Credit Card Fraud Detection

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. I decided to proceed to an undersampling strategy to re-balance the class.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, details about the original features and more background information is not available.

In [8]:

```
#Packages import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [9]:

```
# Load the dataset    EDA
credit_card_df = pd.read_csv("/Users/aswathchary/Desktop/Northeastern Univer
print("Total number of records: "+str(credit_card_df.shape[0]))
print("Total number of columns: "+str(credit_card_df.shape[1]))
credit_card_df.head(10)
```

```
Total number of records: 284807
Total number of columns: 31
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.00000	-1.35981	-0.07278	2.53635	1.37816	-0.33832	0.46239	0.23960	0.09870
1	0.00000	1.19186	0.26615	0.16648	0.44815	0.06002	-0.08236	-0.07880	0.08510
2	1.00000	-1.35835	-1.34016	1.77321	0.37978	-0.50320	1.80050	0.79146	0.24768
3	1.00000	-0.96627	-0.18523	1.79299	-0.86329	-0.01031	1.24720	0.23761	0.37744
4	2.00000	-1.15823	0.87774	1.54872	0.40303	-0.40719	0.09592	0.59294	-0.27053
5	2.00000	-0.42597	0.96052	1.14111	-0.16825	0.42099	-0.02973	0.47620	0.26031
6	4.00000	1.22966	0.14100	0.04537	1.20261	0.19188	0.27271	-0.00516	0.08121
7	7.00000	-0.64427	1.41796	1.07438	-0.49220	0.94893	0.42812	1.12063	-3.80786
8	7.00000	-0.89429	0.28616	-0.11319	-0.27153	2.66960	3.72182	0.37015	0.85108
9	9.00000	-0.33826	1.11959	1.04437	-0.22219	0.49936	-0.24676	0.65158	0.06954

10 rows × 31 columns

As shown above, the data consists of **284807 rows** and **31 columns** .
The variable **Class** is the **dependant variable**.

Data preprocessing

In [10]:
```
pd.set_option('display.float_format', lambda x: '%.5f' % x) # to disable sci
credit_card_df.describe()
```

Out[10]:

	Time	V1	V2	V3	V4	
count	284807.00000	284807.00000	284807.00000	284807.00000	284807.00000	284807.00(
mean	94813.85958	0.00000	0.00000	-0.00000	0.00000	0.00(
std	47488.14595	1.95870	1.65131	1.51626	1.41587	1.38(
min	0.00000	-56.40751	-72.71573	-48.32559	-5.68317	-113.74:
25%	54201.50000	-0.92037	-0.59855	-0.89036	-0.84864	-0.69
50%	84692.00000	0.01811	0.06549	0.17985	-0.01985	-0.05
75%	139320.50000	1.31564	0.80372	1.02720	0.74334	0.61
max	172792.00000	2.45493	22.05773	9.38256	16.87534	34.80

8 rows × 31 columns

The variables V1 to V28 consists of values within a very small range - between -120 to 120. However, the variables **Time & Amount** seems to have a bigger range of values. **Hence, we can scale both the variables before modelling.**

The **StandardScaler function from sklearn** can be used for the same.

In [11]:
```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()

#Standardise the values in 'Amount' variable
credit_card_df['Time_std'] = scaler.fit_transform(credit_card_df['Time'].val
credit_card_df['Amount_std'] = scaler.fit_transform(credit_card_df['Amount']

#removing Time and Amount feature
credit_card_df.drop(["Time","Amount"], axis=1, inplace=True)
```

In [12]:
```
# Lets check the range of values after applying standard scaler
credit_card_df.describe()
```

Out[12]:

	V1	V2	V3	V4	V5	
count	284807.00000	284807.00000	284807.00000	284807.00000	284807.00000	284807.000
mean	0.00000	0.00000	-0.00000	0.00000	0.00000	0.000
std	1.95870	1.65131	1.51626	1.41587	1.38025	1.332
min	-56.40751	-72.71573	-48.32559	-5.68317	-113.74331	-26.16
25%	-0.92037	-0.59855	-0.89036	-0.84864	-0.69160	-0.768
50%	0.01811	0.06549	0.17985	-0.01985	-0.05434	-0.27
75%	1.31564	0.80372	1.02720	0.74334	0.61193	0.398
max	2.45493	22.05773	9.38256	16.87534	34.80167	73.30

8 rows × 31 columns

In [13]:

```
# Check for missing values
credit_card_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   V1          284807 non-null  float64
 1   V2          284807 non-null  float64
 2   V3          284807 non-null  float64
 3   V4          284807 non-null  float64
 4   V5          284807 non-null  float64
 5   V6          284807 non-null  float64
 6   V7          284807 non-null  float64
 7   V8          284807 non-null  float64
 8   V9          284807 non-null  float64
 9   V10         284807 non-null  float64
 10  V11         284807 non-null  float64
 11  V12         284807 non-null  float64
 12  V13         284807 non-null  float64
 13  V14         284807 non-null  float64
 14  V15         284807 non-null  float64
 15  V16         284807 non-null  float64
 16  V17         284807 non-null  float64
 17  V18         284807 non-null  float64
 18  V19         284807 non-null  float64
 19  V20         284807 non-null  float64
 20  V21         284807 non-null  float64
 21  V22         284807 non-null  float64
 22  V23         284807 non-null  float64
 23  V24         284807 non-null  float64
 24  V25         284807 non-null  float64
 25  V26         284807 non-null  float64
 26  V27         284807 non-null  float64
 27  V28         284807 non-null  float64
 28  Class       284807 non-null  int64  
 29  Time_std    284807 non-null  float64
 30  Amount_std  284807 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

There are no missing values in the dataset. Hence, data imputation is not required.

Dependent variable distribution

In [15]:

```
print('Non Frauds', round(credit_card_df['Class'].value_counts()[0]/len(cred
print('Frauds', round(credit_card_df['Class'].value_counts()[1]/len(credit_c
```

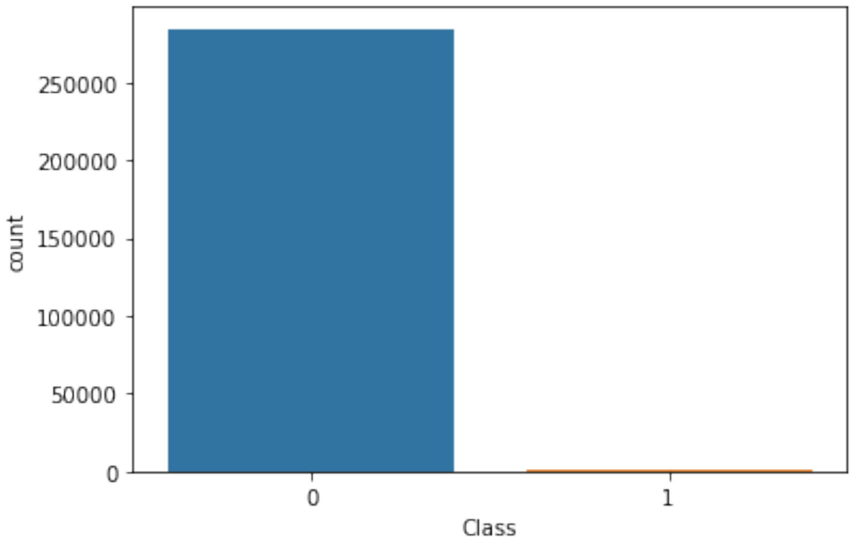
```
Non Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
```

In [8]:

```
sns.countplot(x="Class", data=credit_card_df)
```

Out[8]:

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



The dataset is highly imbalanced ! Most of the transactions are non-fraud. If we use this data as the base for our predictive models and analysis we might get a lot of errors and our algorithms will probably overfit since it will "assume" that most transactions are not fraud.

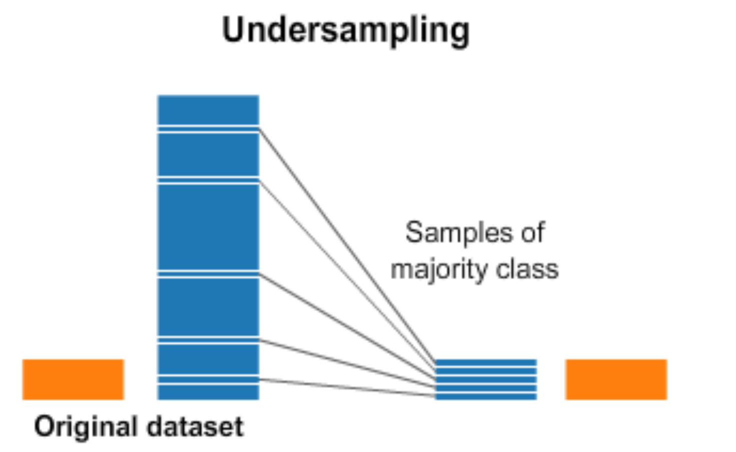
Resampling to handle imbalance

Dealing with imbalanced datasets entails strategies such as improving classification algorithms or balancing classes in the training data (data preprocessing) before providing the data as input to the machine learning algorithm. The later technique is preferred as it has wider application.

The main objective of balancing classes is to either increasing the frequency of the minority class or decreasing the frequency of the majority class. This is done in order to obtain approximately the same number of instances for both the classes

The simplest resampling method is using the undersampling technique which randomly selecting examples from the majority class and deleting them from the training dataset. This is referred to as random undersampling.

Although simple and effective, a limitation of this technique is that examples are removed without any concern for how useful or important they might be in determining the decision boundary between the classes. This means it is possible, or even likely, that useful information will be deleted.



For resampling methods, we can use the **imblearn package** which comes in-built with various resampling strategies.

When we use any sampling technique, we divide the data first into train & test sets and then apply sampling on the training data only. Once the model is trained, we evaluate the model on the test set that contains only the original samples.

In [9]:
```
from sklearn.model_selection import train_test_split

# define the input columns and target columns
cols = credit_card_df.columns.tolist()
cols = [c for c in cols if c not in ["Class"]]
target = "Class"

#define X and Y
X = credit_card_df[cols]
Y = credit_card_df[target]

X_train, X_test, y_train, y_test = train_test_split(X, Y, stratify=Y, test_s
```

In [10]:
```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
undersample = RandomUnderSampler(sampling_strategy=0.4)

#undersample the training data
X_train_undersampled, Y_train_undersampled = undersample.fit_resample(X_trai
```

In [11]:
```
print("Class variable distrubution before undersampling:\n"+str(y_train.valu
```

```
Class variable distrubution before undersampling:
0    213236
1       369
Name: Class, dtype: int64
```

In [12]:
```
print("Class variable distribution after undersampling:\n"+str(Y_train_under
```

```
Class variable distribution after undersampling:
0    922
1    369
Name: Class, dtype: int64
```

Model development

We can explore several machine learning classification models in order to find which one performs best on our data. The following models will be used:

- Logistic regression
- Random Forest Classifier
- Support Vector Classifier
- Gradient Boosting Classifier

The methodology used to train each model is as follows:

- Select the set of hyperparameters to tune for each model.
- Define the metric we'll get when measuring the performance of a model. In this case, we'll use the accuracy. (The final metric of choice will be the AUROC score of the model since accuracy is not a good metric for imbalanced classification).
- Perform a Randomized Search Cross Validation process in order to find the hyperparameter region in which we get higher values of accuracy.
- Use a Grid Search Cross Validation process to exhaustively find the best combination of hyperparameters around the best region identified using Randomized Search Cross Validation.
- Evaluate the models on the test set and choose the best model based on the evaluation metrics.

In [13]:
```
#importing packages for modeling
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn import metrics
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import precision_recall_curve

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from pprint import pprint
```

In [14]:
```
# Turn the values into an array for feeding the classification algorithms.
X_train = X_train_undersampled.values
y_train = Y_train_undersampled.values
```

In [15]:
```
# Let's implement the classifiers without tuning any hyperparameters (Baseli

classifiers = {
    "LogisiticRegression": LogisticRegression(),
    "SupportVectorClassifier": SVC(),
    "RandomForestClassifier": RandomForestClassifier(),
    "GradientBoostingClassifier": GradientBoostingClassifier()
}
```

In [16]:
```
# We will use 5-fold cross validation to check our model performance scores
from sklearn.model_selection import cross_val_score


for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    training_score = cross_val_score(classifier, X_train, y_train, cv=5)
    print("Classifiers: ", classifier.__class__.__name__, "has a training sc
```

```
Classifiers:  LogisticRegression has a training score of 96.0 % accuracy sco
re
Classifiers:  SVC has a training score of 95.0 % accuracy score
Classifiers:  RandomForestClassifier has a training score of 95.0 % accuracy
score
Classifiers:  GradientBoostingClassifier has a training score of 95.0 % accu
racy score
```

The baseline scores of the models looks very promising. All the models yield a very good training score. We can further use hyperparameter tuning to improve the performance of each model and further evaluate on the test set to choose our final model.

Logistic Regression

In [17]:
```
base_lr_model = LogisticRegression(random_state = 8)

print('Parameters currently in use:\n')
pprint(base_lr_model.get_params())
```

```
Parameters currently in use:

{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': 8,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

Hyperparameter tuning using Cross Validation

The following hyperparameters will be tuned:

- ***C*** : Inverse of regularization strength. Smaller values specify stronger regularization.
- ***class_weight*** : Weights associated with classes.
- ***penalty*** : Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties.

In [18]:
```
# C
C = [float(x) for x in np.linspace(start = 0.1, stop = 1, num = 10)]

# solver
solver = ['liblinear'] #For small datasets, ‘liblinear’ is a good choice.

# class_weight
class_weight = ['balanced', None]

# penalty
penalty = ['l1','l2']

# Create the random grid
random_grid = {'C': C,
               'solver': solver,
               'class_weight': class_weight,
               'penalty': penalty}

pprint(random_grid)
```

```
{'C': [0.1,
       0.2,
       0.30000000000000004,
       0.4,
       0.5,
       0.6,
       0.7000000000000001,
       0.8,
       0.9,
       1.0],
 'class_weight': ['balanced', None],
 'penalty': ['l1', 'l2'],
 'solver': ['liblinear']}
```

Randomized Search Cross Validation

In [19]:
```
# Definition of the random search
random_search_LRC = RandomizedSearchCV(estimator=base_lr_model,
                                   param_distributions=random_grid,
                                   n_iter=30,
                                   scoring='accuracy',
                                   cv=3,
                                   verbose=1,
                                   random_state=8)

# Fit the random search model
random_search_LRC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

Out[19]:
```
RandomizedSearchCV(cv=3, estimator=LogisticRegression(random_state=8),
                   n_iter=30,
                   param_distributions={'C': [0.1, 0.2, 0.30000000000000004,
                                              0.4, 0.5, 0.6, 0.7000000000000
001,
                                              0.8, 0.9, 1.0],
                                        'class_weight': ['balanced', None],
                                        'penalty': ['l1', 'l2'],
                                        'solver': ['liblinear']},
                   random_state=8, scoring='accuracy', verbose=1)
```

In [20]:
```python
print("The best hyperparameters from Random Search are:")
print(random_search_LRC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(random_search_LRC.best_score_)
```

```
The best hyperparameters from Random Search are:
{'solver': 'liblinear', 'penalty': 'l1', 'class_weight': None, 'C': 0.2}

The mean accuracy of a model with these hyperparameters is:
0.9604849008075685
```

Grid Search Cross Validation

Do a more centered search around the hyperparameter values obtained using Randomized Search CV

In [21]:
```python
# Create the parameter grid based on the results of random search
C = [float(x) for x in np.linspace(start = 0.7, stop = 1, num = 10)]
solver = ['liblinear']
class_weight = [None]
penalty = ['l1']

param_grid = {'C': C,
              'solver': solver,
              'class_weight': class_weight,
              'penalty': penalty}


# Manually create the splits in CV in order to be able to fix a random_state
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

# Instantiate the grid search model
grid_search_LRC = GridSearchCV(estimator=base_lr_model,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=cv_sets,
                           verbose=1)

# Fit the grid search to the data
grid_search_LRC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

Out[21]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=LogisticRegression(random_state=8),
             param_grid={'C': [0.7, 0.7333333333333333, 0.7666666666666666,
                               0.7999999999999999, 0.8333333333333333,
                               0.8666666666666667, 0.9, 0.9333333333333333,
                               0.9666666666666667, 1.0],
                         'class_weight': [None], 'penalty': ['l1'],
                         'solver': ['liblinear']},
             scoring='accuracy', verbose=1)
```

In [22]:
```
print("The best hyperparameters from Grid Search are:")
print(grid_search_LRC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search_LRC.best_score_)
```

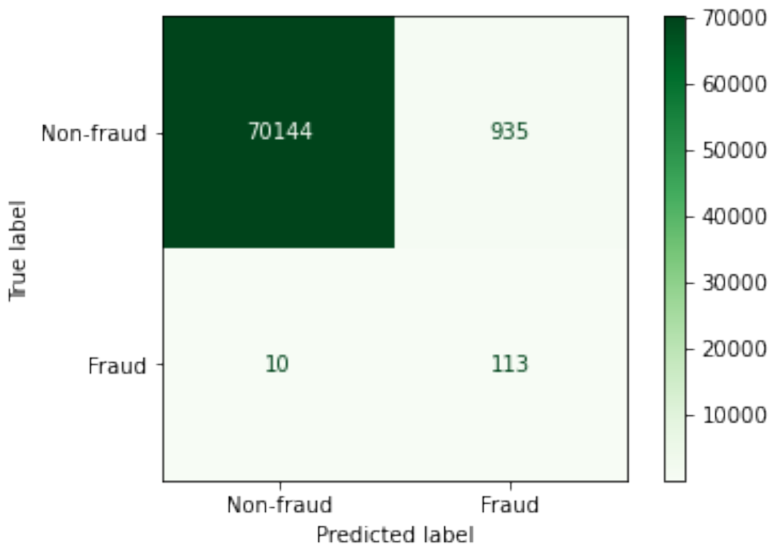
```
The best hyperparameters from Grid Search are:
{'C': 0.7, 'class_weight': None, 'penalty': 'l1', 'solver': 'liblinear'}

The mean accuracy of a model with these hyperparameters is:
0.9640905542544886
```

In [23]:
```
best_lrc = grid_search_LRC.best_estimator_
best_lrc.fit(X_train,y_train)

#Predict on test data
lrc_pred = best_lrc.predict(X_test)
```

Logistic regression model - Test data evaluation

In [24]:
```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, lrc_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,lrc_pred, target_names=['Non-fraud', 'Fra
print("\n")

#Confusion matrix
print(plot_confusion_matrix(best_lrc, X_test, y_test, values_format = '',cma
```

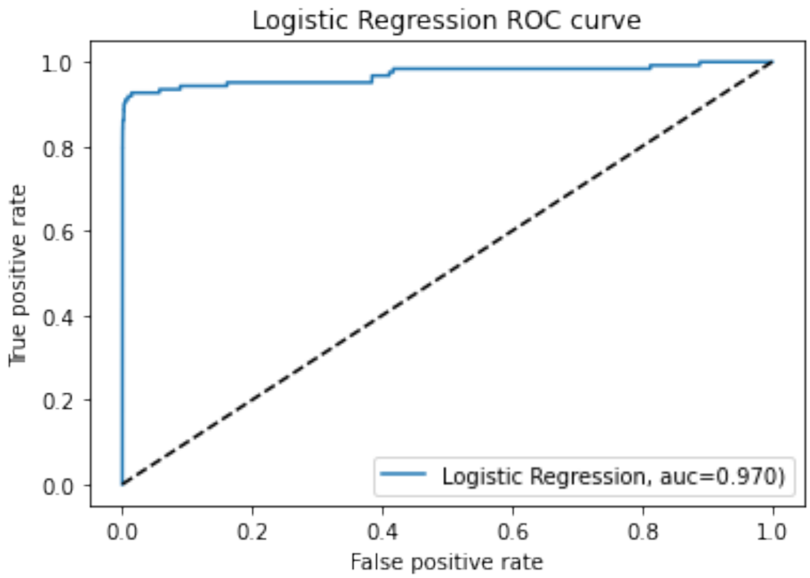
```
The test accuracy is:
0.9867279009016601


Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      0.99     71079
       Fraud       0.11      0.92      0.19       123

    accuracy                           0.99     71202
   macro avg       0.55      0.95      0.59     71202
weighted avg       1.00      0.99      0.99     71202


<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe28da6130>
```



In [25]:

```python
# Plot ROC AUC curve
#AUC
lrc_pred_proba = best_lrc.predict_proba(X_test)[::,1]
lrc_fpr, lrc_tpr, _ = metrics.roc_curve(y_test,  lrc_pred_proba)
LRC_auc = metrics.roc_auc_score(y_test, lrc_pred_proba)
print("AUC Logistic Regression :", LRC_auc)

#ROC
plt.plot(lrc_fpr,lrc_tpr,label="Logistic Regression, auc={:.3f})".format(LRC
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Logistic Regression ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Logistic Regression : 0.9702271616478035
```



Support Vector Classifier

In [26]:
```
base_svc = svm.SVC(random_state=8)

print('Parameters currently in use:\n')
pprint(base_svc.get_params())
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': 8,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

Hyperparameter Tuning using Cross Validation

The following hyperparameters will be tuned:

- ***C***: Penalty parameter C of the error term.
- ***kernel***: Specifies the kernel type to be used in the algorithm.
- ***gamma***: Kernel coefficient.
- ***degree***: Degree of the polynomial kernel function.

Randomized Search Cross Validation

In [27]:
```python
# C
C = [.0001, .001, .01]

# gamma
gamma = [.0001, .001, .01, .1, 1, 10, 100]

# degree
degree = [1, 2, 3, 4, 5]

# kernel
kernel = ['linear', 'rbf', 'poly']

# probability
probability = [True]

# Create the random grid
random_grid = {'C': C,
              'kernel': kernel,
              'gamma': gamma,
              'degree': degree,
              'probability': probability
             }

pprint(random_grid)
```

```
{'C': [0.0001, 0.001, 0.01],
 'degree': [1, 2, 3, 4, 5],
 'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
 'kernel': ['linear', 'rbf', 'poly'],
 'probability': [True]}
```

In [28]:
```python
# First create the base model to tune
svc = svm.SVC(random_state=8)

# Definition of the random search
random_search_SVC = RandomizedSearchCV(estimator=base_svc,
                                   param_distributions=random_grid,
                                   n_iter=50,
                                   scoring='accuracy',
                                   cv=3,
                                   verbose=1,
                                   random_state=8)

# Fit the random search model
random_search_SVC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Out[28]:
```
RandomizedSearchCV(cv=3, estimator=SVC(random_state=8), n_iter=50,
                   param_distributions={'C': [0.0001, 0.001, 0.01],
                                        'degree': [1, 2, 3, 4, 5],
                                        'gamma': [0.0001, 0.001, 0.01, 0.1,
1,
                                                  10, 100],
                                        'kernel': ['linear', 'rbf', 'poly'],
                                        'probability': [True]},
                   random_state=8, scoring='accuracy', verbose=1)
```

In [29]:
```python
print("The best hyperparameters from Random Search are:")
print(random_search_SVC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(random_search_SVC.best_score_)
```

```
The best hyperparameters from Random Search are:
{'probability': True, 'kernel': 'poly', 'gamma': 1, 'degree': 3, 'C': 0.0001
}

The mean accuracy of a model with these hyperparameters is:
0.9550585442184212
```

Grid Search Cross Validation

In [30]:
```python
# Create the parameter grid based on the results of random search
C = [.0001, .001, .01]
degree = [3, 4, 5]
gamma = [0.0001, 0.001, 0.01]
probability = [True]
kernel = ['linear']

param_grid = [
  {'C': C, 'kernel':['linear'], 'probability':probability},
  {'C': C, 'kernel':['poly'], 'degree':degree, 'probability':probability},
  {'C': C, 'kernel':['rbf'], 'gamma':gamma, 'probability':probability}
]


# Manually create the splits in CV in order to be able to fix a random_state
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

# Instantiate the grid search model
grid_search_SVC = GridSearchCV(estimator=base_svc,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=cv_sets,
                           verbose=1)

# Fit the grid search to the data
grid_search_SVC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 21 candidates, totalling 63 fits
```

Out[30]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=SVC(random_state=8),
             param_grid=[{'C': [0.0001, 0.001, 0.01], 'kernel': ['linear'],
                          'probability': [True]},
                         {'C': [0.0001, 0.001, 0.01], 'degree': [3, 4, 5],
                          'kernel': ['poly'], 'probability': [True]},
                         {'C': [0.0001, 0.001, 0.01],
                          'gamma': [0.0001, 0.001, 0.01], 'kernel': ['rbf'],
                          'probability': [True]}],
             scoring='accuracy', verbose=1)
```

In [31]:
```
print("The best hyperparameters from Grid Search are:")
print(grid_search_SVC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search_SVC.best_score_)
```

```
The best hyperparameters from Grid Search are:
{'C': 0.01, 'kernel': 'linear', 'probability': True}

The mean accuracy of a model with these hyperparameters is:
0.9547228727556597
```

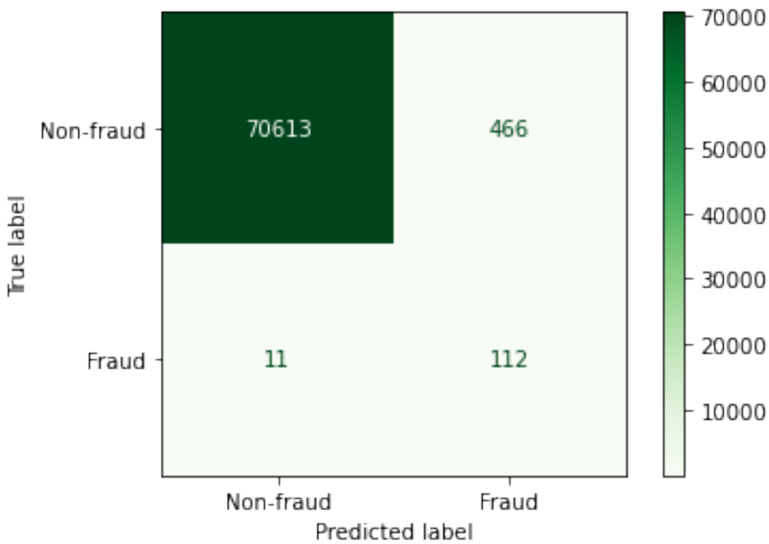
In [32]:
```
best_svc = grid_search_SVC.best_estimator_
best_svc.fit(X_train,y_train)

#Predict on test data
svc_pred = best_svc.predict(X_test)
```

Support Vector Classifier model - Test data evaluation

In [33]:
```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, svc_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,svc_pred, target_names=['Non-fraud', 'Fra
print("\n")

#Confusion matrix
print(plot_confusion_matrix(best_svc, X_test, y_test, values_format = '',cma
```

```
The test accuracy is:
0.9933007499789331


Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      1.00     71079
       Fraud       0.19      0.91      0.32       123

    accuracy                           0.99     71202
   macro avg       0.60      0.95      0.66     71202
weighted avg       1.00      0.99      1.00     71202
```

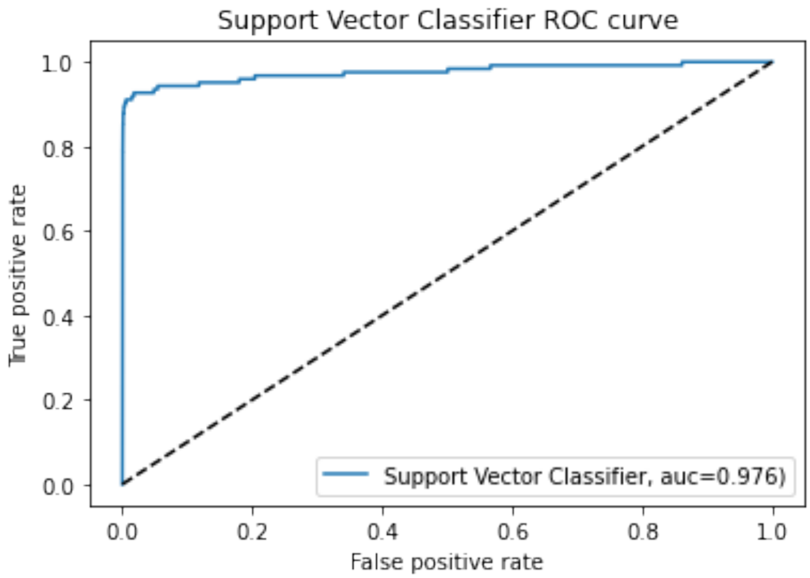
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe28baef70>
```



In [34]:

```
# Plot ROC AUC curve
#AUC
svc_pred_proba = best_svc.predict_proba(X_test)[::,1]
svc_fpr, svc_tpr, _ = metrics.roc_curve(y_test,  svc_pred_proba)
svc_auc = metrics.roc_auc_score(y_test, svc_pred_proba)
print("AUC Support Vector Classifier :", svc_auc)

#ROC
plt.plot(svc_fpr,svc_tpr,label="Support Vector Classifier, auc={:.3f})".form
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Support Vector Classifier ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Support Vector Classifier : 0.9760357678282391
```



Random Forest Classifier

In [35]:
```
base_rfc = RandomForestClassifier(random_state = 8)

print('Parameters currently in use:\n')
pprint(base_rfc.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 8,
 'verbose': 0,
 'warm_start': False}
```

Hyperparameter Tuning for Cross Validation

The following hyperparameters will be tuned:

- ***n_estimators*** = number of trees in the forest.
- ***max_features*** = max number of features considered for splitting a node
- ***max_depth*** = max number of levels in each decision tree
- ***min_samples_split*** = min number of data points placed in a node before the node is split
- ***min_samples_leaf*** = min number of data points allowed in a leaf node
- ***bootstrap*** = method for sampling data points (with or without replacement)

Randomized Search Cross Validation

In [36]:

```
# n_estimators
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 600, num = 3

# max_features
max_features = ['auto', 'sqrt']

# max_depth
max_depth = [int(x) for x in np.linspace(20, 100, num = 5)]
max_depth.append(None)

# min_samples_split
min_samples_split = [2, 5, 10]

# min_samples_leaf
min_samples_leaf = [1, 2, 4]

# bootstrap
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [20, 40, 60, 80, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600]}
```

In [37]:
```
# Definition of the random search
random_search_RFC = RandomizedSearchCV(estimator=base_rfc,
                                   param_distributions=random_grid,
                                   n_iter=50,
                                   scoring='accuracy',
                                   cv=3,
                                   verbose=1,
                                   random_state=8,
                                       n_jobs=-1)

# Fit the random search model
random_search_RFC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Out[37]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(random_state=8),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [20, 40, 60, 80, 100,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600]},
                   random_state=8, scoring='accuracy', verbose=1)
```

In [38]:
```
print("The best hyperparameters from Random Search are:")
print(random_search_RFC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(random_search_RFC.best_score_)
```

```
The best hyperparameters from Random Search are:
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_f
eatures': 'sqrt', 'max_depth': 100, 'bootstrap': True}

The mean accuracy of a model with these hyperparameters is:
0.9542869476069713
```

Grid Search Cross Validation

In [39]:
```
# Create the parameter grid based on the results of random search
bootstrap = [False]
max_depth = [60, 70, 80]
max_features = ['sqrt']
min_samples_leaf = [1, 2, 4]
min_samples_split = [2, 5, 10]
n_estimators = [200]

param_grid = {
    'bootstrap': bootstrap,
    'max_depth': max_depth,
    'max_features': max_features,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'n_estimators': n_estimators
}


# Manually create the splits in CV in order to be able to fix a random_state
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

# Instantiate the grid search model
grid_search_RFC = GridSearchCV(estimator=base_rfc,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=cv_sets,
                           verbose=1,
                           n_jobs=-1)

# Fit the grid search to the data
grid_search_RFC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
```

Out[39]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=RandomForestClassifier(random_state=8), n_jobs=-1,
             param_grid={'bootstrap': [False], 'max_depth': [60, 70, 80],
                         'max_features': ['sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [200]},
             scoring='accuracy', verbose=1)
```

In [40]:
```
print("The best hyperparameters from Grid Search are:")
print(grid_search_RFC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search_RFC.best_score_)
```

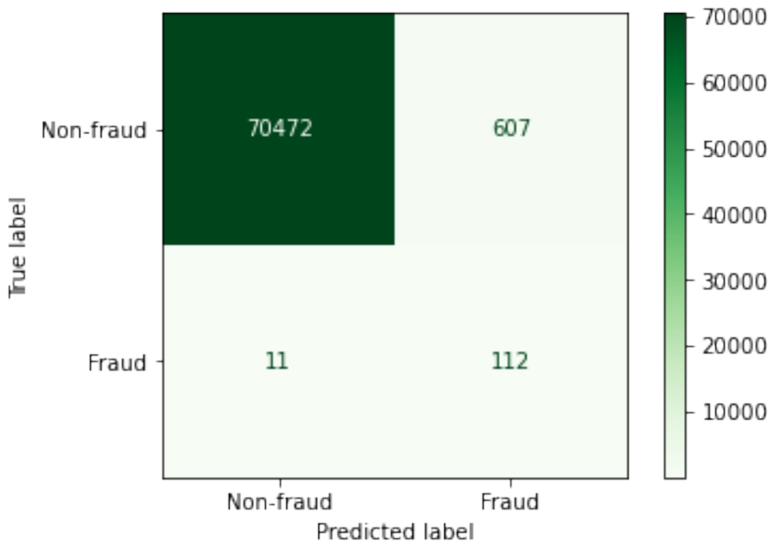
```
The best hyperparameters from Grid Search are:
{'bootstrap': False, 'max_depth': 60, 'max_features': 'sqrt', 'min_samples_l
eaf': 1, 'min_samples_split': 5, 'n_estimators': 200}

The mean accuracy of a model with these hyperparameters is:
0.9594067135050741
```

In [41]:
```
best_rfc = grid_search_RFC.best_estimator_
best_rfc.fit(X_train,y_train)

#Predict on test data
rfc_pred = best_rfc.predict(X_test)
```

Random Forest Classifier model - Test data evaluation

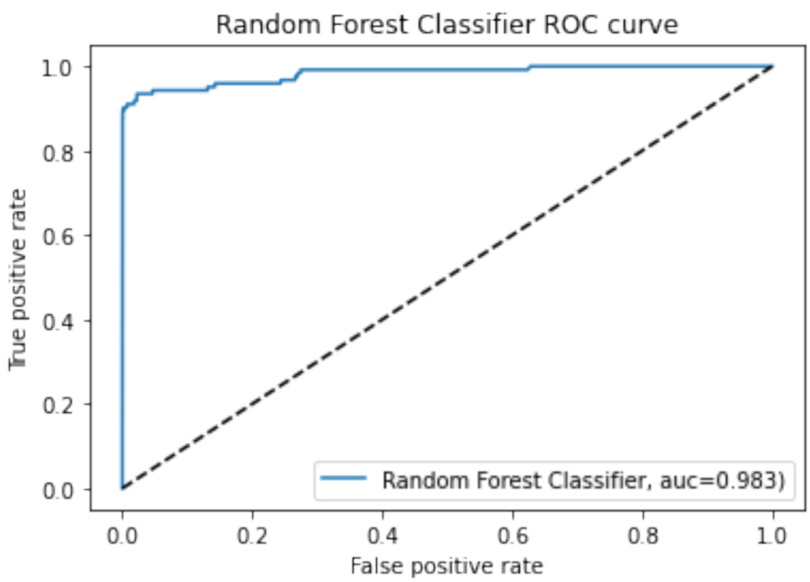
In [42]:
```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, rfc_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,rfc_pred, target_names=['Non-fraud', 'Fra
print("\n")

#Confusion matrix
print(plot_confusion_matrix(best_rfc, X_test, y_test, values_format = '',cma
```

```
The test accuracy is:
0.991320468526165


Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      1.00     71079
       Fraud       0.16      0.91      0.27       123

    accuracy                           0.99     71202
   macro avg       0.58      0.95      0.63     71202
weighted avg       1.00      0.99      0.99     71202



<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe4908e100>
```



In [43]:
```
# Plot ROC AUC curve
#AUC
rfc_pred_proba = best_rfc.predict_proba(X_test)[::,1]
rfc_fpr, rfc_tpr, _ = metrics.roc_curve(y_test,  rfc_pred_proba)
rfc_auc = metrics.roc_auc_score(y_test, rfc_pred_proba)
print("AUC Random Forest Classifier :", rfc_auc)

#ROC
plt.plot(rfc_fpr,rfc_tpr,label="Random Forest Classifier, auc={:.3f})".forma
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Random Forest Classifier ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Random Forest Classifier : 0.9830674491694057
```



Gradient Boosting Classifier

In [44]:
```python
base_gbc = GradientBoostingClassifier(random_state = 8)

print('Parameters currently in use:\n')
pprint(base_gbc.get_params())
```

```
Parameters currently in use:

{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': 8,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

Hyperparameter Tuning for Cross Validation

The following hyperparameters will be tuned:

Tree-related hyperparameters:

- ***n_estimators*** = number of trees in the forest.
- ***max_features*** = max number of features considered for splitting a node
- ***max_depth*** = max number of levels in each decision tree
- ***min_samples_split*** = min number of data points placed in a node before the node is split
- ***min_samples_leaf*** = min number of data points allowed in a leaf node

Boosting-related hyperparameters:

- ***learning_rate*** = learning rate shrinks the contribution of each tree by learning_rate.
- ***subsample*** = the fraction of samples to be used for fitting the individual base learners.

Randomized Search Cross Validation

In [45]:
```
# n_estimators
n_estimators = [200, 400, 600]

# max_features
max_features = ['auto', 'sqrt']

# max_depth
max_depth = [10, 40]
max_depth.append(None)

# min_samples_split
min_samples_split = [10, 30, 50]

# min_samples_leaf
min_samples_leaf = [1, 2, 4]

# learning rate
learning_rate = [.1, .5]

# subsample
subsample = [.5, 1.]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'learning_rate': learning_rate,
               'subsample': subsample}

pprint(random_grid)
```

```
{'learning_rate': [0.1, 0.5],
 'max_depth': [10, 40, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [10, 30, 50],
 'n_estimators': [200, 400, 600],
 'subsample': [0.5, 1.0]}
```

In [46]:
```
# Definition of the random search
random_search_GBC = RandomizedSearchCV(estimator=base_gbc,
                                       param_distributions=random_grid,
                                       n_iter=50,
                                       scoring='accuracy',
                                       cv=3,
                                       verbose=1,
                                       random_state=8,
                                       n_jobs=-1)

# Fit the random search model
random_search_GBC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Out[46]:
```
RandomizedSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=8
),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'learning_rate': [0.1, 0.5],
                                        'max_depth': [10, 40, None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [10, 30, 50],
                                        'n_estimators': [200, 400, 600],
                                        'subsample': [0.5, 1.0]},
                   random_state=8, scoring='accuracy', verbose=1)
```

In [47]:
```
print("The best hyperparameters from Random Search are:")
print(random_search_GBC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(random_search_GBC.best_score_)
```

```
The best hyperparameters from Random Search are:
{'subsample': 1.0, 'n_estimators': 600, 'min_samples_split': 10, 'min_sample
s_leaf': 2, 'max_features': 'auto', 'max_depth': 40, 'learning_rate': 0.1}

The mean accuracy of a model with these hyperparameters is:
0.9535153509955214
```

Grid Search Cross Validation

In [48]:
```
# Create the parameter grid based on the results of random search
max_depth = [ 10, 15]
max_features = ['auto']
min_samples_leaf = [4]
min_samples_split = [10, 20, 50]
n_estimators = [600]
learning_rate = [.1, .5]
subsample = [1.]

param_grid = {
    'max_depth': max_depth,
    'max_features': max_features,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'n_estimators': n_estimators,
    'learning_rate': learning_rate,
    'subsample': subsample

}


# Manually create the splits in CV in order to be able to fix a random_state
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

# Instantiate the grid search model
grid_search_GBC = GridSearchCV(estimator=base_gbc,
                               param_grid=param_grid,
                               scoring='accuracy',
                               cv=cv_sets,
                               verbose=1,
                               n_jobs=-1)

# Fit the grid search to the data
grid_search_GBC.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

Out[48]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=GradientBoostingClassifier(random_state=8), n_jobs=-1
,
             param_grid={'learning_rate': [0.1, 0.5], 'max_depth': [10, 15],
                         'max_features': ['auto'], 'min_samples_leaf': [4],
                         'min_samples_split': [10, 20, 50],
                         'n_estimators': [600], 'subsample': [1.0]},
             scoring='accuracy', verbose=1)
```

In [49]:
```
print("The best hyperparameters from Grid Search are:")
print(grid_search_GBC.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search_GBC.best_score_)
```

```
The best hyperparameters from Grid Search are:
{'learning_rate': 0.5, 'max_depth': 10, 'max_features': 'auto', 'min_samples
_leaf': 4, 'min_samples_split': 10, 'n_estimators': 600, 'subsample': 1.0}

The mean accuracy of a model with these hyperparameters is:
0.9562841530054644
```

In [50]:

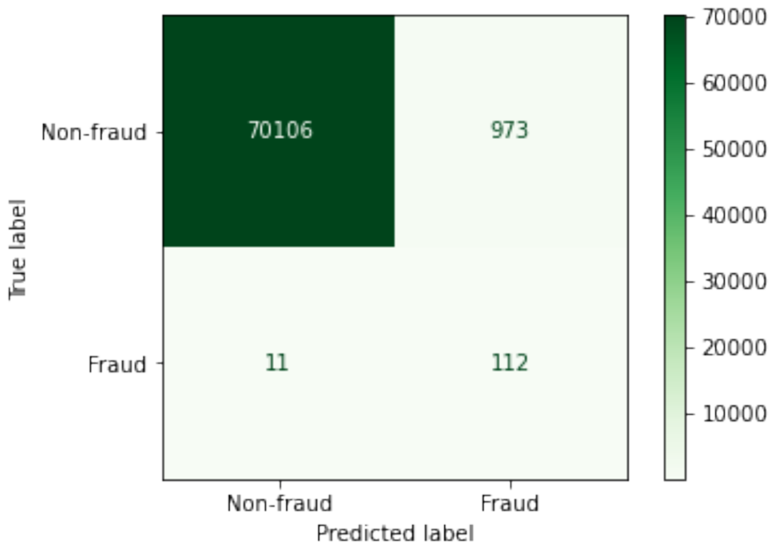
```
best_gbc = grid_search_GBC.best_estimator_
best_gbc.fit(X_train,y_train)

#Predict on test data
gbc_pred = best_rfc.predict(X_test)
```

Gradient Boosting Classifier model - Test data evaluation

In [51]:

```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, gbc_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,gbc_pred, target_names=['Non-fraud', 'Fra
print("\n")

#Confusion matrix
print(plot_confusion_matrix(best_gbc, X_test, y_test, values_format = '',cma
```

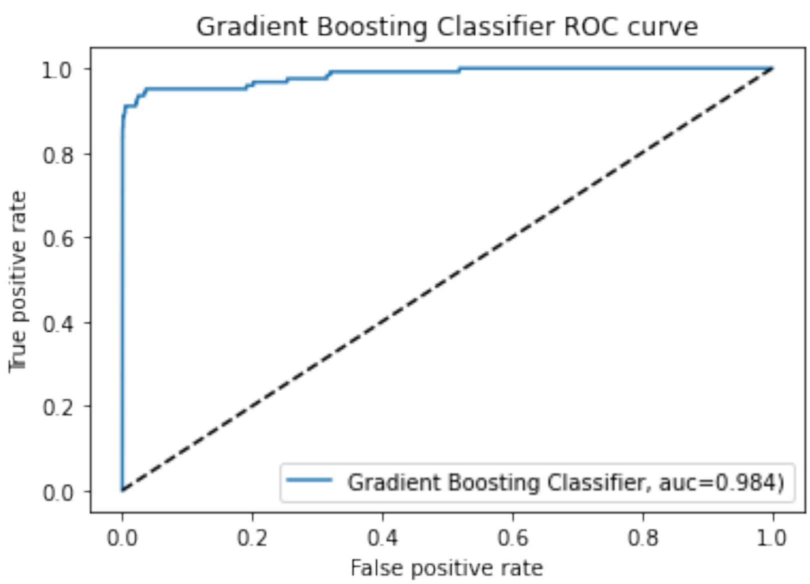
```
The test accuracy is:
0.991320468526165


Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      1.00     71079
       Fraud       0.16      0.91      0.27       123

    accuracy                           0.99     71202
   macro avg       0.58      0.95      0.63     71202
weighted avg       1.00      0.99      0.99     71202



<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe292e9250>
```



In [52]:

```
# Plot ROC AUC curve
#AUC
gbc_pred_proba = best_gbc.predict_proba(X_test)[::,1]
gbc_fpr, gbc_tpr, _ = metrics.roc_curve(y_test,  gbc_pred_proba)
gbc_auc = metrics.roc_auc_score(y_test, gbc_pred_proba)
print("AUC Gradient Boosting Classifier :", gbc_auc)

#ROC
plt.plot(gbc_fpr,gbc_tpr,label="Gradient Boosting Classifier, auc={:.3f})".f
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Gradient Boosting Classifier ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Gradient Boosting Classifier : 0.9840562150187407
```



- All the four models are extremely good in classifying a Fraud from Non-Fraud transaction.
- The AUC Scores obtained are all above 0.96 which indicates that the predicting power of these models are very good with **Gradient Bosoting Classifier having the highest accuracy of 0.994 and AUC score of 0.983 on the test set**.

Ensembling using Voting Classifier

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Voting Classifier supports two types of votings.

- **Hard Voting**: In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class(A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.
- **Soft Voting**: In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier.

Note: We will use Soft Voting since we need the predicted probabilities to get the AUC scores.

In [53]:
```
from sklearn.ensemble import VotingClassifier
```

Model 1 - RFC, SVC and GBC

```
In [54]: """Model 1 - Using RFC,GBC and SVC"""
models= []
models.append(('RFC',best_rfc))
models.append(('SVC',best_svc))
models.append(('GBC',best_gbc))

vot_classifier_1 = VotingClassifier(estimators=models,voting='soft')
vot_classifier_1.fit(X_train,y_train)
```

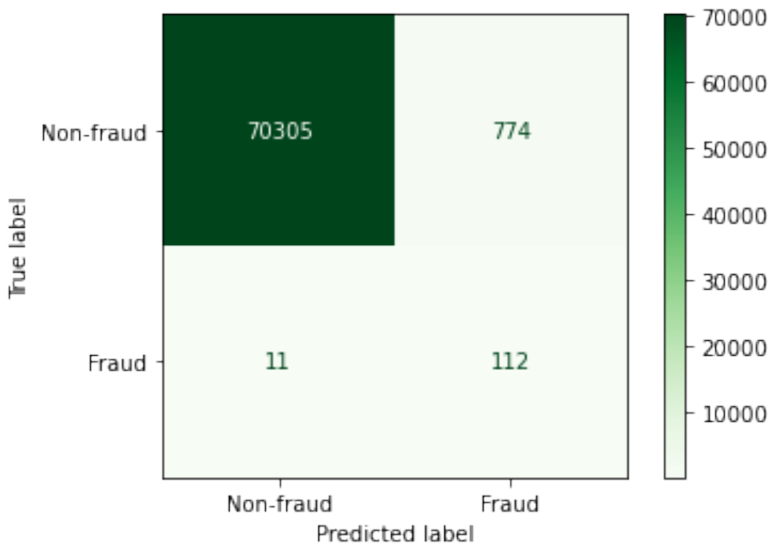
```
Out[54]: VotingClassifier(estimators=[('RFC',
                              RandomForestClassifier(bootstrap=False,
                                                     max_depth=60,
                                                     max_features='sqrt',
                                                     min_samples_split=5,
                                                     n_estimators=200,
                                                     random_state=8)),
                             ('SVC',
                              SVC(C=0.01, kernel='linear', probability=True,
                                  random_state=8)),
                             ('GBC',
                              GradientBoostingClassifier(learning_rate=0.5,
                                                         max_depth=10,
                                                         max_features='auto'
,
                                                         min_samples_leaf=4,
                                                         min_samples_split=1
0,
                                                         n_estimators=600,
                                                         random_state=8))],
                 voting='soft')
```

```
In [55]: vot_classifier_1_pred = vot_classifier_1.predict(X_test)

# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, vot_classifier_1_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,vot_classifier_1_pred, target_names=['Non
print("\n")

#Confusion matrix
print(plot_confusion_matrix(vot_classifier_1, X_test, y_test, values_format
```

```
The test accuracy is:
0.988975028791326
```

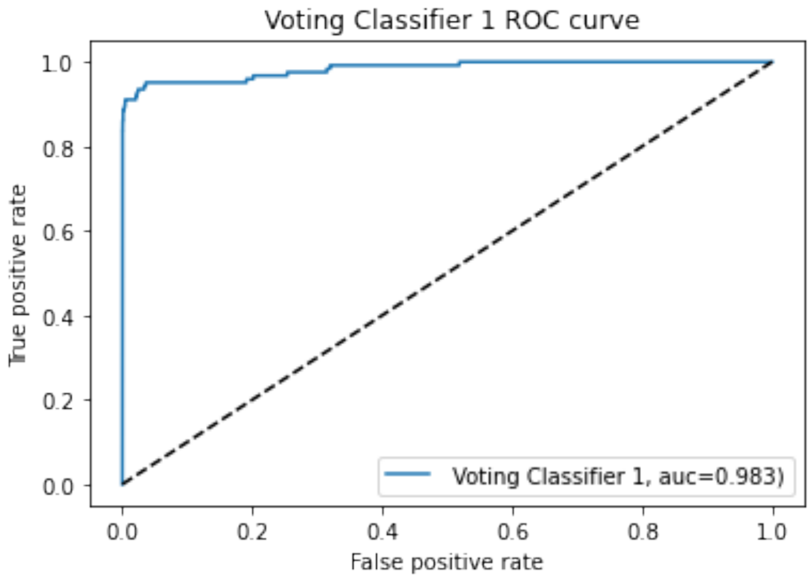
```
Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      0.99     71079
       Fraud       0.13      0.91      0.22       123

    accuracy                           0.99     71202
   macro avg       0.56      0.95      0.61     71202
weighted avg       1.00      0.99      0.99     71202
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe0956aa00>
```



In [56]:
```
# Plot ROC AUC curve
#AUC
vot_classifier_1_pred_proba = vot_classifier_1.predict_proba(X_test)[::,1]
vot_classifier_1_fpr, vot_classifier_1_tpr, _ = metrics.roc_curve(y_test,  v
vot_classifier_1_auc = metrics.roc_auc_score(y_test, vot_classifier_1_pred_p
print("AUC Voting Classifier 1 :", vot_classifier_1_auc)

#ROC
plt.plot(gbc_fpr,gbc_tpr,label=" Voting Classifier 1, auc={:.3f})".format(vo
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title(' Voting Classifier 1 ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Voting Classifier 1 : 0.9829660504852209
```



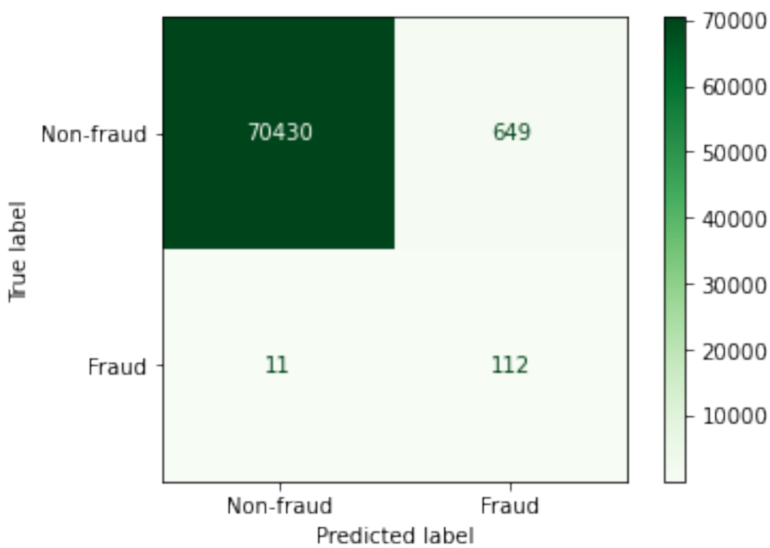
Model 2 - Using, RFC, SVC, LRC

In [57]:
```
"""Model 2 - Using RFC,SVC and LRC"""
models= []
models.append(('RFC',best_rfc))
models.append(('SVC',best_svc))
models.append(('LRC',best_lrc))

vot_classifier_2 = VotingClassifier(estimators=models,voting='soft')
vot_classifier_2.fit(X_train,y_train)
```

Out[57]:
```
VotingClassifier(estimators=[('RFC',
                              RandomForestClassifier(bootstrap=False,
                                                     max_depth=60,
                                                     max_features='sqrt',
                                                     min_samples_split=5,
                                                     n_estimators=200,
                                                     random_state=8)),
                             ('SVC',
                              SVC(C=0.01, kernel='linear', probability=True,
                                  random_state=8)),
                             ('LRC',
                              LogisticRegression(C=0.7, penalty='l1',
                                                 random_state=8,
                                                 solver='liblinear'))],
                 voting='soft')
```

In [58]:
```
vot_classifier_2_pred = vot_classifier_2.predict(X_test)

# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, vot_classifier_2_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,vot_classifier_2_pred, target_names=['Non
print("\n")

#Confusion matrix
print(plot_confusion_matrix(vot_classifier_2, X_test, y_test, values_format
```

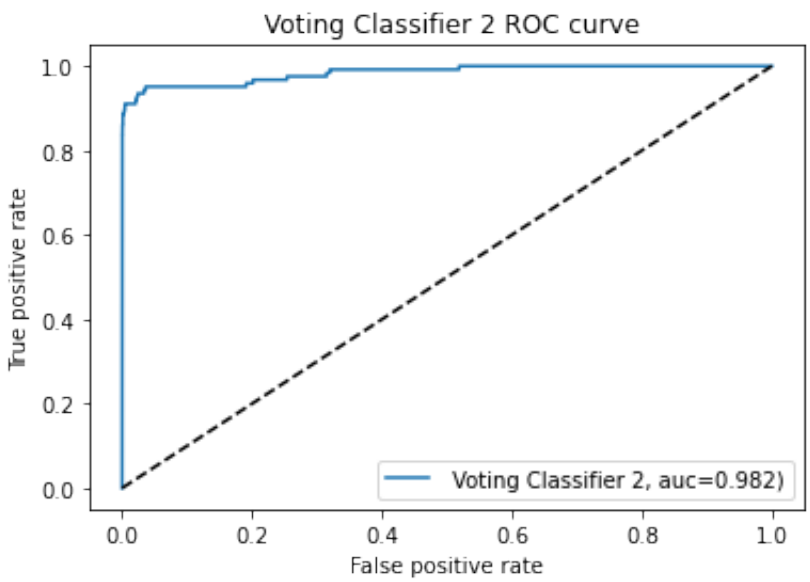
```
The test accuracy is:
0.9907305974551277


Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      1.00     71079
       Fraud       0.15      0.91      0.25       123

    accuracy                           0.99     71202
   macro avg       0.57      0.95      0.62     71202
weighted avg       1.00      0.99      0.99     71202



<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe39c35670>
```



In [59]:

```
# Plot ROC AUC curve
#AUC
vot_classifier_2_pred_proba = vot_classifier_2.predict_proba(X_test)[::,1]
vot_classifier_2_fpr, vot_classifier_2_tpr, _ = metrics.roc_curve(y_test,  v
vot_classifier_2_auc = metrics.roc_auc_score(y_test, vot_classifier_2_pred_p
print("AUC Voting Classifier 2 :", vot_classifier_2_auc)

#ROC
plt.plot(gbc_fpr,gbc_tpr,label=" Voting Classifier 2, auc={:.3f})".format(vo
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title(' Voting Classifier 2 ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Voting Classifier 2 : 0.9816276221682574
```



Model 3 - Using SVC, LRC and GBC

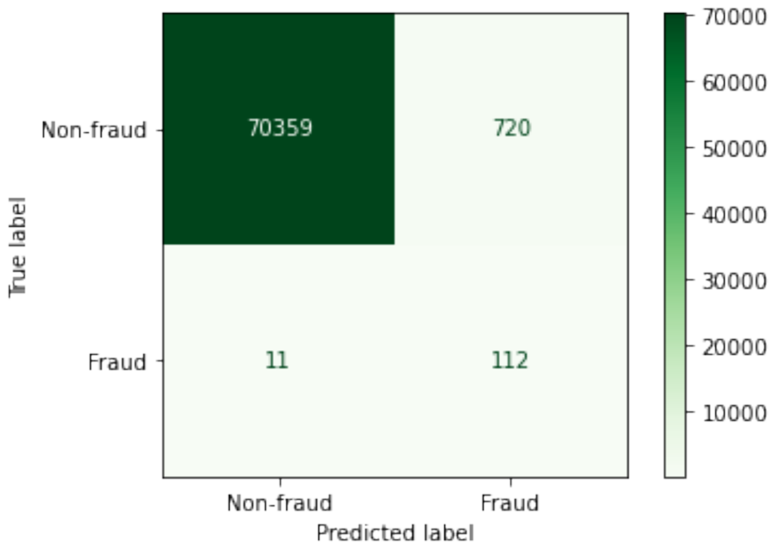
In [60]:

```
"""Model 3 - Using SVC, LRC and GBC"""
models= []
models.append(('SVC',best_svc))
models.append(('LRC',best_lrc))
models.append(('GBC',best_gbc))

vot_classifier_3 = VotingClassifier(estimators=models,voting='soft')
vot_classifier_3.fit(X_train,y_train)
```

Out[60]:
```
VotingClassifier(estimators=[('SVC',
                              SVC(C=0.01, kernel='linear', probability=True,
                                  random_state=8)),
                             ('LRC',
                              LogisticRegression(C=0.7, penalty='l1',
                                                 random_state=8,
                                                 solver='liblinear')),
                             ('GBC',
                              GradientBoostingClassifier(learning_rate=0.5,
                                                         max_depth=10,
                                                         max_features='auto'
,
                                                         min_samples_leaf=4,
                                                         min_samples_split=1
0,
                                                         n_estimators=600,
                                                         random_state=8))],
                 voting='soft')
```

In [61]:
```python
vot_classifier_3_pred = vot_classifier_3.predict(X_test)

# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, vot_classifier_3_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,vot_classifier_3_pred, target_names=['Non
print("\n")

#Confusion matrix
print(plot_confusion_matrix(vot_classifier_3, X_test, y_test, values_format
```

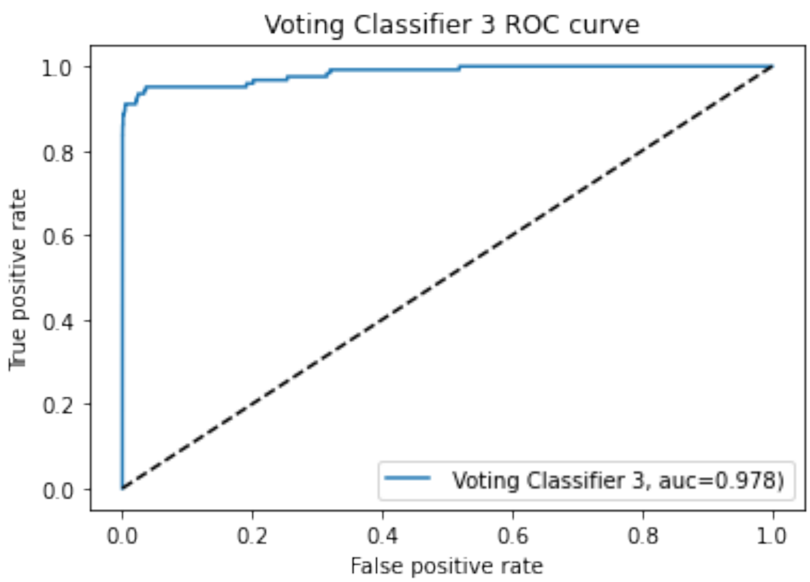
```
The test accuracy is:
0.9897334344540883


Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      0.99     71079
       Fraud       0.13      0.91      0.23       123

    accuracy                           0.99     71202
   macro avg       0.57      0.95      0.61     71202
weighted avg       1.00      0.99      0.99     71202



&lt;sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe099764f0&gt;
```



In [62]:

```
# Plot ROC AUC curve
#AUC
vot_classifier_3_pred_proba = vot_classifier_3.predict_proba(X_test)[::,1]
vot_classifier_3_fpr, vot_classifier_3_tpr, _ = metrics.roc_curve(y_test,  v
vot_classifier_3_auc = metrics.roc_auc_score(y_test, vot_classifier_3_pred_p
print("AUC Voting Classifier 3 :", vot_classifier_3_auc)

#ROC
plt.plot(gbc_fpr,gbc_tpr,label=" Voting Classifier 3, auc={:.3f})".format(vo
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title(' Voting Classifier 3 ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Voting Classifier 3 : 0.9782345694136045
```



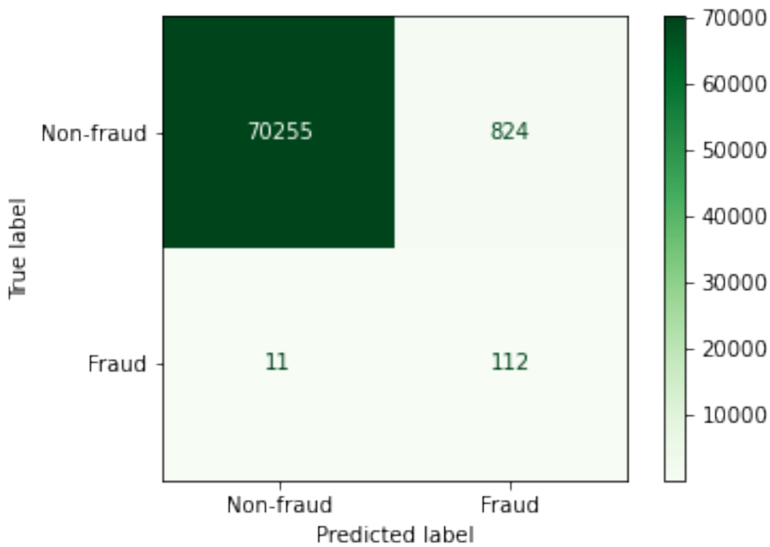
Model 4 - Using RFC, LRC, GBC

In [63]:
```
"""Model 4 - Using RFC, LRC, GBC"""
models= []
models.append(('RFC',best_rfc))
models.append(('LRC',best_lrc))
models.append(('GBC',best_gbc))

vot_classifier_4 = VotingClassifier(estimators=models,voting='soft')
vot_classifier_4.fit(X_train,y_train)
```

Out[63]:
```
VotingClassifier(estimators=[('RFC',
                              RandomForestClassifier(bootstrap=False,
                                                     max_depth=60,
                                                     max_features='sqrt',
                                                     min_samples_split=5,
                                                     n_estimators=200,
                                                     random_state=8)),
                             ('LRC',
                              LogisticRegression(C=0.7, penalty='l1',
                                                 random_state=8,
                                                 solver='liblinear')),
                             ('GBC',
                              GradientBoostingClassifier(learning_rate=0.5,
                                                         max_depth=10,
                                                         max_features='auto'
,
                                                         min_samples_leaf=4,
                                                         min_samples_split=1
0,
                                                         n_estimators=600,
                                                         random_state=8))],
                 voting='soft')
```

In [64]:
```
vot_classifier_4_pred = vot_classifier_4.predict(X_test)

# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(y_test, vot_classifier_4_pred))
print("\n")

# Classification report
print("Classification report")
print(classification_report(y_test,vot_classifier_4_pred, target_names=['Non
print("\n")

#Confusion matrix
print(plot_confusion_matrix(vot_classifier_4, X_test, y_test, values_format
```

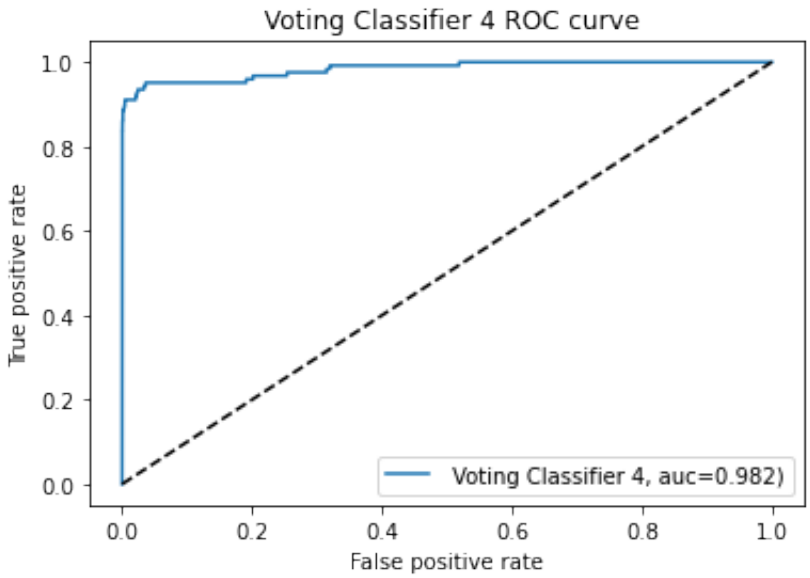
```
The test accuracy is:
0.9882728013258054


Classification report
              precision    recall  f1-score   support

   Non-fraud       1.00      0.99      0.99     71079
       Fraud       0.12      0.91      0.21       123

    accuracy                           0.99     71202
   macro avg       0.56      0.95      0.60     71202
weighted avg       1.00      0.99      0.99     71202


<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7
fbe099dbe50>
```



In [65]:

```
# Plot ROC AUC curve
#AUC
vot_classifier_4_pred_proba = vot_classifier_4.predict_proba(X_test)[::,1]
vot_classifier_4_fpr, vot_classifier_4_tpr, _ = metrics.roc_curve(y_test,  v
vot_classifier_4_auc = metrics.roc_auc_score(y_test, vot_classifier_4_pred_p
print("AUC Voting Classifier 4 :", vot_classifier_4_auc)

#ROC
plt.plot(gbc_fpr,gbc_tpr,label=" Voting Classifier 4, auc={:.3f})".format(vc
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title(' Voting Classifier 4 ROC curve')
plt.legend(loc=4)
plt.show()
```

```
AUC Voting Classifier 4 : 0.9818280747277991
```



Save all the models as pickle files

In [66]:
```
import pickle

#LRC
with open('/Users/aswathchary/Desktop/Picklefiles/best_LRC_fraud_detection.p
    pickle.dump(best_lrc,model_file)
#SVC
with open('/Users/aswathchary/Desktop/Picklefiles/best_SVC_fraud_detection.p
    pickle.dump(best_svc,model_file)
#RFC
with open('/Users/aswathchary/Desktop/Picklefiles/best_RFC_fraud_detection.p
    pickle.dump(best_rfc,model_file)
#GBC
with open('/Users/aswathchary/Desktop/Picklefiles/best_GBC_fraud_detection.p
    pickle.dump(best_gbc,model_file)

with open('/Users/aswathchary/Desktop/Picklefiles/voting_ensemble_1_fraud_de
    pickle.dump(vot_classifier_1,model_file)

with open('/Users/aswathchary/Desktop/Picklefiles/voting_ensemble_2_fraud_de
    pickle.dump(vot_classifier_2,model_file)

with open('/Users/aswathchary/Desktop/Picklefiles/voting_ensemble_3_fraud_de
    pickle.dump(vot_classifier_3,model_file)

with open('/Users/aswathchary/Desktop/Picklefiles/voting_ensemble_4_fraud_de
    pickle.dump(vot_classifier_4,model_file)
```

Final Model Selection

For selecting the final model, we can compare the model performance metrics for the various models. We can also look at the **Recall score** of our models to see how good the models are in classifying fraudulent and non-fraduelent transcations.

In [67]:
```
from sklearn.metrics import recall_score
```

In [68]:
```
# let us create a dataset with a model summary to compare models:
d = {
     'Model': 'Logistic Regression (LRC)',
     'Training Set Accuracy': accuracy_score(y_train, best_lrc.predict(X_tra
     'Test Set Accuracy': accuracy_score(y_test, lrc_pred),
     'AUC Score' : LRC_auc,
     'Recall score': recall_score(y_test, lrc_pred)

}

df_models_lrc = pd.DataFrame(d, index=[0])
print(df_models_lrc)
```

```
                        Model  Training Set Accuracy  Test Set Accuracy  \
0   Logistic Regression (LRC)                0.96437            0.98673

   AUC Score  Recall score
0    0.97023       0.91870
```

In [69]:
```
#SVC
d = {
     'Model': 'Support Vector Classifier (SVC)',
     'Training Set Accuracy': accuracy_score(y_train, best_svc.predict(X_tra
     'Test Set Accuracy': accuracy_score(y_test, svc_pred),
     'AUC Score' : svc_auc,
     'Recall score': recall_score(y_test, svc_pred)

}

df_models_svc = pd.DataFrame(d, index=[0])

#RFC
d = {
     'Model': 'Random Forest Classifier (RFC)',
     'Training Set Accuracy': accuracy_score(y_train, best_rfc.predict(X_tra
     'Test Set Accuracy': accuracy_score(y_test, rfc_pred),
     'AUC Score' : rfc_auc,
     'Recall score': recall_score(y_test, rfc_pred)

}

df_models_rfc = pd.DataFrame(d, index=[0])

#GBC
d = {
```

```
    'Model': 'Gradient Boosting Classifier (GBC)',
    'Training Set Accuracy': accuracy_score(y_train, best_gbc.predict(X_tra
    'Test Set Accuracy': accuracy_score(y_test, gbc_pred),
    'AUC Score' : gbc_auc,
    'Recall score': recall_score(y_test, gbc_pred)

}

df_models_gbc = pd.DataFrame(d, index=[0])

#Voting Classifier 1
d = {
    'Model': 'Voting Classifier 1 (RFC, GBC & SVC)',
    'Training Set Accuracy': accuracy_score(y_train, vot_classifier_1.predi
    'Test Set Accuracy': accuracy_score(y_test, vot_classifier_1_pred),
    'AUC Score' : vot_classifier_1_auc,
    'Recall score': recall_score(y_test, vot_classifier_1_pred)

}

df_models_vot_1 = pd.DataFrame(d, index=[0])

#Voting Classifier 2
d = {
    'Model': 'Voting Classifier 2 (RFC, SVC & LRC)',
    'Training Set Accuracy': accuracy_score(y_train, vot_classifier_2.predi
    'Test Set Accuracy': accuracy_score(y_test, vot_classifier_2_pred),
    'AUC Score' : vot_classifier_2_auc,
    'Recall score': recall_score(y_test, vot_classifier_2_pred)

}

df_models_vot_2 = pd.DataFrame(d, index=[0])

#Voting Classifier 3
d = {
    'Model': 'Voting Classifier 3 (SVC, LRC & GBC)',
    'Training Set Accuracy': accuracy_score(y_train, vot_classifier_3.predi
    'Test Set Accuracy': accuracy_score(y_test, vot_classifier_3_pred),
    'AUC Score' : vot_classifier_3_auc,
    'Recall score': recall_score(y_test, vot_classifier_3_pred)

}

df_models_vot_3 = pd.DataFrame(d, index=[0])

#Voting Classifier 4
d = {
    'Model': 'Voting Classifier 4 (RFC, LRC & GBC)',
    'Training Set Accuracy': accuracy_score(y_train, vot_classifier_4.predi
    'Test Set Accuracy': accuracy_score(y_test, vot_classifier_4_pred),
    'AUC Score' : vot_classifier_4_auc,
    'Recall score': recall_score(y_test, vot_classifier_4_pred)

}

df_models_vot_4 = pd.DataFrame(d, index=[0])
```

In [70]:
```
# Combine all the results together
final_results_df = pd.concat([df_models_lrc, df_models_svc, df_models_rfc, d
final_results_df = final_results_df.reset_index().drop('index', axis=1)
```

In [71]:
```
#sort by AUC and recall score to get the best model
final_results_df.sort_values(['AUC Score', 'Recall score'], ascending=[False
```

Out[71]:

	Model	Training Set Accuracy	Test Set Accuracy	AUC Score	Recall score
3	Gradient Boosting Classifier (GBC)	1.00000	0.99132	0.98406	0.91057
2	Random Forest Classifier (RFC)	1.00000	0.99132	0.98307	0.91057
4	Voting Classifier 1 (RFC, GBC & SVC)	1.00000	0.98898	0.98297	0.91057
7	Voting Classifier 4 (RFC, LRC & GBC)	1.00000	0.98827	0.98183	0.91057
5	Voting Classifier 2 (RFC, SVC & LRC)	0.97134	0.99073	0.98163	0.91057
6	Voting Classifier 3 (SVC, LRC & GBC)	0.97134	0.98973	0.97823	0.91057
1	Support Vector Classifier (SVC)	0.95740	0.99330	0.97604	0.91057
0	Logistic Regression (LRC)	0.96437	0.98673	0.97023	0.91870

THE BEST MODEL IS GRADIENT BOOSTING CLASSIFIER

To sum up,

- All the models have produced good results in both training and testing with very minute differences in performance.
- Given that the dataset is imbalanced, getting a good accuracy is not the goal. However, **the high AUC scores and Recall scores** indicate that the models are quite robust in classifiying fraudulent transactions and non-fraudelent transcations.
- Also, during the model training phase, **by using Randomized Search and Grid Search Cross Validation for Hyperparameter Tuning, the training accuracy of the models improved considerably in comparison with the base estimators (without hyperparameter tuning).**

In [ ]: