In [1]:

```
#DECLARING LIBRARIES


#basic libraries
import math
import numpy as np

#visualization and plotting
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns

#sklearn
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score, roc_curve
from sklearn.metrics import precision_recall_curve, auc, make_scorer, recall_score, accuracy_score, precision_score, confusion_matrix
from sklearn.utils import class_weight
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor #RANDOM FOREST ALGORITHM

#oversampling with imbalanced classes
from imblearn.over_sampling import SMOTE

#time and data related
import time
import datetime
from time import mktime
from datetime import timezone

# Seaborn visualization library
import seaborn as sns

#pandas dataframe
import pandas as pd
```

In [2]:

```
#READING DATASET

#open training dataset
file_handler = open("training2APP.csv", "r")
df_train= pd.read_csv(file_handler, sep = ";")
file_handler.close()
instances_train=df_train.shape[0] #count the number of instances in the training set

#open test dataset
file_handler = open("test2APP.csv", "r")
df_test= pd.read_csv(file_handler, sep = ";")
instances_test=df_test.shape[0]#count the number of instances in the test set
file_handler.close()

#visualizing test dataset
df_train.head(10)
```

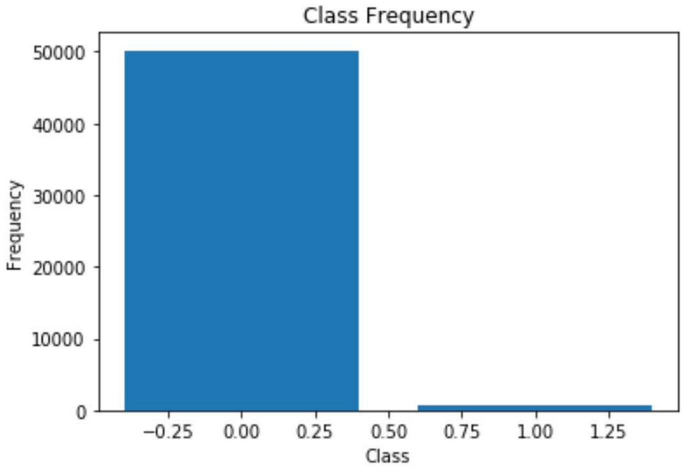
Out[2]:

| | Application | Date_mesure | Nb_requetes | comparaison_prec | Erreur_4xx5xx | Ratio_ |
|---|---|---|---|---|---|---|
| **0** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:00 | 1033 | 100 | 11 | |
| **1** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:05 | 1896 | 183 | 23 | |
| **2** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:10 | 2045 | 107 | 14 | |
| **3** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:15 | 2116 | 103 | 19 | |
| **4** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:20 | 2060 | 97 | 10 | |
| **5** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:25 | 2176 | 105 | 8 | |
| **6** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:30 | 2235 | 102 | 10 | |
| **7** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:35 | 2319 | 103 | 15 | |
| **8** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:40 | 2335 | 100 | 13 | |
| **9** | CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916 | 15/10/2018 8:45 | 2351 | 100 | 25 | |

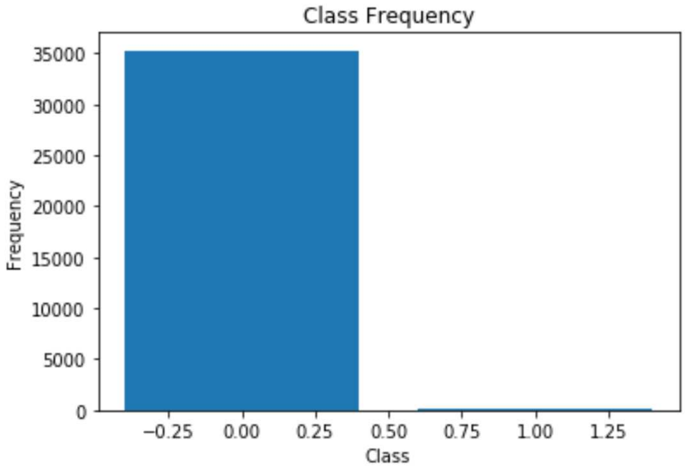
In [3]:

```
#DATA PREPARATION: only incidents are tagged, we must tag as zero the non incidents replace in incidents empty spaces by zeros

#evidence the imbalance in classes
target_count_training=df_train['Incident_global'].value_counts()

print('TRAINING: Class 0 (Non incident):', target_count_training[0])
print('TRAINING: Class 1(Incident):', target_count_training[1])
print('TRAINING: Incident Proportion (%): ', (target_count_training[1] / instances_train)*100)

unique, counts = np.unique(df_train['Incident_global'].values, return_counts=True)
plt.bar(unique,counts)
plt.title('Class Frequency')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()

target_count_test=df_test['Incident_global'].value_counts()

print('TEST: Class 0 (Non incident):', target_count_test[0])
print('TEST: Class 1(Incident):', target_count_test[1])
print('TEST: Incident Proportion (%):', (target_count_test[1] / instances_test)*100)

unique, counts = np.unique(df_test['Incident_global'].values, return_counts=True)
plt.bar(unique,counts)
plt.title('Class Frequency')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

```
TRAINING: Class 0 (Non incident): 50174
TRAINING: Class 1(Incident): 568
TRAINING: Incident Proportion (%):  1.1193882779551456
```



```
TEST: Class 0 (Non incident): 35292
TEST: Class 1(Incident): 206
TEST: Incident Proportion (%): 0.5803143839089526
```



In [4]:

```
#DATA DISCOVERY: PARSING NAMES OF APPLICATIONS TO CATEGORIAL VALUES
print("BEFORE PARSING TO CATEGORICAL VALUE: ")
print(df_train.iloc[0,:])

app = {'CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916': 1,
         'GEOFIBRE_ID_15216': 2,
        'IPSITE_IHM_ALT_IASGP_ID_08685': 3,
        'MONCRM_ALT_ID_17409': 4,
         'ORCHESTRA_ALT_ID_03554': 5,
          'PATH_ALT_ID_12766': 6,
           'SAVI_ALT_ID_04385': 7,
            'SOFT_ALT_ID_12461': 8,
            'SPAS_ALT_ID_04769': 9,
            'SUIVICOM_ALT_ID_04854':10}

#training
df_train.Application = [app[item] for item in df_train.Application]
keys=list(app.keys())
values=list(app.values())
#test
df_test.Application = [app[item] for item in df_test.Application]
keys=list(app.keys())
values=list(app.values())

print("AFTER PARSING TO CATEGORICAL VALUE: ")
print(df_train.iloc[0,:])
```

```
BEFORE PARSING TO CATEGORICAL VALUE: 
Application           CUSTOMER-SERVICES_CUSTOMER_ALT_ID_20916
Date_mesure                                   15/10/2018 8:00
Nb_requetes                                              1033
comparaison_prec                                          100
Erreur_4xx5xx                                              11
Ratio_err                                                   1
Pages_lentes                                                0
Ratio_pages_lentes                                          0
Temps_reponse                                             291
Ratio_tps_rep                                              36
Incident_global                                             0
Name: 0, dtype: object
AFTER PARSING TO CATEGORICAL VALUE: 
Application                         1
Date_mesure           15/10/2018 8:00
Nb_requetes                      1033
comparaison_prec                  100
Erreur_4xx5xx                      11
Ratio_err                           1
Pages_lentes                        0
Ratio_pages_lentes                  0
Temps_reponse                     291
Ratio_tps_rep                      36
Incident_global                     0
Name: 0, dtype: object
```

In [5]:

```
#TIME SERIES: DECOMPOSING DATA_MESURE FEATURE INTO DIFFERENT FEATURES TO EXTRACT
USEFUL KNOWLEDGE
print("BEFORE DECOMPOSING TEMPORAL INFORMATION: ")
print(df_train.iloc[0,:])
#train
df_train['day_of_week']=0
df_train['month_of_the_year']=0
df_train['week_number']=0
df_train['region_hour_of_day']=0
df_train['time_of_day']=0
df_train['season']=0

Weekday= lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).weekday()
Month= lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).month
Strftime= lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).strftime('%
V')

df_train['day_of_week']=df_train['Date_mesure'].map(Weekday)
df_train['month_of_the_year']=df_train['Date_mesure'].map(Month)
df_train['season']=df_train['Date_mesure'].map(Strftime)

seasons = [0,0,1,1,1,2,2,2,3,3,3,0] #dec - feb is winter, then spring, summer, f
all etc
season = lambda x: seasons[(datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).mon
th-1)]
df_train['season']=df_train['Date_mesure'].map(season)

# sleep: 12-5, 6-9: breakfast, 10-14: lunch, 14-17: dinner prep, 17-21: dinner,
21-23: deserts!
hours_of_day = [0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5
, 5 ]
region_hour_of_day = lambda x: hours_of_day[datetime.datetime.strptime(x, "%d/%m
/%Y %H:%M").hour]
hour_of_day = lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M").hour
minute_of_day = lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M").minute

df_train['region_hour_of_day']=df_train['Date_mesure'].map(hour_of_day)
df_train['time_of_day']=df_train['Date_mesure'].map(minute_of_day) + 60*(df_trai
n['Date_mesure'].map(hour_of_day))

#test
df_test['day_of_week']=0
df_test['month_of_the_year']=0
df_test['week_number']=0
df_test['region_hour_of_day']=0
df_test['time_of_day']=0
df_test['season']=0

Weekday= lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).weekday()
Month= lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).month
Strftime= lambda x: datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).strftime('%
V')

df_test['day_of_week']=df_test['Date_mesure'].map(Weekday)
df_test['month_of_the_year']=df_test['Date_mesure'].map(Month)
df_test['season']=df_test['Date_mesure'].map(Strftime)

seasons = [0,0,1,1,1,2,2,2,3,3,3,0] #dec - feb is winter, then spring, summer, f
all etc
season = lambda x: seasons[(datetime.datetime.strptime(x, "%d/%m/%Y %H:%M" ).mon
th-1)]
df_test['season']=df_test['Date_mesure'].map(season)

# sleep: 12-5, 6-9: breakfast, 10-14: lunch, 14-17: dinner prep, 17-21: dinner,
21-23: deserts!
hours_of_day = [0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5
, 5 ]
region_hour_of_day = lambda x: hours_of_day[datetime.datetime.strptime(x, "%d/%m
/%Y %H:%M").hour]
```

```
BEFORE DECOMPOSING TEMPORAL INFORMATION:
Application                         1
Date_mesure           15/10/2018 8:00
Nb_requetes                      1033
comparaison_prec                  100
Erreur_4xx5xx                      11
Ratio_err                           1
Pages_lentes                        0
Ratio_pages_lentes                  0
Temps_reponse                     291
Ratio_tps_rep                      36
Incident_global                     0
Name: 0, dtype: object
AFTER DECOMPOSING TEMPORAL INFORMATION:
Application                         1
Date_mesure           15/10/2018 8:00
Nb_requetes                      1033
comparaison_prec                  100
Erreur_4xx5xx                      11
Ratio_err                           1
Pages_lentes                        0
Ratio_pages_lentes                  0
Temps_reponse                     291
Ratio_tps_rep                      36
Incident_global                     0
day_of_week                         0
month_of_the_year                  10
week_number                         0
region_hour_of_day                  8
time_of_day                       480
season                              3
Name: 0, dtype: object
```

In [6]:

```
#MISSING OBSERVATIONS IN TRAINING AND TEST SETS
df_trainingapp=df_train.groupby('Application')
df_testapp=df_test.groupby('Application')

#FEATURE SELECTION
features = ['region_hour_of_day','time_of_day','Nb_requetes','Erreur_4xx5xx','Ratio_err',
            'Ratio_pages_lentes','Temps_reponse','Application',
            'comparaison_prec','Ratio_tps_rep','Pages_lentes' ]

training_gaps=0
test_gaps=0

for i in range(1,11): #for each app we cluster to find anomalies
    df_app_i_train = df_trainingapp.get_group(i).loc[:,features] #get the application i and the feature j
    df_app_i_test = df_testapp.get_group(i).loc[:,features]
    counts_train=5280-df_app_i_train.shape[0]
    counts_test=3600-df_app_i_test.shape[0]
    training_gaps=training_gaps+counts_train
    test_gaps=test_gaps+counts_test
    print("-----------------------------------------------")
    #print(df_app_i_train.shape[0])
    #print(df_app_i_test.shape[0])
    print("TRAINING: Number of Missing observations in application "+str(keys[values.index(i)])+": "+str(counts_train))
    print("TEST: Number of Missing observations in application "+str(keys[values.index(i)])+": "+str(counts_test))

print("-----------------------------------------------")
print("TRAINING: Number of Missing observations in the dataset "+str(training_gaps))
print("TRAINING: Percentage of missing data "+str((training_gaps/df_train.shape[0])*100))
print("-----------------------------------------------")
print("TEST: Number of Missing observations in the dataset "+str(test_gaps))
print("TEST: Percentage of missing data "+str((test_gaps/df_test.shape[0])*100))
```

```
-----------------------------------------------
TRAINING: Number of Missing observations in application CUSTOMER-SERVICES_CUST
OMER_ALT_ID_20916: 2
TEST: Number of Missing observations in application CUSTOMER-SERVICES_CUSTOMER
_ALT_ID_20916: 0
-----------------------------------------------
TRAINING: Number of Missing observations in application GEOFIBRE_ID_15216: 243
TEST: Number of Missing observations in application GEOFIBRE_ID_15216: 1
-----------------------------------------------
TRAINING: Number of Missing observations in application IPSITE_IHM_ALT_IASGP_I
D_08685: 1
TEST: Number of Missing observations in application IPSITE_IHM_ALT_IASGP_ID_08
685: 0
-----------------------------------------------
TRAINING: Number of Missing observations in application MONCRM_ALT_ID_17409: 2
40
TEST: Number of Missing observations in application MONCRM_ALT_ID_17409: 0
-----------------------------------------------
TRAINING: Number of Missing observations in application ORCHESTRA_ALT_ID_03554
: 120
TEST: Number of Missing observations in application ORCHESTRA_ALT_ID_03554: 0
-----------------------------------------------
TRAINING: Number of Missing observations in application PATH_ALT_ID_12766: 120
TEST: Number of Missing observations in application PATH_ALT_ID_12766: 1
-----------------------------------------------
TRAINING: Number of Missing observations in application SAVI_ALT_ID_04385: 240
TEST: Number of Missing observations in application SAVI_ALT_ID_04385: 0
-----------------------------------------------
TRAINING: Number of Missing observations in application SOFT_ALT_ID_12461: 485
TEST: Number of Missing observations in application SOFT_ALT_ID_12461: 250
-----------------------------------------------
TRAINING: Number of Missing observations in application SPAS_ALT_ID_04769: 605
TEST: Number of Missing observations in application SPAS_ALT_ID_04769: 250
-----------------------------------------------
TRAINING: Number of Missing observations in application SUIVICOM_ALT_ID_04854:
2
TEST: Number of Missing observations in application SUIVICOM_ALT_ID_04854: 0
-----------------------------------------------
TRAINING: Number of Missing observations in the dataset 2058
TRAINING: Percentage of missing data 4.0558117535769185
-----------------------------------------------
TEST: Number of Missing observations in the dataset 502
TEST: Percentage of missing data 1.414164178263564
```

In [ ]:

```
#HISTOGRAM REPRESENTATION PER FEATURE
features = ['week_number','region_hour_of_day','time_of_day','Nb_requetes','Erre
ur_4xx5xx','Ratio_err',
            'Ratio_pages_lentes','Temps_reponse','Application',
            'comparaison_prec','Ratio_tps_rep','Pages_lentes']

feature=['Pages_lentes']

#separate dataset per application
groupedapp=df_train.groupby('Application') #already normalized

#separate dataset per incident
groupedinc=df_train.groupby('Incident_global') #already normalized

application=10
df_app = groupedapp.get_group(application).loc[:,feature] #get the application i
and the feature j
plt.hist(df_app,density=True, bins=50)
#plt.title("Value distribution of the feature' " + str(feature) + "' in applicat
ion " + str(keys[values.index(application)]))
```

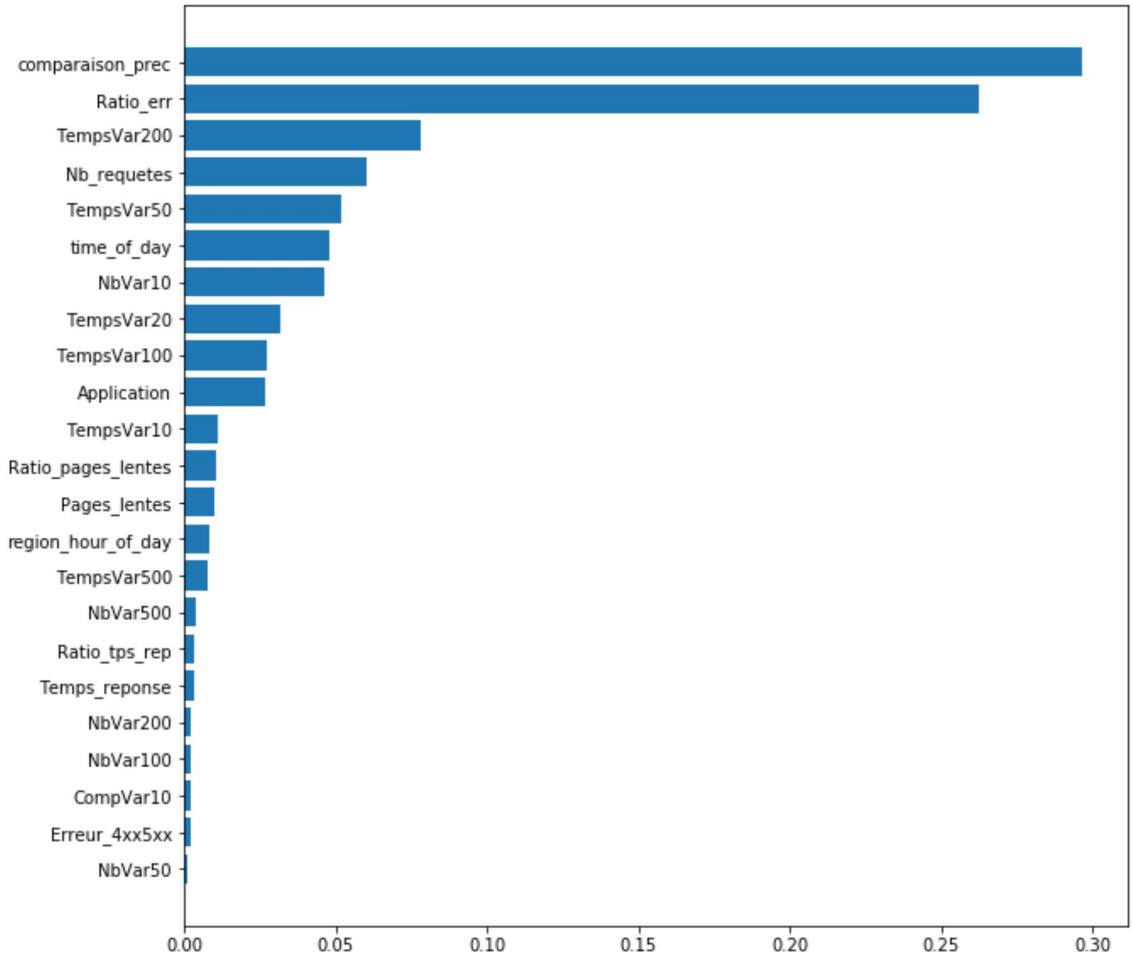
In [7]:

```
#ADDITION OF NEW FEATURES BASED ON PERCENTUAL VARIATION
#COMMENT DIFF
features2=['Nb_requetes','Temps_reponse','comparaison_prec'] #features over whic
h i will compute percentage variation

print("BEFORE ADDING PORCENTAGE VARIATION INFORMATION: ")
print(df_train.iloc[50,:])
#training
df_train10=df_train.loc[:,features2].pct_change(periods=1)
df_train10 = df_train10.mask(np.isinf(df_train10))
df_train10=df_train10.fillna(0)
df_train20=df_train.loc[:,features2].pct_change(periods=5)
df_train20 = df_train20.mask(np.isinf(df_train20))
df_train20=df_train20.fillna(0)
df_train50=df_train.loc[:,features2].pct_change(periods=10)
df_train50 = df_train50.mask(np.isinf(df_train50))
df_train50=df_train50.fillna(0)
df_train100=df_train.loc[:,features2].pct_change(periods=15)
df_train100 = df_train100.mask(np.isinf(df_train100))
df_train100=df_train100.fillna(0)
df_train200=df_train.loc[:,features2].pct_change(periods=20)
df_train200 = df_train200.mask(np.isinf(df_train200))
df_train200=df_train200.fillna(0)
df_train500=df_train.loc[:,features2].pct_change(periods=25)
df_train500 = df_train500.mask(np.isinf(df_train500))
df_train500=df_train500.fillna(0)

#test
df_test10=df_test.loc[:,features2].pct_change(periods=1)
df_test10 = df_test10.mask(np.isinf(df_test10))
df_test10=df_test10.fillna(0)
df_test20=df_test.loc[:,features2].pct_change(periods=5)
df_test20 = df_test20.mask(np.isinf(df_test20))
df_test20=df_test20.fillna(0)
df_test50=df_test.loc[:,features2].pct_change(periods=10)
df_test50 = df_test50.mask(np.isinf(df_test50))
df_test50=df_test50.fillna(0)
df_test100=df_test.loc[:,features2].pct_change(periods=15)
df_test100 = df_test100.mask(np.isinf(df_test100))
df_test100=df_test100.fillna(0)
df_test200=df_test.loc[:,features2].pct_change(periods=20)
df_test200 = df_test200.mask(np.isinf(df_test200))
df_test200=df_test200.fillna(0)
df_test500=df_test.loc[:,features2].pct_change(periods=25)
df_test500 = df_test500.mask(np.isinf(df_test500))
df_test500=df_test500.fillna(0)

df_train10.rename(columns={'Nb_requetes':'NbVar10','Temps_reponse':'TempsVar10',
                        'comparaison_prec':'CompVar10'},inplace=True)
df_test10.rename(columns={'Nb_requetes':'NbVar10','Temps_reponse':'TempsVar10',
                        'comparaison_prec':'CompVar10'},inplace=True)
df_train20.rename(columns={'Nb_requetes':'NbVar20','Temps_reponse':'TempsVar20',
                        'comparaison_prec':'CompVar20'},inplace=True)
df_test20.rename(columns={'Nb_requetes':'NbVar20','Temps_reponse':'TempsVar20',
                        'comparaison_prec':'CompVar20'},inplace=True)
df_train50.rename(columns={'Nb_requetes':'NbVar50','Temps_reponse':'TempsVar50',
                        'comparaison_prec':'CompVar50'},inplace=True)
df_test50.rename(columns={'Nb_requetes':'NbVar50','Temps_reponse':'TempsVar50',
                        'comparaison_prec':'CompVar50'},inplace=True)
df_train100.rename(columns={'Nb_requetes':'NbVar100','Temps_reponse':'TempsVar10
0',
                        'comparaison_prec':'CompVar100'},inplace=True)
df_test100.rename(columns={'Nb_requetes':'NbVar100','Temps_reponse':'TempsVar100
',
                        'comparaison_prec':'CompVar100'},inplace=True)
df_train200.rename(columns={'Nb_requetes':'NbVar200','Temps_reponse':'TempsVar20
0',
                        'comparaison_prec':'CompVar200'},inplace=True)
df_test200.rename(columns={'Nb_requetes':'NbVar200','Temps_reponse':'TempsVar200
BEFORE ADDING PORCENTAGE VARIATION INFORMATION:
Application                          1
Date_mesure           15/10/2018 12:10
Nb_requetes                       3558
comparaison_prec                   101
Erreur_4xx5xx                       41
Ratio_err                            1
Pages_lentes                         1
Ratio_pages_lentes                   0
Temps_reponse                      294
Ratio_tps_rep                       36
Incident_global                      0
day_of_week                          0
month_of_the_year                   10
week_number                          0
region_hour_of_day                  12
time_of_day                        730
season                               3
Name: 50, dtype: object
-----------------------------------------------
AFTER ADDING PORCENTAGE VARIATION INFORMATION:
Application              1.000000
Nb_requetes           3558.000000
comparaison_prec       101.000000
Erreur_4xx5xx           41.000000
Ratio_err                1.000000
Pages_lentes             1.000000
Ratio_pages_lentes       0.000000
Temps_reponse          294.000000
Ratio_tps_rep           36.000000
day_of_week              0.000000
month_of_the_year       10.000000
week_number              0.000000
region_hour_of_day      12.000000
time_of_day            730.000000
season                   3.000000
NbVar10                  0.012521
TempsVar10              -0.092593
CompVar10                0.086022
NbVar20                 -0.188227
TempsVar20              -0.111782
CompVar20                0.010000
NbVar50                 -0.203314
TempsVar50              -0.092593
CompVar50               -0.038095
NbVar100                -0.136827
TempsVar100             -0.114458
CompVar100              -0.019417
NbVar200                -0.161837
TempsVar200             -0.111782
CompVar200              -0.028846
NbVar500                -0.128155
TempsVar500             -0.075472
CompVar500               0.000000
Name: 50, dtype: float64
```

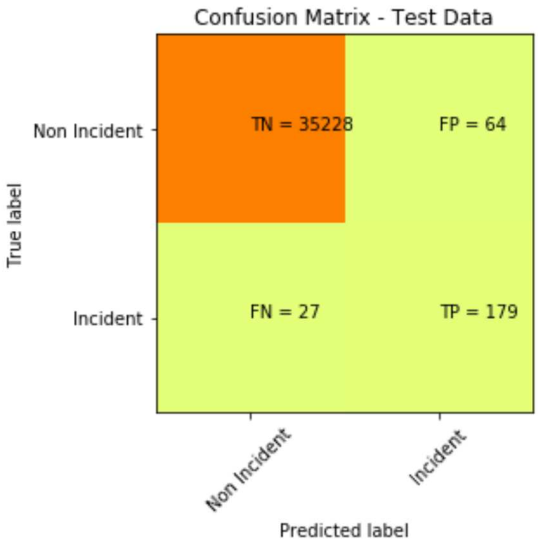
In [8]:

```
#TRAINING PHASE
# Separating out the target
y_test= df_test.loc[:,['Incident_global']].values
y_train= df_train.loc[:,['Incident_global']].values
X_train = df_train_final.loc[:, features].values
X_test = df_test_final.loc[:, features].values

#oversampling
#sm = SMOTE(random_state=2)
#sm = RandomOverSampler(sampling_strategy='minority')
#X_train, y_train = sm.fit_resample(X_train, y_train)

#algorithm choice: Random Forest (examples of the same model but with different
hyperparameters)
model1= RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features=n_features, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
            oob_score=True, random_state=44, verbose=0,
            warm_start=False)

model2 = RandomForestClassifier(bootstrap=True, class_weight={0:1,1:5}, criterio
n='gini',
            max_depth=6, max_features=n_features, max_leaf_nodes=23,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
            oob_score=True, random_state=44, verbose=0,
            warm_start=False)

#fitting the trained model
model2.fit(X_train, y_train)

#FEATURE IMPORTANCE
imp=model2.feature_importances_
names=features
plt.figure(figsize=(10,10))
imp,names=zip(*sorted(zip(imp,names)))
plt.barh(range(len(names)),imp,align = 'center')
plt.yticks(range(len(names)),names)
plt.show()

#TEST PHASE
#model prediciton on the test set
y_pred=model2.predict(X_test)

#model evaluation based on confusion matrix
print("Score:", accuracy_score(y_test, y_pred, normalize=True))
print(classification_report(y_test,y_pred))
cm = confusion_matrix(y_test, y_pred)

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['Non Incident','Incident']
plt.title('Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]

for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))

plt.show()
```

```
C:\Users\dnkx4622\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykerne
l_launcher.py:31: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples,), for exampl
e using ravel().
```



```
Score: 0.9974364752943827
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     35292
           1       0.74      0.87      0.80       206

   micro avg       1.00      1.00      1.00     35498
   macro avg       0.87      0.93      0.90     35498
weighted avg       1.00      1.00      1.00     35498
```



```
PRECISION: 0.7366255144032922
RECALL: 0.8689320388349514
F-SCORE: 0.7973273942093541
AUC: 0.9335592983475448
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```