Backorder Classifier Report

Prepared for

Alia Stanciu

Operations Manager

By

Kevin Collier

December 8, 2022

Abstract:

This data analysis report explains the implementation, analysis, and performance of three classification models to determine which products are most likely to experience backorder.

1. Introduction

*Summary:*

Alicia Stanciu has expressed concern about products that are experiencing stock out. I have been asked to develop a classification model to find a pattern inside the historical product data to predict the likelihood of backordered products. This report details the process of developing three classification models, their implementation, analysis, and performance. I conclude that the decision tree classification model is the most effective and provide the propensities for the fifty most likely backordered products in the conclusion.

*Data:*

Alicia Stanciau provided two datasets to create this model: an inventory training dataset and an inventory test dataset. The training dataset provided information on around 1.3 million different products and their inventory levels, sales, forecasts, risk flags, and whether or not they have been backordered. The test dataset provided information about around two hundred thousand items but did include whether or not the items have been on backorder. The training dataset will be used to develop the classification model. The model will then be implemented on the inventory test dataset to determine which products are most likely to experience stock out for the company to change their operations accordingly.

*Outcome:*

The resulting classification model predicts the likelihood of backordered products 92.38% of the time based on the training dataset provided. The three most likely products to go on backorder are the products with item numbers 3507096, 3306758, and 3462286. I have provided the list of the fifty most likely products to go on backorder in the conclusion of the report.

1. Body

To predict which items are most likely to experience stock out, I developed three Classification models: Decision Tree, Logistic Regression, and KNN. I will discuss the implementation and results of each model below.

*Data Cleaning:*

Before developing any classification models, I first performed data cleaning. The transit time from order to delivery (lead\_time) field had 82,389 missing entries. To fill these entries, I calculated the median transit time and filled the empty entries with this value. Similarly, the field indicating the proportion of orders placed that were not backordered for the previous six and nine months (perf\_6\_month\_avg and perf\_9\_month\_avg) had missing values. I replaced these values with the median of the respective fields. I replaced the empty values with the median to account for skewed distributions and outliers. Additionally, I changed the target value (went\_on\_backorder) from ‘Yes’ to 0 and “No’ to 1 in order to calculate propensities later on.

*Identifying Predictors:*

Based on the exploratory data analysis, I identified twelve potential predictors, nine of which are numerical and three of which are categorical risk flags. When looking at the distribution of the risk flags, I found three that were never used, so I eliminated them as potential predictors. When looking at the numerical fields, I eliminated the fields that had identical distributions as this would have no impact on the classification model's performance. The numerical predictors consist of the current inventory level in units, the forecasted sales for the next three, six, and nine months, the sales quantity for the previous month, three months, six months, and nine months, and the recommended minimum amount of stock. The risk flags that were used were deck\_risk, ppap\_risk, and stop\_auto\_buy.

*Classification Models:*

1. *Decision Tree*

When developing the decision tree classification model, I used 60% of the inventory dataset as training data and 40% as validation data. This allowed me to build the model on 60% of the data and test its performance on the remaining 40%. To develop the most accurate tree and understand the impact of different parameters, I performed a grid search. The parameters explored were max depth, minimum sample splits, and minimum sample leaves. The result of the grid search indicated a model with the best average score (determined by AUC for ROC) of 91.67%. After testing the model on the validation data, the model had an AUC of 92.38% indicating that this model predicts that a product is backordered better than a random prediction 92.38% of the time.

1. *Logistic Regression*

Before implementing the logistic regression classification model, I standardized the data. I used a normalizer scaler for the standardization because each row represents a different item with unique levels of inventory, sales, forecasts, and risk flag measures. Similar to the Decision Tree model, I used 60% of the inventory dataset as training data and 40% as validation data. After creating the model on the training dataset, I was able to identify the coefficients for the different predictors. The two most important coefficients to analyze are the current inventory levels and the sales of the previous month, as these had the most significant influence on predicting if an item went on backorder. After testing the model on the validation data, the model had an AUC of 91.06% indicating that this model predicts that a product is backordered better than a random prediction 91.06% of the time.

1. *KNN*

To minimize the time required to run a KNN model and avoid overfitting the model, I took a sample of 10% of the inventory dataset. This sample had a near identical frequency of items that were back-ordered in comparison to the entire dataset. Before developing the KNN classification model, I standardized the rows as I did for logistic regression for the subset of data. I then standardized by the features, or columns, to find the standard deviations for each entry. I used 60% of the subset dataset as training data and 40% as validation data to build the model. To develop the most accurate model and understand the impact of different numbers of neighbors, I performed a grid search. The result of the grid search indicated a model with the best average score (determined by AUC for ROC) of 76.08% using 19 neighbors. Before testing the model on the validation data, I repeated the same standardization process for the train data. After testing the model on the validation data, the model had an AUC of 75.93% indicating that this model predicts that a product is backordered better than a random prediction 75.93% of the time.

| Table 1: Summary of Classification Models | | |
| --- | --- | --- |
| Models | Data Used | Performance |
| Classifier 1: Decision Tree | All inventory data split into 60% training data | AUC: 92.38% |
| Classifier 2: Logistic Regression | All inventory data split into 60% of training data | AUC: 91.06% |
| Classifier 3: KNN | A subset of 10% split into 60% of training data | AUC: 75.93% |

1. Conclusion

Based on the performance of the three models, I recommend Alia Stanciu focus her attention on the outcomes of Classifier 1: Decision Tree. Compared to the other models developed, this classifier provides the most accurate predictions of items that will experience stock out. The decision tree classifier provides an area under the ROC curve of 92.38% indicating the model predicts that a product is backordered better than a random prediction 92.38% of the time.

After creating the model and determining the decision tree is the most effective, I tested the model on the test inventory dataset to identify the predicted propensity scores for potentially backordered products. Although I can not exactly predict if certain products will stock out, I was able to identify the top 50 items that are most likely to. The three most likely products to go on backorder are the products with item numbers 3507096, 3306758, and 3462286. The remaining products are shown below. I would adjust your operations accordingly based on this information and plan to have more available inventory for these items.

Top 50 most likely backordered products:

[3507096, 3306758, 3462286, 3358139, 3509659, 3298504, 3424442, 3470727, 3285571, 3440082, 3460600, 3457380, 3516692, 3314441, 3375959, 3365812, 3330838, 3498869, 3387524, 3287826, 3520375, 3389339, 3526106, 3365020, 3445802, 3485459, 3432729, 3348749, 3311679, 3492949, 3320174, 3385788, 3345934, 3383270,

3430603, 3394801, 3505750, 3506164, 3361762, 3315744, 3420586, 3318634, 3448724, 3297637, 3345487, 3333326, 3363800, 3363744, 3292363, 3484438]

1. Appendix

The following is the code for this report imported as an html file.

# **Backorder Classifier Code**

In [1]:

# import packages

import warnings

warnings.filterwarnings('ignore')

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot\_tree

from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score

from sklearn.metrics import confusion\_matrix, roc\_curve, roc\_auc\_score, ConfusionMatrixDisplay

from sklearn.metrics import accuracy\_score, roc\_curve, auc, confusion\_matrix,plot\_confusion\_matrix

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.preprocessing import StandardScaler, Normalizer

from dmba import classificationSummary, gainsChart, liftChart

# may have to put this after DMBA import command

%matplotlib inline

no display found. Using non-interactive Agg backend

# **Read in Data and look at basic statistics**

In [2]:

inventory\_df = pd.read\_csv("Data/inventory\_train.csv")

inventory\_df.head()

Out[2]:

|  | **sku** | **national\_inv** | **lead\_time** | **in\_transit\_qty** | **forecast\_3\_month** | **forecast\_6\_month** | **forecast\_9\_month** | **sales\_1\_month** | **sales\_3\_month** | **sales\_6\_month** | **...** | **pieces\_past\_due** | **perf\_6\_month\_avg** | **perf\_12\_month\_avg** | **local\_bo\_qty** | **deck\_risk** | **oe\_constraint** | **ppap\_risk** | **stop\_auto\_buy** | **rev\_stop** | **went\_on\_backorder** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1026827 | 0.0 | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | -99.00 | -99.00 | 0.0 | No | No | No | Yes | No | No |
| **1** | 1043384 | 2.0 | 9.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.99 | 0.99 | 0.0 | No | No | No | Yes | No | No |
| **2** | 1043696 | 2.0 | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | -99.00 | -99.00 | 0.0 | Yes | No | No | Yes | No | No |
| **3** | 1043852 | 7.0 | 8.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.10 | 0.13 | 0.0 | No | No | No | Yes | No | No |
| **4** | 1044048 | 8.0 | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | -99.00 | -99.00 | 0.0 | Yes | No | No | Yes | No | No |

5 rows × 23 columns

In [3]:

# gather basic information on dataset

inventory\_df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1360571 entries, 0 to 1360570

Data columns (total 23 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sku 1360571 non-null int64

1 national\_inv 1360571 non-null float64

2 lead\_time 1278182 non-null float64

3 in\_transit\_qty 1360571 non-null float64

4 forecast\_3\_month 1360571 non-null float64

5 forecast\_6\_month 1360571 non-null float64

6 forecast\_9\_month 1360571 non-null float64

7 sales\_1\_month 1360571 non-null float64

8 sales\_3\_month 1360571 non-null float64

9 sales\_6\_month 1360571 non-null float64

10 sales\_9\_month 1360571 non-null float64

11 min\_bank 1360571 non-null float64

12 potential\_issue 1360571 non-null object

13 pieces\_past\_due 1360571 non-null float64

14 perf\_6\_month\_avg 1360571 non-null float64

15 perf\_12\_month\_avg 1360571 non-null float64

16 local\_bo\_qty 1360571 non-null float64

17 deck\_risk 1360571 non-null object

18 oe\_constraint 1360571 non-null object

19 ppap\_risk 1360571 non-null object

20 stop\_auto\_buy 1360571 non-null object

21 rev\_stop 1360571 non-null object

22 went\_on\_backorder 1360571 non-null object

dtypes: float64(15), int64(1), object(7)

memory usage: 238.7+ MB

In [4]:

# look at basic statistics of dataset

inventory\_df.describe()

Out[4]:

|  | **sku** | **national\_inv** | **lead\_time** | **in\_transit\_qty** | **forecast\_3\_month** | **forecast\_6\_month** | **forecast\_9\_month** | **sales\_1\_month** | **sales\_3\_month** | **sales\_6\_month** | **sales\_9\_month** | **min\_bank** | **pieces\_past\_due** | **perf\_6\_month\_avg** | **perf\_12\_month\_avg** | **local\_bo\_qty** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 1.360571e+06 | 1.360571e+06 | 1.278182e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 | 1.360571e+06 |
| **mean** | 1.852106e+06 | 4.837912e+02 | 7.858978e+00 | 4.374903e+01 | 1.780070e+02 | 3.454916e+02 | 5.067209e+02 | 5.517012e+01 | 1.730748e+02 | 3.390355e+02 | 5.240252e+02 | 5.245381e+01 | 2.535347e+00 | -6.939863e+00 | -6.512312e+00 | 5.250457e-01 |
| **std** | 5.013569e+05 | 2.806845e+04 | 7.056853e+00 | 1.371530e+03 | 4.933240e+03 | 9.770796e+03 | 1.441222e+04 | 1.959227e+03 | 5.086128e+03 | 9.529091e+03 | 1.485647e+04 | 1.232674e+03 | 2.628731e+02 | 2.666420e+01 | 2.596508e+01 | 3.136197e+01 |
| **min** | 1.026827e+06 | -2.541400e+04 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | -9.900000e+01 | -9.900000e+01 | 0.000000e+00 |
| **25%** | 1.452832e+06 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 6.300000e-01 | 6.600000e-01 | 0.000000e+00 |
| **50%** | 1.794584e+06 | 1.500000e+01 | 8.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 1.000000e+00 | 2.000000e+00 | 4.000000e+00 | 0.000000e+00 | 0.000000e+00 | 8.200000e-01 | 8.100000e-01 | 0.000000e+00 |
| **75%** | 2.134726e+06 | 8.000000e+01 | 9.000000e+00 | 0.000000e+00 | 4.000000e+00 | 1.200000e+01 | 2.000000e+01 | 4.000000e+00 | 1.500000e+01 | 3.100000e+01 | 4.700000e+01 | 3.000000e+00 | 0.000000e+00 | 9.600000e-01 | 9.500000e-01 | 0.000000e+00 |
| **max** | 3.284914e+06 | 1.233440e+07 | 5.200000e+01 | 4.894080e+05 | 1.218328e+06 | 2.461360e+06 | 3.777304e+06 | 7.417740e+05 | 1.094112e+06 | 2.146625e+06 | 3.201035e+06 | 3.133190e+05 | 1.464960e+05 | 1.000000e+00 | 1.000000e+00 | 1.253000e+04 |

# **Data Cleaning**

In [5]:

# replace NA values with median

median\_leadtime = inventory\_df.lead\_time.median()

inventory\_df.lead\_time = inventory\_df.lead\_time.fillna( value = median\_leadtime)

# replace -99 values with NA

inventory\_df.perf\_6\_month\_avg = inventory\_df.perf\_6\_month\_avg.replace(to\_replace = -99.00, value = np.nan)

inventory\_df.perf\_12\_month\_avg = inventory\_df.perf\_12\_month\_avg.replace(to\_replace = -99.00, value = np.nan)

# calculate the median averages

perf\_6\_median = inventory\_df.perf\_6\_month\_avg.median()

perf\_12\_median = inventory\_df.perf\_12\_month\_avg.median()

# fill NA values with median value

inventory\_df.perf\_6\_month\_avg = inventory\_df.perf\_6\_month\_avg.fillna(perf\_6\_median)

inventory\_df.perf\_12\_month\_avg = inventory\_df.perf\_6\_month\_avg.fillna(perf\_12\_median)

# Change target variable from 'Yes' and 'No' to 1 or 0

inventory\_df.went\_on\_backorder = inventory\_df.went\_on\_backorder.replace(to\_replace = "Yes", value = 0)

inventory\_df.went\_on\_backorder = inventory\_df.went\_on\_backorder.replace(to\_replace = "No", value = 1)

In [6]:

#Explore the frequency of the target variable

freq = inventory\_df.went\_on\_backorder.value\_counts()

freq/sum(freq)\*100

Out[6]:

1 99.214447

0 0.785553

Name: went\_on\_backorder, dtype: float64

## **This data is very imbalanced. When creating the models I must take this into account.**

## **Take a subset of the data to use for the KNN model later on.**

In [7]:

# Take a subset of 10% of the data for the KNN model

subset\_df = inventory\_df.sample(frac=0.1)

# Check that the frequency is similar to the entire data set

freq = subset\_df.went\_on\_backorder.value\_counts()

freq/sum(freq)\*100

Out[7]:

1 99.223855

0 0.776145

Name: went\_on\_backorder, dtype: float64

# **Perform EDA to determine predictors and understand the data**

In [8]:

inventory\_df.columns

Out[8]:

Index(['sku', 'national\_inv', 'lead\_time', 'in\_transit\_qty',

'forecast\_3\_month', 'forecast\_6\_month', 'forecast\_9\_month',

'sales\_1\_month', 'sales\_3\_month', 'sales\_6\_month', 'sales\_9\_month',

'min\_bank', 'potential\_issue', 'pieces\_past\_due', 'perf\_6\_month\_avg',

'perf\_12\_month\_avg', 'local\_bo\_qty', 'deck\_risk', 'oe\_constraint',

'ppap\_risk', 'stop\_auto\_buy', 'rev\_stop', 'went\_on\_backorder'],

dtype='object')

In [9]:

# Separate the numerical and categorical columns to make EDA easier

quant\_cols = ['national\_inv', 'lead\_time', 'in\_transit\_qty',

'forecast\_3\_month', 'forecast\_6\_month', 'forecast\_9\_month',

'sales\_1\_month', 'sales\_3\_month', 'sales\_6\_month', 'sales\_9\_month',

'min\_bank', 'pieces\_past\_due', 'perf\_6\_month\_avg',

'perf\_12\_month\_avg', 'local\_bo\_qty']

cat\_cols = ['potential\_issue','deck\_risk', 'oe\_constraint',

'ppap\_risk', 'stop\_auto\_buy', 'rev\_stop']

In [12]:

# Create a scatterplot

g = sns.relplot(x='forecast\_3\_month', y ='sales\_1\_month', data=inventory\_df, hue='went\_on\_backorder', alpha=0.5, height=8, aspect=10/8)

plt.ticklabel\_format(style='plain', axis='y')

plt.show(g)

In [13]:

# Look at distribution of categories in cat columns with small numbers of categories

for column in cat\_cols:

if inventory\_df[column].nunique() < 10:

sns.countplot(y=column, data=inventory\_df)

plt.show()

## **The fields potential\_issue, oe\_contraint, and rev\_stop are irrelevant. They will not be used as predictors.**

## **I will use deck\_risk, ppapp\_risk, stop\_auto\_buy as categorical predictors.**

# **To perform more EDA, lets standardize the data to look at the distributions of quantitative fields**

In [14]:

# Standardize the data by rows

scaler = Normalizer()

QtyFields = quant\_cols

scaler.fit(inventory\_df[QtyFields])

inventory\_df[QtyFields] = scaler.transform(inventory\_df[QtyFields])

In [15]:

sns.set\_style('darkgrid')

inventory\_df.hist(figsize=(14,14),xrot=45);

plt.tight\_layout()

In [16]:

# Make a heatmap to find correlations between variables

corr = inventory\_df.corr()

sns.heatmap(corr, xticklabels=corr.columns,

yticklabels=corr.columns, vmin=-1, vmax=1, center=0, cmap="RdBu")

Out[16]:

<AxesSubplot:>

## **Let's use national\_inv, forecast\_3\_month, forecast\_6\_month, forecast\_9\_month, sale\_1\_month, sales\_3\_month, sales\_6\_month, sales\_9\_month, min\_bank as quantitative predictors because there distributions show to have an impact on the data.**

# **Classifier 1: Decision Tree**

### **Decision Trees should be made without standardizing the data so lets read in the data again and clean it like we did previously.**

In [17]:

inventory\_df = pd.read\_csv("Data/inventory\_train.csv")

# replace NA values with median

median\_leadtime = inventory\_df.lead\_time.median()

inventory\_df.lead\_time = inventory\_df.lead\_time.fillna( value = median\_leadtime)

# replace -99 values with NA

inventory\_df.perf\_6\_month\_avg = inventory\_df.perf\_6\_month\_avg.replace(to\_replace = -99.00, value = np.nan)

inventory\_df.perf\_12\_month\_avg = inventory\_df.perf\_12\_month\_avg.replace(to\_replace = -99.00, value = np.nan)

# calculate the median averages

perf\_6\_median = inventory\_df.perf\_6\_month\_avg.median()

perf\_12\_median = inventory\_df.perf\_12\_month\_avg.median()

# fill NA values with median value

inventory\_df.perf\_6\_month\_avg = inventory\_df.perf\_6\_month\_avg.fillna(perf\_6\_median)

inventory\_df.perf\_12\_month\_avg = inventory\_df.perf\_6\_month\_avg.fillna(perf\_12\_median)

# Change target variable from 'Yes' and 'No' to 1 or 0

inventory\_df.went\_on\_backorder = inventory\_df.went\_on\_backorder.replace(to\_replace = "Yes", value = 0)

inventory\_df.went\_on\_backorder = inventory\_df.went\_on\_backorder.replace(to\_replace = "No", value = 1)

In [18]:

#Set the predictor and outcome fields

predictors = ['national\_inv', 'forecast\_3\_month', 'forecast\_6\_month',

'forecast\_9\_month', 'sales\_1\_month', 'sales\_3\_month', 'sales\_6\_month',

'sales\_9\_month', 'min\_bank', 'deck\_risk', 'ppap\_risk', 'stop\_auto\_buy']

outcome = 'went\_on\_backorder'

In [19]:

# Seperate predictor fields into numerical and categorical lists

quant\_predictors = ['national\_inv', 'forecast\_3\_month', 'forecast\_6\_month',

'forecast\_9\_month', 'sales\_1\_month', 'sales\_3\_month', 'sales\_6\_month',

'sales\_9\_month', 'min\_bank']

cat\_predictors = ['deck\_risk', 'ppap\_risk', 'stop\_auto\_buy']

In [20]:

# Get dummies for the categorical fields

X = pd.get\_dummies(inventory\_df[predictors],

columns = cat\_predictors,

drop\_first = True)

y = inventory\_df[outcome]

#split the dataset into training and validation sets

train\_X, valid\_X, train\_y, valid\_y = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

## **In order to find the best parameters, let's use a grid search to determine the most accurate DT parameters.**

In [21]:

dt\_param\_grid = {

'max\_depth': [4,5,6],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf' :[1, 2, 5]

}

dt\_gridSearch = GridSearchCV(DecisionTreeClassifier(), dt\_param\_grid, cv=5, n\_jobs=-1, scoring='roc\_auc')

dt\_gridSearch.fit(train\_X, train\_y)

print('Best Average Score: {:.2f}%'.format(dt\_gridSearch.best\_score\_\*100))

print('Best Parameters: ', dt\_gridSearch.best\_params\_)

Best Average Score: 91.67%

Best Parameters: {'max\_depth': 6, 'min\_samples\_leaf': 5, 'min\_samples\_split': 2}

In [22]:

# Look at the performance of the grid search

pd.DataFrame.from\_dict(dt\_gridSearch.cv\_results\_).head()

Out[22]:

|  | **mean\_fit\_time** | **std\_fit\_time** | **mean\_score\_time** | **std\_score\_time** | **param\_max\_depth** | **param\_min\_samples\_leaf** | **param\_min\_samples\_split** | **params** | **split0\_test\_score** | **split1\_test\_score** | **split2\_test\_score** | **split3\_test\_score** | **split4\_test\_score** | **mean\_test\_score** | **std\_test\_score** | **rank\_test\_score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 3.341740 | 0.021809 | 0.111938 | 0.001017 | 4 | 1 | 2 | {'max\_depth': 4, 'min\_samples\_leaf': 1, 'min\_s... | 0.89647 | 0.900149 | 0.896969 | 0.893212 | 0.893738 | 0.896108 | 0.002498 | 19 |
| **1** | 2.821187 | 0.655247 | 0.111271 | 0.005768 | 4 | 1 | 5 | {'max\_depth': 4, 'min\_samples\_leaf': 1, 'min\_s... | 0.89647 | 0.900149 | 0.896969 | 0.893212 | 0.893738 | 0.896108 | 0.002498 | 19 |
| **2** | 2.038571 | 0.029889 | 0.101496 | 0.013023 | 4 | 1 | 10 | {'max\_depth': 4, 'min\_samples\_leaf': 1, 'min\_s... | 0.89647 | 0.900149 | 0.896969 | 0.893212 | 0.893738 | 0.896108 | 0.002498 | 19 |
| **3** | 2.086425 | 0.048071 | 0.106931 | 0.003088 | 4 | 2 | 2 | {'max\_depth': 4, 'min\_samples\_leaf': 2, 'min\_s... | 0.89647 | 0.900149 | 0.896969 | 0.893212 | 0.893738 | 0.896108 | 0.002498 | 19 |
| **4** | 2.120618 | 0.034054 | 0.104474 | 0.003454 | 4 | 2 | 5 | {'max\_depth': 4, 'min\_samples\_leaf': 2, 'min\_s... | 0.89647 | 0.900149 | 0.896969 | 0.893212 | 0.893738 | 0.896108 | 0.002498 | 19 |

In [23]:

# Based on the grid search create the model with the best parameters

bestTree = DecisionTreeClassifier(criterion='entropy', max\_depth= 6, min\_samples\_leaf= 5, min\_samples\_split=2 ,random\_state=0)

bestTree.fit(train\_X, train\_y)

Out[23]:

DecisionTreeClassifier(criterion='entropy', max\_depth=6, min\_samples\_leaf=5,

random\_state=0)

In [24]:

#Training vs Validation Data Performance

classificationSummary(train\_y, bestTree.predict(train\_X))

classificationSummary(valid\_y, bestTree.predict(valid\_X))

Confusion Matrix (Accuracy 0.9922)

Prediction

Actual 0 1

0 5 6400

1 2 809935

Confusion Matrix (Accuracy 0.9921)

Prediction

Actual 0 1

0 1 4282

1 4 539942

In [25]:

# Display the decision tree

fig, ax = plt.subplots(figsize=(40, 30))

plot\_tree(bestTree, fontsize = 10, filled=True, feature\_names=X.columns, proportion=True)

plt.show()

In [26]:

classTree\_proba = bestTree.predict\_proba(valid\_X)

ct\_result1 = pd.DataFrame({'actual': valid\_y,

'p(Class = 0)': [p[0] for p in classTree\_proba],

'p(Class = 1)': [p[1] for p in classTree\_proba]})

ct\_result1.head()

Out[26]:

|  | **actual** | **p(Class = 0)** | **p(Class = 1)** |
| --- | --- | --- | --- |
| **983487** | 1 | 0.000279 | 0.999721 |
| **1023005** | 1 | 0.000075 | 0.999925 |
| **359194** | 1 | 0.000075 | 0.999925 |
| **992041** | 1 | 0.068160 | 0.931840 |
| **917338** | 1 | 0.000279 | 0.999721 |

In [27]:

# Confusion matrix for the best threshold

THRESHOLD = [.5, .6, .7, .8]

# output formatting

pd.options.display.float\_format = '{:,.3f}'.format

np.set\_printoptions(precision=3,suppress=True)

# Create dataframe to store results

results = pd.DataFrame(columns=["THRESHOLD", "accuracy", "true pos rate (sens)", "true neg rate (spec)", "false pos rate (1-spec)", "precision"])

# Create threshold row

results['THRESHOLD'] = THRESHOLD

j = 0

# Iterate over the thresholds

# The model is already fit, so just find the predictions

for i in THRESHOLD:

# If prob for test set > threshold predict 1

preds = np.where(bestTree.predict\_proba(valid\_X)[:,1] > i, 1, 0)

# create confusion matrix (as percentage)

cm = (confusion\_matrix(valid\_y, preds,labels=[0, 1], sample\_weight=None) / len(valid\_y))\*100

# OR create confusion matrix as counts

#cm = confusion\_matrix(train\_y, preds,labels=[0, 1], sample\_weight=None)

print('Confusion matrix for threshold =',i)

print(cm)

print(' ')

TN = cm[0][0] # True Negatives

FP = cm[0][1] # False Positives

FN = cm[1][0] # False Negatives

TP = cm[1][1] # True Positives

results.iloc[j,1] = accuracy\_score(valid\_y, preds)

results.iloc[j,2] = recall\_score(valid\_y, preds) # Recall is the TPR or sensitivity

results.iloc[j,3] = TN/(FP+TN) # True negative rate or specificity

results.iloc[j,4] = FP/(FP+TN) # False positive rate or (1-specificity)

results.iloc[j,5] = precision\_score(valid\_y, preds)

j += 1

print('ALL METRICS')

print( results.T)

Confusion matrix for threshold = 0.5

[[ 0. 0.787]

[ 0.001 99.212]]

Confusion matrix for threshold = 0.6

[[ 0. 0.787]

[ 0.001 99.212]]

Confusion matrix for threshold = 0.7

[[ 0.006 0.781]

[ 0.019 99.194]]

Confusion matrix for threshold = 0.8

[[ 0.122 0.665]

[ 0.385 98.828]]

ALL METRICS

0 1 2 3

THRESHOLD 0.500 0.600 0.700 0.800

accuracy 0.992 0.992 0.992 0.989

true pos rate (sens) 1.000 1.000 1.000 0.996

true neg rate (spec) 0.000 0.000 0.008 0.155

false pos rate (1-spec) 1.000 1.000 0.992 0.845

precision 0.992 0.992 0.992 0.993

In [28]:

# Create ROC curve

fpr, tpr, \_ = roc\_curve(valid\_y, bestTree.predict\_proba(valid\_X)[:,1])

roc\_auc = auc(fpr,tpr)

In [29]:

plt.figure(figsize=(8,6)) # format the plot size

lw = 1.5

plt.plot(fpr, tpr, color='darkorange', marker='.',

lw=lw, label='Decision Tree (AUC = {:0.4f})'.format(roc\_auc))

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--',

label='Random Prediction (AUC = 0.5)' )

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic curve')

plt.legend(loc="lower right")

plt.show()

# **This model performs very well. Its AUC is 92.38%.**

## **Classifier 2: LOGISTIC REGRESSION**

In [30]:

# Use the same predictors as before

predictors = ['national\_inv', 'forecast\_3\_month', 'forecast\_6\_month',

'forecast\_9\_month', 'sales\_1\_month', 'sales\_3\_month', 'sales\_6\_month',

'sales\_9\_month', 'min\_bank', 'deck\_risk', 'ppap\_risk', 'stop\_auto\_buy']

outcome = 'went\_on\_backorder'

In [31]:

# Lets standardize the data by rows again

scaler = Normalizer()

QtyFields = quant\_predictors

scaler.fit(inventory\_df[QtyFields])

inventory\_df[QtyFields] = scaler.transform(inventory\_df[QtyFields])

In [32]:

# Split the data into the predictors and outcomes

# Get dummie values for the predictors

X = pd.get\_dummies(inventory\_df[predictors],

drop\_first = True)

y = inventory\_df[outcome]

In [33]:

#Split the data into training and validation sets

#The training data will be 60% of the data

train\_X, valid\_X, train\_y, valid\_y = train\_test\_split(X,y,train\_size=0.6,random\_state=1)

In [34]:

# Create the logistic regression model

logit\_reg = LogisticRegression(penalty="l2", C=1e42, solver='liblinear')

