In [1]:
```
#importing librarys
import pandas as pd;
```

In [2]:
```
dataset=pd.read_csv('C:/Users/Prachi/Desktop/MBA/Fall 2023/Data analytics for busir
```

In [3]:
```
dataset.columns
```

Out[3]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

In [4]:
```
dataset.isnull().values.any()
```

Out[4]:
```
False
```

In [5]:
```
dataset.dtypes
```

Out[5]:
```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
campaign      int64
pdays         int64
previous      int64
poutcome     object
y            object
dtype: object
```

In [6]:
```
dataset['age'].describe()
```

Out[6]:
```
count    45211.000000
mean        40.936210
std         10.618762
min         18.000000
25%         33.000000
50%         39.000000
75%         48.000000
max         95.000000
Name: age, dtype: float64
```

In [7]:
```
dataset['job'].unique()
```

Out[7]:
```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

In [8]:
```
dataset['marital'].unique()
```

Out[8]:
```
array(['married', 'single', 'divorced'], dtype=object)
```

In [9]:
```
dataset['education'].unique()
```

Out[9]:
```
array(['tertiary', 'secondary', 'unknown', 'primary'], dtype=object)
```

In [10]:
```
dataset['education'].value_counts()
```

Out[10]:
```
education
secondary    23202
tertiary     13301
primary       6851
unknown       1857
Name: count, dtype: int64
```

In [11]:
```
dataset['default'].unique()
```

Out[11]:
```
array(['no', 'yes'], dtype=object)
```

In [12]:
```
dataset['balance'].describe()
```

Out[12]:
```
count     45211.000000
mean       1362.272058
std        3044.765829
min       -8019.000000
25%          72.000000
50%         448.000000
75%        1428.000000
max      102127.000000
Name: balance, dtype: float64
```

In [13]:
```
dataset['housing'].unique()
```

Out[13]:
```
array(['yes', 'no'], dtype=object)
```

In [14]:
```
dataset['loan'].unique()
```

Out[14]:
```
array(['no', 'yes'], dtype=object)
```

In [15]:
```
dataset['contact'].unique()
```

Out[15]:
```
array(['unknown', 'cellular', 'telephone'], dtype=object)
```

In [16]:
```
dataset['day'].unique()
```

Out[16]:
```
array([ 5,  6,  7,  8,  9, 12, 13, 14, 15, 16, 19, 20, 21, 23, 26, 27, 28,
       29, 30,  2,  3,  4, 11, 17, 18, 24, 25,  1, 10, 22, 31],
      dtype=int64)
```

In [17]:
```
dataset['month'].unique()
```

Out[17]:
```
array(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb',
       'mar', 'apr', 'sep'], dtype=object)
```

In [18]:
```
dataset['duration'].describe()
```

Out[18]:
```
count    45211.000000
mean       258.163080
std        257.527812
min          0.000000
25%        103.000000
50%        180.000000
75%        319.000000
max       4918.000000
Name: duration, dtype: float64
```

In [19]:
```
dataset['pdays'].describe()
```

Out[19]:
```
count    45211.000000
mean        40.197828
std        100.128746
min         -1.000000
25%         -1.000000
50%         -1.000000
75%         -1.000000
max        871.000000
Name: pdays, dtype: float64
```

In [20]:
```
dataset['previous'].unique()
```

Out[20]:
```
array([  0,   3,   1,   4,   2,  11,  16,   6,   5,  10,  12,   7,  18,
         9,  21,   8,  14,  15,  26,  37,  13,  25,  20,  27,  17,  23,
        38,  29,  24,  51, 275,  22,  19,  30,  58,  28,  32,  40,  55,
        35,  41], dtype=int64)
```

In [21]:
```
dataset['poutcome'].unique()
```

Out[21]:
```
array(['unknown', 'failure', 'other', 'success'], dtype=object)
```

In [22]:
```
dataset['y'].value_counts()
```

Out[22]:
```
y
no     39922
yes     5289
Name: count, dtype: int64
```

In [23]:
```
dataset['y'].unique()
```

Out[23]:
```
array(['no', 'yes'], dtype=object)
```

In [24]:
```
dataset['campaign'].unique()
```

Out[24]:
```
array([ 1,  2,  3,  5,  4,  6,  7,  8,  9, 10, 11, 12, 13, 19, 14, 24, 16,
       32, 18, 22, 15, 17, 25, 21, 43, 51, 63, 41, 26, 28, 55, 50, 38, 23,
       20, 29, 31, 37, 30, 46, 27, 58, 33, 35, 34, 36, 39, 44],
      dtype=int64)
```

In [25]:
```
#creating dummy variable for marital as marital_divorsed, ,maritial_single, maritic
dataset = pd.concat([dataset, pd.get_dummies(dataset['marital']).rename(columns=lam
dataset.drop(['marital'], inplace=True, axis=1)
```

In [26]:
```
#creating dummy variable for education as education_primary, education_secondary, e
dataset = pd.concat([dataset, pd.get_dummies(dataset['education']).rename(columns=l
dataset.drop(['education'], inplace=True, axis=1)
```

In [27]:
```
#creating dummy variable for contact as contact_cellular, contact_unknown, contact_
dataset = pd.concat([dataset, pd.get_dummies(dataset['contact']).rename(columns=lam
dataset.drop(['contact'], inplace=True, axis=1)
```

In [28]:
```
#creating dummy variable for poutcome as poutcome_unknown, poutcome_failure, poutcc
dataset = pd.concat([dataset, pd.get_dummies(dataset['poutcome']).rename(columns=la
dataset.drop(['poutcome'], inplace=True, axis=1)
```

In [29]:
```
#creating dummy variable for job as job_unemployed, job_services, job_management, j
#     job_housemaid, job_retired, job_unknown
dataset = pd.concat([dataset, pd.get_dummies(dataset['job']).rename(columns=lambda
dataset.drop(['job'], inplace=True, axis=1)
```

In [30]:
```
#creating dummy variable for month as month_oct, month_may, month_apr, month_jun, m
dataset = pd.concat([dataset, pd.get_dummies(dataset['month']).rename(columns=lambd
dataset.drop(['month'], inplace=True, axis=1)
```

In [31]:
```
# replace the 'default','housing", "loan" column contains
# the values 'yes' and 'no'  with
# True and  False:
dataset = dataset.replace( {'yes':1 ,'no':0})
  # replace the 'other dummy variables' column contains
# the values 'yes' and 'no'  with
# True and  False:
dataset = dataset.replace( {True:1 ,False:0})
```

In [32]:
```
y=dataset['y']
df=dataset
df.drop(['y'], inplace=True, axis=1)


X=df
```

In [33]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [34]:
```
from sklearn.linear_model import LogisticRegression
# instantiate the model (using the default parameters)
logreg = LogisticRegression(random_state=16)
# fit the model with data
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

```
C:\Users\misht\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

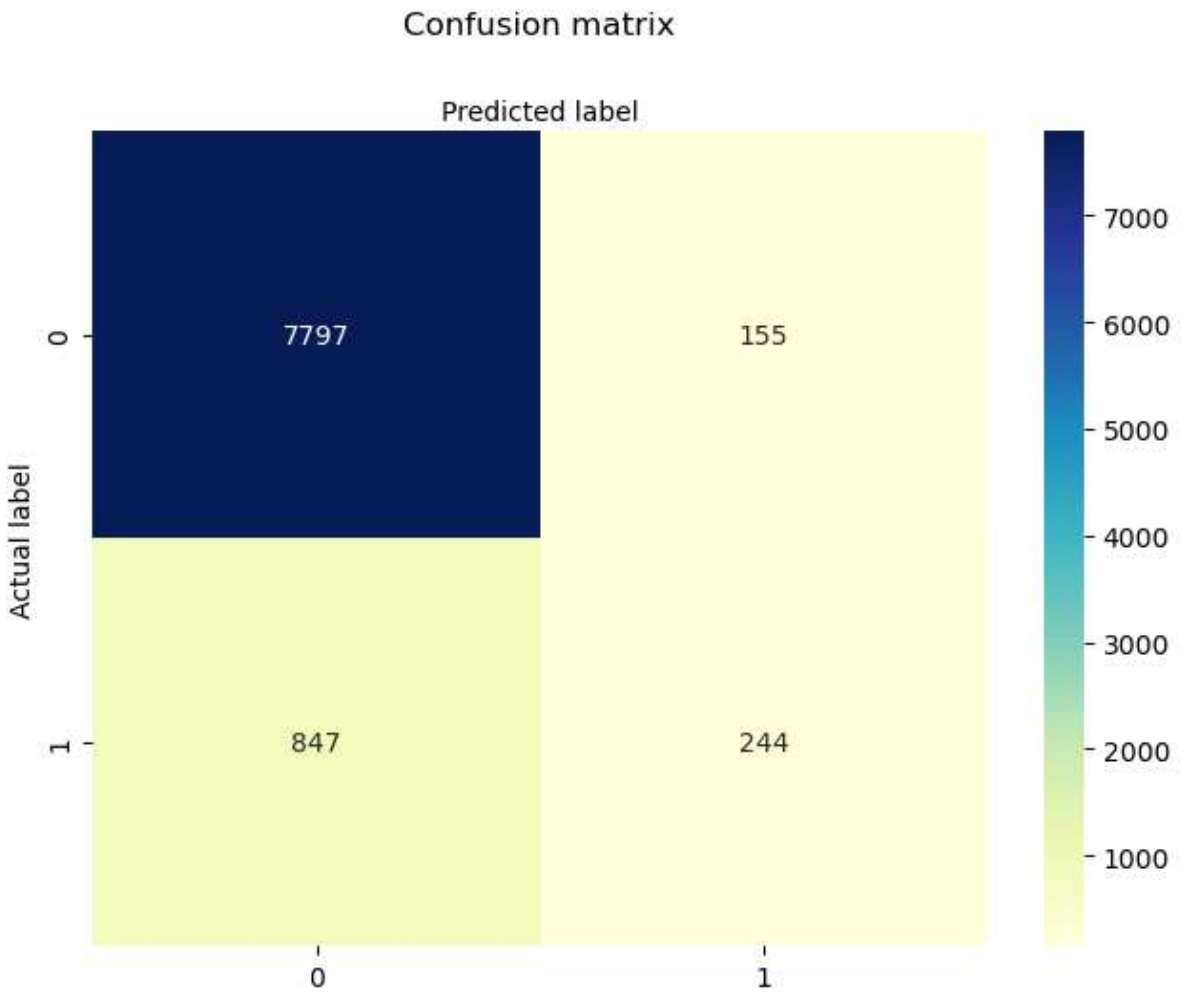
In [35]:
```
from sklearn import metrics

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[35]:
```
array([[7797,  155],
       [ 847,  244]], dtype=int64)
```

In [36]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=['no','yes'] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
```

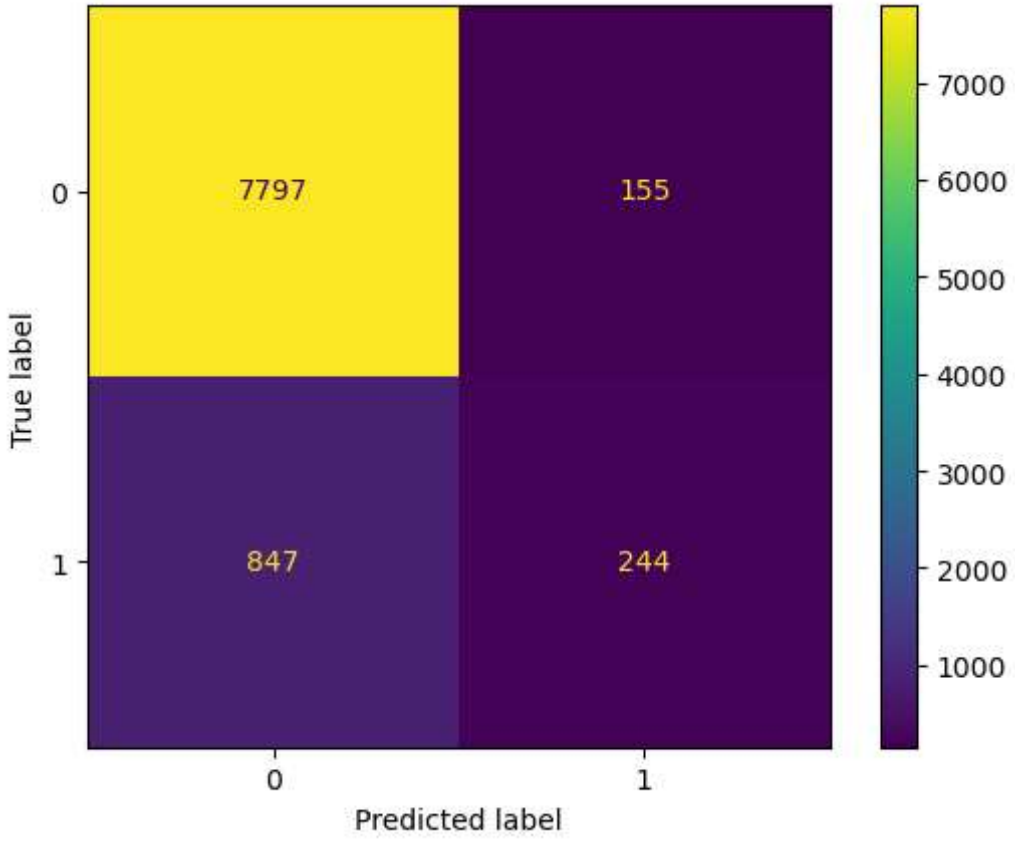
```
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[36]: Text(0.5, 427.9555555555555, 'Predicted label')



In [37]:

```
# Create the confusion matrix
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, reca
ConfusionMatrixDisplay(confusion_matrix=cnf_matrix).plot();
```



In [38]:
```
from sklearn.metrics import classification_report
target_names = ['Did not subscribed', 'Subscribed']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                   precision    recall  f1-score   support

Did not subscribed       0.90      0.98      0.94      7952
        Subscribed       0.61      0.22      0.33      1091

          accuracy                           0.89      9043
         macro avg       0.76      0.60      0.63      9043
      weighted avg       0.87      0.89      0.87      9043
```

In [39]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8891960632533451
```

In [40]:
```
errors = abs(y_pred - y_test)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.11 degrees.
```

In [41]:
```
#Decision trees

import pandas as pd
from sklearn import tree


#choose from different tunable hyper parameters
clf = tree.DecisionTreeClassifier(max_depth=3,criterion='entropy')

#Printing all the parameters of Decision Trees
print(clf)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

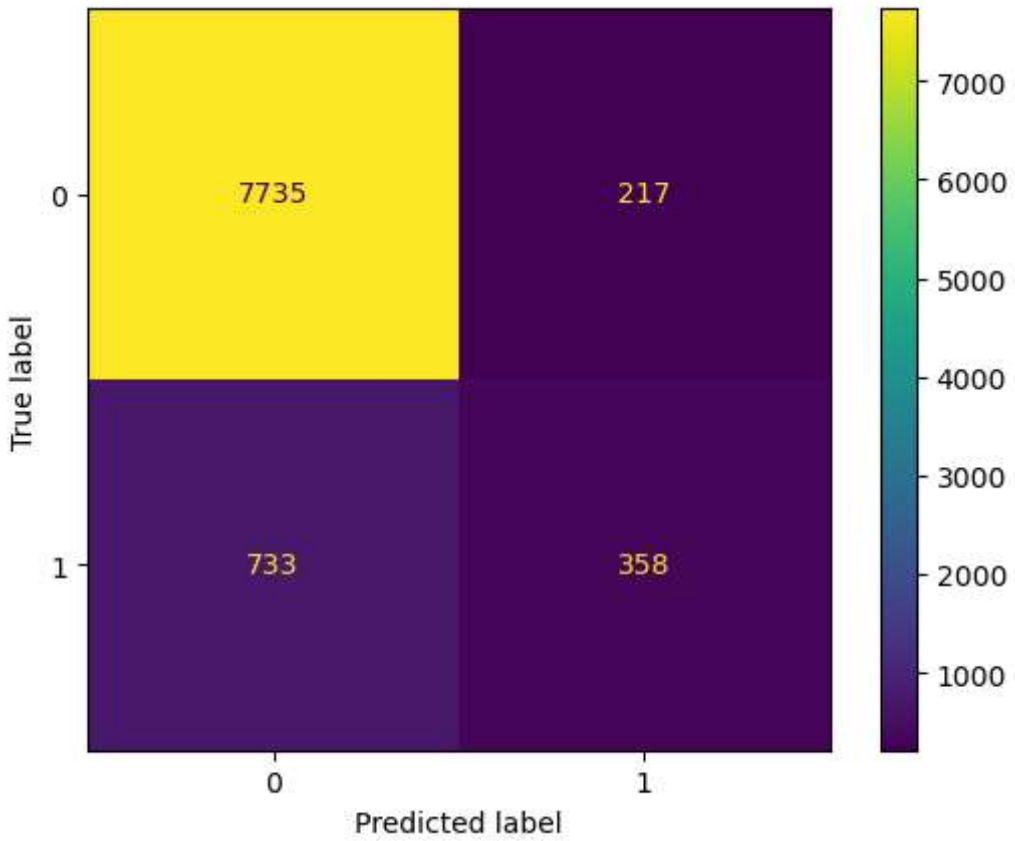
In [42]:
```
#Creating the model on Training Data
DTree=clf.fit(X_train,y_train)
prediction=DTree.predict(X_test)

#Measuring accuracy on Testing Data
from sklearn import metrics
target_names = ['Did not subscribed', 'Subscribed']
print(metrics.classification_report(y_test, prediction,target_names=target_names))
```

```
                    precision    recall  f1-score   support

Did not subscribed       0.91      0.97      0.94      7952
        Subscribed       0.62      0.33      0.43      1091

          accuracy                           0.89      9043
         macro avg       0.77      0.65      0.69      9043
      weighted avg       0.88      0.89      0.88      9043
```

In [43]:
```
print(metrics.confusion_matrix(y_test, prediction))
```

```
[[7735  217]
 [ 733  358]]
```

In [44]:
```
# Create the confusion matrix
cm = confusion_matrix(y_test, prediction)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```



In [45]:
```
print("Accuracy:",metrics.accuracy_score(y_test, prediction))
```

```
Accuracy: 0.8949463673559659
```

In [46]:
```
errors = abs(prediction - y_test)
# Print out the mean absolute error (mae)
```

```
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.11 degrees.
```

In [47]:

```
feature_cols = ['age', 'default', 'balance', 'housing', 'loan', 'day', 'duration',
       'campaign', 'pdays', 'previous', 'marital_divorced', 'marital_married',
       'marital_single', 'education_primary', 'education_secondary',
       'education_tertiary', 'education_unknown', 'contact_cellular',
       'contact_telephone', 'contact_unknown', 'poutcome_failure',
       'poutcome_other', 'poutcome_success', 'poutcome_unknown','job_admin.',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
       'month_apr', 'month_aug', 'month_dec', 'month_feb', 'month_jan',
       'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov',
       'month_oct', 'month_sep']
```

In [48]:

```
X.columns
```

Out[48]:

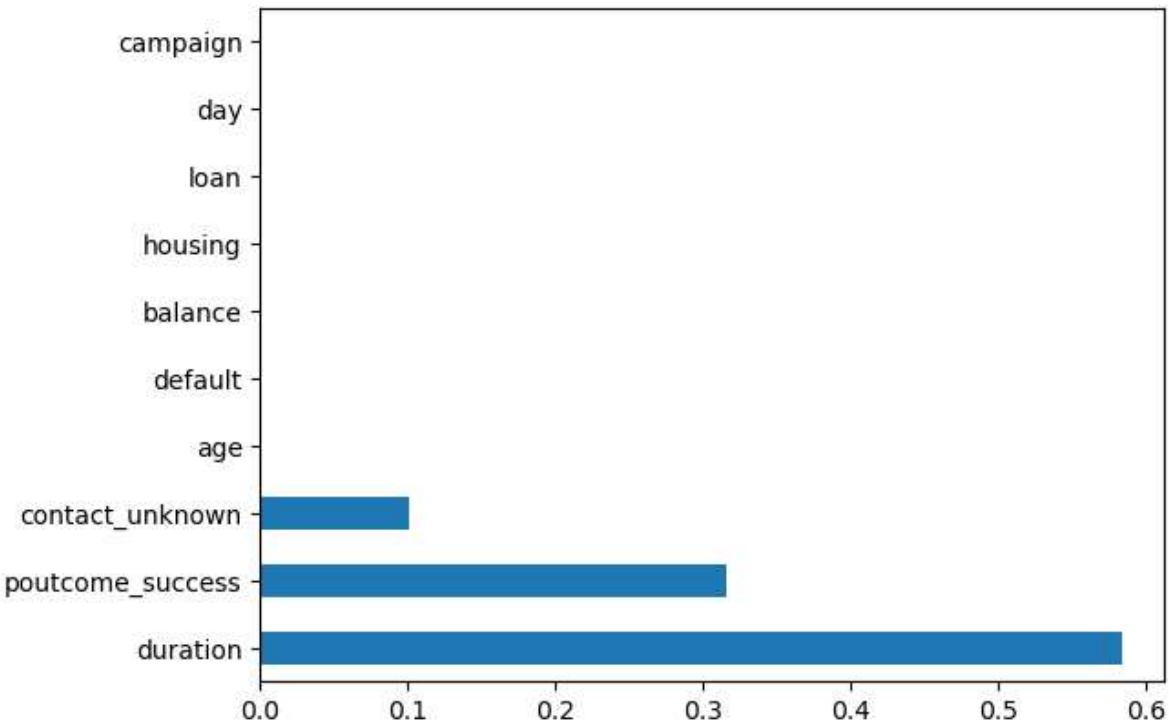
```
Index(['age', 'default', 'balance', 'housing', 'loan', 'day', 'duration',
       'campaign', 'pdays', 'previous', 'marital_divorced', 'marital_married',
       'marital_single', 'education_primary', 'education_secondary',
       'education_tertiary', 'education_unknown', 'contact_cellular',
       'contact_telephone', 'contact_unknown', 'poutcome_failure',
       'poutcome_other', 'poutcome_success', 'poutcome_unknown', 'job_admin.',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
       'month_apr', 'month_aug', 'month_dec', 'month_feb', 'month_jan',
       'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov',
       'month_oct', 'month_sep'],
      dtype='object')
```

In [49]:

```
#Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(DTree.feature_importances_, index=feature_cols)
feature_importances.nlargest(10).plot(kind='barh')
```

Out[49]:

```
<Axes: >
```



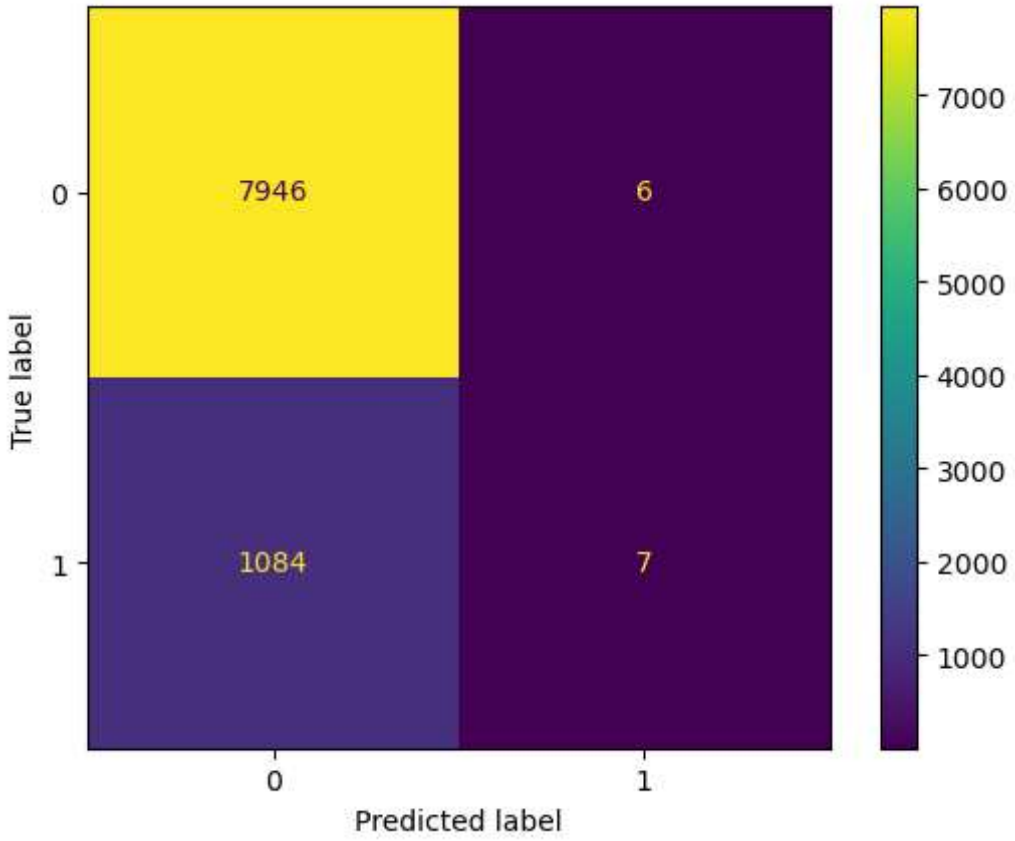
In [50]:
```
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split as tts
from sklearn import svm
from sklearn import metrics
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
y_pr = clf.predict(X_test)
print(y_pr)
```

```
[0 0 0 ... 0 0 0]
```

In [51]:
```
#Measuring accuracy on Testing Data
from sklearn import metrics
target_names = ['Did not subscribed', 'Subscribed']
print(metrics.classification_report(y_test, y_pr,target_names=target_names))
```

```
                   precision    recall  f1-score   support

Did not subscribed       0.88      1.00      0.94      7952
        Subscribed       0.54      0.01      0.01      1091

          accuracy                           0.88      9043
         macro avg       0.71      0.50      0.47      9043
      weighted avg       0.84      0.88      0.82      9043
```

In [52]:
```
# Create the confusion matrix
cm = confusion_matrix(y_test, y_pr)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```



In [53]:
```
print(metrics.confusion_matrix(y_test, y_pr))
```

```
[[7946    6]
 [1084    7]]
```

In [54]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pr))
```

```
Accuracy: 0.8794647793873714
```

In [55]:
```
errors = abs(y_pr - y_test)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.12 degrees.
```

In [56]:
```
#random forests
X.describe()
```

Out[56]:

	age	default	balance	housing	loan	day	
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	4521
mean	40.936210	0.018027	1362.272058	0.555838	0.160226	15.806419	25
std	10.618762	0.133049	3044.765829	0.496878	0.366820	8.322476	25
min	18.000000	0.000000	-8019.000000	0.000000	0.000000	1.000000	
25%	33.000000	0.000000	72.000000	0.000000	0.000000	8.000000	10
50%	39.000000	0.000000	448.000000	1.000000	0.000000	16.000000	18
75%	48.000000	0.000000	1428.000000	1.000000	0.000000	21.000000	31
max	95.000000	1.000000	102127.000000	1.000000	1.000000	31.000000	491

8 rows × 48 columns

In [57]:
```
# Instantiate model with 1000 decision trees
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, reca
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
rfc = RandomForestClassifier()

# Train the model on training data
rfc.fit(X_train,y_train);
```

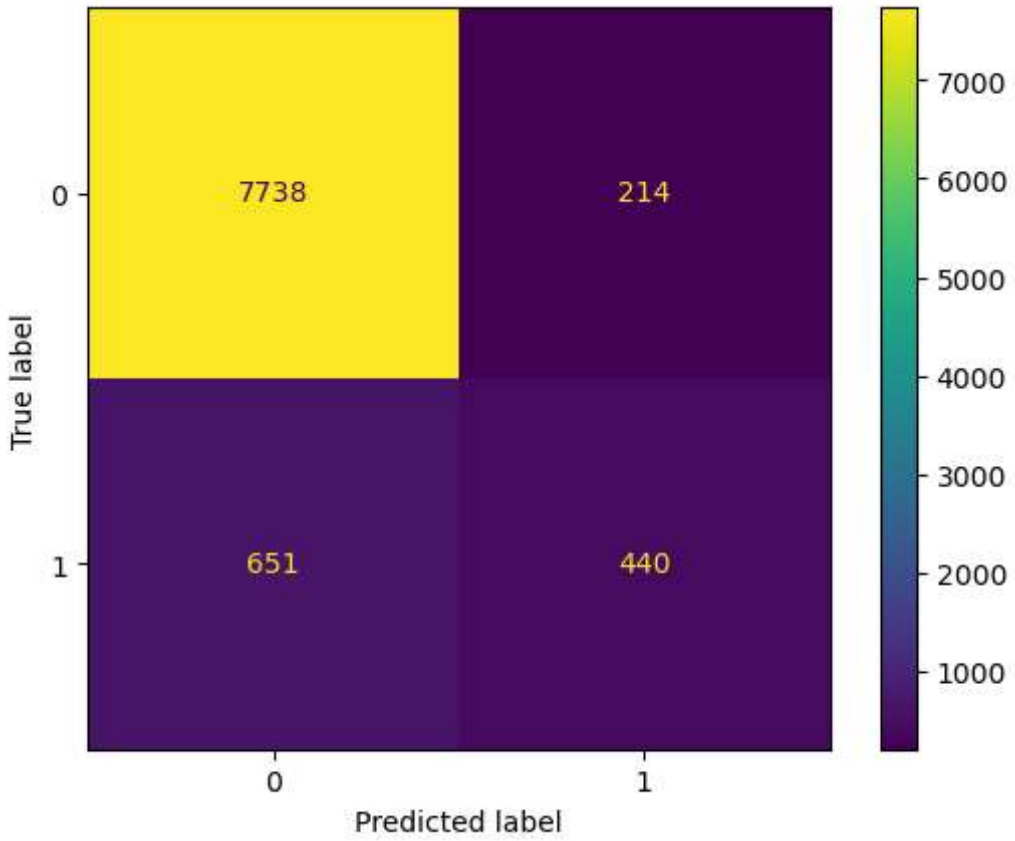
In [58]:
```
import numpy as np
# Use the forest's predict method on the test data
predictions = rfc.predict(X_test)
```

In [59]:
```
#Measuring accuracy on Testing Data
from sklearn import metrics
target_names = ['Did not subscribed', 'Subscribed']
print(metrics.classification_report(y_test, predictions,target_names=target_names))
print(metrics.confusion_matrix(y_test, predictions))
```

```
                    precision    recall  f1-score   support

Did not subscribed       0.92      0.97      0.95      7952
        Subscribed       0.67      0.40      0.50      1091

          accuracy                           0.90      9043
         macro avg       0.80      0.69      0.73      9043
      weighted avg       0.89      0.90      0.89      9043

[[7738  214]
 [ 651  440]]
```

In [60]:

```
# Create the confusion matrix
cm = confusion_matrix(y_test, predictions)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```



In [61]:

```
print("Accuracy:",metrics.accuracy_score(y_test, predictions))
```

```
Accuracy: 0.9043459029083268
```

In [62]:

```
# Calculate the absolute errors
errors = abs(predictions - y_test)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.1 degrees.
```

In [ ]: