Entrée [65]:

```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

# load datasets
users_features = pd.read_csv("data/Social_spammers_dataset/users_features/features.
labels = pd.read_csv("data/Social_spammers_dataset/users/coded_ids_labels_train.csv
code_ids_label = pd.read_csv("data/Social_spammers_dataset/users/coded_ids.csv")
test_submission = pd.read_csv("data/Social_spammers_dataset/users/coded_ids_labels_
users_features = pd.merge(users_features, code_ids_label, on='user_id')
```

Entrée [66]:

```
# merge features and tarin label
users_features_with_labels = pd.merge(users_features, labels, on='coded_id')
users_features.head()
```

Out[66]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_of_
0	0.055	2.600	
1	154.333	3.447	
2	40.000	9.938	
3	0.334	2.600	
4	4.494	0.000	

5 rows × 146 columns

Exploration of data

Entrée [67]:

```
# informations about data
users_features_with_labels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 686 entries, 0 to 685
Columns: 147 entries, active_tweeting_frequency_per_day to label
dtypes: bool(2), float64(93), int64(44), object(8)
memory usage: 783.8+ KB
```

Let's compute the number of non-number data.

Entrée [68]:

```
# total null values
users_features_with_labels.isna().sum().sum()
```

Out[68]:

```
1022
```

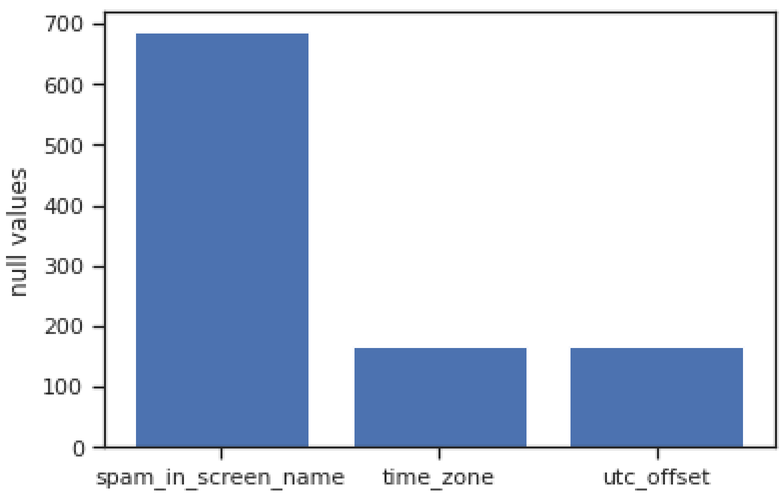
Entrée [69]:

```
# number of null values in columns
import matplotlib.pyplot as plt

########################################################################
columns = list(users_features_with_labels.iloc[0:0,])
x=[]
y=[]
for col in columns :
    s = users_features_with_labels[col].isna().sum()
    if (s != 0):
        print (col)
        x.append(col)
        print(s)
        y.append(s)
    else:
        pass
##########################################################################

plt.bar(x,y,align='center') # A bar chart
plt.ylabel('null values')
fig1 = plt.gcf()
plt.show()
plt.draw()
fig1.savefig("null_values.png", dpi=100)
```

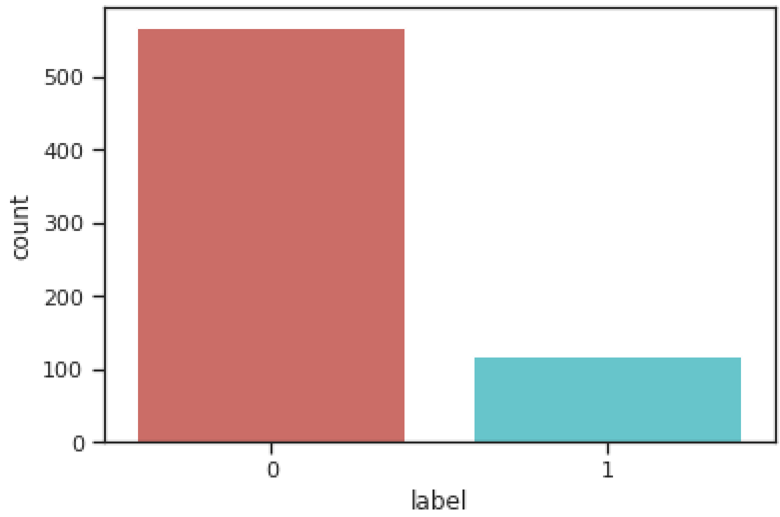
```
spam_in_screen_name
686
time_zone
168
utc_offset
168
```




```
<Figure size 432x288 with 0 Axes>
```

Entrée [70]:

```
# we have unbalanced data
import seaborn as sb
%matplotlib inline
users = users_features_with_labels
sb_label = sb.countplot(x='label', data = users, palette = 'hls')
sb_label
sb_label.figure.savefig("label.png")
```

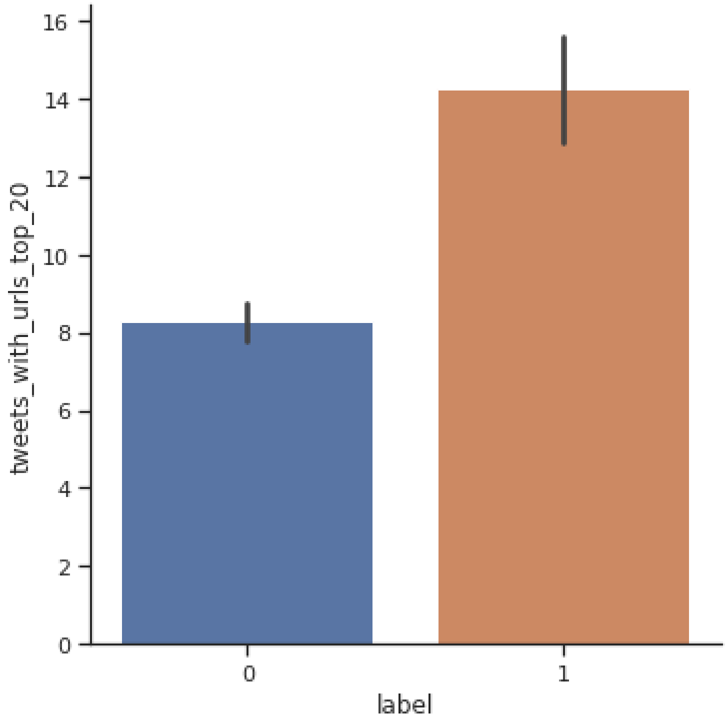


Entrée [71]:

```
# how column influence in detection of spammers
def explore_column (name_column):
    sb_to20 = sb.catplot(y=name_column, x= 'label', kind="bar" ,data = users)
    sb_to20
    sb_to20.savefig(name_column+".png")
```

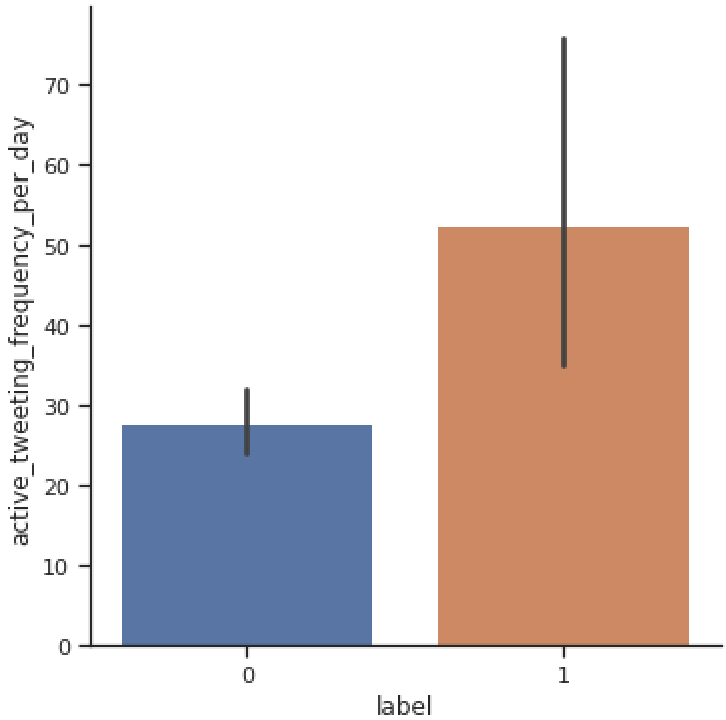
Entrée [72]:

```
# how column tweets_with_urls_top_20 influence in detection of spammers
explore_column ('tweets_with_urls_top_20')
```



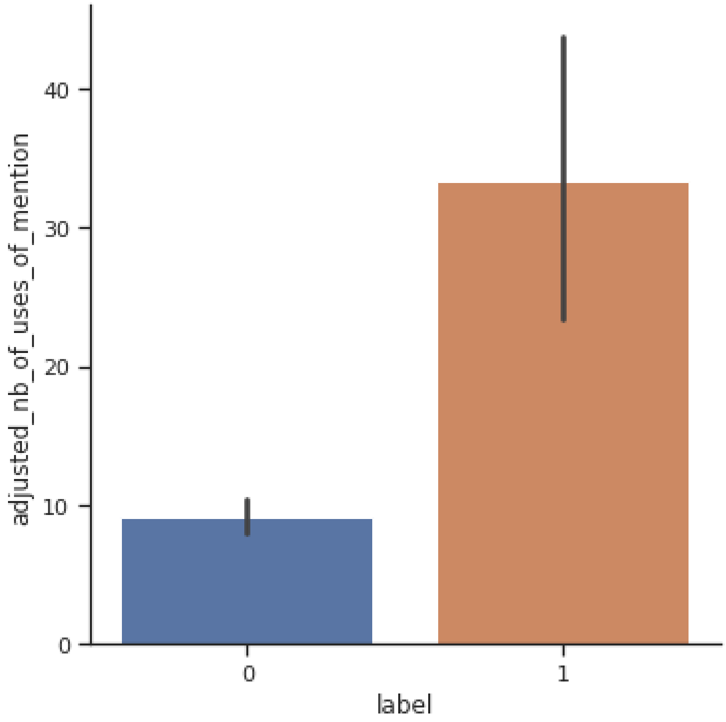
Entrée [73]:

```
explore_column ('active_tweeting_frequency_per_day')
```



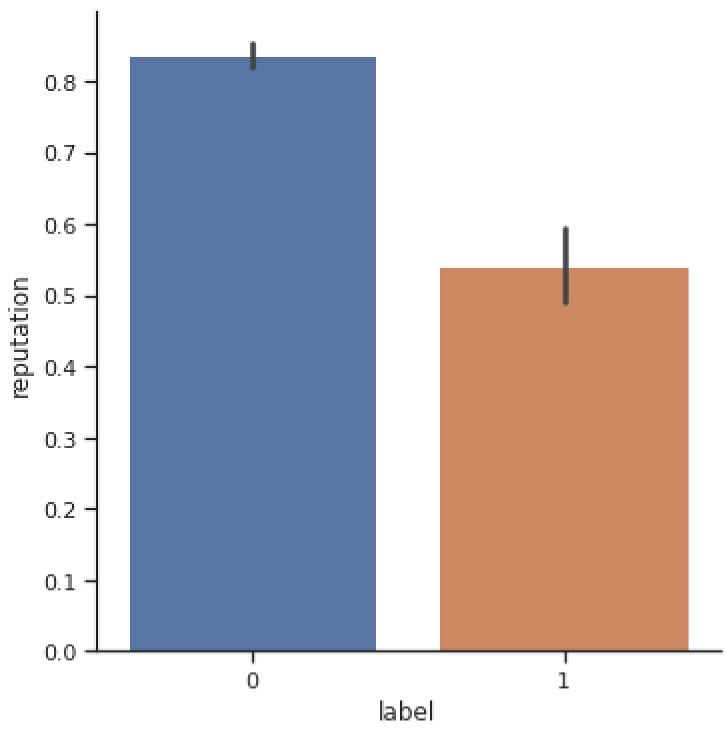
Entrée [74]:

```
explore_column ('adjusted_nb_of_uses_of_mention')
```



Entrée [75]:

```
explore_column ('reputation')
```



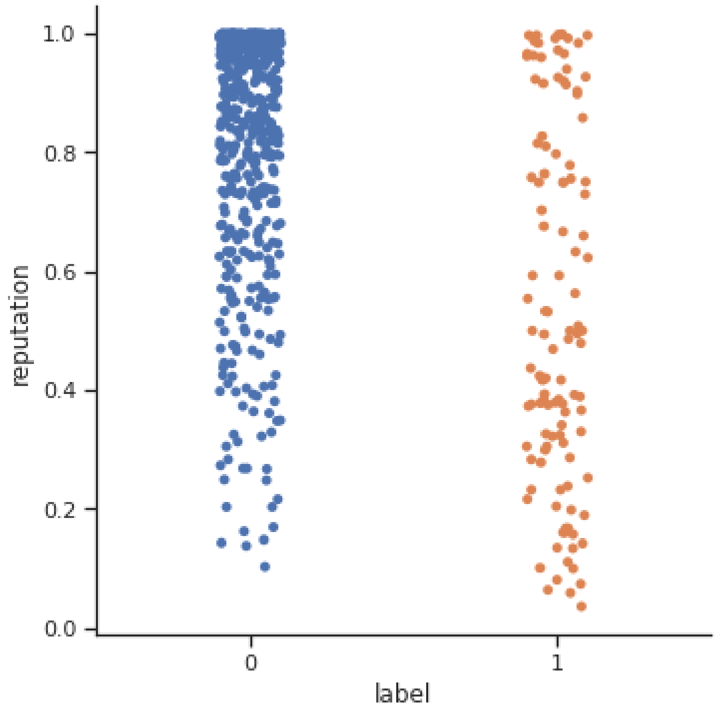
Let show the reputation distribution over the class

Entrée [76]:

```
# reputation destribution
sb.set_theme(style="ticks")
sb.catplot(y="reputation", x="label", data=users)
```

Out[76]:

```
<seaborn.axisgrid.FacetGrid at 0x7f7812eb8910>
```



Features extraction

Entrée [77]:

```
## all values of column spam_in_screen_name are null
users_features_with_labels['spam_in_screen_name']
```

Out[77]:

```
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
        ..
681    NaN
682    NaN
683    NaN
684    NaN
685    NaN
Name: spam_in_screen_name, Length: 686, dtype: float64
```

Entrée [78]:

```
#so we can delete this column spam_in_screen_name
users = users_features_with_labels.drop(['spam_in_screen_name'], axis=1)
```

Entrée [79]:

```
'''Frequent Category Imputation
This technique is used to fill the missing values in categorical data.
In this, we replace NaN values with the most Frequent label.
First, we find the most frequent label and then replace NaN with it.'''

def impute_nan(df,variable):
    most_frequent_category=df[variable].mode()[0] ##Most Frequent
    df[variable].fillna(most_frequent_category,inplace=True)

for feature in ['time_zone']:           ##List of Categorical Features
    impute_nan(users,feature)

for feature in ['utc_offset']:           ##List of Categorical Features
    impute_nan(users,feature)

users[['time_zone','utc_offset']].head(8)
```

Out[79]:

	time_zone	utc_offset
0	Hawaii	-36000.0
1	Riyadh	10800.0
2	Pacific Time (US & Canada)	-28800.0
3	Pacific Time (US & Canada)	-28800.0
4	Central Time (US & Canada)	-21600.0
5	Pacific Time (US & Canada)	-28800.0
6	Riyadh	10800.0
7	Eastern Time (US & Canada)	-14400.0

Let define the type of data in our dataset

Entrée [80]:

```
# types of columns
columns = list(users.iloc[0:0,])
float_columns = []
int_columns = []
boolean_columns = []
else_columns = []
object_columns = []
for i in columns :
    #print (users[i].head(5))
    if users[i].dtype == 'float64':
        float_columns.append(i)
        #print (users[i].dtype)
    if users[i].dtype == 'int64':
        int_columns.append(i)
        #print (users[i].dtype)
    if users[i].dtype == 'bool':
        boolean_columns.append(i)
        #print (users[i].dtype)
    if users[i].dtype == 'object':
        object_columns.append(i)
        #print (users[i].dtype)
    else :
        else_columns.append(i)
        #print (users[i].dtype)
```

Entrée [81]:

```
# columns contain objects values
users[object_columns].head()
```

Out[81]:

	avg_intertweet_times	date_newest_tweet	date_oldest_tweet	lang	max_intertweet_times	min
0	19 days 05:12:37.409091000	26/12/2017 14:45:25	29/10/2016 20:07:42	ar	176 days 23:35:57.000000000	0(
1	0 days 00:39:20.897243000	10/02/2018 17:00:37	30/01/2018 19:20:39	en	1 days 18:22:38.000000000	0(
2	16 days 16:04:30.509317000	04/07/2011 03:37:09	05/03/2010 06:21:35	en	673 days 21:00:01.000000000	0(
3	0 days 05:24:08.857143000	09/02/2018 12:43:09	11/11/2017 17:08:15	ar	9 days 05:21:55.000000000	0(
4	0 days 00:19:59.997494000	10/02/2018 17:15:00	05/02/2018 04:15:01	en	0 days 02:00:01.000000000	0(

Entrée [82]:

```
# columns contain boolean values
users[boolean_columns].head()
```

Out[82]:

	default_profile	default_profile_image
0	False	False
1	False	False
2	True	False
3	True	False
4	True	False

Entrée [ ]:

```

```

Entrée [83]:

```
# function for delete columns
def delete_columns (data,liste_columns) :
    return(data.drop(liste_columns, axis=1))

# we delete 'user_id','coded_id','utc_offset' because its not influence in spam det
users=delete_columns(users,['user_id','coded_id','utc_offset'])
users.head()
```

Out[83]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_of_
0	0.055	2.600	
1	40.000	9.938	
2	0.334	2.600	
3	4.494	0.000	
4	80.000	395.000	

5 rows × 143 columns

Extraction of only features (X).

Entrée [84]:

```
# features
X = users.iloc[:,0:142]
X
```

Out[84]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_
0	0.055	2.600	
1	40.000	9.938	
2	0.334	2.600	
3	4.494	0.000	
4	80.000	395.000	
...	...	...	
681	0.995	1.000	
682	100.000	42.436	
683	2.020	0.000	
684	1.418	1.912	
685	4.000	2.682	

686 rows × 142 columns

Entrée [85]:

```
# types of columns
columns = list(X.iloc[0:0,])
float_columns = []
int_columns = []
boolean_columns = []
else_columns = []
object_columns = []
for i in columns :
    #print (X[i].head(5))
    if X[i].dtype == 'float64':
        float_columns.append(i)
        #print (X[i].dtype)
    if X[i].dtype == 'int64':
        int_columns.append(i)
        #print (X[i].dtype)
    if X[i].dtype == 'bool':
        boolean_columns.append(i)
        #print (X[i].dtype)
    if X[i].dtype == 'object':
        object_columns.append(i)
        #print (X[i].dtype)
    else :
        else_columns.append(i)
        #print (X[i].dtype)
```

Entrée [86]:

```
# features head
X.head()
```

Out[86]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_of_
0	0.055	2.600	
1	40.000	9.938	
2	0.334	2.600	
3	4.494	0.000	
4	80.000	395.000	

5 rows × 142 columns

Entrée [87]:

```
# encode non numbers values
from sklearn import preprocessing
labelencoder = preprocessing.LabelEncoder()
categorical_cols = boolean_columns + object_columns
# apply le on categorical feature columns
X[categorical_cols] = X[categorical_cols].apply(lambda col: labelencoder.fit_transf
X[categorical_cols].head()
```

Out[87]:

	default_profile	default_profile_image	avg_intertweet_times	date_newest_tweet	date_oldest_tw
0	0	0	663	659	
1	0	0	125	138	
2	1	0	662	16	
3	1	0	465	50	
4	1	0	64	148	

Entrée [88]:

```
# label
Y = users.iloc[:,142]
Y.head()
```

Out[88]:

```
0    0
1    0
2    1
3    1
4    1
Name: label, dtype: int64
```

Entrée [89]:

```
X
```

Out[89]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_(
0	0.055	2.600	
1	40.000	9.938	
2	0.334	2.600	
3	4.494	0.000	
4	80.000	395.000	
...	...	...	
681	0.995	1.000	
682	100.000	42.436	
683	2.020	0.000	
684	1.418	1.912	
685	4.000	2.682	

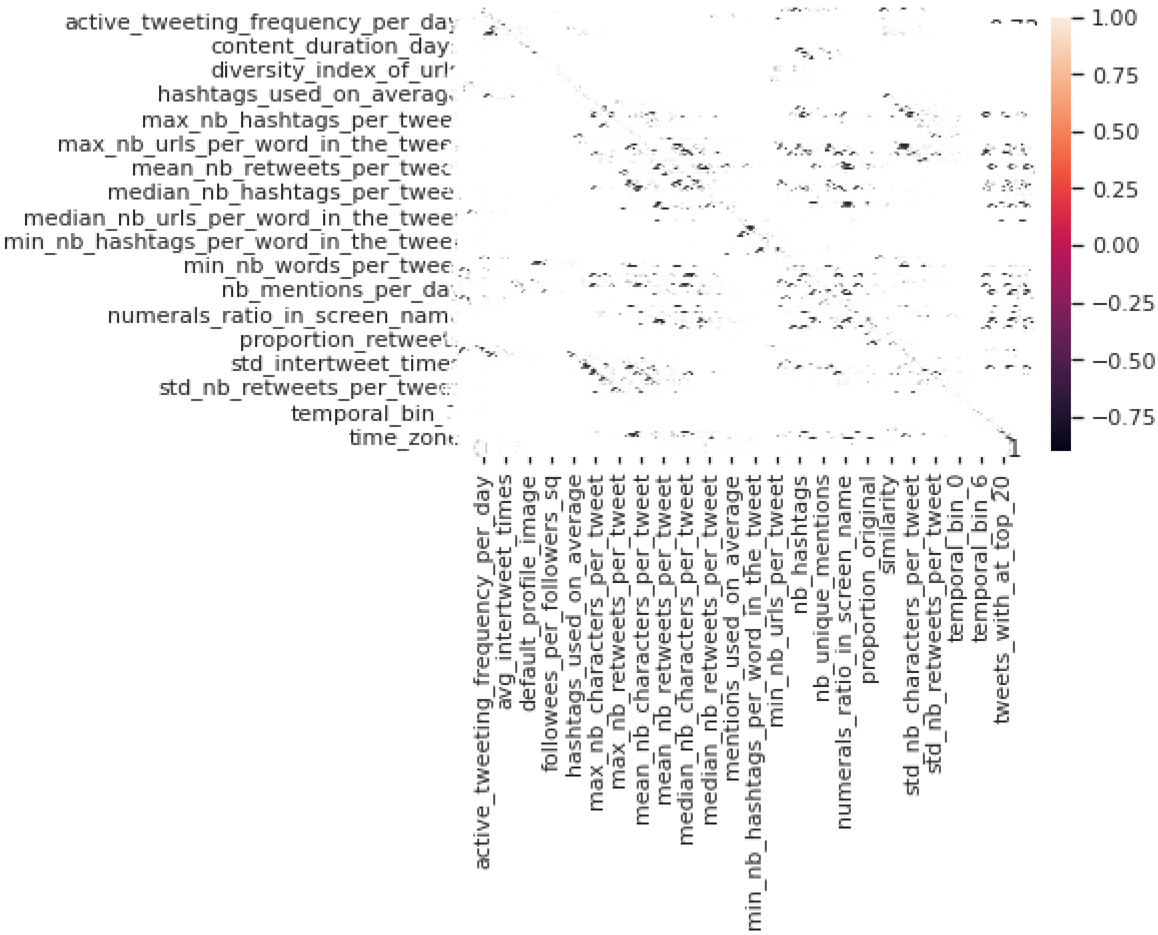
686 rows × 142 columns

Let build the correlation matrix of our domain.

Entrée [90]:

```
correlation = X.corr()
import seaborn as sn

import matplotlib.pyplot as plt
sn.heatmap(correlation, annot=True)
fig1 = plt.gcf()
plt.show()
plt.draw()
fig1.savefig("null_values.png", dpi=300)
```



```
<Figure size 432x288 with 0 Axes>
```

```
Normalization with StandardScaler
```

Entrée [91]:

```
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# function for calculate the variance in data
def variance_threshold_selector(data, threshold=0.5):
    selector = VarianceThreshold(threshold)
    selector.fit(data)
    return data[data.columns[selector.get_support(indices=True)]]

## we built the function prepare_features for making easy test with differents para
'''
faetures : data
value_variance : value of the variance, if we don't want using it we make a 0
value_centralize : if we want centralize values we can put 1, if not we make a 0
value_PCA : dimention reduction we put the size, if we don't want using it we make
'''
def prepare_features(features,value_variance,value_centralize,value_PCA ):
    X_pr = features
    if (value_variance != 0):
        #add variance
        X_pr = variance_threshold_selector(X,value_variance)
        print(" dimension afer variance ",X_pr.shape)

    if (value_centralize != 0):
        #centralize value
        st = StandardScaler()
        X_pr = st.fit_transform(X)

    if (value_PCA != 0):
        # Dimention reduction
        pca = PCA(value_PCA)
        X_pr = pca.fit_transform(X)

    return X_pr

#prepare_features(features,value_variance,value_centralize,value_PCA )
# we prepared our data with using centralization of data and dimention reduction wi
new_x = prepare_features(X, 0 , 1, 60)
new_x

# create balance with data points
from imblearn.over_sampling import SMOTE,ADASYN
sm=SMOTE()
new_x,Y=sm.fit_resample(new_x,Y)
new_x
```

Out[91]:

```
array([[ 1.29929414e+07, -1.61090845e+05,  3.39437087e+06, ...,
         3.66215819e+00,  3.11224452e+00, -3.35653093e+00],
       [-2.38992268e+06, -2.15325671e+05, -2.84498576e+04, ...,
        -1.06027304e+00,  5.97958821e+00, -1.27091628e+00],
       [ 5.58935462e+07,  4.42658965e+04,  1.50224536e+06, ...,
         3.40639531e+00, -6.85801146e+00,  4.13398089e+00],
       ...,
       [ 3.05605213e+07, -6.65370881e+04, -3.32273721e+05, ...,
        -1.67937677e-01, -2.90824888e+00,  3.50040393e+00],
```

```
       [-2.51627304e+06, -2.23459425e+05, -2.61450428e+04, ...,
         1.27554282e+00, -8.78968197e-01,  1.70116010e+00],
       [-2.50973534e+06, -2.23408009e+05, -2.37776191e+04, ...,
        -3.67896089e-01,  2.42884462e+00, -1.83028512e+00]])
```

Evaluate with cross validation

In this section, we will do the prediction training task. We will train the models with cross validation techniques.

Entrée [92]:

```
import pickle
from sklearn.naive_bayes import  MultinomialNB,BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold,GroupKFold, cross_val_score
```

Entrée [93]:

```
# function for evaluate our model using cross validation (we use 4 fold)
def cross_validation(model,x,y):
  cv = StratifiedKFold(4)
  score=cross_val_score(model,x,y,cv=cv,scoring="recall")
  #print(f"Mean:{score.mean()}\n Std:{score.std()}\n")
  return score
```

Entrée [ ]:

```

```

Entrée [94]:

```
# we test our model using 4 classifiers
warnings.filterwarnings("ignore")
Classifiers = [DecisionTreeClassifier(),XGBClassifier(),KNeighborsClassifier(3),Log
str_calassifiers =['DecisionTreeClassifier','XGBClassifier','KNeighborsClassifier',
scores = []
for model in Classifiers :
    score = cross_validation(model,new_x,Y)
    #print(name)
    scores.append(score)
print('\n ####################################################################')
print("################ Results of Recall for each model #####################")
for i in range (0,4):
    print(str_calassifiers[i])
    print(scores[i])
```

```
[17:37:14] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
[17:37:14] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
[17:37:14] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
[17:37:14] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.

 ####################################################################
################ Results of Recall for each model ####################
#
DecisionTreeClassifier
[0.94366197 0.97887324 0.95774648 0.96478873]
XGBClassifier
[0.99295775 1.         0.97183099 0.99295775]
KNeighborsClassifier
[0.95774648 0.95070423 0.93661972 0.93661972]
LogisticRegression
[0.94366197 0.96478873 0.97887324 0.95070423]
```

Entrée [ ]:

```
Eval
```

Entrée [95]:

```
# evaluate our model using simple test split
from sklearn.metrics import precision_score, \
    recall_score, confusion_matrix, \
    accuracy_score, f1_score

X_train,X_test,Y_train,Y_test=train_test_split(new_x,Y,test_size=0.25,random_state=

def evaluation(Y_test,Y_predict):
   print(' Accuracy:', round(accuracy_score(Y_test, Y_predict),4)*100,"%")
   print(' Recall:', round(recall_score(Y_test, Y_predict),2)*100,"%")
   print(' Precision:', round(precision_score(Y_test,Y_predict ),4)*100,"%")
   print(' F1 score:', round(f1_score(Y_test, Y_predict),4)*100,"%")



forest=XGBClassifier() #RandomForestClassifier()
model = forest.fit(X_train,Y_train)
pickle.dump(model, open("XGBClassifier.pickle.dat", "wb"))
Y_predict=forest.predict(X_test)
print("*******Classification avec XGBClassifierClassifier*******")
evaluation(Y_test,Y_predict)
```

```
[17:37:14] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
*******Classification avec XGBClassifierClassifier*******
 Accuracy: 98.94 %
 Recall: 99.0 %
 Precision: 98.61999999999999 %
 F1 score: 98.96000000000001 %
```

Entrée [96]:

```
forest=DecisionTreeClassifier() #RandomForestClassifier()
model = forest.fit(X_train,Y_train)
pickle.dump(model, open("DecisionTreeClassifier.pickle.dat", "wb"))
Y_predict=forest.predict(X_test)
print("*******Classification avec DecisionTreeClassifier*******")
evaluation(Y_test,Y_predict)
```

```
*******Classification avec DecisionTreeClassifier*******
 Accuracy: 94.72 %
 Recall: 94.0 %
 Precision: 95.1 %
 F1 score: 94.77 %
```

Entrée [97]:

```
forest=LogisticRegression() #RandomForestClassifier()
model = forest.fit(X_train,Y_train)
pickle.dump(model, open("LogisticRegression.pickle.dat", "wb"))
Y_predict=forest.predict(X_test)
print("*******Classification avec LogisticRegression*******")
evaluation(Y_test,Y_predict)
```

```
*******Classification avec LogisticRegression*******
 Accuracy: 71.83 %
 Recall: 98.0 %
 Precision: 64.68 %
 F1 score: 77.9 %
```

Entrée [98]:

```
forest=KNeighborsClassifier(3) #RandomForestClassifier()
model = forest.fit(X_train,Y_train)
pickle.dump(model, open("Kneighbors.pickle.dat", "wb"))
Y_predict=forest.predict(X_test)
print("*******Classification avec Kneighbors*******")
evaluation(Y_test,Y_predict)
```

```
*******Classification avec Kneighbors*******
 Accuracy: 90.85 %
 Recall: 95.0 %
 Precision: 87.82 %
 F1 score: 91.33 %
```

Entrée [99]:

```
str_calassifiers
```

Out[99]:

```
['DecisionTreeClassifier',
 'XGBClassifier',
 'KNeighborsClassifier',
 'LogisticRegression']
```

Entrée [100]:

```
scores
```

Out[100]:

```
[array([0.94366197, 0.97887324, 0.95774648, 0.96478873]),
 array([0.99295775, 1.        , 0.97183099, 0.99295775]),
 array([0.95774648, 0.95070423, 0.93661972, 0.93661972]),
 array([0.94366197, 0.96478873, 0.97887324, 0.95070423])]
```

Entrée [101]:

```
scores[0]
```

Out[101]:

```
array([0.94366197, 0.97887324, 0.95774648, 0.96478873])
```

Entrée [102]:

```
flods = ["Fold1","Fold2","Fold3","Fold4"]
```

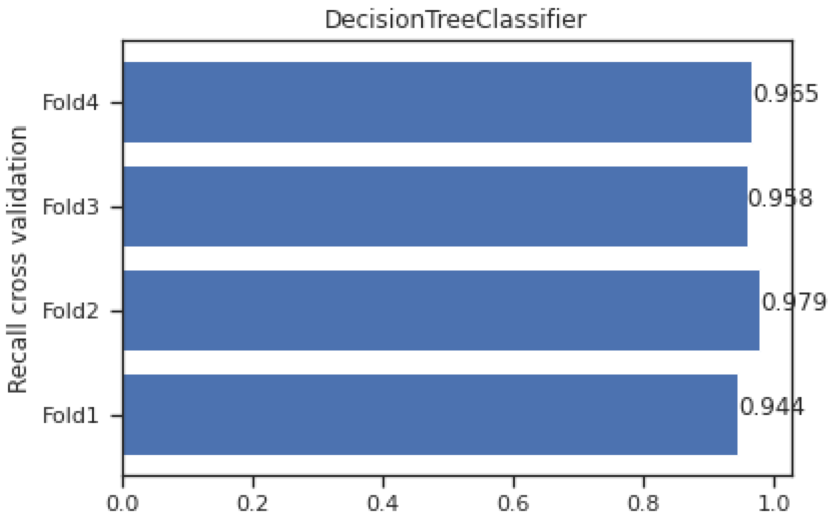
Entrée [ ]:

```

```

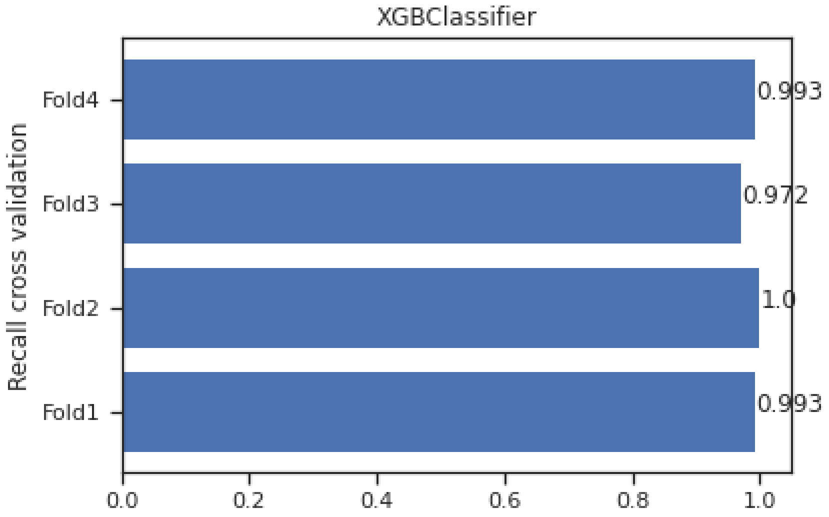
Entrée [103]:

```
# results and comparison with charts
x = flods
y = scores[0]
plt.barh(x, y)
plt.ylabel('Recall cross validation')
plt.title('DecisionTreeClassifier')
for index, value in enumerate(y):
    plt.text(round(value,3), index, str(round(value,3)))
plt.savefig("DecisionTreeClassifier.png")
```



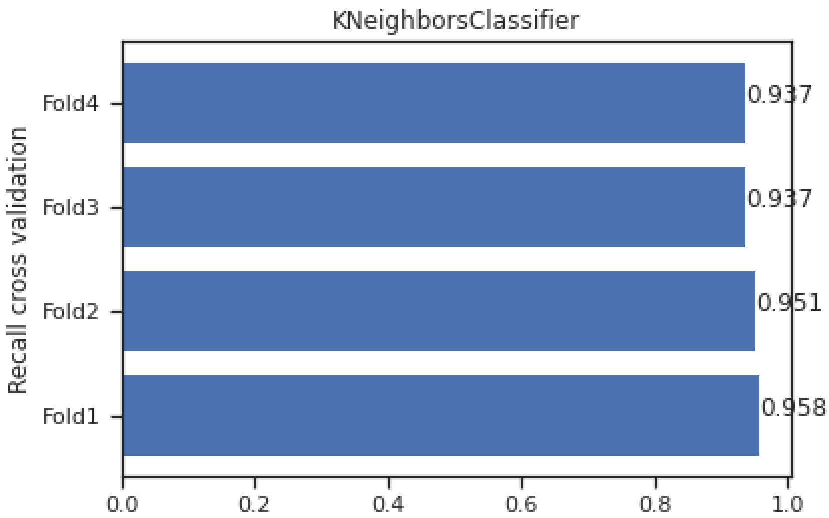
Entrée [104]:

```
x = flods
y = scores[1]
plt.barh(x, y)
plt.ylabel('Recall cross validation')
plt.title('XGBClassifier')
for index, value in enumerate(y):
    plt.text(round(value,3), index, str(round(value,3)))
plt.savefig("XGBClassifier.png")
```



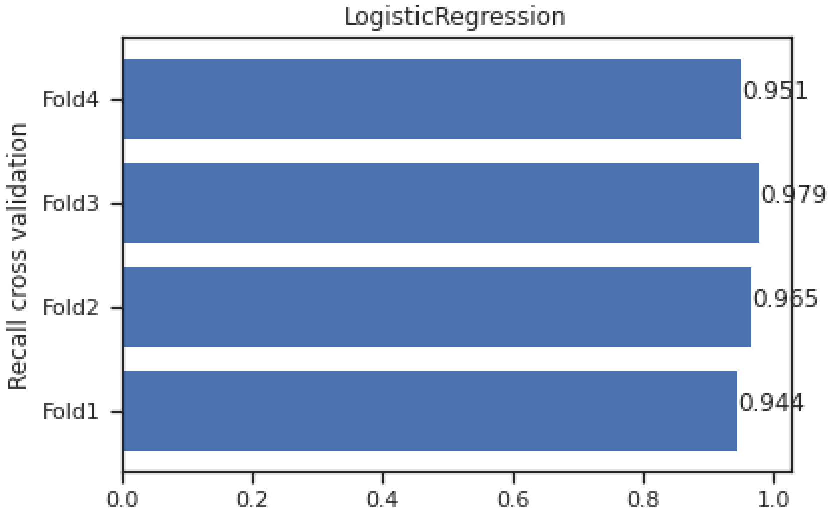
Entrée [105]:

```
x = flods
y = scores[2]
plt.barh(x, y)
plt.ylabel('Recall cross validation')
plt.title('KNeighborsClassifier')
for index, value in enumerate(y):
    plt.text(round(value,3), index, str(round(value,3)))
plt.savefig("KNeighborsClassifier.png")
```



Entrée [106]:

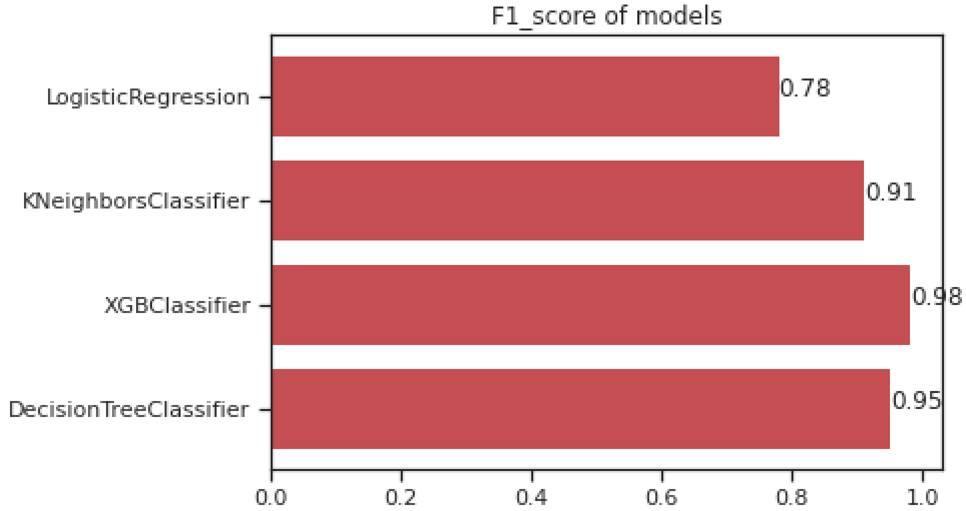
```
x = flods
y = scores[3]
plt.barh(x, y)
plt.ylabel('Recall cross validation')
plt.title('LogisticRegression')
for index, value in enumerate(y):
    plt.text(round(value,3), index, str(round(value,3)))
plt.savefig("LogisticRegression.png")
```



Entrée [107]:

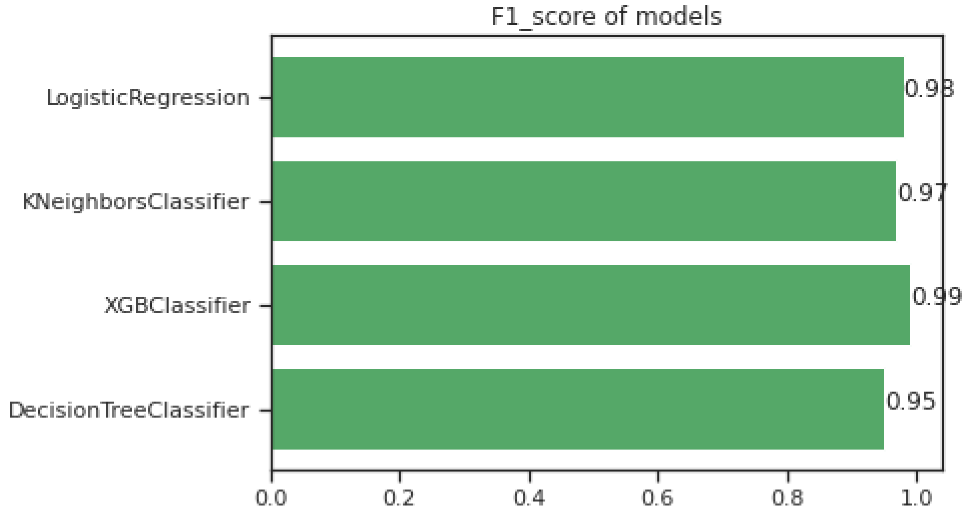
```
F1_score = [0.95,0.98,0.91,0.78]
Recall = [0.95,0.99,0.97,0.98]

x = str_calassifiers
y = F1_score
plt.barh(x, y, color ='r')
plt.ylabel('')
plt.title('F1_score of models')
for index, value in enumerate(y):
    plt.text(round(value,3), index, str(round(value,3)))
plt.savefig("f1_score.png")
```



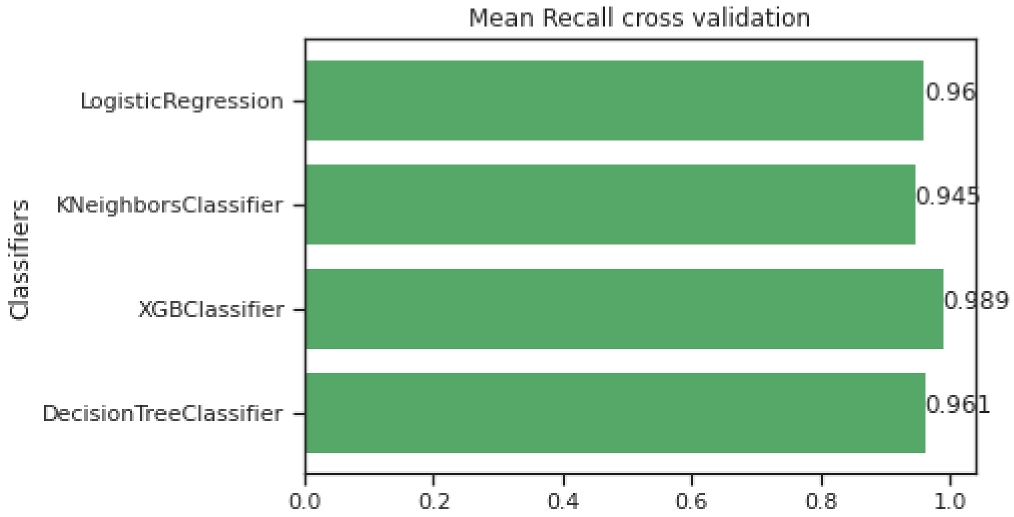
Entrée [108]:

```
x = str_calassifiers
y = Recall
plt.barh(x, y, color ='g')
plt.ylabel('')
plt.title('F1_score of models')
for index, value in enumerate(y):
    plt.text(round(value,3), index, str(round(value,3)))
plt.savefig("recal_test_split.png")
```



Entrée [109]:

```
# Mean Recall cross validation
import matplotlib.pyplot as plt
x = str_calassifiers
y = [scores[0].mean(),scores[1].mean(),scores[2].mean(),scores[3].mean()]
plt.barh(x, y, color ='g')
plt.ylabel('Classifiers')
plt.title('Mean Recall cross validation')
for index, value in enumerate(y):
    plt.text(round(value,3), index, str(round(value,3)))
plt.savefig("classifiersMean.png")
```



Let fill the test_submission file with our prediction

Entrée [110]:

```
# submission file
test_submission.head()
```

Out[110]:

	coded_id	label
0	5	NaN
1	26	NaN
2	37	NaN
3	40	NaN
4	52	NaN

Entrée [111]:

```
# merging features and users for predict
users_features_test = pd.merge(users_features, test_submission, on='coded_id')
users_features_test
```

Out[111]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_o
0	154.333	3.447	
1	44.222	3.335	
2	0.769	0.000	
3	398.000	1.074	
4	0.851	0.000	
...	...	...	
76	40.000	3.727	
77	9.524	6.479	
78	3.226	0.000	
79	22.222	1.000	
80	200.000	1.540	

81 rows × 147 columns

Entrée [112]:

```
users_features_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 81 entries, 0 to 80
Columns: 147 entries, active_tweeting_frequency_per_day to label
dtypes: bool(2), float64(94), int64(43), object(8)
memory usage: 92.5+ KB
```

Entrée [113]:

```
# column content null values
df1 = users_features_test.iloc[0:0,]
columns = list(df1)
xx=[]
yy=[]
for col in columns :
    s = users_features_test[col].isna().sum()
    if (s != 0):
        print (col)
        xx.append(col)
        print(s)
        yy.append(s)
    else:
        pass
users_features_test.shape
```

```
spam_in_screen_name
81
time_zone
17
utc_offset
17
label
81
```

Out[113]:

```
(81, 147)
```

Entrée [114]:

```
#so we can delete this column spam_in_screen_name
users_test = users_features_test.drop(['spam_in_screen_name'], axis=1)
```

Entrée [115]:

```
for feature in ['time_zone']:           ##List of Categorical Features
    impute_nan(users_test,feature)

for feature in ['utc_offset']:           ##List of Categorical Features
    impute_nan(users_test,feature)

users_test[['time_zone','utc_offset']].head(8)
```

Out[115]:

	time_zone	utc_offset
0	Eastern Time (US & Canada)	-14400.0
1	Eastern Time (US & Canada)	-14400.0
2	Eastern Time (US & Canada)	-14400.0
3	Eastern Time (US & Canada)	-14400.0
4	Eastern Time (US & Canada)	-14400.0
5	Eastern Time (US & Canada)	-14400.0
6	Eastern Time (US & Canada)	-14400.0
7	Eastern Time (US & Canada)	-14400.0

Entrée [116]:

```
# types of columns
# types of columns
columns = list(users_test.iloc[0:0,])
float_columns = []
int_columns = []
boolean_columns = []
else_columns = []
object_columns = []
for i in columns :
    #print (users_test[i].head(5))
    if users_test[i].dtype == 'float64':
        float_columns.append(i)
        #print (users_test[i].dtype)
    if users_test[i].dtype == 'int64':
        int_columns.append(i)
        #print (users_test[i].dtype)
    if users_test[i].dtype == 'bool':
        boolean_columns.append(i)
        #print (users_test[i].dtype)
    if users_test[i].dtype == 'object':
        object_columns.append(i)
        #print (users_test[i].dtype)
    else :
        else_columns.append(i)
        #print (users_test[i].dtype)
```

Entrée [117]:

```
def delete_columns (data,liste_columns) :
    return(data.drop(liste_columns, axis=1))
users_predict = users_test
users_test=delete_columns(users_test,['user_id','coded_id','utc_offset'])
users_test.head()
```

Out[117]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_of_
0	154.333	3.447	
1	44.222	3.335	
2	0.769	0.000	
3	398.000	1.074	
4	0.851	0.000	

5 rows × 143 columns

Entrée [118]:

```
# features
X = users_test.iloc[:,0:142]
X
```

Out[118]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_o
0	154.333	3.447	
1	44.222	3.335	
2	0.769	0.000	
3	398.000	1.074	
4	0.851	0.000	
...	...	...	
76	40.000	3.727	
77	9.524	6.479	
78	3.226	0.000	
79	22.222	1.000	
80	200.000	1.540	

81 rows × 142 columns

Entrée [119]:

```
# types of columns
columns = list(X.iloc[0:0,])
float_columns = []
int_columns = []
boolean_columns = []
else_columns = []
object_columns = []
for i in columns :
    #print (X[i].head(5))
    if X[i].dtype == 'float64':
        float_columns.append(i)
        #print (X[i].dtype)
    if X[i].dtype == 'int64':
        int_columns.append(i)
        #print (X[i].dtype)
    if X[i].dtype == 'bool':
        boolean_columns.append(i)
        #print (X[i].dtype)
    if X[i].dtype == 'object':
        object_columns.append(i)
        #print (X[i].dtype)
    else :
        else_columns.append(i)
        #print (X[i].dtype)
```

Entrée [120]:

```
X.head()
```

Out[120]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_of_
0	154.333	3.447	
1	44.222	3.335	
2	0.769	0.000	
3	398.000	1.074	
4	0.851	0.000	

5 rows × 142 columns

Entrée [121]:

```
from sklearn import preprocessing
labelencoder = preprocessing.LabelEncoder()
categorical_cols = boolean_columns + object_columns
# apply le on categorical feature columns
X[categorical_cols] = X[categorical_cols].apply(lambda col: labelencoder.fit_transf
X[categorical_cols]
```

Out[121]:

	default_profile	default_profile_image	avg_intertweet_times	date_newest_tweet	date_oldest_
0	1	0	8	76	
1	1	0	13	14	
2	1	0	76	74	
3	1	0	2	75	
4	1	0	75	11	
...	...	...	...	...	
76	0	0	15	62	
77	0	0	45	64	
78	0	0	60	19	
79	0	0	24	40	
80	0	0	5	63	

81 rows × 10 columns

Entrée [122]:

```
# label
Y = users_test.iloc[:,142]
Y.head()
```

Out[122]:

```
0   NaN
1   NaN
2   NaN
3   NaN
4   NaN
Name: label, dtype: float64
```

Entrée [123]:

```
#prepare_features(features,value_variance,value_centralize,value_PCA )

new_x = prepare_features(X, 0 , 1, 60)
new_x
```

Out[123]:

```
array([[-2.22244972e+06, -2.61369056e+05,  3.96345789e+04, ...,
        -2.75557584e-01, -1.59683424e+00, -1.04089237e+00],
       [-2.15562982e+06, -2.59128070e+05,  3.91488310e+04, ...,
         7.79726170e-01, -1.52004686e+00, -2.23830365e+00],
       [ 3.16109834e+07,  1.79087869e+04, -1.96302644e+06, ...,
        -1.97887723e-01,  7.99316050e-02, -3.01724383e-01],
       ...,
       [-1.67521607e+06,  1.33107171e+05,  3.40626022e+04, ...,
        -3.22232951e-01,  5.77581229e-02, -9.39862066e-03],
       [-2.16068898e+06, -2.22466888e+05,  4.28466699e+04, ...,
         7.38468128e-01, -3.02692805e-01, -2.17873243e+00],
       [-2.21174050e+06, -2.27907079e+05,  3.71744611e+04, ...,
         2.85585829e+00,  2.02947063e+00, -7.51327640e-01]])
```

Entrée [124]:

```
# load model from file
import pickle
import pandas as pd
loaded_model = pickle.load(open("XGBClassifier.pickle.dat", "rb"))
loaded_model2 = pickle.load(open("DecisionTreeClassifier.pickle.dat", "rb"))
loaded_model3 = pickle.load(open("LogisticRegression.pickle.dat", "rb"))

# make predictions for test data with three classifiers
y_pred = loaded_model.predict(new_x)
print(y_pred)
y_pred2 = loaded_model2.predict(new_x)
print(y_pred2)
y_pred3 = loaded_model3.predict(new_x)
print(y_pred3)
```

```
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 1 0 0 0 1]
[1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0
 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0
0 0
 0 0 1 0 0 0 1]
[1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 1 1 1 1 1
1 1
 0 1 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 0 1
1 0
 0 1 1 1 1 0 1]
```

Entrée [125]:

```
y_pred2
```

Out[125]:

```
array([1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1])
```

Entrée [126]:

```
# last submission file
users_predict["label"] = y_pred
users_predict [['coded_id','label']]
ordoned = users_predict [['coded_id','label']].sort_values(by=['coded_id'])
ordoned.to_csv('./results_test/test_XGB.csv',index=False)
ordoned
```

Out[126]:

	coded_id	label
24	5	0
13	26	0
14	37	0
12	40	0
7	52	1
...	...	...
73	729	0
42	745	0
72	746	0
70	757	0
62	762	0

81 rows × 2 columns

Entrée [127]:

```
users_predict["label"] = y_pred2
users_predict [['coded_id','label']]
users_predict [['coded_id','label']]
ordoned = users_predict [['coded_id','label']].sort_values(by=['coded_id'])
ordoned.to_csv('./results_test/test_DT.csv',index=False)
ordoned
```

Out[127]:

	coded_id	label
24	5	0
13	26	0
14	37	1
12	40	0
7	52	1
...	...	...
73	729	0
42	745	1
72	746	0
70	757	1
62	762	0

81 rows × 2 columns

Entrée [128]:

```
merged_predect_XGB = users_predict
merged_predect_XGB["label"] = y_pred

merged_predect_DTree = users_predict
merged_predect_DTree["label"] = y_pred2

merged_predect_LR = users_predict
merged_predect_LR["label"] = y_pred3

submited_test = merged_predect_LR [['coded_id','label']]
merged_predect_XGB
```

Out[128]:

	active_tweeting_frequency_per_day	adjusted_nb_of_uses_of_hashtag	adjusted_nb_of_uses_o
0	154.333	3.447	
1	44.222	3.335	
2	0.769	0.000	
3	398.000	1.074	
4	0.851	0.000	
...	...	...	
76	40.000	3.727	
77	9.524	6.479	
78	3.226	0.000	
79	22.222	1.000	
80	200.000	1.540	

81 rows × 146 columns

Entrée [ ]:

```
```