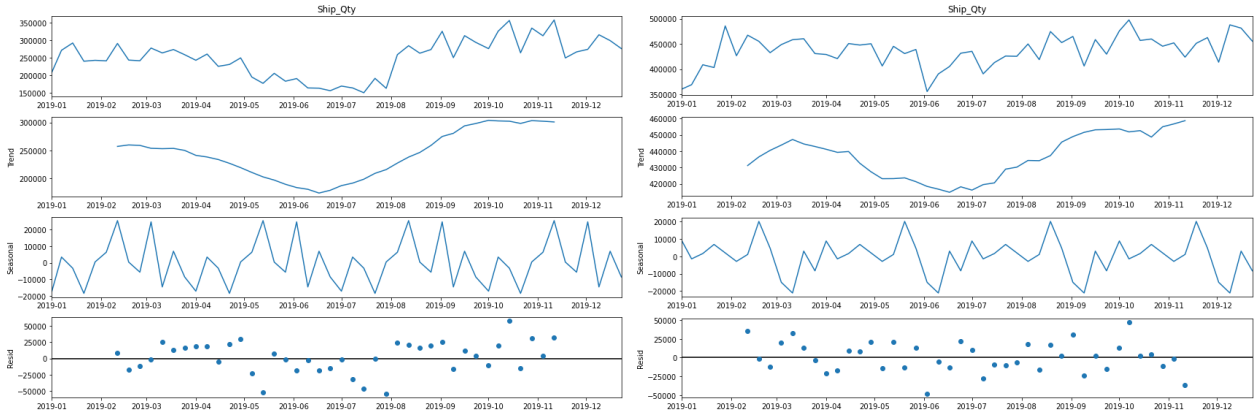


Figure 5: Time-Series Decomposition Plots: a) CA-FS, b) Cluster 3

We performed the same decomposition analysis for all divisions and cluster levels and learned that there is a clear seasonality and trend in weekly aggregated demands as evident from plots in Fig 5. In order to capture both these characteristics of time series data, we forecasted the demand using time-series models that perform well under similar conditions: **1. Holt Winter 2. SARIMA, and 3. FB-Prophets**. We then also explored the capability of **4. Random Forest** in time-series-based forecast predictions. Our research suggested that they could give reliable results in a time-series setting.

a) Holt Winter

Holt Winter's model was used to capture the suspected seasonality and trend in the demand patterns. We analyzed the results for both additive and multiplicative holt winter models and found that the forecast results were better for additive models. Hence we used it as our base model for demand forecasting.

b) SARIMA

Seasonal Autoregressive Integrated Moving Average (SARIMA) is a very efficient univariate time-series forecasting method. In order to use a SARIMA model, we must determine the values of its parameters (p, d, q) that capture the trend component of the time series, and parameters (P, D, Q, m) that determine the seasonal components. To find the trend parameters p and q, we used PACF and ACF plots from Figure 6. This was done for all levels of aggregation, and we have found that the values of p and q were the same for all.

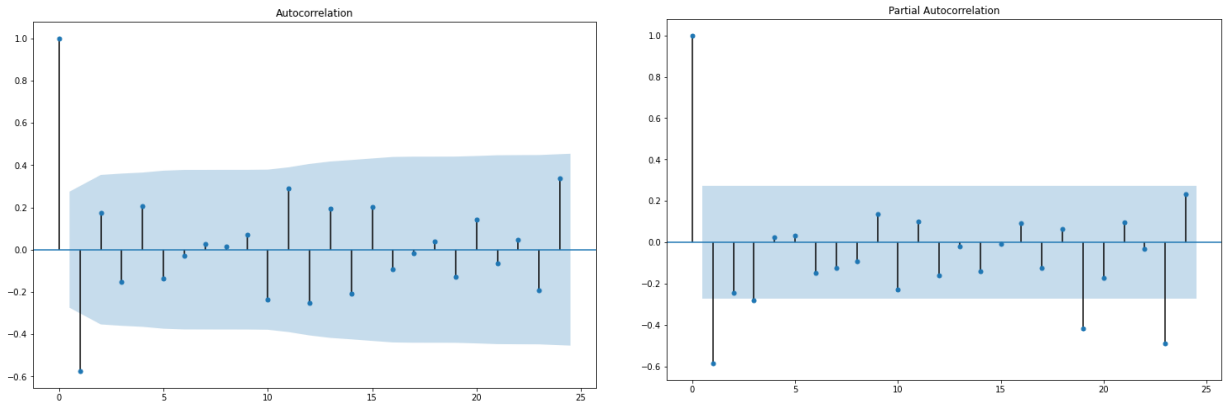


Figure 6: a) ACF Plot, b) PACF Plot

The value for parameter d - level of trend differencing required to make the data stationary was determined from the differencing plot in Fig 7. The stationarity of the time-series datasets was then confirmed using the Augmented Dickey fuller test where we got a significant p-value for the first difference. From these verifications, we concluded d=1. This was also confirmed to be the same for all levels of aggregation.

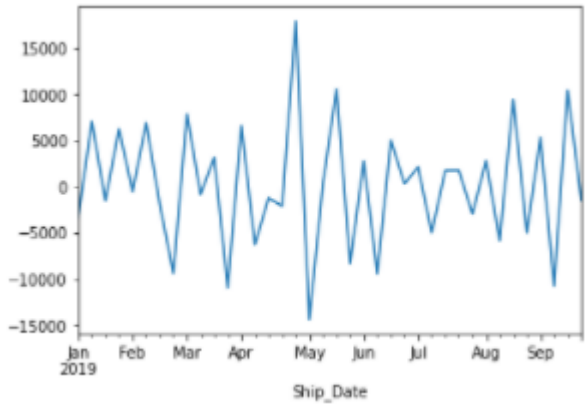


Figure 7: Stationarity Check - 1st order Differencing

To find the optimal values for seasonality parameters (P, D, Q, m) that gave the best forecast results, we ran a grid search algorithm over a pre-determined range for all levels of aggregation. The hyperparameters that gave the lowest AIC value were found for each of the divisions and clusters. All seven optimal hyperparameters were then given as input to the model function and the demand was forecasted.

c) Prophet

Facebook's open-source time-series forecasting model is another technique that deals well with univariate and multivariate time-series datasets having seasonal and trend components.

It is a Generalized additive model (GAM), that is very robust in handling shifts in trends and we also have the flexibility to specify seasonal periods and change points. Since we have noticed that there is a weekly seasonality with a periodicity of 13, we have also added that to the FB-prophet models.

d) Random Forest

Random Forest is a very effective ensemble technique used in supervised machine learning problems. However, like any other supervised learning algorithm, it requires a set of input features to make predictions on the target variable. But since ours is a univariate time-series dataset, it does not have any other input variables. Therefore, the data had to be first transformed into a supervised learning problem. This was done by restructuring the dataset to take quantities from previous weeks as input variables to predict the demands for the next week.

t-8	t-7	t-6	t-5	t-4	t-3	t-2	t-1	Quantity
								10231
							10231	9532
						10231	9532	12750
					10231	9532	12750	12876
				10231	9532	12750	12876	15400
			10231	9532	12750	12876	15400	10872
		10231	9532	12750	12876	15400	10872	9568
	10231	9532	12750	12876	15400	10872	9568	11488
10231	9532	12750	12876	15400	10872	9568	11488	15527
9532	12750	12876	15400	10872	9568	11488	15527	11009
12750	12876	15400	10872	9568	11488	15527	11009	12105
12876	15400	10872	9568	11488	15527	11009	12105	18795
15400	10872	9568	11488	15527	11009	12105	18795	14136
10872	9568	11488	15527	11009	12105	18795	14136	17392
9568	11488	15527	11009	12105	18795	14136	17392	13246
11488	15527	11009	12105	18795	14136	17392	13246	11839
15527	11009	12105	18795	14136	17392	13246	11839	16625

Figure 8: Time-Series transformation: Sliding window representation

This method of time-series forecasting required a technique called walk-forward validation for evaluating the results on the test set; because one cannot use data that you do not have at the time of prediction when forecasting for the next period.

Forecast Disaggregation

Once the forecasts were calculated at both division and cluster levels, we disaggregated the data to its respective SKU level. This was done using the fixed proportions we calculated based on the percentage contributions of each SKU to the total quantity of the aggregation level. We have considered only one method of disaggregation in this project, but we have identified a scope to experiment with different disaggregation techniques in future extensions of the project.

Inventory Management

Reasonable assumptions were made for inventory holding costs and ordering costs for the 20 SKUs selected to make the inventory optimization models. Integer Linear Programming models were implemented using Pyomo and glpk libraries to find the optimal order quantities and ordering periods for these SKUs based on the quantity and demand constraints. Since we did not consider capacity constraints, a Big M constraint was placed on the order quantities to achieve optimality. The objective function and the constraints were formulated to minimize the total ordering and holding costs for these products.

$$Min : Co * \sum_{i=1}^{8} Y_i + Ch * \sum_{i=1}^{8} S_i$$

Constraints

$$S_0 = 0$$

$$Q_i + S_{i-1} - S_i = D_i$$

$$Q_i <= M * Y_i$$

$$Y_i \in [0, 1]$$

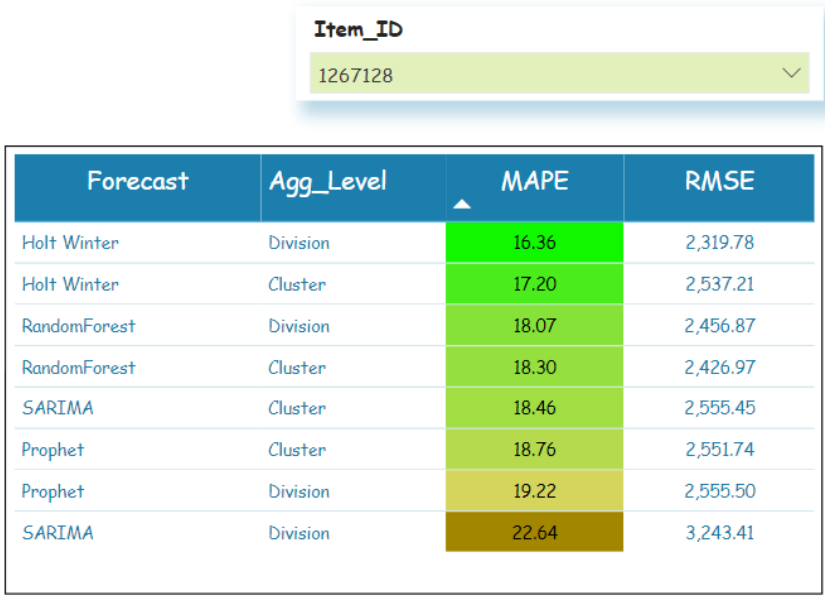
$$i \in [1, 8]$$

Currently, we do not have any knowledge regarding the inventory management solution implemented by the company. Hence, we considered a Lot for Lot model as a base inventory management model to calculate the cost savings of using the Integer linear programming method. Since we did not have accurate information about the costs, the prescriptive solution is currently a prototype. However, this is an excellent reference for when the company wishes to implement the solution using the current production costs.

3. Results

Predictive

The best-performing forecasting technique and the corresponding level of aggregation for each SKU were chosen by comparing the results of the forecast models. This analysis was done based on the different MAPE values; the aggregation level and forecasting technique that gave the lowest value for MAPE was selected as the best forecasting model for that SKU. The table below (Figure 9) shows the performance of all forecasting techniques for the selected Item ID.



Forecast	Agg_Level	MAPE	RMSE
Holt Winter	Division	16.36	2,319.78
Holt Winter	Cluster	17.20	2,537.21
RandomForest	Division	18.07	2,456.87
RandomForest	Cluster	18.30	2,426.97
SARIMA	Cluster	18.46	2,555.45
Prophet	Cluster	18.76	2,551.74
Prophet	Division	19.22	2,555.50
SARIMA	Division	22.64	3,243.41

Figure 9: SKU level forecast results

From the table in Figure 10, we can propose that for Item ID-1267128, the best-performing forecast model is Holt Winter when forecasted at division level aggregation. A similar analysis was carried out for all the SKUs considered in the scope of study and the following results table consolidates the best techniques identified for each SKU.

Item_ID	Agg_Level	Forecast	RMSE	MAPE
1267319	Cluster	RandomForest	4996.57	10.06
1267908	Cluster	RandomForest	1052.29	7.74
1267133	Cluster	RandomForest	1414.12	13.05
1267128	Division	Holt Winter	2319.78	16.36
1267375	Division	Holt Winter	3168.2	23.66
1268762	Division	Prophet	2639.96	60.9
1269810	Division	Prophet	5344.37	35.92
1269903	Division	Holt Winter	1025.34	54.15
1269987	Division	SARIMA	326.31	20.96
1270709	Division	Prophet	451.86	24.53
1270349	Cluster	Holt Winter	760.6	9.59
1269237	Cluster	SARIMA	348.32	18.42
1269461	Cluster	SARIMA	4624.26	36.55
1269685	Cluster	SARIMA	4553.08	38.63
1267261	Cluster	Prophet	9201.92	11.18
1267504	Cluster	Prophet	7343.81	16.22
1267746	Cluster	Prophet	19779.71	16.5
1269265	Cluster	Prophet	205.92	11.77
1269603	Cluster	Prophet	3317.7	7.58
1270707	Cluster	Prophet	5870.98	38.5

Figure 10: Best performing model for each SKUs

For the scope of our study, we have only chosen 20 items; top 5 items based on quantity from each of the 4 divisions (US FS, US RTL, CA FS, and CA RTL). We found out that all the forecasting methods used in the project were giving decent results, and the cluster level aggregation resulted in better forecasts for 65% of items, as represented by the donut chart below. But this may change if you retrain the models with more data.

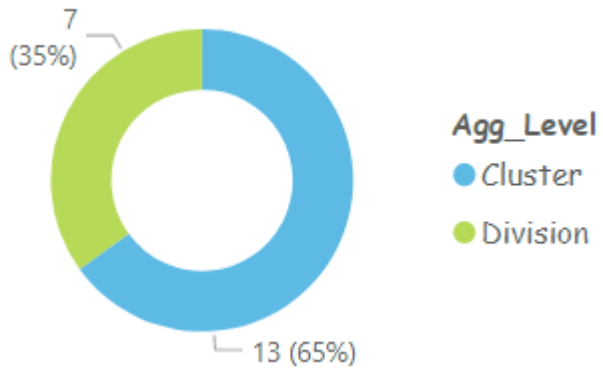


Figure 11: Number of items as per aggregation levels

We then examined whether the predicted demands from the best-performing models can capture the time series elements for our targeted SKUs. The demand forecasting plots at

the SKU level, similar to those below, were plotted to study results from the predictive models.

Figure 12: SKU level Demand Forecasting plot for best performing model of two items

As we can see from the line plots in Figure 12, the predictions for the test data are closely representing the actual demand for that period and are reasonably accounting for both the trend and seasonality in the demand patterns.

Prescriptive

Inventory optimization models were developed using the forecasted demands from predictive models as input for each of the 20 SKUs. Since the results are based on our inventory and holding cost assumptions, and we do not have information on the company's current holding policies, we have refrained from making recommendations based on cost benefits from the prescriptive tool. However, we have made the tool in such a way that these cost values can be easily adjusted for each SKU when the company decides to test its performance. Also, we have added the costs calculated from a Lot for Lot model as a reference to compare the results of our inventory solution.

Based on the current inputs, a sample output from the inventory model is shown in Figure 13.

Item_ID	Week	ForecastedDemand	OrderQty	Inventory	Order(yes/no)
1267128	44	13745	56172	42428	1
1267128	45	13295	0	29132	0
1267128	46	15255	0	13877	0
1267128	48	13877	0	0	0
1267128	49	12822	53957	41135	1
1267128	50	12622	0	28513	0
1267128	51	13922	0	14591	0
1267128	52	14591	0	0	0

Figure 13: Inventory Management results table

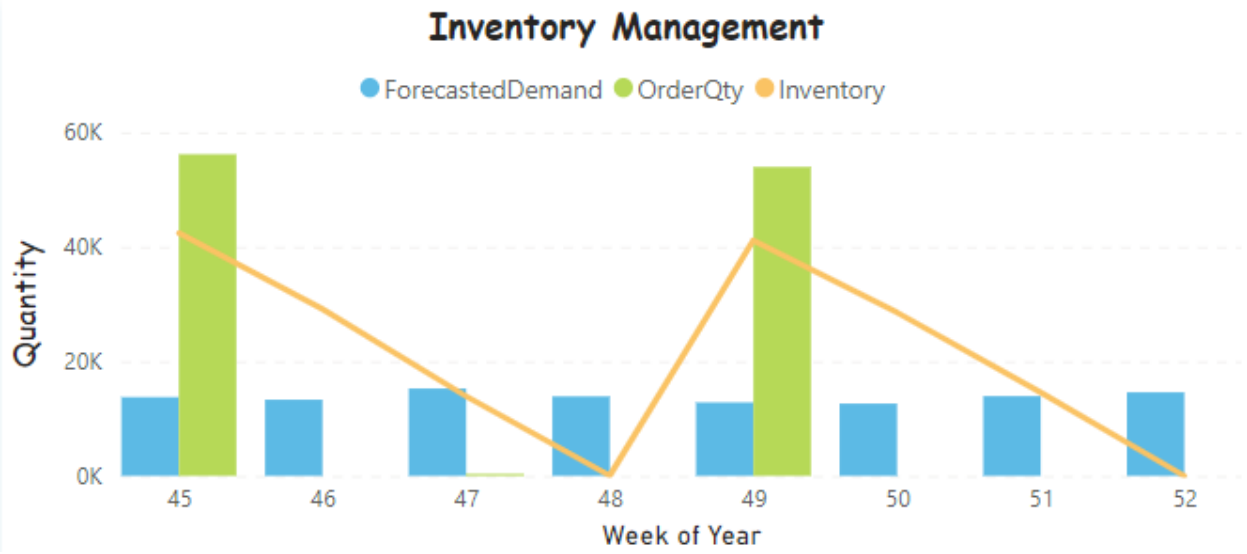


Figure 14: Inventory Management Plot

To quickly analyze the forecast predictions and inventory management solution, we have also provided a Power BI dashboard that can be filtered by the SKU of choice. The Dashboard can provide the demand planner with a lot of elements that can ease their planning process; such as: a demand forecasting plot of the best-performing model by item, a performance matrix comprising the different aggregation levels and forecasting models, an Inventory Management plot for the proposed MILP solution, and a KPI index to provide an estimate of cost reduction if the suggested inventory management solution(MILP) is implemented instead of a Lot for Lot model.

4. Conclusion

From our analysis, we observed that there is no one-size-fits model that works for any SKU. There is also potential for the different forecasting models to be combined using an optimal weightage factor for increasing the accuracy of prediction even further, but this is beyond the scope of our current project. However, the results showed that the cluster level aggregation gives better forecast accuracies. But we cannot give a conclusive suggestion regarding the best technique and the aggregation levels because our models were limited to just one year (52 weeks) of data. To give a more accurate recommendation, we would need to retrain the models with historical data from more years. This would enable our predictive models to better represent the seasonal and trend patterns in the data.

The predictive and prescriptive models developed in this project, and the Power-BI dashboard are dynamic and can be easily adapted to include several years of historical data and more complex constraints. This demand forecasting and inventory optimization tool we developed will be a great reference for the company if they wish to implement these solutions on a larger scale.

5. References

• Coffey, C. (2013, March 28). Of Data and Science: Capital Bikeshare: Time Series Clustering. Of Data and Science. http://ofdataandscience.blogspot.com/2013/03/capital-bikeshare-time-series-clustering.html

• Jiang, G., Wang, W., & Zhang, W. (2019). A novel distance measure for time series: Maximum shifting correlation distance. Pattern Recognition Letters, 117, 58–65. https://doi.org/10.1016/j.patrec.2018.11.013

• Jupyter Notebook Viewer. (n.d.). Retrieved April 1, 2022, from https://nbviewer.org/github/jckantor/ND-Pyomo-Cookbook/blob/master/notebooks/02.01-Production-Models-with-Linear-Constraints.ipynb

• Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., & Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '12, 262. https://doi.org/10.1145/2339530.2339576

• Random Forest for Time Series Forecasting for Data Science. (2021, June 2). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/random-forest-for-time-series-forecasting/

6. Appendix

Time Series Clustering

APPENDIX 1: Kmeans Clustering

```
#Silhouette score
ssd1=[]
ssd2=[]
for i in range(2,9):
    kmeans=TimeSeriesKMeans(n_clusters=i,\
                            metric="dtw",\
                            max_iter=5,\
                            max_iter_barycenter=5,\
                            #metric_params={"gamma": .5},\
                            random_state=0)
    kmeans.fit(ts_pivot)
    cluster_labels = kmeans.labels_
    silhouette_avg = silhouette_score(ts_pivot, cluster_labels, metric="dtw")
    print("For n_clusters={0}, the silhouette score is {1}".format(i, silhouette_avg))
    ssd1.append([i,silhouette_avg])
    ssd2.append((i,kmeans.inertia_))

plt.plot(pd.DataFrame(ssd1)[0], pd.DataFrame(ssd1)[1])
plt.savefig("Images/kmeansSilhouetteScore.png")
plt.clf()

# Elbow Curve
plt.plot(pd.DataFrame(ssd2)[0], pd.DataFrame(ssd2)[1])
plt.savefig("Images/kmeansElbowCurve.png")
plt.clf()

km_sdtw = TimeSeriesKMeans(n_clusters=4,\
                           metric="dtw",\
                           max_iter=10,\
                           max_iter_barycenter=10,\
                           #metric_params={"gamma": .5},\
                           random_state=0).fit(ts_pivot)
cluster_kmeans = list(km_sdtw.predict(ts_pivot))
```

```python
def plot_clusters(df, cluster_labels, title, row_mx=2, column_mx=2):
    """
    Plot the time-series cluster groups based on the method
    Parameters:
        df : (dataframe) Input time-series dataframe used for clustering.
        cluster_labels : (list)Ouput of cluster model.
        title : (string) Name of clustering model.
        row_mx : (int, optional) Number of rows to show in subplot. The default is 2.
        column_mx : (int, optional) Number of columns in subplot. The default is 2.

    Returns:
        Outputs cluster plots as png and saves them in the Images folder
    """
    fig, axs = plt.subplots(row_mx, column_mx, figsize=(25,25))
    fig.suptitle('Clusters')
    loc = []
    row = 0
    column = 0
    for i in set(cluster_labels):
        loc = [x for x,y in enumerate(cluster_labels) if y==i]
        for j in range(10):
            try:
                axs[row, column].plot(df.iloc[loc[j],:].values, c="gray", alpha=0.4)
            except:
                axs[row, column].plot(df.iloc[loc,:].values, c="gray", alpha=0.4)
        if j>1:
            axs[row, column].plot(np.average(np.vstack(df.iloc[loc,:].values), axis=0),c="red")
        column+=1
        if column>=column_mx:
            row+=1
            column=0
    plt.savefig(f'Images/{title}.png')
```

Demand Forecasting

APPENDIX 2: Holt Winter Model

```
#Holt Winter Forecasting - Returns training and testing datasets with predicted values
def Holt_Winter_Forecasting(training_dataset,testing_dataset,operation):
  training_dataset.head(5)
  training_dataset=training_dataset.set_index("Ship_Date")
  testing_dataset=testing_dataset.set_index("Ship_Date")
  y_train=training_dataset["Ship_Qty"]
  fitted_model = ExponentialSmoothing(y_train,trend=operation,seasonal=operation,seasonal_periods=13).fit(optimized=True)
  testing_dataset["Prediction_Holt"] = list(fitted_model.forecast(len(testing_dataset)))

  training_dataset["Prediction_Holt"]=list(fitted_model.forecast(len(training_dataset)))
  return training_dataset.reset_index(),testing_dataset.reset_index()
```

APPENDIX 3:SARIMAX Model

```
def SARIMAX_Forecasting(training_dataset,testing_dataset,order,seasonal_order):
  model = sarimax.SARIMAX(training_dataset["Ship_Qty"], order=order, seasonal_order=seasonal_order)
  model_fit=model.fit()
  testing_dataset["Prediction_Sarima"]= model_fit.predict(start=44,end=52).reset_index().drop("index",1)
  return testing_dataset
```

APPENDIX 4:PROPHET Model

```
def Prophet_Forecasting(training_dataset,testing_dataset):
  df_train = training_dataset.rename({'Ship_Date': 'ds', 'Ship_Qty': 'y'}, axis=1)
  df_test=pd.DataFrame()
  df_test["ds"]=testing_dataset["Ship_Date"]
  model=Prophet()
  model.fit(df_train)
  testing_dataset["Prediction_Prophet"]=model.predict(df_test)["yhat"]
  return testing_dataset
```

APPENDIX 5:RANDOM FOREST Model

```
def random_forest_prep(df, level):
    """
    Parameters
    ----------
    df : DataFrame
        Receives the quantity data by each item and processes it for input to the random_forest model

    level : String
        Level of aggregation being called : Division or Cluster

    Returns
    -------
    out : DataFrame
        Output from random_forest model

    """

    df["Ship_Date"] = pd.to_datetime(df["Ship_Date"])
    train_len = datetime.datetime(2019,11,1)
    for i in range(5,0,-1):
        df["t-"+str(i)] = df["Ship_Qty"].shift(i)
    df.dropna(inplace=True)
    train = df[df["Ship_Date"]<train_len].copy()
    test = df[df["Ship_Date"]>=train_len].copy()
    X_test = test.iloc[:,3:]
    X_train = train.iloc[:,3:]
    y_train = train.iloc[:,2]
    y_pred = pd.Series(random_forest(X_train, y_train, X_test), name="Qty_Predicted")
    out = pd.concat((test[[level, "Ship_Date", "Ship_Qty"]]\
                     .reset_index().drop("index",axis=1), y_pred), axis=1)
    return out
```

```python
#RMSE AND MAPE
def calculate_rmse(actual,predicted):
  rmse = np.sqrt(mean_squared_error(actual, predicted)).round(2)
  return rmse
def calculate_mape(actual,predicted):
  mape= np.round(np.mean(np.abs(actual-predicted)/actual)*100,2)
  return mape


def random_forest(X_train, y_train, X_test):
    """
    Parameters
    ----------
    X_train : DataFrame
        Predictor variables for model input - train data - in sample
    y_train : DataFrame
        Target variable for model input - train data - in sample
    X_test : DataFrame
        Predictor variables for model input - test data - out of sample

    Returns
    -------
    y_pred : Series
        Out from the random forest model for input X_test - test set - out of sample prediction

    """
    rfr = RandomForestRegressor(n_estimators=1000, random_state=1)
    rfr.fit(X_train, y_train)

    index_start = X_test.index[0]
    y_pred = []
    for i in range(X_test.shape[0]):
        pred = rfr.predict(pd.DataFrame(X_test.iloc[i, :]).T)[0]
        y_pred.append(pred)
        for j in range(1,6):
            try:
                X_test.at[index_start+i+j ,"t-"+str(j)] = pred
            except:
                continue
    return y_pred
```

Inventory Management

APPENDIX 6: Inventory Optimization function

```
def optimizer(x, Co):
  """
    Parameters
    ----------
    x : numpy array
        Forecasted shipment quantities for 8 weeks
    Co : float
        Order cost multiplier
    Returns
    -------
    model : pyomo object
            Optimized inventory management model based on the obj and constraints of shortest path method (Integer Linear Programming Model)
    Co : float
          Lot4Lot inventory calculation
  """
  Co = x.mean() * Co
  model = ConcreteModel()
  #Variables
  model.y = Var(list(range(1, len(x)+1)), within=Binary) # Order yes or no
  model.s = Var(list(range(1, len(x)+1)), within=NonNegativeReals) # Inventory at period i
  model.q = Var(list(range(1, len(x)+1)), within=NonNegativeReals) # Order Quantity at period i
  #Objective
  obj1 = 0
  obj2 = 0
  for i in range(1, len(x)+1):
    obj1 += model.y[i]
    obj2 += model.s[i]
  model.OBJ = Objective(sense=minimize, expr = Co*obj1 + Ch*obj2)
  #Constraints
  model.order = ConstraintList()
  model.inv = ConstraintList()
  #Constraint 1: Constraint for order quantity
  for i in range(1, len(x)+1):
    const_expr = model.q[i] <= model.y[i]*maxx
    model.order.add(expr = const_expr)
  #Constraint 2: Constraint for quantity meeting expected demand
  model.inv.add(expr = model.s[1] - model.q[1] + x[0] == 0) #Initial Inventory is considered as 0
  for i in range(2, len(x)+1):
    const_expr = model.s[i] - model.s[i-1] - model.q[i] + x[i-1] == 0
    model.inv.add(expr = const_expr)
  #model.pprint()
  opt = SolverFactory('glpk')
  opt.solve(model)
  #model.display()
  return model, Co*len(x)
```

APPENDIX 7: Dashboard