# Fit the model to the training data set

logit\_reg.fit(train\_X, train\_y)

print('Intercept {:.5f}'.format(logit\_reg.intercept\_[0]))

print(pd.DataFrame({'Predictor': train\_X.columns, 'Coefficient': logit\_reg.coef\_[0]}),'\n')

Intercept 4.30400

Predictor Coefficient

0 national\_inv 5.639

1 forecast\_3\_month -1.921

2 forecast\_6\_month -1.407

3 forecast\_9\_month 0.281

4 sales\_1\_month -4.013

5 sales\_3\_month -3.203

6 sales\_6\_month -0.301

7 sales\_9\_month 0.095

8 min\_bank 1.357

9 deck\_risk\_Yes 0.332

10 ppap\_risk\_Yes -0.199

11 stop\_auto\_buy\_Yes 0.352

In [35]:

logit\_reg\_proba = logit\_reg.predict\_proba(train\_X)

logit\_result1 = pd.DataFrame({'Actual Class': train\_y,

'p(Class = 0)': [p[0] for p in logit\_reg\_proba],

'p(Class = 1)': [p[1] for p in logit\_reg\_proba]})

logit\_result1.sample(10,random\_state=1)

Out[35]:

|  | **Actual Class** | **p(Class = 0)** | **p(Class = 1)** |
| --- | --- | --- | --- |
| **386056** | 1 | 0.000 | 1.000 |
| **57499** | 1 | 0.000 | 1.000 |
| **1029782** | 1 | 0.017 | 0.983 |
| **284618** | 1 | 0.013 | 0.987 |
| **99408** | 1 | 0.032 | 0.968 |
| **884435** | 1 | 0.000 | 1.000 |
| **1178744** | 1 | 0.000 | 1.000 |
| **757421** | 1 | 0.000 | 1.000 |
| **1290633** | 1 | 0.000 | 1.000 |
| **39312** | 1 | 0.038 | 0.962 |

In [36]:

# Explore differnet cut offs for the data

logit\_reg\_pred = logit\_reg.predict(train\_X)

logit\_result2 = pd.DataFrame({'Actual Class': train\_y,

'p(Class = 0)': [p[0] for p in logit\_reg\_proba],

'p(Class = 1)': [p[1] for p in logit\_reg\_proba],

'Predicted Class (0.5 cutoff)': logit\_reg\_pred })

logit\_result2.sample(10,random\_state=1)

Out[36]:

|  | **Actual Class** | **p(Class = 0)** | **p(Class = 1)** | **Predicted Class (0.5 cutoff)** |
| --- | --- | --- | --- | --- |
| **386056** | 1 | 0.000 | 1.000 | 1 |
| **57499** | 1 | 0.000 | 1.000 | 1 |
| **1029782** | 1 | 0.017 | 0.983 | 1 |
| **284618** | 1 | 0.013 | 0.987 | 1 |
| **99408** | 1 | 0.032 | 0.968 | 1 |
| **884435** | 1 | 0.000 | 1.000 | 1 |
| **1178744** | 1 | 0.000 | 1.000 | 1 |
| **757421** | 1 | 0.000 | 1.000 | 1 |
| **1290633** | 1 | 0.000 | 1.000 | 1 |
| **39312** | 1 | 0.038 | 0.962 | 1 |

In [37]:

cutoff = 0.8

logit\_reg\_pred2 = (logit\_reg.predict\_proba(train\_X)[:,1] >= cutoff).astype(int)

logit\_result3 = pd.DataFrame({'Actual Class': train\_y,

'p(Class = 0)': [p[0] for p in logit\_reg\_proba],

'p(Class = 1)': [p[1] for p in logit\_reg\_proba],

'Predicted Class (0.5 cutoff)': logit\_reg\_pred,

'Predicted Class (0.8 cutoff)': logit\_reg\_pred2 })

logit\_result3.sample(10,random\_state=1)

Out[37]:

|  | **Actual Class** | **p(Class = 0)** | **p(Class = 1)** | **Predicted Class (0.5 cutoff)** | **Predicted Class (0.8 cutoff)** |
| --- | --- | --- | --- | --- | --- |
| **386056** | 1 | 0.000 | 1.000 | 1 | 1 |
| **57499** | 1 | 0.000 | 1.000 | 1 | 1 |
| **1029782** | 1 | 0.017 | 0.983 | 1 | 1 |
| **284618** | 1 | 0.013 | 0.987 | 1 | 1 |
| **99408** | 1 | 0.032 | 0.968 | 1 | 1 |
| **884435** | 1 | 0.000 | 1.000 | 1 | 1 |
| **1178744** | 1 | 0.000 | 1.000 | 1 | 1 |
| **757421** | 1 | 0.000 | 1.000 | 1 | 1 |
| **1290633** | 1 | 0.000 | 1.000 | 1 | 1 |
| **39312** | 1 | 0.038 | 0.962 | 1 | 1 |

In [38]:

Cutoff = 0.8

# Need to match how the actual classes are identified. Can use strings as long as they match exactly

class\_names = [0, 1]

# list comprehension to classify the records based on the predicted probability AND the

# cutoff

predicted = [1 if p > Cutoff else 0 for p in logit\_reg\_proba[:,1]]

classificationSummary(train\_y, predicted, class\_names=class\_names)

Confusion Matrix (Accuracy 0.9915)

Prediction

Actual 0 1

0 85 6320

1 631 809306

In [39]:

# Accuracy using sklearn

accuracy = accuracy\_score(train\_y, predicted)

print('The accuracy of this model with the chosen cutoff is: {:.2f}%'.format(accuracy\*100))

# Confusion Matrix from sklearn

cm = confusion\_matrix(train\_y, predicted)

print(cm)

disp = ConfusionMatrixDisplay(cm)

disp.plot()

The accuracy of this model with the chosen cutoff is: 99.15%

[[ 85 6320]

[ 631 809306]]

Out[39]:

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7fa20a5fc8b0>

In [40]:

# Calculate the receiver operating curve and the AUC measure

# Build the model - here using the validation data

lr\_prob=logit\_reg.predict\_proba(valid\_X)

# Just get the prediction probabilities for Class 1 (target class)

lr\_prob=lr\_prob[:, 1]

lr\_auc=roc\_auc\_score(valid\_y, lr\_prob)

print("AUC predictions from logistic regression model = {:.4f}".format(lr\_auc))

# the roc\_curve function returns a series of false postive rates (x-axis)

# and a series of true positive rates

# for a series of potential cutoffs (1558 of them in this case!)

# The roc is just the plot of the connected scatterplot of these points

lr\_fpr,lr\_tpr,\_=roc\_curve(valid\_y,lr\_prob)

plt.figure(figsize=[8, 8])

# plot the dotted line that would show a random classifier with

# AUC of 0.5.

plt.plot([0, 1], [0, 1], linestyle='--',label='Random Predction')

plt.plot(lr\_fpr,lr\_tpr,marker='.',label='Logistic Regression')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.legend()

plt.show()

AUC predictions from logistic regression model = 0.9106

# **This model also performs very well. The AUC is 91.06%.**

# **Classifier 3: KNN**

In [41]:

# Use the 10% subset of data for the KNN model

subset\_df.head()

Out[41]:

|  | **sku** | **national\_inv** | **lead\_time** | **in\_transit\_qty** | **forecast\_3\_month** | **forecast\_6\_month** | **forecast\_9\_month** | **sales\_1\_month** | **sales\_3\_month** | **sales\_6\_month** | **...** | **pieces\_past\_due** | **perf\_6\_month\_avg** | **perf\_12\_month\_avg** | **local\_bo\_qty** | **deck\_risk** | **oe\_constraint** | **ppap\_risk** | **stop\_auto\_buy** | **rev\_stop** | **went\_on\_backorder** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **548027** | 1895984 | 51.000 | 8.000 | 0.000 | 15.000 | 31.000 | 55.000 | 11.000 | 27.000 | 54.000 | ... | 0.000 | 0.860 | 0.860 | 0.000 | No | No | No | Yes | No | 1 |
| **471204** | 1819190 | 2.000 | 9.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | ... | 0.000 | 0.680 | 0.680 | 0.000 | No | No | No | Yes | No | 1 |
| **559925** | 1907883 | 5.000 | 10.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 4.000 | ... | 0.000 | 0.890 | 0.890 | 0.000 | No | No | No | Yes | No | 1 |
| **914050** | 2262035 | 42.000 | 8.000 | 139.000 | 205.000 | 249.000 | 304.000 | 52.000 | 108.000 | 212.000 | ... | 0.000 | 0.940 | 0.940 | 99.000 | No | No | No | Yes | No | 1 |
| **180178** | 1291732 | 4.000 | 2.000 | 0.000 | 3.000 | 7.000 | 10.000 | 1.000 | 4.000 | 4.000 | ... | 0.000 | 0.990 | 0.990 | 0.000 | Yes | No | No | Yes | No | 0 |

5 rows × 23 columns

In [42]:

# Standardize the data by rows

scaler = Normalizer()

QtyFields = quant\_predictors

scaler.fit(subset\_df[QtyFields])

subset\_df[QtyFields] = scaler.transform(subset\_df[QtyFields])

In [43]:

# split the data into training ad test data sets

# training data will make up 60% of the subset data

train\_df, test\_df = train\_test\_split(subset\_df, train\_size=0.6, random\_state=1)

In [44]:

# Split the train\_df into our X and y variables

# get dummies for the categorical predictors

train\_X = pd.get\_dummies(train\_df[predictors],

columns = cat\_predictors,

drop\_first = True)

train\_y = train\_df[outcome]

In [45]:

# Standardize the data by features

# This will calculate the standard deviation for every entry

z\_scaler = StandardScaler()

z\_scaler.fit(train\_X)

train\_X\_std = pd.DataFrame(z\_scaler.transform(train\_X),columns=train\_X.columns,

index=train\_X.index)

In [46]:

# Perform a grid search to find the best number of neighbors

param\_grid = {

'n\_neighbors': [k for k in range(5,20,2)]

}

# Scoring is based off of roc\_auc

gridSearch = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5, n\_jobs=-1,scoring='roc\_auc')

# fit and find the "best" parameters from our data.

gridSearch.fit(train\_X\_std, train\_y)

print('Best (Average) Accuracy: {:.2f}%'.format(gridSearch.best\_score\_\*100))

print('Best Parameter: ', gridSearch.best\_params\_)

Best (Average) Accuracy: 76.08%

Best Parameter: {'n\_neighbors': 19}

In [47]:

# My choice is to rebuild (fit) the model with ALL the standardized training data and the best found parameters.

k = 19

# 19 neighbors was best, so let's use it!

neigh\_class = KNeighborsClassifier(n\_neighbors=k)

neigh\_class.fit(train\_X, train\_y)

Out[47]:

KNeighborsClassifier(n\_neighbors=19)

In [48]:

# Repeat the same process for test data

test\_X = pd.get\_dummies(test\_df[predictors],

columns = cat\_predictors,

drop\_first = True)

test\_y = test\_df[outcome]

In [49]:

# standardize the features of test data

test\_X\_std = pd.DataFrame(z\_scaler.transform(test\_X),columns=test\_X.columns,

index=test\_X.index)

In [50]:

# Classify the test data using our selected KNN model (k=19)!!

# Check the predictions for the training data..meaning compute the actual probabilities using your model!

neigh\_class\_proba = neigh\_class.predict\_proba(test\_X\_std)

neighC\_result = pd.DataFrame({'Actual TEST Class': test\_y,

'p(Class = 1)': [p[1] for p in neigh\_class\_proba]})

neighC\_result.head()

Out[50]:

|  | **Actual TEST Class** | **p(Class = 1)** |
| --- | --- | --- |
| **268987** | 1 | 1.000 |
| **1277655** | 1 | 0.947 |
| **61518** | 1 | 1.000 |
| **460147** | 1 | 1.000 |
| **1310518** | 1 | 0.947 |

In [51]:

# Let's check the accuracy on the TEST data

pred\_probs = neigh\_class\_proba[:,1]

predicted = [1 if p > 0.8 else 0 for p in pred\_probs]

# Accuracy using sklearn for the 0.5 cutoff

accuracy = accuracy\_score(test\_y, predicted)

print('The accuracy of this model with the chosen cutoff is: {:.2f}%'.format(accuracy\*100))

# Confusion Matrix from sklearn

cm = confusion\_matrix(test\_y, predicted)

print(cm)

disp = ConfusionMatrixDisplay(cm)

disp.plot();

The accuracy of this model with the chosen cutoff is: 97.70%

[[ 36 384]

[ 870 53133]]

In [52]:

# Calculate the receiver operating curve and the AUC measure

# Build the model - here using the validation data

knn\_auc=roc\_auc\_score(test\_y, pred\_probs)

print("AUC metric using 19-NN model = {:.4f}".format(knn\_auc))

# the roc\_curve function returns a series of false postive rates (x-axis)

# and a series of true positive rates

# for a series of potential cutoffs (1558 of them in this case!)

# The roc is just the plot of the connected scatterplot of these points

knn\_fpr,knn\_tpr,\_= roc\_curve(test\_y,pred\_probs)

plt.figure(figsize=[8, 8])

# plot the dotted line that would show a random classifier with

# AUC of 0.5.

plt.plot([0, 1], [0, 1], linestyle='--',label='Random Predction')

plt.plot(knn\_fpr,knn\_tpr,marker='.',label='KNN')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.legend()

plt.show()

AUC metric using 19-NN model = 0.7593

# **This model did not perform as well as the others. The AUC for ROC is 70.38%.**

# **Create Predictions on Test Data**

## **Perform the same data cleaning as we did on the training data**

In [53]:

inventory\_test\_df = pd.read\_csv("Data/inventory\_test.csv")

inventory\_test\_df.head()

Out[53]:

|  | **sku** | **national\_inv** | **lead\_time** | **in\_transit\_qty** | **forecast\_3\_month** | **forecast\_6\_month** | **forecast\_9\_month** | **sales\_1\_month** | **sales\_3\_month** | **sales\_6\_month** | **...** | **potential\_issue** | **pieces\_past\_due** | **perf\_6\_month\_avg** | **perf\_12\_month\_avg** | **local\_bo\_qty** | **deck\_risk** | **oe\_constraint** | **ppap\_risk** | **stop\_auto\_buy** | **rev\_stop** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 3285085 | 62 | NaN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | No | 0 | -99.000 | -99.000 | 0 | Yes | No | No | Yes | No |
| **1** | 3285131 | 9 | NaN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | No | 0 | -99.000 | -99.000 | 0 | No | No | Yes | No | No |
| **2** | 3285358 | 17 | 8.000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | No | 0 | 0.920 | 0.950 | 0 | No | No | No | Yes | No |
| **3** | 3285517 | 9 | 2.000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | No | 0 | 0.780 | 0.750 | 0 | No | No | Yes | Yes | No |
| **4** | 3285608 | 2 | 8.000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | No | 0 | 0.540 | 0.710 | 0 | No | No | No | Yes | No |

5 rows × 22 columns

In [54]:

# replace NA values with median

median\_leadtime\_test = inventory\_test\_df.lead\_time.median()

inventory\_test\_df.lead\_time = inventory\_test\_df.lead\_time.fillna( value = median\_leadtime\_test)

# replace -99 values with NA

inventory\_test\_df.perf\_6\_month\_avg = inventory\_test\_df.perf\_6\_month\_avg.replace(to\_replace = -99.00, value = np.nan)

inventory\_test\_df.perf\_12\_month\_avg = inventory\_test\_df.perf\_12\_month\_avg.replace(to\_replace = -99.00, value = np.nan)

# calculate the median averages

perf\_6\_median\_test = inventory\_test\_df.perf\_6\_month\_avg.median()

perf\_12\_median\_test = inventory\_test\_df.perf\_12\_month\_avg.median()

# fill NA values with median value

inventory\_test\_df.perf\_6\_month\_avg = inventory\_test\_df.perf\_6\_month\_avg.fillna(perf\_6\_median\_test)

inventory\_test\_df.perf\_12\_month\_avg = inventory\_test\_df.perf\_6\_month\_avg.fillna(perf\_12\_median\_test)

# Add target variable column to test data frame

inventory\_test\_df['went\_on\_backorder'] = 0

In [55]:

# Get dummies for teh predictions

prediction\_df = pd.get\_dummies(inventory\_test\_df[predictors], drop\_first = True)

In [56]:

# Prediction Values for Decision Tree Model

submit\_DT\_df = pd.DataFrame(columns = ['sku', 'propensity'])

submit\_DT\_df["sku"] = inventory\_test\_df["sku"]

submit\_DT\_df['propensity'] = bestTree.predict\_proba(prediction\_df)

# submit\_DT\_df['propensity'].hist()

In [57]:

# Prediction Values for Logistic Regression Model

submit\_log\_df = pd.DataFrame(columns = ['sku', 'propensity'])

submit\_log\_df["sku"] = inventory\_test\_df["sku"]

submit\_log\_df['propensity'] = logit\_reg.predict\_proba(prediction\_df)

# submit\_log\_df['propensity'].hist()

In [58]:

# Prediction Values for KNN Model

submit\_KNN\_df = pd.DataFrame(columns = ['sku', 'propensity'])

submit\_KNN\_df["sku"] = inventory\_test\_df["sku"]

submit\_KNN\_df['propensity'] = neigh\_class.predict\_proba(prediction\_df)

# submit\_KNN\_df['propensity'].hist()

In [60]:

# #Submit prediction propensity file

# submit\_KNN\_df.to\_csv("Final KNN Attempt.csv",index=False)

# **Model Analysis and Evaluation**

## **Classifier 1: Decision Tree performed the best**

### **Let's sort the data and find which items are most likely to be backordered**

In [64]:

top\_50 = submit\_DT\_df.sort\_values(by = 'propensity',

ascending = False).head(50)

In [65]:

#top\_50['sku'].to\_list()

# **Although I can't fully predict if these items will be back ordered, these are the top 50 items that are most likely to be backordered based on my model.**