Use Random Forest to prepare a model on fraud data treating those who have taxable_income <= 30000 as "Risky" and others are "Good"

In [1]:

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

In [2]:

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

Out[2]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
0	NO	Single	68833	50047	10	YES
1	YES	Divorced	33700	134075	18	YES
2	NO	Married	36925	160205	30	YES
3	YES	Single	50190	193264	15	YES
4	NO	Married	81002	27533	28	NO
...	...	...	...	...	...	...
595	YES	Divorced	76340	39492	7	YES
596	YES	Divorced	69967	55369	2	YES
597	NO	Divorced	47334	154058	0	YES
598	YES	Married	98592	180083	17	NO
599	NO	Divorced	96519	158137	16	NO

600 rows × 6 columns

In [3]:

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

In [4]:

```
x = 0
for i in data_raw['Taxable.Income']:
    if i <= 30000:
        data['Taxable.Income'][x] = 'Risky'
    else:
        data['Taxable.Income'][x] = 'Good'
    x += 1
data
```

```
C:\Users\SOWMYA~1\AppData\Local\Temp/ipykernel_32624/2802370999.py:6: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  data['Taxable.Income'][x] = 'Good'
C:\Users\sowmya sandeep\anaconda3\lib\site-packages\pandas\core\indexing.py:
1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  self._setitem_single_block(indexer, value, name)
```

Out[4]:

	Undergrad	Marital.Status	Taxable.Income	City.Population	Work.Experience	Urban
0	NO	Single	Good	50047	10	YES
1	YES	Divorced	Good	134075	18	YES
2	NO	Married	Good	160205	30	YES
3	YES	Single	Good	193264	15	YES
4	NO	Married	Good	27533	28	NO
...	...	...	...	...	...	...
595	YES	Divorced	Good	39492	7	YES
596	YES	Divorced	Good	55369	2	YES
597	NO	Divorced	Good	154058	0	YES
598	YES	Married	Good	180083	17	NO
599	NO	Divorced	Good	158137	16	NO

600 rows × 6 columns

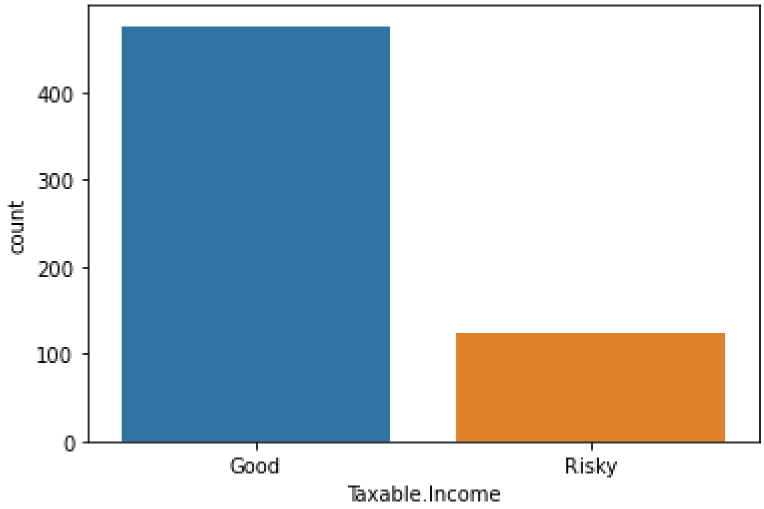
In [5]:

```
sns.countplot(data['Taxable.Income'])
```

```
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[5]:

```
<AxesSubplot:xlabel='Taxable.Income', ylabel='count'>
```



In [6]:

```
y = data['Taxable.Income']
X = data.drop('Taxable.Income', axis = 1)
X['Undergrad'] = X['Undergrad'].map({'NO' : 0, 'YES' : 1})
X['Marital.Status'] = X['Marital.Status'].map({'Single' : 0, 'Married' : 1, 'Divorced' : 2}
X['Urban'] = X['Urban'].map({'NO' : 0, 'YES' : 1})
X
```

Out[6]:

	Undergrad	Marital.Status	City.Population	Work.Experience	Urban
0	0	0	50047	10	1
1	1	2	134075	18	1
2	0	1	160205	30	1
3	1	0	193264	15	1
4	0	1	27533	28	0
...	...	...	...	...	...
595	1	2	39492	7	1
596	1	2	55369	2	1
597	0	2	154058	0	1
598	1	1	180083	17	0
599	0	2	158137	16	0

600 rows × 5 columns

Resampling the data

In [7]:

```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_st
```

In [8]:

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

Out[8]:

```
[0.6233333333333333,
 0.74,
 0.685,
 0.73,
 0.7016666666666667,
 0.74,
 0.7150000000000001,
 0.7383333333333333,
 0.725,
 0.7433333333333333,
 0.7283333333333333,
 0.7416666666666668,
 0.7299999999999999,
 0.7333333333333333,
 0.7283333333333333,
 0.7383333333333333,
 0.7316666666666667,
 0.74,
```

In [9]:

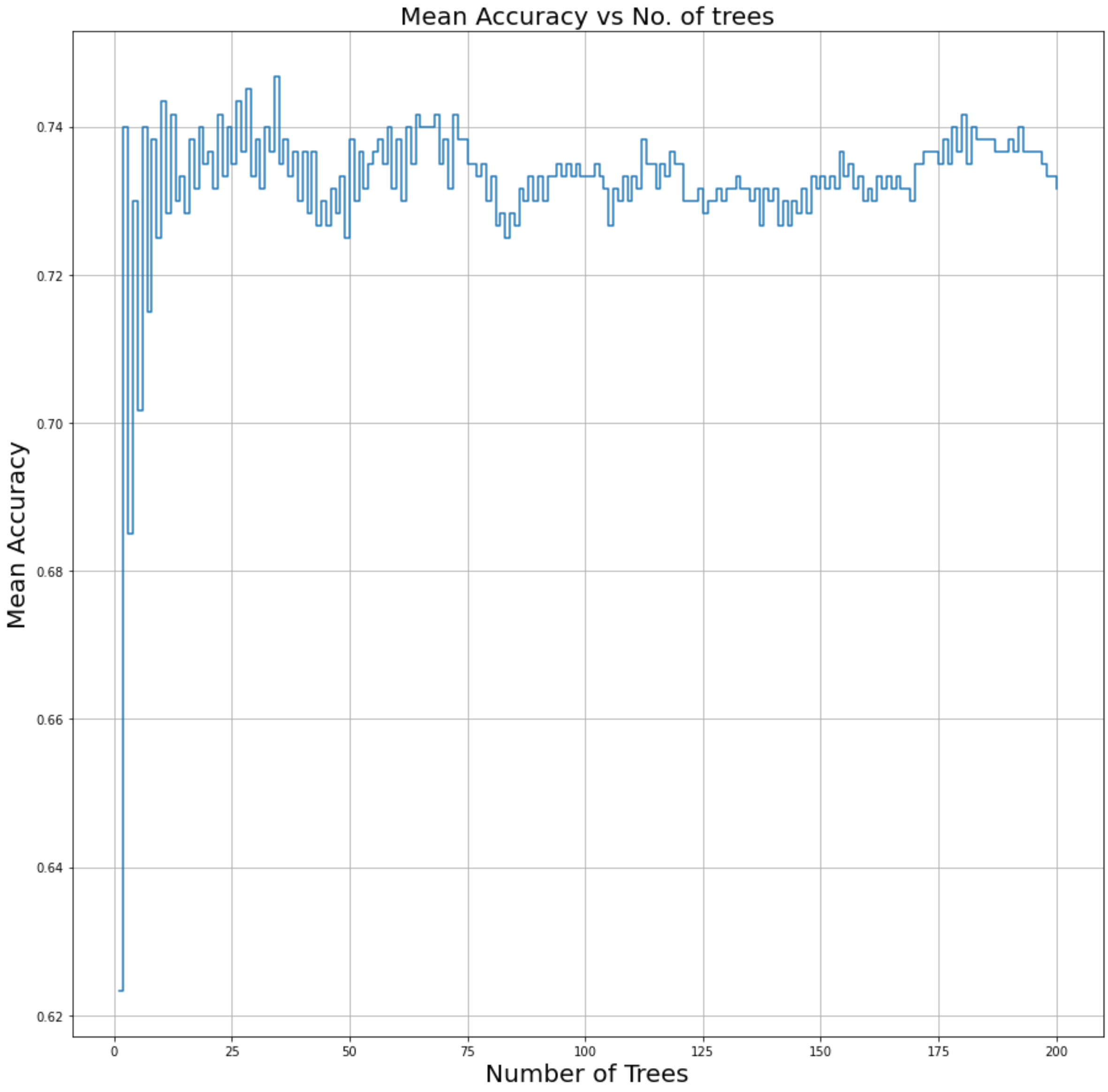
```
n_est_ideal = accuracy.index(max(accuracy[2:]))
n_est_ideal
```

Out[9]:

```
33
```

In [10]:

```
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('Mean Accuracy vs No. of trees', fontsize = 20)
plt.grid()
```



In [33]:

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

Out[33]:

```
((730, 5), (730,))
```

In [46]:

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

In [47]:

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

Out[47]:

```
▾                 RandomForestClassifier

RandomForestClassifier(n_estimators=33, random_state=42)
```

In [48]:

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

Out[48]:

```
0.8583333333333333
```

In [49]:

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

```
              precision    recall  f1-score   support

        Good       0.87      0.97      0.92        97
       Risky       0.75      0.39      0.51        23

    accuracy                           0.86       120
   macro avg       0.81      0.68      0.72       120
weighted avg       0.85      0.86      0.84       120
```

In [50]:

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

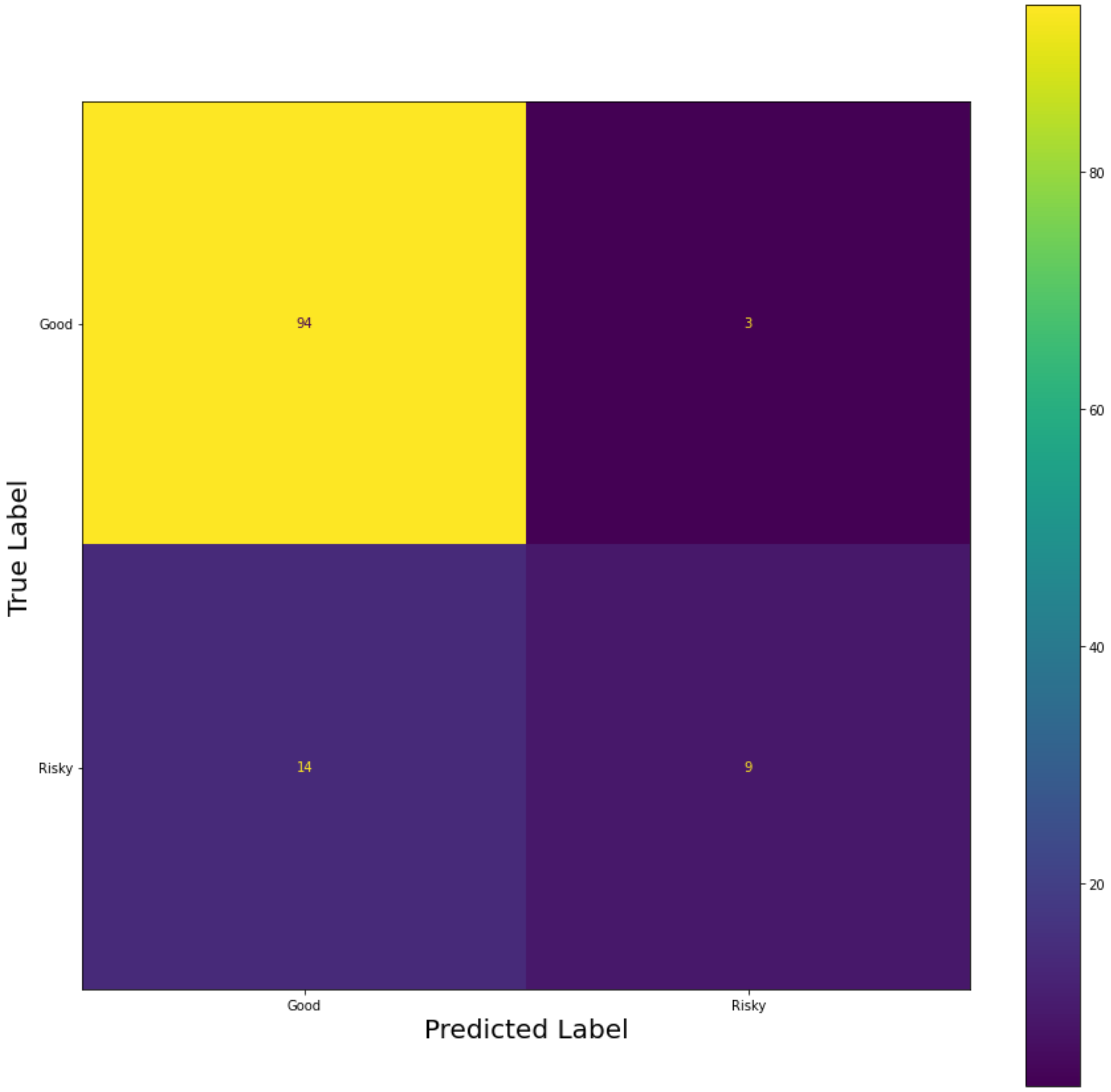
Out[50]:

```
array([[94,  3],
       [14,  9]], dtype=int64)
```

In [51]:

```
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 [ ]:

In [ ]: