About the data:

Let’s consider a Company dataset with around 10 variables and 400 records. The attributes are as follows:

Sales -- Unit sales (in thousands) at each location
Competitor Price -- Price charged by competitor at each location
Income -- Community income level (in thousands of dollars)
Advertising -- Local advertising budget for company at each location (in thousands of dollars)
Population -- Population size in region (in thousands)
Price -- Price company charges for car seats at each site
Shelf Location at stores -- A factor with levels Bad, Good and Medium indicating the quality of the shelving location for the car seats at each site
Age -- Average age of the local population
Education -- Education level at each location
Urban -- A factor with levels No and Yes to indicate whether the store is in an urban or rural location
US -- A factor with levels No and Yes to indicate whether the store is in the US or not

The company dataset looks like this:

Problem Statement:

A cloth manufacturing company is interested to know about the segment or attributes causes high sale.

Approach - A Random Forest can be built with target variable Sales (we will first convert it in categorical variable) & all other variable will be independent in the analysis.

In [46]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from imblearn.combine import SMOTETomek
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, classification_report
from sklearn.svm import SVC
```

In [2]:

```
data_raw = pd.read_csv('Company_Data.csv')
data_raw
```

Out[2]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	U
0	9.50	138	73	11	276	120	Bad	42	17	
1	11.22	111	48	16	260	83	Good	65	10	
2	10.06	113	35	10	269	80	Medium	59	12	
3	7.40	117	100	4	466	97	Medium	55	14	
4	4.15	141	64	3	340	128	Bad	38	13	
...	...	...	...	...	...	...	...	...	...	
395	12.57	138	108	17	203	128	Good	33	14	
396	6.14	139	23	3	37	120	Medium	55	11	
397	7.41	162	26	12	368	159	Medium	40	18	
398	5.94	100	79	7	284	95	Bad	50	12	
399	9.71	134	37	0	27	120	Good	49	16	

400 rows × 11 columns

In [3]:

```
data = data_raw.copy()
```

In [4]:

```
x = 0
for i in data_raw['Sales']:
    if i < np.percentile(data_raw['Sales'] , 25):
        data['Sales'][x] = 'Low'
    elif np.percentile(data_raw['Sales'], 25) <= i < np.percentile(data_raw['Sales'], 75):
        data['Sales'][x] = 'Average'
    else:
        data['Sales'][x] = 'High'
    x += 1
data
```

view-versus-a-copy)
  self._setitem_single_block(indexer, value, name)

Out[4]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	High	138	73	11	276	120	Bad	42	17	Yes	Yes
1	High	111	48	16	260	83	Good	65	10	Yes	Yes
2	High	113	35	10	269	80	Medium	59	12	Yes	Yes
3	Average	117	100	4	466	97	Medium	55	14	Yes	Yes
4	Low	141	64	3	340	128	Bad	38	13	Yes	No
...	...	...	...	...	...	...	...	...	...	...	...
395	High	138	108	17	203	128	Good	33	14	Yes	Yes
396	Average	139	23	3	37	120	Medium	55	11	No	Yes

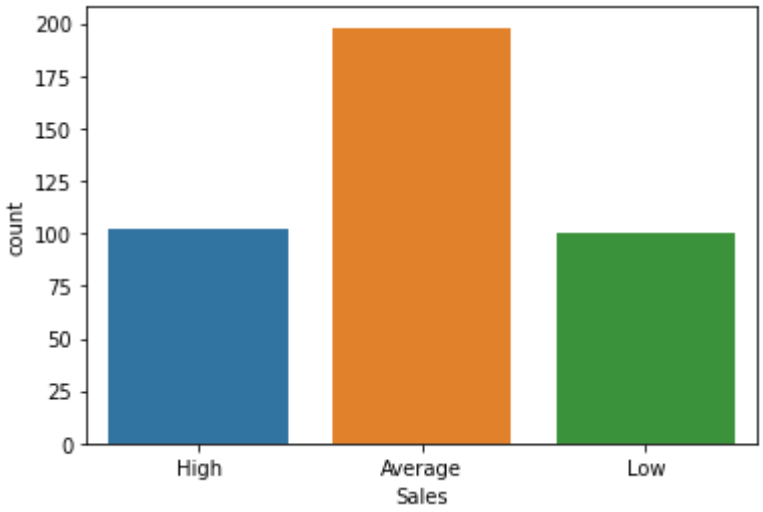
In [27]:

```
sns.countplot(data['Sales'])
```

```
C:\Users\sowmya sandeep\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misinter
pretation.
  warnings.warn(
```

Out[27]:

```
<AxesSubplot:xlabel='Sales', ylabel='count'>
```



In [28]:

```
X = data.drop('Sales', axis = 1)
y = data['Sales']
X['ShelveLoc'] = X['ShelveLoc'].map({'Bad' : 0, 'Medium' : 1, 'Good' : 3})
X['Urban'] = X['Urban'].map({'No' : 0, 'Yes' : 1})
X['US'] = X['US'].map({'No' : 0, 'Yes' : 1})
X
```

Out[28]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	U
0	138	73	11	276	120	0	42	17	1	
1	111	48	16	260	83	3	65	10	1	
2	113	35	10	269	80	1	59	12	1	
3	117	100	4	466	97	1	55	14	1	
4	141	64	3	340	128	0	38	13	1	
...	...	...	...	...	...	...	...	...	...	
395	138	108	17	203	128	3	33	14	1	
396	139	23	3	37	120	1	55	11	0	
397	162	26	12	368	159	1	40	18	1	
398	100	79	7	284	95	0	50	12	1	
399	134	37	0	27	120	3	49	16	1	

400 rows × 10 columns

Resampling the data

In [18]:

```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

In [19]:

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_st
```

In [20]:

```
kfold = KFold(10)
accuracy = []
for i in range (1,201):
    forest = RandomForestClassifier(random_state = 42, n_estimators = i)
    result = cross_val_score(forest, X, y, cv = kfold)
    accuracy.append(np.mean(result))
accuracy
```

```
 0.6149999999999999,
 0.6100000000000001,
 0.6249999999999999,
 0.6425,
 0.6475,
 0.6525000000000001,
 0.655,
 0.66,
 0.655,
 0.6475,
 0.6575,
 0.65,
 0.65,
 0.665,
 0.6675,
 0.675,
 0.665,
 0.6699999999999999,
 0.6625,
```

In [31]:

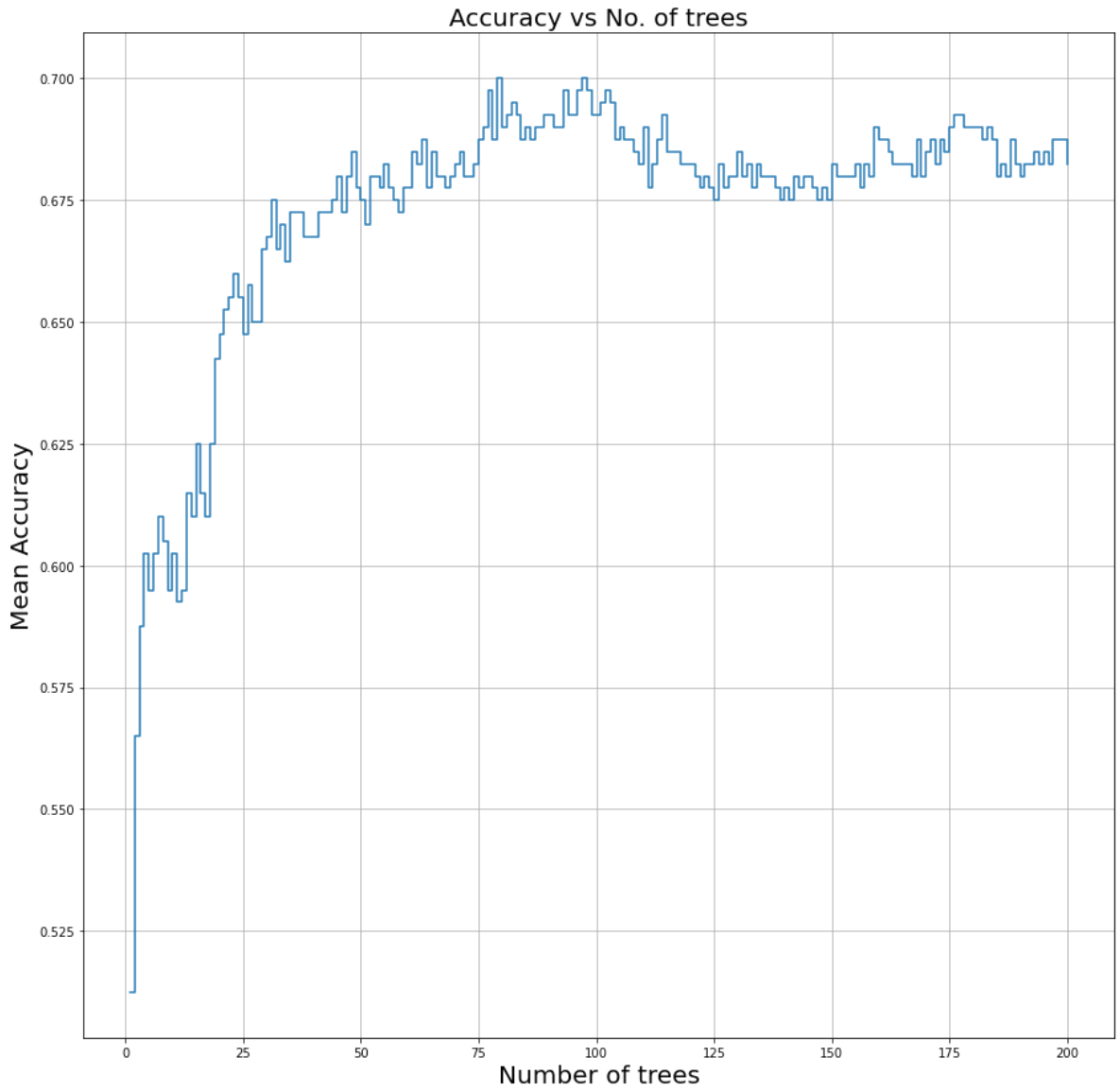
```
n_est_ideal = accuracy.index(max(accuracy))
n_est_ideal
```

Out[31]:

```
96
```

In [32]:

```
plt.figure(figsize = (15,15))
plt.plot(range(1,201),accuracy,drawstyle = 'steps-post')
plt.xlabel('Number of trees', fontsize = 20)
plt.ylabel('Mean Accuracy', fontsize = 20)
plt.title('Accuracy vs No. of trees', fontsize = 20)
plt.grid()
plt.show()
```



In [85]:

```
resample = SMOTETomek(random_state = 42)
X_res, y_res = resample.fit_resample(X, y)
(X_res.shape, y_res.shape)
```

Out[85]:

```
((538, 10), (538,))
```

In [89]:

```
X_train,X_loc_test,y_train,y_loc_test = train_test_split(X,y, train_size=0.6)
```

In [90]:

```
forest = RandomForestClassifier(n_estimators = n_est_ideal, random_state = 42)
forest.fit(X_train, y_train)
```

Out[90]:

```
RandomForestClassifier
RandomForestClassifier(n_estimators=96, random_state=42)
```

In [91]:

```
predictions = forest.predict(X_test)
np.mean(predictions == y_test)
```

Out[91]:

```
0.9375
```

In [92]:

```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

     Average       0.92      0.98      0.95        47
        High       1.00      0.83      0.91        12
         Low       0.95      0.90      0.93        21

    accuracy                           0.94        80
   macro avg       0.96      0.91      0.93        80
weighted avg       0.94      0.94      0.94        80
```

In [93]:

```
cf_mat = confusion_matrix(y_test, predictions)
cf_mat
```

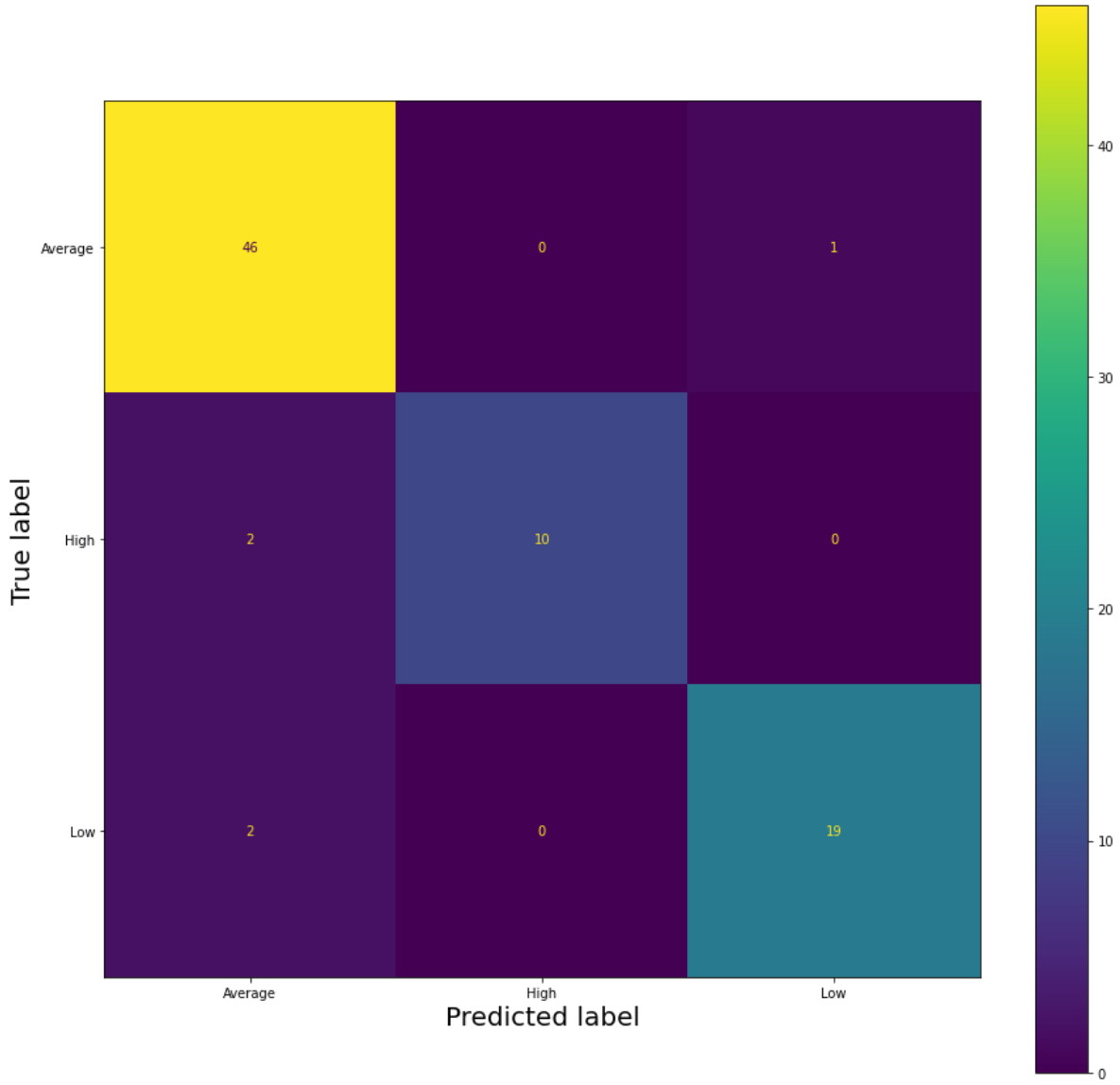
Out[93]:

```
array([[46,  0,  1],
       [ 2, 10,  0],
       [ 2,  0, 19]], dtype=int64)
```

In [94]:

```
fig, ax = plt.subplots(figsize = (15,15))
plot_confusion_matrix(forest, X_test, y_test, ax = ax)
ax.set_xlabel('Predicted label', fontsize = 20)
ax.set_ylabel('True label', fontsize = 20)
plt.show()
```

```
C:\Users\sowmya sandeep\anaconda3\lib\site-packages\sklearn\utils\deprecatio
n.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Functi
on `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2.
Use one of the class methods: ConfusionMatrixDisplay.from_predictions or Con
fusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [ ]:

In [ ]:

In [ ]: