Backorder Prediction

Problem statement.

Data: Back Order Prediction Data

Problem statement :

- Backorders are unavoidable, but by anticipating which things will be backordered,planning can be streamlined at several levels, preventing unexpected strain on production, logistics, and transportation. The datasets positive class corresponds to the product will go on backorder. The negative class corresponds to product will not go in backorder.
- The problem is to reduce the cost of supply chain in backorder. So it is required to minimize the false predictions.

True class	Positive	Negative
Predicted class		
Positive	-	cost_1
Negative	cost_2	

Cost 1 = 10 and Cost 2 = 200

- The total cost of a prediction model the sum of Cost_1 multiplied by the number of Instances with type 1 failure and Cost_2 with the number of instances with type 2 failure, resulting in a Total_cost. In this case Cost_1 refers to the unnessecary supply chain cost for keeping product in stock, while Cost_2 refer to the immediate supplly chain cost which required the product to keep in inventory as fast as possible.
- Total_cost = Cost_1 *No_Instances + Cost_2* No_Instances.
- From the above problem statement we could observe that, we have to reduce false positives and false negatives. More importantly we have to **reduce false negatives, since cost incurred due to false negative is 20 times higher than the false positives.**

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statistics import mean
import warnings

warnings.filterwarnings("ignore")
%matplotlib inline
```

In [2]:
```
#Load csv file
df = pd.read_csv('Kaggle_Training_Dataset_v2.csv')
```

In [3]:
```
#Dataset
df
```

Out[3]:

	sku	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_month	fore
0	1026827	0.0	NaN	0.0	0.0	0.0	
1	1043384	2.0	9.0	0.0	0.0	0.0	
2	1043696	2.0	NaN	0.0	0.0	0.0	
3	1043852	7.0	8.0	0.0	0.0	0.0	
4	1044048	8.0	NaN	0.0	0.0	0.0	
...	...	...	...	...	...	...	
1687856	1373987	-1.0	NaN	0.0	5.0	7.0	
1687857	1524346	-1.0	9.0	0.0	7.0	9.0	
1687858	1439563	62.0	9.0	16.0	39.0	87.0	
1687859	1502009	19.0	4.0	0.0	0.0	0.0	
1687860	(1687860 rows)	NaN	NaN	NaN	NaN	NaN	

1687861 rows × 23 columns

In [4]:
```
#Number of rows and columns of datasets
df.shape
```

Out[4]:
```
(1687861, 23)
```

In [5]:
```
# Dataset datatype information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1687861 entries, 0 to 1687860
Data columns (total 23 columns):
 #   Column             Non-Null Count    Dtype  
---  ------             --------------    -----  
 0   sku                1687861 non-null  object 
 1   national_inv       1687860 non-null  float64
 2   lead_time          1586967 non-null  float64
 3   in_transit_qty     1687860 non-null  float64
 4   forecast_3_month   1687860 non-null  float64
 5   forecast_6_month   1687860 non-null  float64
 6   forecast_9_month   1687860 non-null  float64
 7   sales_1_month      1687860 non-null  float64
 8   sales_3_month      1687860 non-null  float64
 9   sales_6_month      1687860 non-null  float64
 10  sales_9_month      1687860 non-null  float64
 11  min_bank           1687860 non-null  float64
 12  potential_issue    1687860 non-null  object 
 13  pieces_past_due    1687860 non-null  float64
 14  perf_6_month_avg   1687860 non-null  float64
 15  perf_12_month_avg  1687860 non-null  float64
 16  local_bo_qty       1687860 non-null  float64
 17  deck_risk          1687860 non-null  object 
 18  oe_constraint      1687860 non-null  object 
```

```
 19  ppap_risk          1687860 non-null  object
 20  stop_auto_buy      1687860 non-null  object
 21  rev_stop           1687860 non-null  object
 22  went_on_backorder  1687860 non-null  object
dtypes: float64(15), object(8)
memory usage: 296.2+ MB
```

In [6]:

```
# Finding caterical and numerical columns
categorical_col = [feature for feature in df.columns if df[feature].dtypes=='O']
numeircal_col  = [feature for feature in df.columns if df[feature].dtypes!='O']
print(f"no. of categorical features {len(categorical_col)}",categorical_col)
print(f"no. of numerical features {len(numeircal_col)}",numeircal_col)
```

```
no. of categorical features 8 ['sku', 'potential_issue', 'deck_risk', 'oe_constrain
t', 'ppap_risk', 'stop_auto_buy', 'rev_stop', 'went_on_backorder']
no. of numerical features 15 ['national_inv', 'lead_time', 'in_transit_qty', 'foreca
st_3_month', 'forecast_6_month', 'forecast_9_month', 'sales_1_month', 'sales_3_mont
h', 'sales_6_month', 'sales_9_month', 'min_bank', 'pieces_past_due', 'perf_6_month_a
vg', 'perf_12_month_avg', 'local_bo_qty']
```

In [7]:

```
# Checking unique values of target coolumns
df['went_on_backorder'].value_counts()
```

Out[7]:

```
No     1676567
Yes      11293
Name: went_on_backorder, dtype: int64
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
sku                       0
national_inv              1
lead_time            100894
in_transit_qty            1
forecast_3_month          1
forecast_6_month          1
forecast_9_month          1
sales_1_month             1
sales_3_month             1
sales_6_month             1
sales_9_month             1
min_bank                  1
potential_issue           1
pieces_past_due           1
perf_6_month_avg          1
perf_12_month_avg         1
local_bo_qty              1
deck_risk                 1
oe_constraint             1
ppap_risk                 1
stop_auto_buy             1
rev_stop                  1
went_on_backorder         1
dtype: int64
```

In [9]:

```
# Perctage of missing values in each Columns
for item in df.isnull().sum().items():
    print(f"{item[0]} has {round((item[1]/len(df))*100,2)}% missing values")
```

```
sku has 0.0% missing values
national_inv has 0.0% missing values
lead_time has 5.98% missing values
```

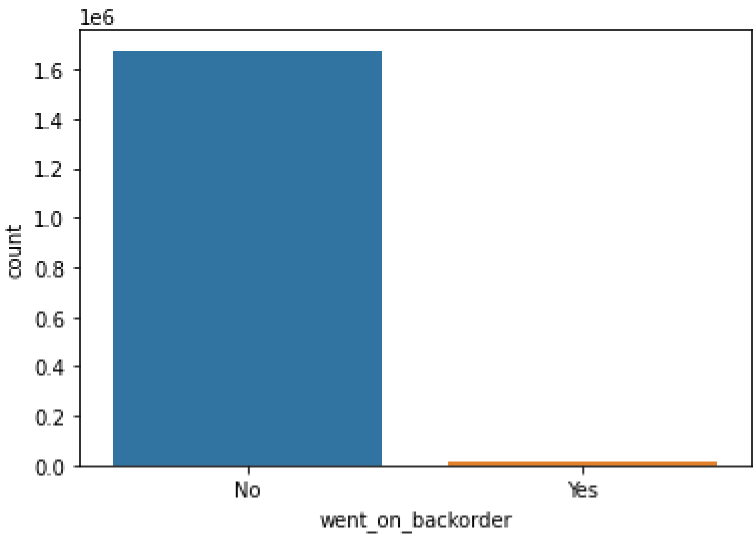
```
in_transit_qty has 0.0% missing values
forecast_3_month has 0.0% missing values
forecast_6_month has 0.0% missing values
forecast_9_month has 0.0% missing values
sales_1_month has 0.0% missing values
sales_3_month has 0.0% missing values
sales_6_month has 0.0% missing values
sales_9_month has 0.0% missing values
min_bank has 0.0% missing values
potential_issue has 0.0% missing values
pieces_past_due has 0.0% missing values
perf_6_month_avg has 0.0% missing values
perf_12_month_avg has 0.0% missing values
local_bo_qty has 0.0% missing values
deck_risk has 0.0% missing values
oe_constraint has 0.0% missing values
ppap_risk has 0.0% missing values
stop_auto_buy has 0.0% missing values
rev_stop has 0.0% missing values
went_on_backorder has 0.0% missing values
```

Visualization of unique values in Target variable

In [10]:

```
a,b=df['went_on_backorder'].value_counts().items()
print(f"{a[0]} : {a[1]} , {b[0]} : {b[1]}")
sns.countplot(x='went_on_backorder',data=df)
plt.show()
```

```
No : 1676567 , Yes : 11293
```



Report

- The target classes are highly imbalanced
- Class imbalance is a scenario that arises when we have unequal distribution of class in a dataset i.e. the no. of data points in the negative class (majority class) very large compared to that of the positive class (minority class)
- If the imbalanced data is not treated beforehand, then this will degrade the performance of the classifier model. Hence we should handle imbalanced data with certain methods.

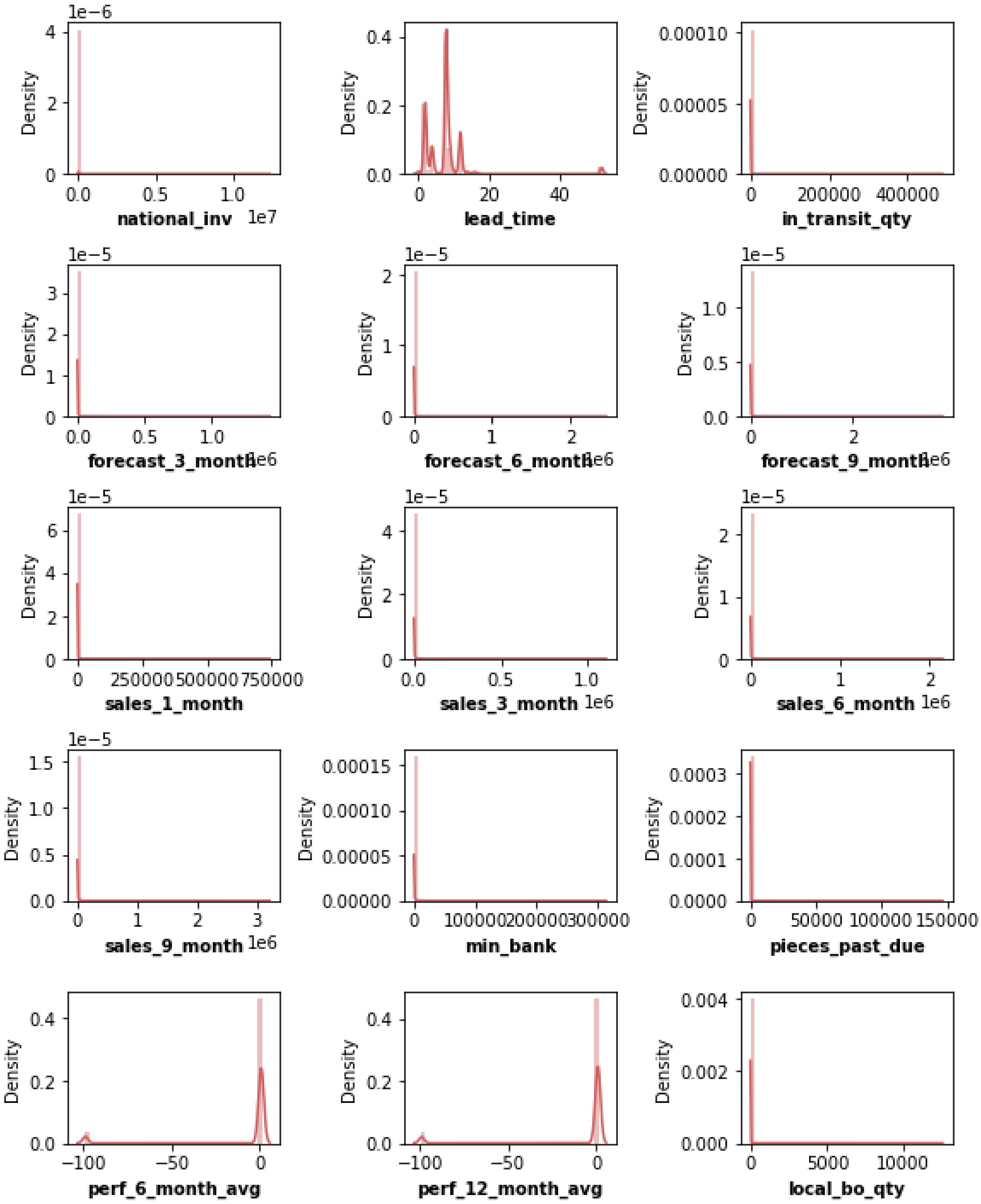
How to handle Imbalance Data ?

- Resampling data is one of the most commonly preferred approaches to deal with an imbalanced dataset. There are broadly two types of methods for this
- **i) Undersampling**
- **ii) Oversampling**
- In most cases, oversampling is preferred over undersampling techniques. The reason being, in undersampling we tend to remove instances from data that may be carrying some important information.
- **SMOTE:** Synthetic Minority Oversampling Technique
- SMOTE is an oversampling technique where the synthetic samples are generated for the minority class.
- Hybridization techniques involve combining both undersampling and oversampling techniques. This is done to optimize the performance of classifier models for the samples created as part of these techniques.
- It only duplicates the data and it won't add and new information. Hence we look at some different techniques.

Plot distribution of all Independent Numerical variables

In [11]:

```
plt.figure(figsize=(8, 10))
for i,col in enumerate(numeircal_col):
    plt.subplot(5,3,i+1)
    sns.distplot(x=df[col], color='indianred')
    plt.xlabel(col, weight='bold')
    plt.tight_layout()
```

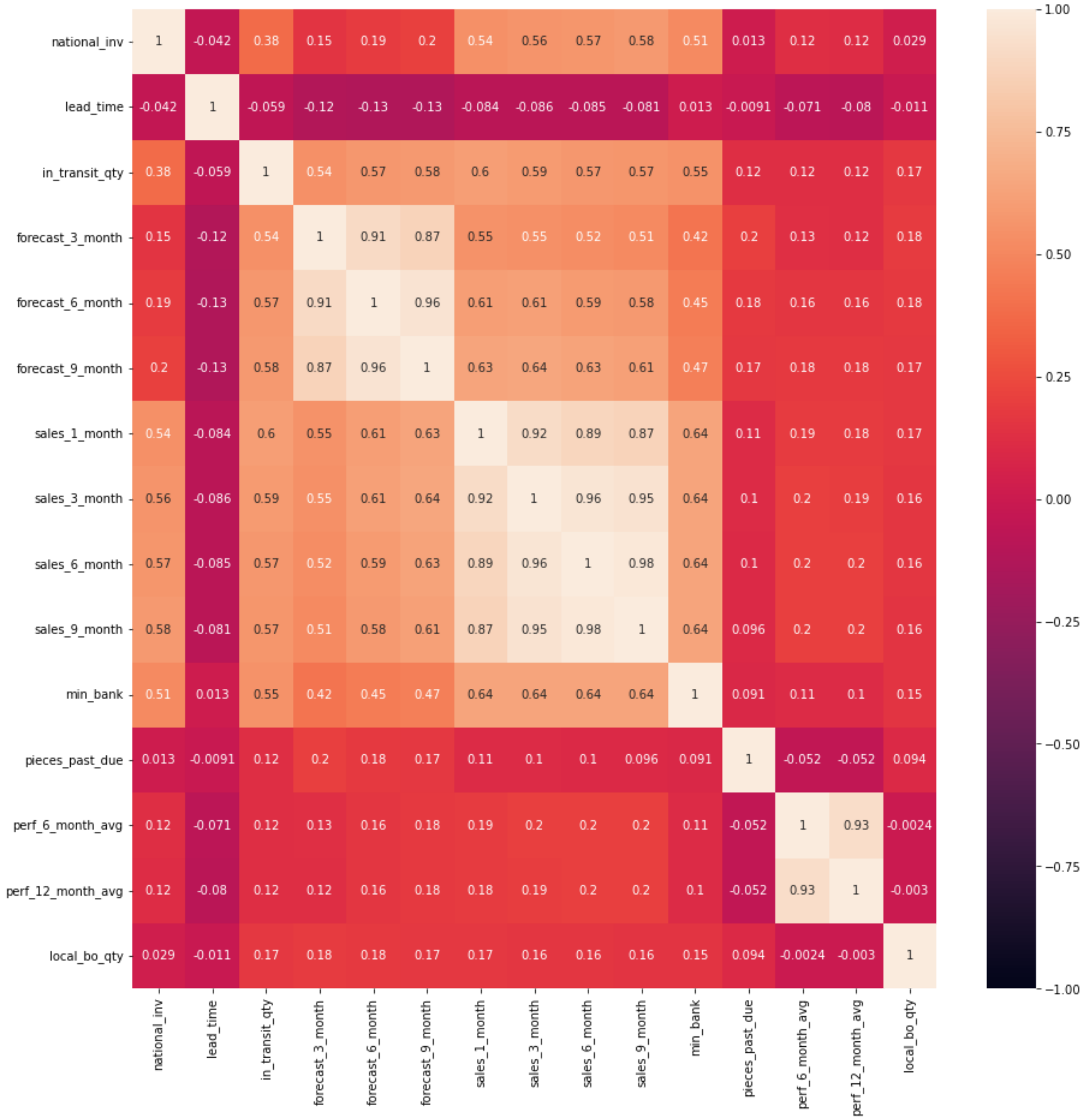


Report

- As per the above plot most of the features are not normally distributed.
- Transformation of data is not of prime importance since it is a classification problem.

In [12]:

```
# Checking co-relation in numerical columns
plt.figure(figsize=(15,15))
sns.heatmap(df[numeircal_col].corr(method='spearman'),annot=True,vmax=1.0,vmin=-1.0)
plt.show()
```



Insights :

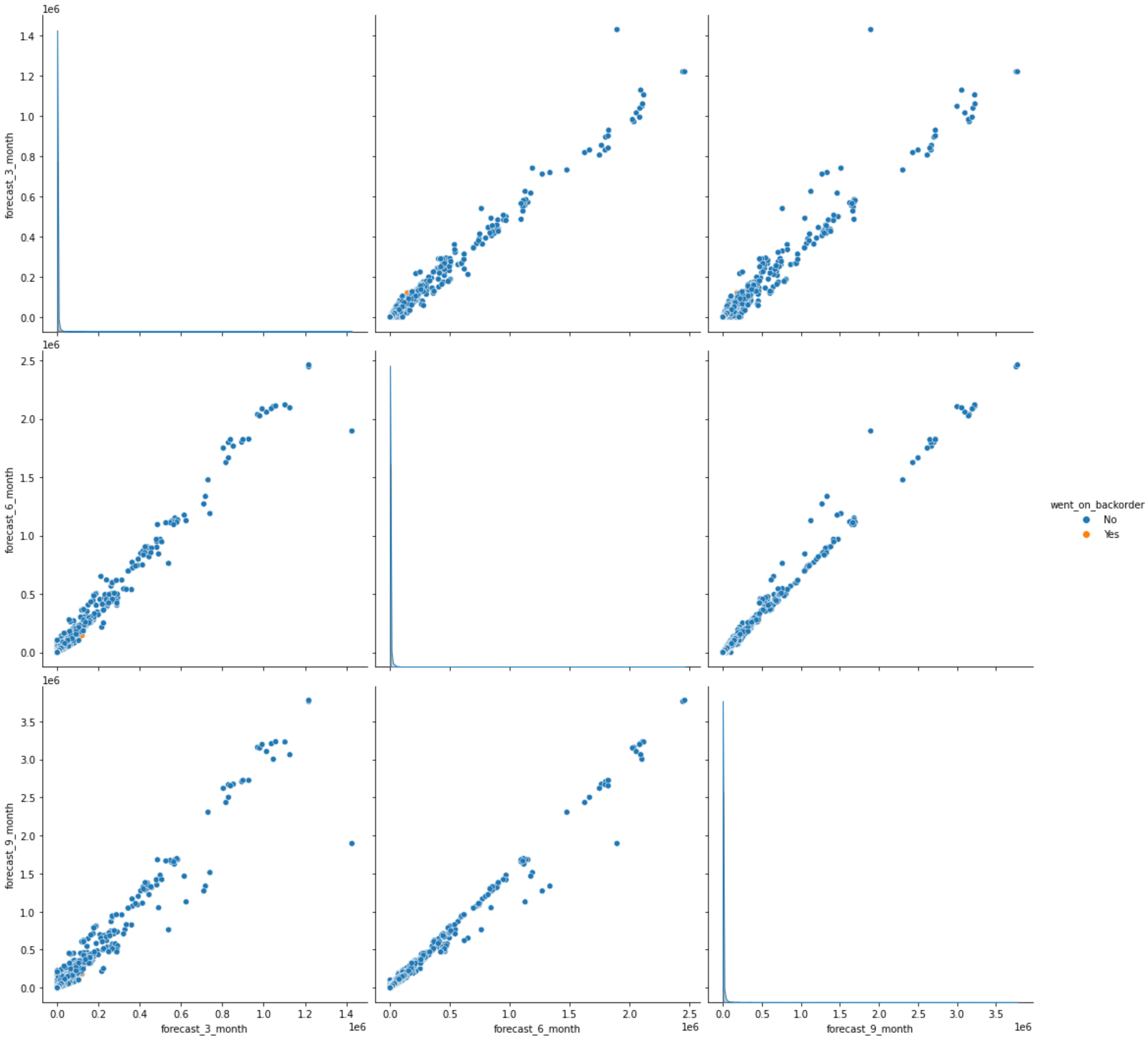
- The correlation matrix shows that the quantity in transit, the forecast sales over 3/6/9 months, the actual sales over the previous 1/3/6/9 months, and minimum recommended stock level are highly correlated.
- If the sales are high over the last 1/3/6/9 months, then it is reasonable for the forecast sales over the next 3/6/9 months to also be high. If forecast sales are high, then it would be useful to have more of the stock in hand and to have more shipped in.
- Besides that, the average performance over the last 6 months strongly correlates with that over the last 12 months.
- Overall, the correlation matrix suggests that the number of features used for predicting whether an item goes on back order can be lower than the number of features in the dataset. In other words, the dimensionality of the problem may be reduced.

In [13]:
```
## Lets take a closer look on the forescast period.
```

```
# Forecast columns
forecasts = ['forecast_3_month','forecast_6_month', 'forecast_9_month']

# Pair-wise scatter plot for the forecasts (3, 6 and 9)
sns.pairplot(df, x_vars=forecasts,y_vars=forecasts, hue='went_on_backorder', size=5)

# Show the plot
fig = plt.figure(figsize = (20 , 12))
plt.show()
```



<Figure size 1440x864 with 0 Axes>

Report:

- The forecast values over each time frame have very close linear correlation with each other, as expected from the correlation matrix.
- The forecast values cover a wide range from 0 to over 1 million.
- Backorders only occur when the forecast value is low.

In [14]:

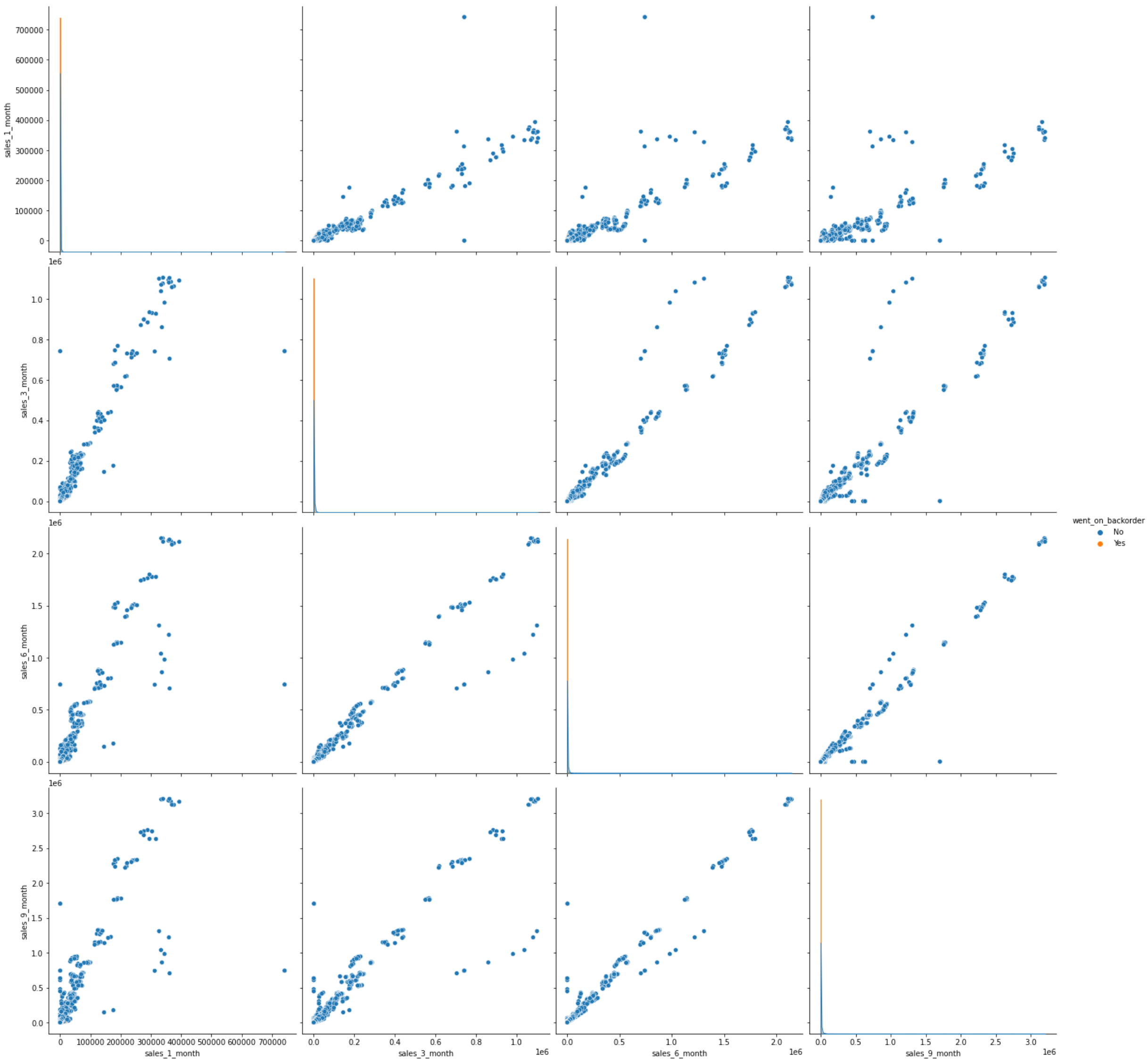
```
# Do a pair-wise scatter plot for sales

## Sales columns
sales = ['sales_1_month', 'sales_3_month', 'sales_6_month', 'sales_9_month']

## Pair-wise scatter plot for the respective sales months
sns.pairplot(df, vars=sales, hue='went_on_backorder', size=5)

## Plot the figure
```

```
fig = plt.figure(figsize = (20 , 12))
plt.show()
```




```
<Figure size 1440x864 with 0 Axes>
```

Report :

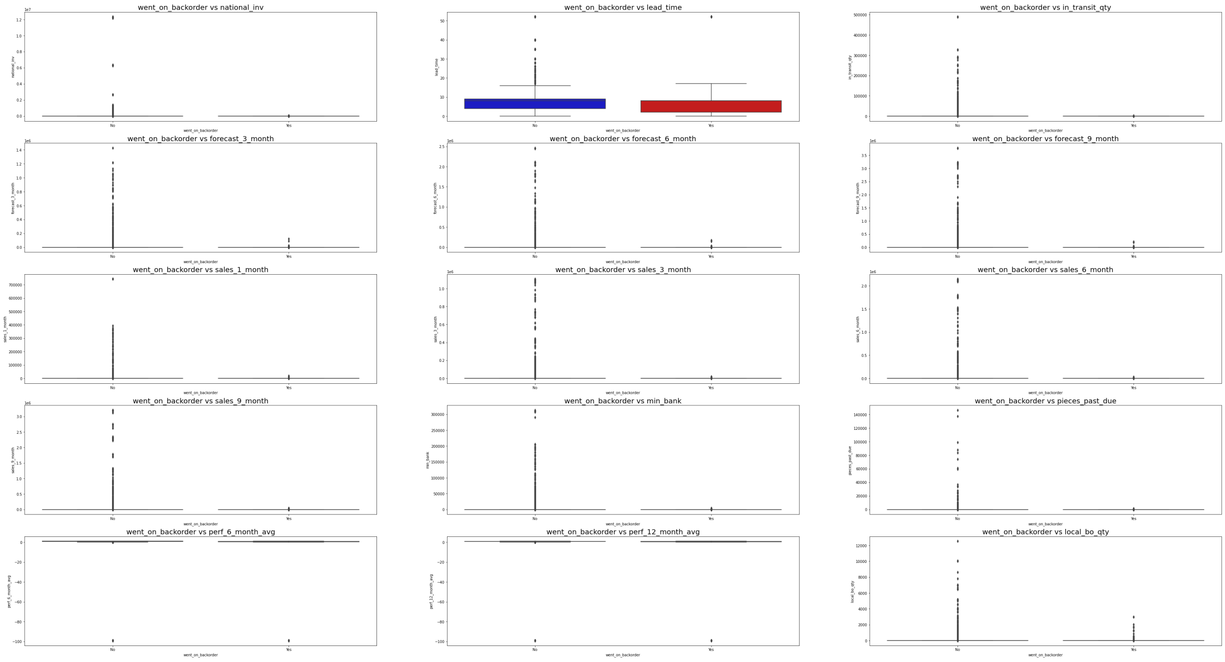
- The sales over each time frame have linear correlations with each other, as expected from the correlation matrix.
- There are some instances when the sales at different time frames fall away from the linear correlation.
- The sales range from 0 to over 1 million. Backorders only occur when sales are low.

In [15]:

```
colors = ["#0101DF", "#DF0101"] ## color combination
```

In [16]:

```
## Creating the Boxplot for Every Feature with respect to "Went on Backorder"
plt.figure(figsize=(60,100))
i=0
for feature in numeircal_col:
    plt.subplot(15,3,i+1)
    sns.boxplot(x="went_on_backorder", y=feature, data=df, palette=colors)
    plt.title(f'went_on_backorder vs {feature}' , fontsize= 20)
    plt.xlabel('went_on_backorder',fontsize = 10)
    plt.ylabel(feature, fontsize = 10)
    i=i+1
```



Report:

- There are plenty of outliers for every features with respect to target (went on backorder)
- We will not be removing these outliers because each it may result in loss of data
- Besides we will using Tree based Algorithms or Ensemble Techniques to predict the backorder
- The above mentioned algorithms are robust to outliers and skewness.

In [ ]:

```
```

In [17]:

```
df[categorical_col]
```

Out[17]:

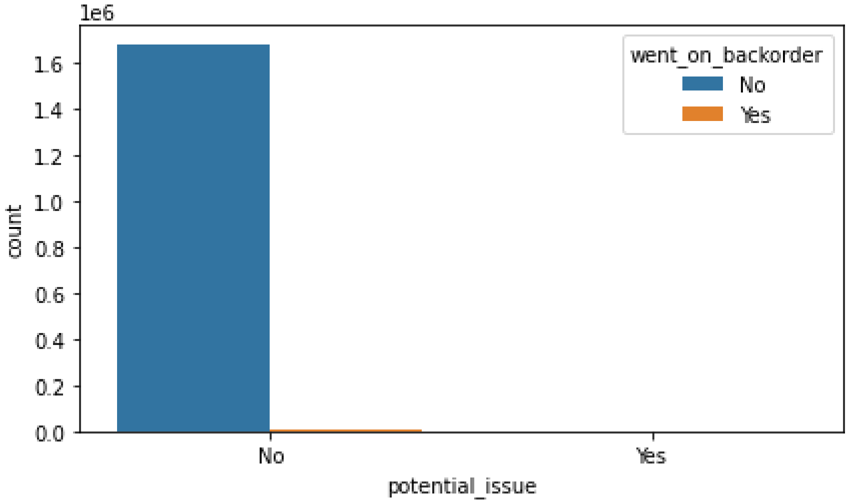
	sku	potential_issue	deck_risk	oe_constraint	ppap_risk	stop_auto_buy	rev_stop	we
0	1026827	No	No	No	No	Yes	No	
1	1043384	No	No	No	No	Yes	No	
2	1043696	No	Yes	No	No	Yes	No	
3	1043852	No	No	No	No	Yes	No	
4	1044048	No	Yes	No	No	Yes	No	
...	...	...	...	...	...	...	...	
1687856	1373987	No	No	No	No	Yes	No	
1687857	1524346	No	Yes	No	No	No	No	
1687858	1439563	No	No	No	No	Yes	No	
1687859	1502009	No	No	No	No	Yes	No	
1687860	(1687860 rows)	NaN	NaN	NaN	NaN	NaN	NaN	

1687861 rows × 8 columns

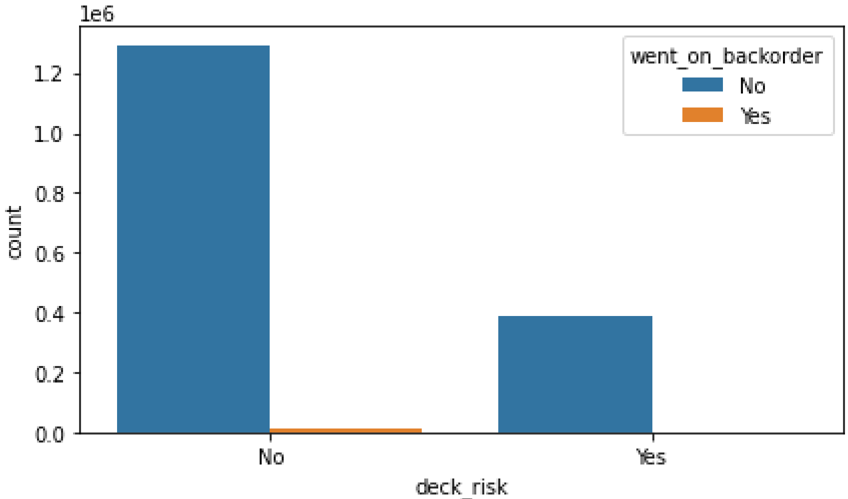
In [18]:
```
pd. set_option('display.max_rows', None)
k=0

for i in categorical_col:
    if i!='sku':
        plt.figure(figsize=(15,35))
        plt.subplot(8,2,k+1)
        print('When went on backorder  ',df[df['went_on_backorder']=='Yes'][i].value
        print('When didnt went on backorder  ',df[df['went_on_backorder']=='No'][i].
        sns.countplot(x=i,data=df,hue='went_on_backorder')
        k=k+1
        plt.show()
```

```
When went on backorder   No     11242
Yes       51
Name: potential_issue, dtype: int64
When didnt went on backorder   No     1675711
Yes        856
Name: potential_issue, dtype: int64
```

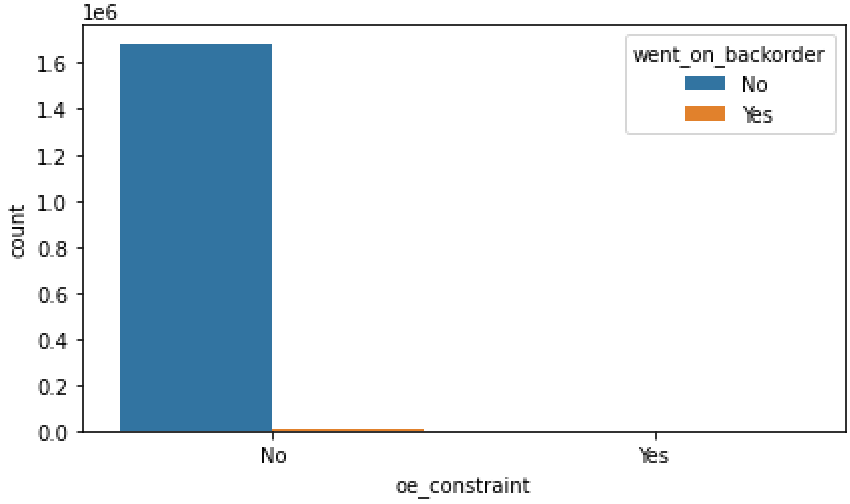



```
When went on backorder   No     9377
Yes    1916
Name: deck_risk, dtype: int64
When didnt went on backorder   No     1291000
Yes     385567
Name: deck_risk, dtype: int64
```

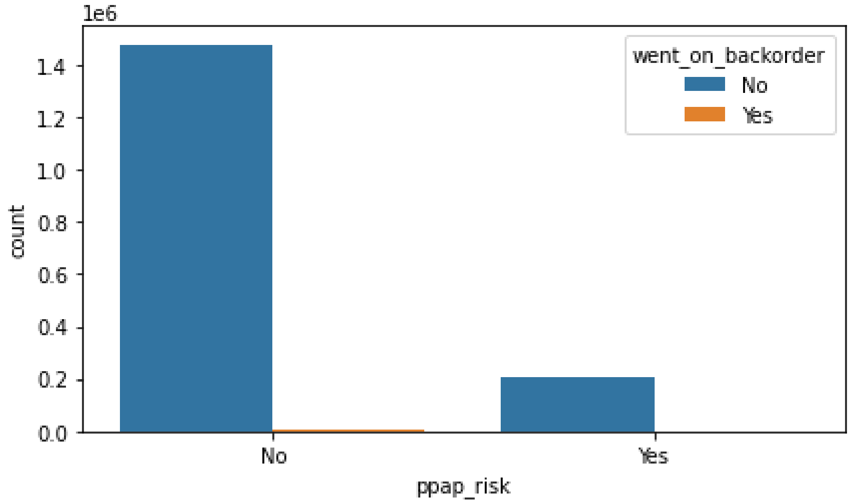



```
When went on backorder   No     11285
Yes        8
Name: oe_constraint, dtype: int64
When didnt went on backorder   No     1676330
```

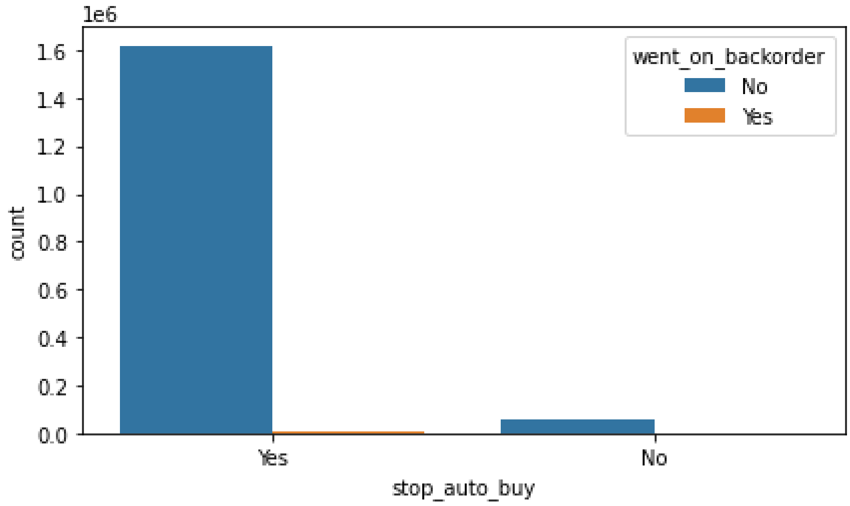
```
Yes        237
Name: oe_constraint, dtype: int64
```




```
When went on backorder   No     9534
Yes    1759
Name: ppap_risk, dtype: int64
When didnt went on backorder   No     1474492
Yes     202075
Name: ppap_risk, dtype: int64
```

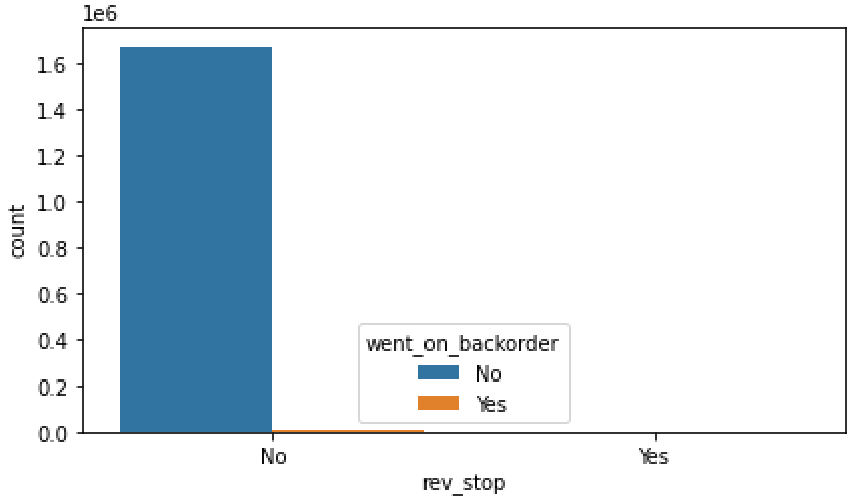



```
When went on backorder   Yes    10822
No       471
Name: stop_auto_buy, dtype: int64
When didnt went on backorder   Yes    1615952
No       60615
Name: stop_auto_buy, dtype: int64
```

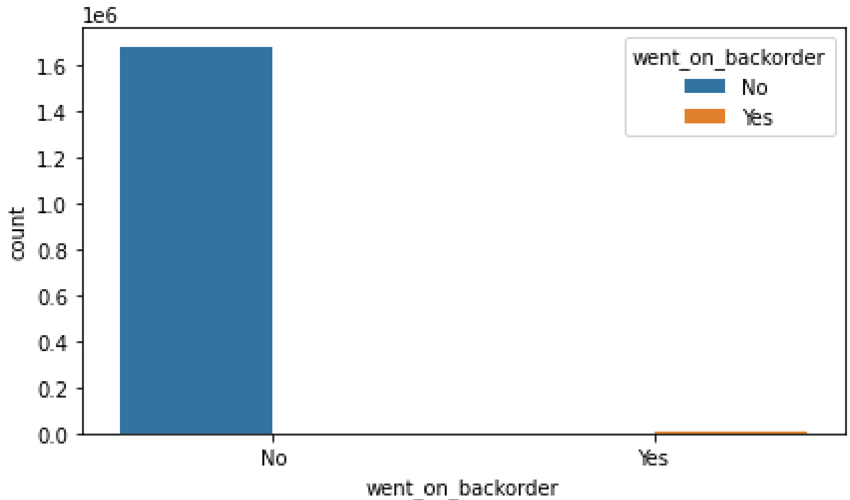



```
When went on backorder   No    11293
```

```
Name: rev_stop, dtype: int64
When didnt went on backorder    No      1675836
Yes         731
Name: rev_stop, dtype: int64
```




```
When went on backorder    Yes     11293
Name: went_on_backorder, dtype: int64
When didnt went on backorder    No     1676567
Name: went_on_backorder, dtype: int64
```



Report:

- As we can see from the plots, there's a very few number of products that actually went to back order.
- This implies the data is extremely imbalanced
- Using this imbalanced data, if we are to predict the backorder activity, the model will overfit
- Hence it is recommended that the data has to be balanced before building the model.

In [ ]: