For the second model I wanted to use decision trees because I thought it would be interesting to see the tree model visualized and possibly gain some additional insight from that. I used the same dataset created previously during EDA and also used the same target and predictors.

In [2]:
```
import pandas as pd
pd.pandas.set_option('display.max_columns', None)
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
```

In [4]:
```
df1 = pd.read_csv('EDA_df.csv')
df1.head()
```

Out[4]:

	Unnamed: 0	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	Reported Time	Arrest Flag	Frisk Flag
0	0	Unknown	Arrest	None	7500	1980s	M	Black or African American	Asian	Male	2015-10-16T00:00:00	11:32:00	N	N
1	1	Unknown	Field Contact	None	5670	1960s	M	White	Unknown	Unknown	2015-03-19T00:00:00	07:59:00	N	N
2	3	Unknown	Field Contact	None	7539	1960s	M	White	Unknown	Unknown	2015-04-01T00:00:00	04:55:00	N	N
3	4	Unknown	Field Contact	None	6973	1970s	M	White	Black or African American	Male	2015-04-03T00:00:00	00:41:00	N	N
4	5	Unknown	Field Contact	None	7402	1970s	M	White	Black or African American	Male	2015-04-05T00:00:00	23:46:00	N	N

After splitting the data for the target and predictors, I had to make some adjustments again to the N/Y values and replace it with 0 and 1 for better calculations. Dummy variables/one-hot-encoding was next completed and then the data was reasy to do a train-test-split.

In [5]:
```
target = df1['Arrest Flag']
predictors = df1.drop(columns = ['Arrest Flag'], axis = 1)
```

In [6]:
```
target.replace({"N": 0, "Y": 1}, inplace=True)
predictors['Frisk Flag'].replace({"N": 0, "Y": 1}, inplace=True)
```

In [7]:
```
dummy_predictors = pd.get_dummies(predictors, drop_first=False)
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(dummy_predictors, target, random_state=11)
```

Fitting my data to the decision tree classifier from scikit-learn.

In [9]:
```
classifier = DecisionTreeClassifier(random_state=11)
classifier.fit(X_train, y_train)
```

Out[9]:
```
DecisionTreeClassifier(random_state=11)
```

In [10]:
```
y_pred_test = classifier.predict(X_test)
y_pred_train = classifier.predict(X_train)
```

To evaluate the model, I brought over the function I made during Logistic Regression to calculate Precision, Recall, Accuracy, and the F1 score of the train and test groups. I also calculate the AUC (Area Under Curve).

In [12]:
```
def model_eval(YTrain, YTest, YHat_Train, YHat_Test):
    PrecisionTrain = precision_score(YTrain, YHat_Train)
    PrecisionTest = precision_score(YTest, YHat_Test)
    RecallTrain = recall_score(YTrain, YHat_Train)
    RecallTest = recall_score(YTest, YHat_Test)
    AccuracyTrain = accuracy_score(YTrain, YHat_Train)
    AccuracyTest = accuracy_score(YTest, YHat_Test)
    F1Train = f1_score(YTrain, YHat_Train)
    F1Test = f1_score(YTest, YHat_Test)
    print('Precision Score:\nTrain: {}  Test: {}\nRecall Score:\nTrain: {}  Test {}\nAccuracy Score:\nTrain: {}  Test: {}\nF1 Score:\nTrain: {}
                                                PrecisionTrain, PrecisionTest, RecallTrain, RecallTest,
                                                AccuracyTrain, AccuracyTest, F1Train, F1Test))
```

In [13]:
```
model_eval(y_train, y_test, y_pred_train, y_pred_test)
```

```
Precision Score:
Train: 1.0  Test: 0.9969183359013868
Recall Score:
Train: 1.0  Test 0.9892966360856269
Accuracy Score:
Train: 1.0  Test: 0.9991973602069028
F1 Score:
Train: 1.0  Test: 0.9930928626247123
```

In [11]:
```
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_test)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('AUC is :{0}'.format(round(roc_auc, 2)))
```

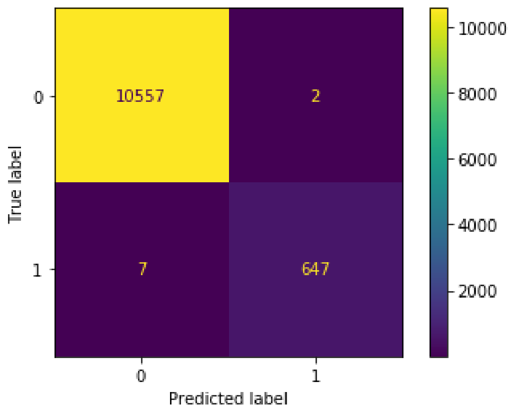
```
AUC is :0.99
```

It appears that this model performed very well since the test scores are all above 98%, as opposed to the logistic regression model where only recall and accuracy had high scores and precision and F1 were much lower. While I am skeptical of such good results, the confusion matrix shows the overwelming amount of true positives.

In [15]:
```
cnf_matrix = confusion_matrix(y_test, y_pred_test)
cnf_matrix
```

Out[15]:
```
array([[10557,     2],
       [    7,   647]], dtype=int64)
```

In [17]:
```
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```



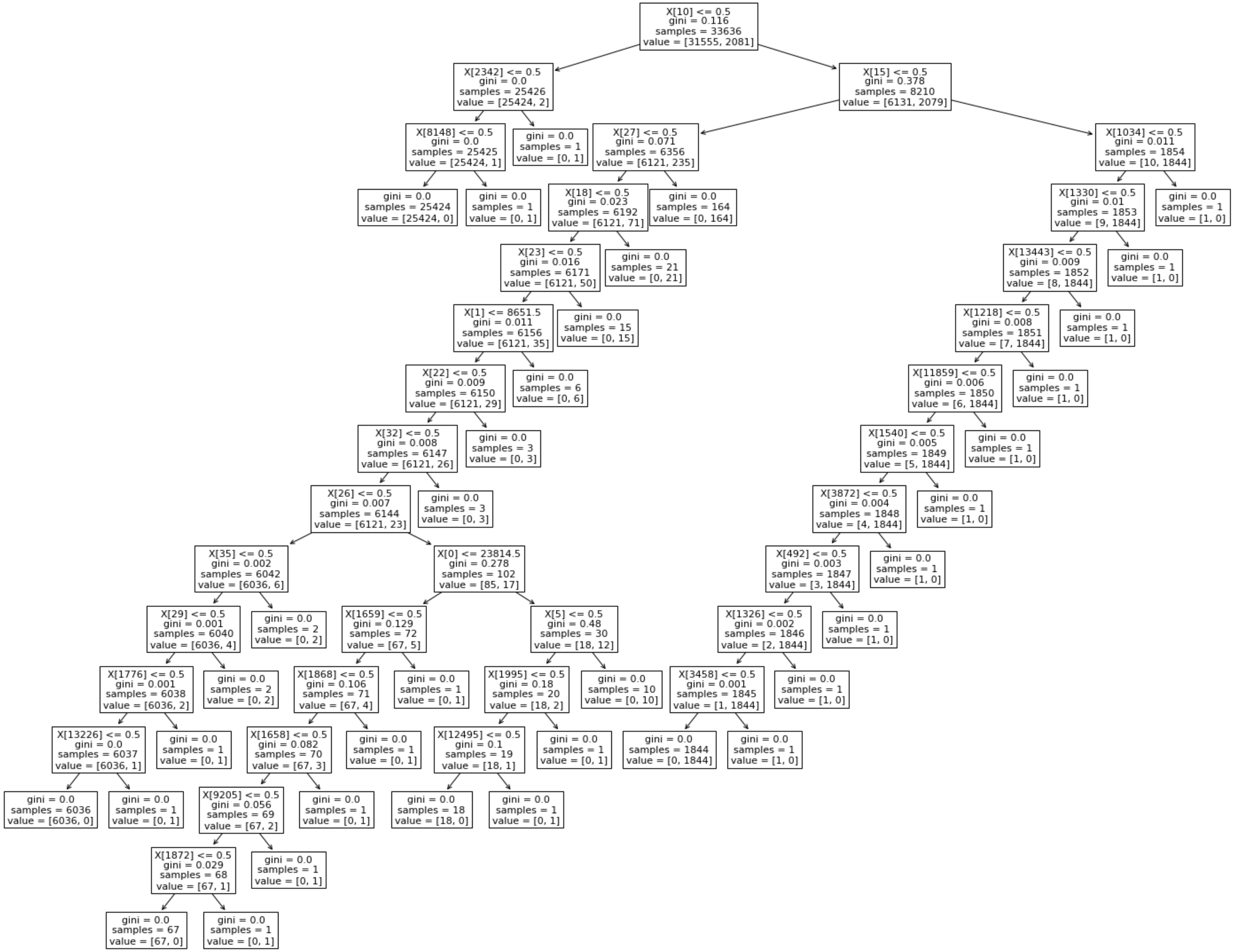
Lastly I wanted a visual representation of the tree that was created. There is both a text version (for easy loading and running) and the bigger visualizations that show each leaf node more easily.

In [18]:
```
text_representation = tree.export_text(classifier)
print(text_representation)
```

```
|--- feature_10 <= 0.50
|   |--- feature_2342 <= 0.50
|   |   |--- feature_8148 <= 0.50
|   |   |   |--- class: 0
|   |   |--- feature_8148 >  0.50
|   |   |   |--- class: 1
|   |--- feature_2342 >  0.50
|   |   |--- class: 1
|--- feature_10 >  0.50
|   |--- feature_15 <= 0.50
|   |   |--- feature_27 <= 0.50
|   |   |   |--- feature_18 <= 0.50
|   |   |   |   |--- feature_23 <= 0.50
|   |   |   |   |   |--- feature_1 <= 8651.50
|   |   |   |   |   |   |--- feature_22 <= 0.50
|   |   |   |   |   |   |   |--- feature_32 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_26 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_35 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- feature_29 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 3
|   |   |   |   |   |   |   |   |   |   |--- feature_29 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_35 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- feature_26 >  0.50
|   |   |   |   |   |   |   |   |   |--- feature_0 <= 23814.50
|   |   |   |   |   |   |   |   |   |   |--- feature_1659 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 5
|   |   |   |   |   |   |   |   |   |   |--- feature_1659 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_0 >  23814.50
|   |   |   |   |   |   |   |   |   |   |--- feature_5 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 3
|   |   |   |   |   |   |   |   |   |   |--- feature_5 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_32 >  0.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- feature_22 >  0.50
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_1 >  8651.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_23 >  0.50
|   |   |   |   |   |--- class: 1
|   |   |   |--- feature_18 >  0.50
|   |   |   |   |--- class: 1
|   |   |--- feature_27 >  0.50
|   |   |   |--- class: 1
|   |--- feature_15 >  0.50
|   |   |--- feature_1034 <= 0.50
|   |   |   |--- feature_1330 <= 0.50
```

```
|   |   |   |   |--- feature_13443 <= 0.50
|   |   |   |   |   |--- feature_1218 <= 0.50
|   |   |   |   |   |   |--- feature_11859 <= 0.50
|   |   |   |   |   |   |   |--- feature_1540 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_3872 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_492 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- feature_1326 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- feature_1326 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- feature_492 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- feature_3872 >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- feature_1540 >  0.50
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_11859 >  0.50
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_1218 >  0.50
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_13443 >  0.50
|   |   |   |   |   |--- class: 0
|   |   |   |--- feature_1330 >  0.50
|   |   |   |   |--- class: 0
|   |   |--- feature_1034 >  0.50
|   |   |   |--- class: 0
```

In [37]:
```
plt.figure(figsize=(25,20))
tree.plot_tree(classifier, fontsize=11);
plt.show()
```



While the results were very good, this brought up concerns of overfitting since I am still skeptical of the results.