Homework 3

Part 1: Imbalanced Dataset

In this homework, you will be working with an imbalanced Dataset. The dataset is Credit Card Fraud Detection dataset which was hosted on Kaggle. The aim is to detect fraudlent transactions.

Instructions

Please push the .ipynb, .py, and .pdf to Github Classroom prior to the deadline. Please include your UNI as well.

Setup

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

In [2]:
```
# Feel free to import any other packages you'd like to
```

Data Preprocessing and Exploration

Download the Kaggle Credit Card Fraud data set. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

In [3]:
```
raw_df = pd.read_csv('https://storage.googleapis.com/download.tensorflow.org/dat
raw_df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

1.1 Examining the class Imbalance

1.1.1 How many observations are in this dataset? How many are positive and negative?

(Note: Positive labels are labeled as 1)

In [4]:
```
# Your Code here
raw_df.shape[0]
```

Out[4]:
```
284807
```

In [5]:
```
raw_df['Class'].value_counts()
```

Out[5]:
```
0    284315
1       492
Name: Class, dtype: int64
```

Ans: there are 284807 observations, while 492 are positive, and 284315 are negative

1.2 Cleaning and normalizing the data

The raw data has a few issues. We are not sure what the time column actually means so drop the Time column. The Amount column also has a wide range of values covered so we take the log of the Amount column to reduce its range.

In [6]:
```
cleaned_df = raw_df.copy()

# You don't want the `Time` column.
cleaned_df.pop('Time')

# The `Amount` column covers a huge range. Convert to log-space.
eps = 0.001 # 0 => 0.1¢
cleaned_df['Log Ammount'] = np.log(cleaned_df.pop('Amount')+eps)
```

In [7]:
```
cleaned_df
```

Out[7]:

	V1	V2	V3	V4	V5	V6	V7	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053
...	...	...	...	...	...	...	...	.
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.30533
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.29486
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.70841
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.67914

	V1	V2	V3	V4	V5	V6	V7	V
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.41465

284807 rows × 30 columns

In [8]:
```
cleaned_df["Class"].value_counts(normalize=True)
```

Out[8]:
```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

1.2.1 Split the dataset into development and test sets. Please set test size as 0.2 and random state as 42. Print the shape of your development and test features

In [9]:
```
# Your Code Here
X = cleaned_df.drop("Class", axis = 1)
y = cleaned_df["Class"]
```

In [10]:
```
from sklearn.model_selection import train_test_split
X_dev, X_test, y_dev, y_test = train_test_split(X, y==1, stratify = y, test_size
print(f"X_dev shape: {X_dev.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_dev shape: {y_dev.shape}")
print(f"y_test shape: { y_test.shape}")
```

```
X_dev shape: (227845, 29)
X_test shape: (56962, 29)
y_dev shape: (227845,)
y_test shape: (56962,)
```

1.2.2 Normalize the features using Standard Scaler from Sklearn.

In [11]:
```
# Your Code Here
from sklearn.preprocessing import StandardScaler

scaler1 = StandardScaler()
X_test = scaler1.fit_transform(X_test)
X_dev = scaler1.fit_transform(X_dev)
```

1.3 Defining Model and Performance Metrics

1.3.1 First, let us fit a default Decision tree classifier. (use max_depth=10 and random_state=42). Print the AUC and Average Precision values of 5 Fold Cross Validation

In [12]:
```
# Your Code here
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_validate
tree1 = DecisionTreeClassifier(max_depth = 10, random_state = 42)
score1 = cross_validate(tree1, X_dev, y_dev, cv = 5, scoring = ['roc_auc', 'aver
```

In [13]:
```
print(score1["test_roc_auc"])
```

```
print(score1["test_average_precision"])
```

```
[0.88756328 0.88400873 0.81260522 0.77454358 0.82104149]
[0.62653551 0.71014615 0.60399496 0.58692296 0.68112078]
```

In [14]:

```
print(f'mean AUC: {score1["test_roc_auc"].mean()}')
print(f'mean AP: {score1["test_average_precision"].mean()}')
```

```
mean AUC: 0.8359524571279693
mean AP: 0.641744070059615
```

1.3.2 Perform random oversampling on the development dataset.

- What many positive and negative labels do you observe after random oversampling?
- What is the shape of your development dataset?

(Note: Set random state as 42 when performing oversampling)

In [15]:

```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in /Users/clarencestudy/opt/anaconda3/li
b/python3.8/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /Users/clarencestudy/opt/anac
onda3/lib/python3.8/site-packages (from imblearn) (0.9.1)
Requirement already satisfied: numpy>=1.17.3 in /Users/clarencestudy/opt/anacond
a3/lib/python3.8/site-packages (from imbalanced-learn->imblearn) (1.21.4)
Requirement already satisfied: scipy>=1.3.2 in /Users/clarencestudy/opt/anaconda
3/lib/python3.8/site-packages (from imbalanced-learn->imblearn) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/clarencestudy/opt/
anaconda3/lib/python3.8/site-packages (from imbalanced-learn->imblearn) (3.0.0)
Requirement already satisfied: scikit-learn>=1.1.0 in /Users/clarencestudy/opt/a
naconda3/lib/python3.8/site-packages (from imbalanced-learn->imblearn) (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in /Users/clarencestudy/opt/anacond
a3/lib/python3.8/site-packages (from imbalanced-learn->imblearn) (1.1.0)
```

In [16]:

```
# Your Code here
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state = 42)
X_dev_oversample1, y_dev_oversample1 = ros.fit_resample(X_dev, y_dev)
print(y_dev_oversample1.value_counts())
print(X_dev_oversample1.shape)
```

```
False    227451
True     227451
Name: Class, dtype: int64
(454902, 29)
```

Ans: Both positive and negative labels have 227451 samples, and the shape is about (454902, 29)

1.3.3 Repeat 1.3.1 using the dataset you created in the above step(1.3.2 Random oversampling). (Make sure you use the same hyperparameters as 1.3.1. i.e., max_depth=10 and random_state=42. This will help us to compare the models)

In [17]:

```
tree2 = DecisionTreeClassifier(max_depth = 10, random_state = 42)
```

In [18]:
```
score22 = cross_validate(tree2, X_dev_oversample1, y_dev_oversample1, cv = 5, sc
```

In [19]:
```
print(score22["test_roc_auc"])
print(score22["test_average_precision"])
```

```
[0.99886808 0.99929956 0.99900977 0.99926441 0.99932258]
[0.99816396 0.99884446 0.99826911 0.9986672  0.99874564]
```

In [20]:
```
print(f'mean AUC: {score22["test_roc_auc"].mean()}')
print(f'mean AP: {score22["test_average_precision"].mean()}')
```

```
mean AUC: 0.9991528810903473
mean AP: 0.9985380738652964
```

1.3.4 Perform Random undersampling on the development dataset.

- What many positive and negative labels do you observe after random undersampling?
- What is the shape of your development dataset?

(Note: Set random state as 42 when performing undersampling)

In [21]:
```
# Your Code here
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(replacement = False, random_state = 42)
X_dev_subsample1, y_dev_subsample1 = rus.fit_resample(X_dev, y_dev)
print(y_dev_subsample1.value_counts())
print(X_dev_subsample1.shape)
```

```
False    394
True     394
Name: Class, dtype: int64
(788, 29)
```

Ans: Both positive and negative labels have 394 samples, and the shape is about (788, 29)

1.3.5 Repeat 1.3.1 using the dataset you created in the above step(1.3.4 Random undersampling). (Make sure you use the same hyperparameters as 1.3.1. i,e., max_depth=10 and random_state=42. This will help us to compare the models)

In [22]:
```
tree3 = DecisionTreeClassifier(max_depth = 10, random_state = 42)
```

In [23]:
```
# tree3 = DecisionTreeClassifier(max_depth = 10, random_state = 42)

score33 = cross_validate(tree3, X_dev_subsample1, y_dev_subsample1, cv = 5, scor
```

In [24]:
```
print(score33["test_roc_auc"])
print(score33["test_average_precision"])
```

```
[0.93662875 0.94936709 0.9097901  0.92583577 0.92989289]
[0.91336577 0.92661432 0.86678359 0.90629539 0.89830283]
```

In [25]:
```
print(f'mean AUC: {score33["test_roc_auc"].mean()}')
print(f'mean AP: {score33["test_average_precision"].mean()}')
```

```
mean AUC: 0.9303029182535673
mean AP: 0.9022723811037444
```

1.3.6 Perform Synthetic Minority Oversampling Technique(SMOTE) on the development dataset

- What many positive and negative labels do you observe after performing SMOTE?
- What is the shape of your development dataset? (Note: Set random state as 42 when performing SMOTE)

In [26]:
```
# Your code here
from imblearn.over_sampling import SMOTE
smote1 = SMOTE(random_state = 42)
X_dev_smote1, y_dev_smote1 = smote1.fit_resample(X_dev, y_dev)
print(y_dev_smote1.value_counts())
print(X_dev_smote1.shape)
```

```
False    227451
True     227451
Name: Class, dtype: int64
(454902, 29)
```

Ans: Both positive and negative labels have 227451 samples, and the shape is about (454902, 29)

1.3.7 Repeat 1.3.1 using the dataset you created in the above step(1.3.6 SMOTE). (Make sure you use the same hyperparameters as 1.3.1. i.e., max_depth=10 and random_state=42. This will help us to compare the models)

In [27]:
```
tree4 = DecisionTreeClassifier(max_depth = 10, random_state = 42)
```

In [28]:
```
score44 = cross_validate(tree4, X_dev_smote1, y_dev_smote1, cv = 5, scoring = ['
```

In [29]:
```
print(score44["test_roc_auc"])
print(score44["test_average_precision"])
```

```
[0.99757923 0.99738426 0.99772842 0.9972463  0.99714931]
[0.99667622 0.99645425 0.99685462 0.99619248 0.99611009]
```

In [30]:
```
print(f'mean AUC: {score44["test_roc_auc"].mean()}')
print(f'mean AP: {score44["test_average_precision"].mean()}')
```

```
mean AUC: 0.9974175030591891
mean AP: 0.9964575330041552
```

1.3.8 Make predictions on the test set using the four models that you built and report their AUC values.

In [31]:
```
# Your Code here
from sklearn.metrics import roc_auc_score
```

```
tree1.fit(X_dev, y_dev)
score_normal = tree1.predict_proba(X_test)[:, 1]
print(f"The normal model has an AUC of {roc_auc_score(y_test, score_normal)}")

tree2.fit(X_dev_oversample1, y_dev_oversample1)
score_over = tree2.predict_proba(X_test)[:, 1]
print(f"The oversampling model has an AUC of {roc_auc_score(y_test, score_over)}


tree3.fit(X_dev_subsample1, y_dev_subsample1)
score_under = tree3.predict_proba(X_test)[:, 1]
print(f"The undersampling model has an AUC of {roc_auc_score(y_test, score_under


tree4.fit(X_dev_smote1, y_dev_smote1)
score_smote = tree4.predict_proba(X_test)[:, 1]
print(f"The smote model has an AUC of {roc_auc_score(y_test, score_smote)}")
```

```
The normal model has an AUC of 0.819676270198569
The oversampling model has an AUC of 0.9017043350120015
The undersampling model has an AUC of 0.896389918516647
The smote model has an AUC of 0.8753799792989789
```

In [ ]:

```

```

1.3.9 Plot Confusion Matrices for all the four models on the test set. Comment your results

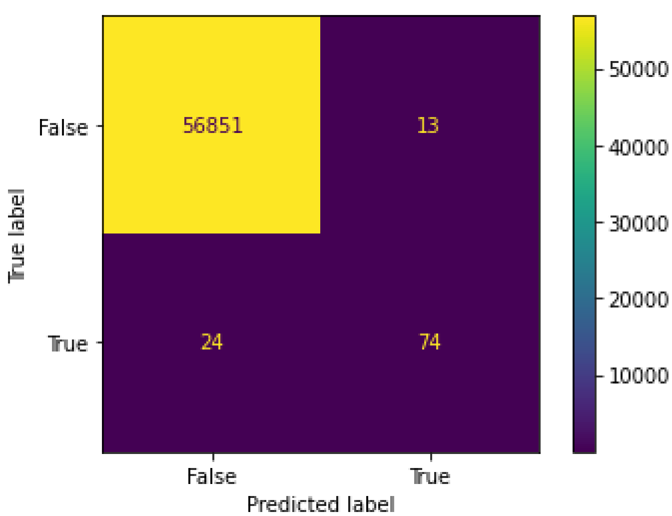
In [32]:

```
# Your Code here
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(tree1, X_test, y_test)
```

```
/Users/clarencestudy/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/dep
recation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Fun
ction `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. U
se one of the class methods: ConfusionMatrixDisplay.from_predictions or Confusio
nMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[32]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdb6032c250
>
```

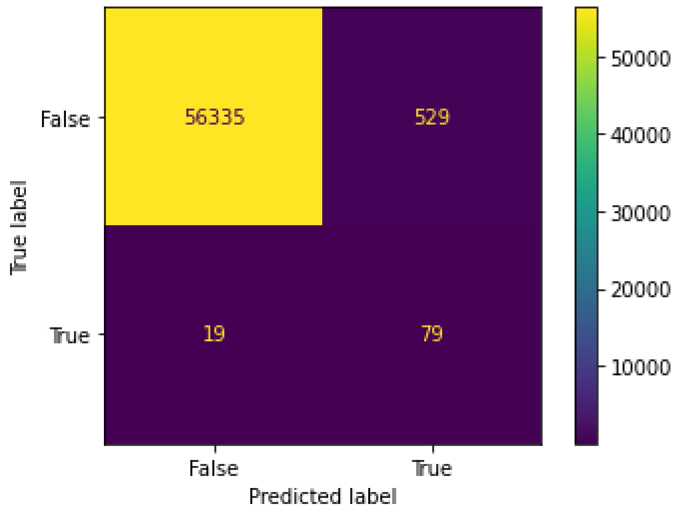


In [34]:

```
plot_confusion_matrix(tree2, X_test, y_test)
```

```
/Users/clarencestudy/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/dep
recation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Fun
ction `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. U
se one of the class methods: ConfusionMatrixDisplay.from_predictions or Confusio
nMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

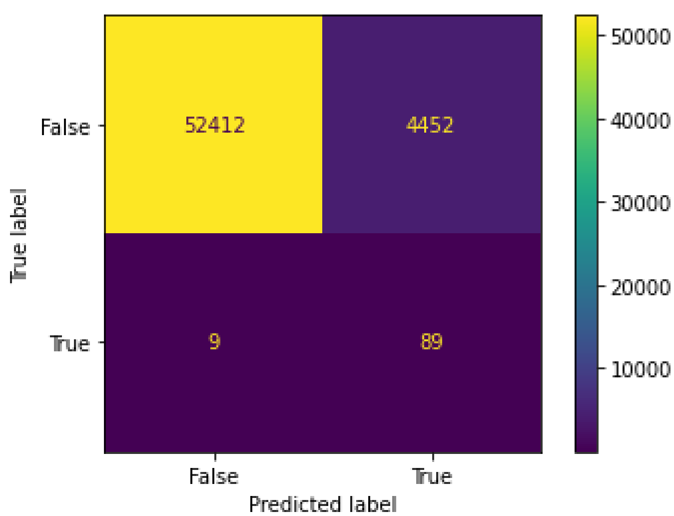
Out[34]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdb8329ccd0
>
```



In [35]:
```
plot_confusion_matrix(tree3, X_test, y_test)
```

```
/Users/clarencestudy/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/dep
recation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Fun
ction `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. U
se one of the class methods: ConfusionMatrixDisplay.from_predictions or Confusio
nMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[35]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdb8329cd30
>
```



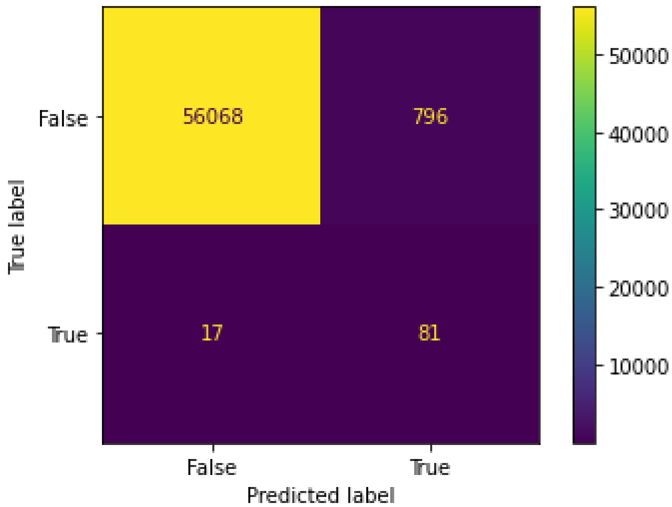
In [36]:
```
plot_confusion_matrix(tree4, X_test, y_test)
```

```
/Users/clarencestudy/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/dep
recation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Fun
ction `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. U
se one of the class methods: ConfusionMatrixDisplay.from_predictions or Confusio
```

```
nMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[36]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdb60492df0
>
```



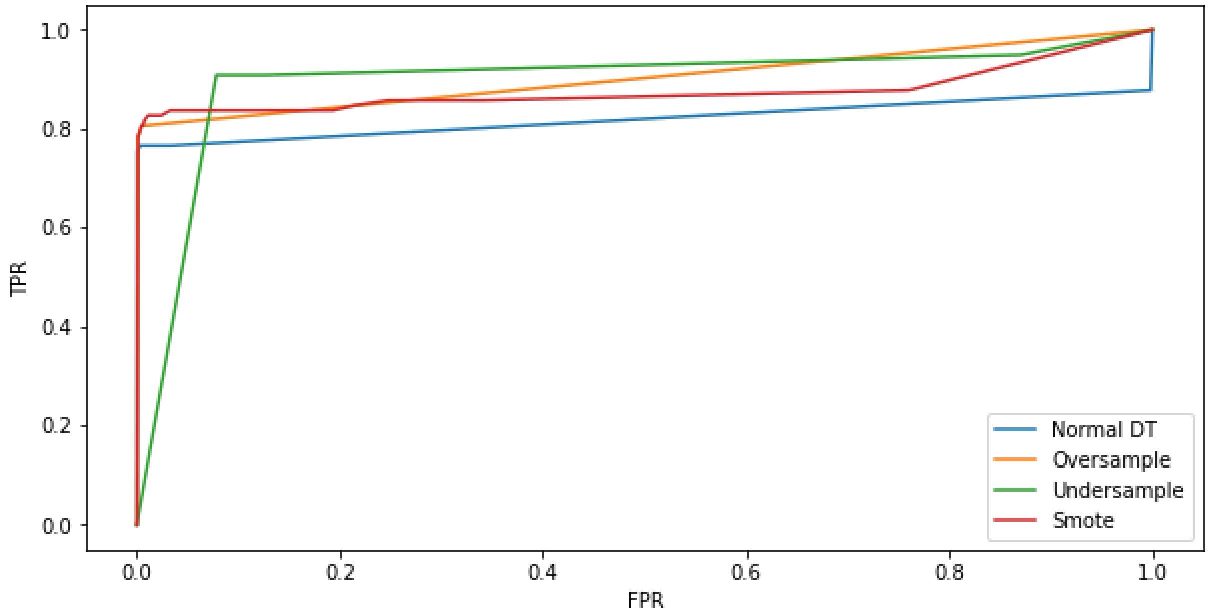
Comment: the normal decision tree classifier has the highest accuracy, while the undersampling tree has the lowest accuracy. The undersampling tree has the best recall, while the normal decision tree has lowest recall. The normal decision tree has the highest precision, while the undersampling has the lowest precision.

1.3.10 Plot ROC for all the four models on the test set in a single plot. Make sure you label axes and legend properly. Comment your results

In [37]:

```
# Your code
from sklearn.metrics import roc_curve
normal_fpr, normal_tpr, normal_thr = roc_curve(y_test, score_normal, pos_label =
over_fpr, over_tpr, over_thr = roc_curve(y_test, score_over, pos_label = 1)
under_fpr, under_tpr, under_thr = roc_curve(y_test, score_under, pos_label = 1)
smote_fpr, smote_tpr, smote_thr = roc_curve(y_test, score_smote, pos_label = 1)

plt.figure(figsize = (10,5))
plt.plot(normal_fpr, normal_tpr, label = "Normal DT")
plt.plot(over_fpr, over_tpr, label = "Oversample")
plt.plot(under_fpr, under_tpr, label = "Undersample")
plt.plot(smote_fpr, smote_tpr, label = "Smote")
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



ANS: The normal decision tree has the worst AUC, and undersampling has the best AUC, which is relatively higher than oversampling and smote. While smote and oversampling have similar AUC

1.3.11 Train a balanced default Decision tree classifier, using max_depth = 10 and random_state = 42 (balance the class weights). Print the AUC and average precision on dev set

In [38]:

```
# Your code here
tree5 = DecisionTreeClassifier(max_depth = 10, random_state = 42, class_weight='
score_balanced_default = cross_validate(tree5, X_dev, y_dev, scoring = ['roc_auc
print(score_balanced_default["test_roc_auc"])
print(score_balanced_default["test_average_precision"])
```

```
[0.88026358 0.95432645 0.90415573 0.91647782 0.87235086]
[0.51935587 0.573068   0.50745277 0.56269097 0.4556111 ]
```

In [39]:

```
print(f'Mean AUC: {score_balanced_default["test_roc_auc"].mean()}')
print(f'Mean AP: {score_balanced_default["test_average_precision"].mean()}')
```

```
Mean AUC: 0.905514887383234
Mean AP: 0.5236357414975382
```

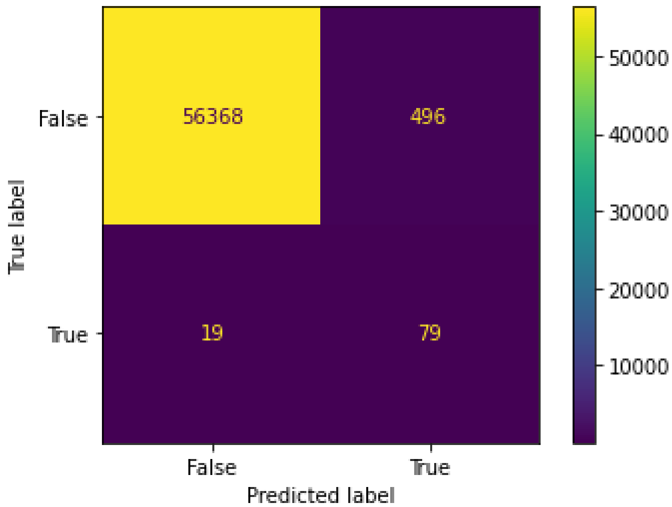
1.3.12 Plot confusion matrix on test set using the above model and comment on your results

In [40]:

```
# Your code here
tree5.fit(X_dev, y_dev)
plot_confusion_matrix(tree5, X_test, y_test)
```

```
/Users/clarencestudy/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/dep
recation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Fun
ction `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. U
se one of the class methods: ConfusionMatrixDisplay.from_predictions or Confusio
nMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[40]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdb500d75b0
>



ANS:It has an accuracy similar to oversampling decision tree, which is only slightly less than the normal decision tree. Its recall is also similar to the oversampling decision tree, which is relatively less than other models. Its precision is also similar to the oversampling decision tree, which is the only slightly less than the normal DT

In [ ]: