Adopt a Pet

You are in charge of an animal shelter and you want to predict if the animals you have in your possession can be adopted within 30 days or not.

The dataset at your disposal contains different information about the animals in the shelter: data about the breed or color, data about a cost, data about its health. You even have a short description written by the former owner and a picture of the animal.

We provide you only with the train part and a small test subset so that you can test the whole process.

Deadline: Jannuary 15, 2022.

You must submit a zip archive to LMS that contains 3 documents: - A pdf report that outlines the various stages of your work. You will insist on the different hyperparameters of your treatment and for each of them, you will specify on which ranges of values you have tested them. This report will also contain the precision obtained on the train set and on the test set. - the executable notebook containing only the chosen hyper-parameters and not their research. You will leave in this one the execution traces. - A ".joblib" file so that we can execute your code. Of course, the test dataset will be modified and only the predict function of the pipeline will be executed.

The final grade will be based on the quality of the prediction (accuracy score) for 25% and the quality of the work for 75%.

Table of Contents

In [ ]:

```
import os
from tqdm import tqdm
```

```
import warnings
warnings.filterwarnings("ignore")

import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

In [ ]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

Load train data

In [ ]:
```
path = "https://www.i3s.unice.fr/~riveill/dataset/petfinder-adoption-prediction/"
```

In [ ]:
```
breeds = pd.read_csv(path+'breed_labels.csv')
colors = pd.read_csv(path+'color_labels.csv')
states = pd.read_csv(path+'state_labels.csv')

train = pd.read_csv(path+'train.csv')

train['dataset_type'] = 'train'
```

In [ ]:
```
len(train)
```

Out[ ]:
```
8168
```

In [ ]:
```
# In this example notebook, we will only work with a small part of the dataset
N = 10
train = train[:N]
```

In [ ]:
```
if 'dataset_type' in train.columns:
    train = train.drop(labels='dataset_type', axis=1)
```

In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['Type', 'Age', 'Gender', 'Color1', 'Color2', 'Color3', 'MaturitySize',
       'FurLength', 'Vaccinated', 'Dewormed', 'Sterilized', 'Health', 'Fee',
       'Description', 'Images', 'Breed', 'target'],
      dtype='object')
```

In [ ]:
```
y_train = train['target']
X_train = train.drop(['target'], axis=1)
X_train.head()
```

Out[ ]:

	Type	Age	Gender	Color1	Color2	Color3	MaturitySize	FurLength	Vaccinated	Dewormed	Sterilized	Health	Fee	Description	Images
0	Cat	12.0	Female	White	Unknown	Unknown	Medium	Yes	Unknown	Unknown	Unknown	Healthy	0.0	We got Luna when she was a kitten in Feb 15'. ...	880e13787-4.jpg
1	Cat	4.0	Male	Golden	White	Unknown	Medium	Yes	No	Yes	No	Healthy	0.0	Ginger Boy was found starving and hungry so I ...	7abe9a0a1-2.jpg
2	Cat	12.0	Female	Black	Golden	White	Medium	No	No	No	No	Healthy	0.0	An indoor cat with nice green/ yellowish eyes....	605d07d33-5.jpg
3	Dog	60.0	Male	Black	Gray	White	Medium	No	Yes	Unknown	Unknown	Healthy	0.0	My dog name called boo. He is a male. I feedin...	7ed568ab9-1.jpg
4	Cat	36.0	Female	Cream	Gray	White	Large	No	No	No	Yes	Healthy	0.0	1) Foxy is a stray cat which I feed regularly,...	8969b314b-5.jpg

In [ ]:
```
y_train.head()
```

Out[ ]:
```
0     True
1    False
2     True
3    False
4     True
Name: target, dtype: bool
```

In [ ]:
```
cat_cols = ['Type', 'Gender', 'Breed', 'Color1', 'Color2', 'Color3',
       'MaturitySize', 'FurLength', 'Vaccinated', 'Dewormed',
       'Sterilized', 'Health']
num_cols = ['Age', 'Fee']
txt_cols = ['Description']
img_cols = ['Images']
```

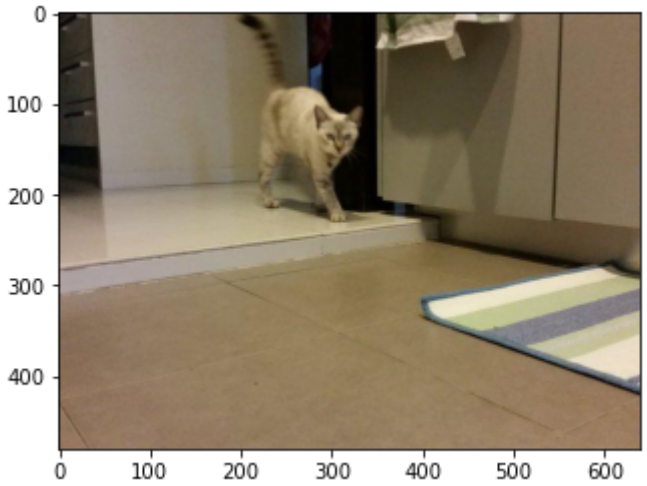
Load the images

In [ ]:
```
# Build the image list of the training set
img_dir = "train_images/"
X_train['Images'] = [path+img_dir+img for img in train['Images']]
```

In [ ]:
```
from skimage import io

# Read the first image of the list
img = io.imread(X_train['Images'][0])
# have a look to the image
plt.imshow(img)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f96d7583cd0>
```



Compute SIFT detector and descriptors for one image

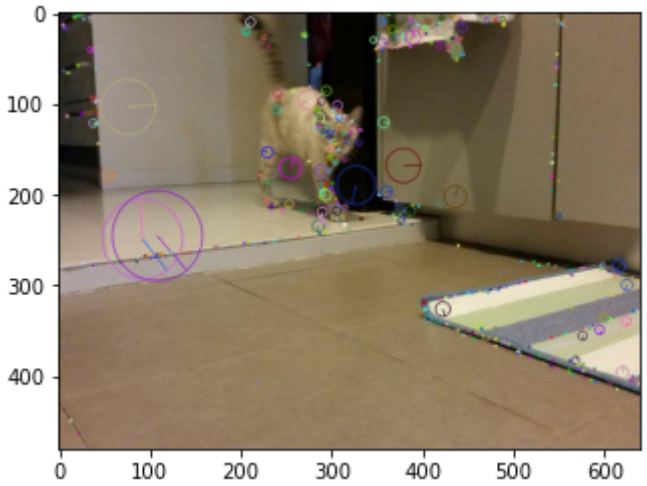
In [ ]:
```
# convert the image to grey levels
import cv2

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

In [ ]:
```
# compute SIFT detector and descriptors
sift = cv2.SIFT_create()
kp,des = sift.detectAndCompute(gray,None)
```

In [ ]:
```
# plot image and descriptors
cv2.drawKeypoints(img,kp,img,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.imshow(img)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f96dbf8d790>
```



Extract features and build BOFs

In [ ]:

```
# First step, extract the SIFTs of each image
# Be carefull: very long process

def extract_SIFT(img_lst):
    nbSIFTs = 0    # Nomber of SIFTs
    SIFTs = []  # List of SIFTs descriptors
    #dimImgs = []   # Nb of descriptors associated to each images

    for pathImg in tqdm(img_lst, position=0, leave=True):
        img = io.imread(pathImg)
        if len(img.shape)==2: # this is a grey level image
            gray = img
        else: # we expect the image to be a RGB image or RGBA
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        sift = cv2.SIFT_create()
        kp, des = sift.detectAndCompute(gray, None)
        if len(kp) == 0 and img.shape[2]==4: #some images are mask on alpha channel: we thus extract this channel if nc
            gray = img[:,:,3]
            sift = cv2.SIFT_create()
            kp, des = sift.detectAndCompute(gray, None)

        nbSIFTs += des.shape[0]
        SIFTs.append(des)
```

```
        #dimImgs.append(des.shape[0])
    return nbSIFTs, SIFTs#, dimImgs
```

In [ ]:

```
nbSIFTs, SIFTs = extract_SIFT(X_train['Images'])
print('nbSifts: ', nbSIFTs)
```

```
100%|██████████| 10/10 [00:00<00:00, 15.12it/s]
nbSifts:  4179
```

In [ ]:

```
# Step 2: clusterize the SIFT
from sklearn.cluster import MiniBatchKMeans

def clusterize(SIFTs, nb_img_features=5, verbose=False):
    clusterizer = MiniBatchKMeans(n_clusters=nb_img_features)   # nb_img_features is a hyperparameter
    # learning of the clustering
    flat_list = SIFTs[0]
    for des in SIFTs[1:]:
        flat_list = np.concatenate((flat_list, des))
        if verbose:
            print("shape:", des.shape, flat_list.shape)
    clusterizer.fit(flat_list)
    # we now know the label of each SIFT descriptor
    return clusterizer
```

In [ ]:

```
clusterizer = clusterize(SIFTs, verbose=True)
```

```
shape: (75, 128) (466, 128)
shape: (117, 128) (583, 128)
shape: (302, 128) (885, 128)
shape: (138, 128) (1023, 128)
shape: (655, 128) (1678, 128)
shape: (1306, 128) (2984, 128)
shape: (461, 128) (3445, 128)
shape: (622, 128) (4067, 128)
shape: (112, 128) (4179, 128)
```

In [ ]:

```
# Step 3: build the BOW representation of each images (i.e. construction of the BOFs)

def build_BOFs(SIFTs, clusterizer, verbose=False):
```

```
    ok, nok = 0, 0
    #BOF initialization
    nb_img_features = clusterizer.get_params()['n_clusters']
    BOFs = np.empty(shape=(0, nb_img_features), dtype=int)

    # Build label list
    flat_list = SIFTs[0]
    for des in SIFTs[1:]:
        flat_list = np.concatenate((flat_list, des))
        if verbose:
            print("shape:", des.shape, flat_list.shape)
    labels = clusterizer.predict(flat_list)

    # loop on images
    i = 0 # index for the loop on SIFTs
    for des in SIFTs:
        #initialisation of the bof for the current image
        tmpBof = np.array([0]*nb_img_features)
        j = 0
        # for every SIFT of the current image:
        nbs = des.shape[0]
        while j < nbs:
            tmpBof[labels[i]] += 1
            j+=1
            i+=1
        BOFs = np.concatenate((BOFs, tmpBof.reshape(1,-1)), axis=0)
    if verbose:
        print("BOFs : ", BOFs)

    return BOFs
```

In [ ]:

```
BOFs = build_BOFs(SIFTs, clusterizer, verbose=True)
BOFs.shape
```

```
shape: (75, 128) (466, 128)
shape: (117, 128) (583, 128)
shape: (302, 128) (885, 128)
shape: (138, 128) (1023, 128)
shape: (655, 128) (1678, 128)
shape: (1306, 128) (2984, 128)
shape: (461, 128) (3445, 128)
shape: (622, 128) (4067, 128)
```

```
shape: (112, 128) (4179, 128)
BOFs :  [[ 37 156 129  40  29]
 [ 12  14  14  25  10]
 [ 29  22  22  26  18]
 [ 38  91  52  85  36]
 [ 28  32  44  21  13]
 [165  70  78 223 119]
 [341 177 170 395 223]
 [ 90  50  92 149  80]
 [162  71  62 222 105]
 [  6  35  42  19  10]]
(10, 5)
```

Out[ ]:

In [ ]:

```python
from sklearn.base import BaseEstimator,TransformerMixin

def list_comparaison(l1, l2):
    if not l1 is None \
        and not l2 is None \
        and len(l1)==len(l2) \
        and len(l1)==sum([1 for i,j in zip(l1, l2) if i==j]):
        return True
    return False

class BOF_extractor(BaseEstimator,TransformerMixin):
    X = None
    SIFTs = None
    nbSIFTs = 0

    def __init__(self, nb_img_features=10, verbose=False):
        self.nb_img_features = nb_img_features
        self.verbose = verbose
        self.path = path
        if self.verbose:
            print("BOF.init()")

    def fit(self, X, y=None):
        if self.verbose:
            print("BOF.fit()")
        if list_comparaison(X, self.X):
            SIFTs = self.SIFTs
            nbSIFTs = self.nbSIFTs
        else:
```

```
            if self.verbose:
                print("extract_SIFT")
            nbSIFTs, SIFTs = extract_SIFT(X)
        self.X = X
        self.SIFTs = SIFTs
        self.nbSIFTs = nbSIFTs
        self.clusterizer = clusterize(SIFTs, self.nb_img_features, self.verbose)

    def transform(self, X, y=None):
        if self.verbose:
            print("BOF.transform()")
        if list_comparaison(X, self.X):
            SIFTs = self.SIFTs
            nbSIFTs = self.nbSIFTs
        else:
            if self.verbose:
                print("extract_SIFT")
            nbSIFTs, SIFTs = extract_SIFT(X)

        if self.verbose:
            print("nbSIFTs:", nbSIFTs)
        return build_BOFs(SIFTs, self.clusterizer, self.verbose)

    def fit_transform(self, X, y=None):
        if self.verbose:
            print("BOF.fit_transform()")
        if list_comparaison(X, self.X):
            SIFTs = self.SIFTs
            nbSIFTs = self.nbSIFTs
        else:
            if self.verbose:
                print("extract_SIFT")
            nbSIFTs, SIFTs = extract_SIFT(X)
        self.X = X
        self.SIFTs = SIFTs
        self.nbSIFTs = nbSIFTs
        self.clusterizer = clusterize(SIFTs, self.nb_img_features, self.verbose)
        return build_BOFs(SIFTs, self.clusterizer, self.verbose)
```

In [ ]:

```
test_BOF_extractor = BOF_extractor(nb_img_features=5, verbose=True)
```

```
BOF.init()
```

In [ ]:
```
test_BOF_extractor.fit(X_train['Images'])
```

```
 10%|          | 1/10 [00:00<00:01,  6.65it/s]
BOF.fit()
extract_SIFT
100%|██████████| 10/10 [00:00<00:00, 14.67it/s]
shape: (75, 128) (466, 128)
shape: (117, 128) (583, 128)
shape: (302, 128) (885, 128)
shape: (138, 128) (1023, 128)
shape: (655, 128) (1678, 128)
shape: (1306, 128) (2984, 128)
shape: (461, 128) (3445, 128)
shape: (622, 128) (4067, 128)
shape: (112, 128) (4179, 128)
```

In [ ]:
```
BOFs = test_BOF_extractor.transform(X_train['Images'])
BOFs.shape
```

```
BOF.transform()
nbSIFTs: 4179
shape: (75, 128) (466, 128)
shape: (117, 128) (583, 128)
shape: (302, 128) (885, 128)
shape: (138, 128) (1023, 128)
shape: (655, 128) (1678, 128)
shape: (1306, 128) (2984, 128)
shape: (461, 128) (3445, 128)
shape: (622, 128) (4067, 128)
shape: (112, 128) (4179, 128)
BOFs :  [[ 46  36  51 138 120]
 [ 24  14   2  13  22]
 [ 29  19  33  13  23]
 [ 92  51  13  58  88]
 [ 21  17  27  23  50]
 [222 188  98  74  73]
 [395 381 189 164 177]
 [116  99  54 113  79]
 [223 162  96  52  89]
 [ 24   9   6  29  44]]
```

Out[ ]:
```
(10, 5)
```

In [ ]:
```
BOFs = test_BOF_extractor.fit_transform(X_train['Images'])
BOFs.shape
```

```
BOF.fit_transform()
shape: (75, 128) (466, 128)
shape: (117, 128) (583, 128)
shape: (302, 128) (885, 128)
shape: (138, 128) (1023, 128)
shape: (655, 128) (1678, 128)
shape: (1306, 128) (2984, 128)
shape: (461, 128) (3445, 128)
shape: (622, 128) (4067, 128)
shape: (112, 128) (4179, 128)
shape: (75, 128) (466, 128)
shape: (117, 128) (583, 128)
shape: (302, 128) (885, 128)
shape: (138, 128) (1023, 128)
shape: (655, 128) (1678, 128)
shape: (1306, 128) (2984, 128)
shape: (461, 128) (3445, 128)
shape: (622, 128) (4067, 128)
shape: (112, 128) (4179, 128)
BOFs :  [[ 60  30  49 225  27]
 [ 26   9   3  25  12]
 [ 27  11  30  24  25]
 [ 96  37  21 109  39]
 [ 27  13  24  57  17]
 [242 104  96  94 119]
 [435 262 174 198 237]
 [154  95  44 110  58]
 [240 115  97  75  95]
 [ 26   5   8  65   8]]
(10, 5)
```

Out[ ]:

In [ ]:
```
test = pd.read_csv(path+"test.csv")
y_test = test['target']
X_test = test.drop(['target'], axis=1)

img_dir = "test_images/"
```

```
X_test['Images'] = [path+img_dir+img for img in test['Images']]
len(X_test)
```

Out[ ]:
```
250
```

In [ ]:
```
BOFs = test_BOF_extractor.transform(X_test['Images'])
BOFs.shape
```

```
  1%|          | 2/250 [00:00<00:23, 10.78it/s]
BOF.transform()
extract_SIFT
100%|██████████| 250/250 [00:19<00:00, 13.00it/s]
nbSIFTs: 177122
shape: (2011, 128) (2350, 128)
shape: (913, 128) (3263, 128)
shape: (1712, 128) (4975, 128)
shape: (2479, 128) (7454, 128)
shape: (332, 128) (7786, 128)
shape: (372, 128) (8158, 128)
shape: (329, 128) (8487, 128)
shape: (230, 128) (8717, 128)
shape: (983, 128) (9700, 128)
shape: (296, 128) (9996, 128)
shape: (430, 128) (10426, 128)
shape: (319, 128) (10745, 128)
shape: (816, 128) (11561, 128)
shape: (235, 128) (11796, 128)
shape: (1316, 128) (13112, 128)
shape: (1175, 128) (14287, 128)
shape: (417, 128) (14704, 128)
shape: (184, 128) (14888, 128)
shape: (243, 128) (15131, 128)
shape: (333, 128) (15464, 128)
shape: (936, 128) (16400, 128)
shape: (548, 128) (16948, 128)
shape: (1096, 128) (18044, 128)
shape: (730, 128) (18774, 128)
shape: (383, 128) (19157, 128)
shape: (577, 128) (19734, 128)
shape: (1348, 128) (21082, 128)
shape: (855, 128) (21937, 128)
```

```
shape: (691, 128) (22628, 128)
shape: (159, 128) (22787, 128)
shape: (488, 128) (23275, 128)
shape: (615, 128) (23890, 128)
shape: (1637, 128) (25527, 128)
shape: (293, 128) (25820, 128)
shape: (362, 128) (26182, 128)
shape: (292, 128) (26474, 128)
shape: (93, 128) (26567, 128)
shape: (97, 128) (26664, 128)
shape: (582, 128) (27246, 128)
shape: (1586, 128) (28832, 128)
shape: (677, 128) (29509, 128)
shape: (836, 128) (30345, 128)
shape: (217, 128) (30562, 128)
shape: (2918, 128) (33480, 128)
shape: (1114, 128) (34594, 128)
shape: (296, 128) (34890, 128)
shape: (401, 128) (35291, 128)
shape: (400, 128) (35691, 128)
shape: (413, 128) (36104, 128)
shape: (992, 128) (37096, 128)
shape: (3085, 128) (40181, 128)
shape: (389, 128) (40570, 128)
shape: (624, 128) (41194, 128)
shape: (745, 128) (41939, 128)
shape: (555, 128) (42494, 128)
shape: (420, 128) (42914, 128)
shape: (2369, 128) (45283, 128)
shape: (561, 128) (45844, 128)
shape: (1001, 128) (46845, 128)
shape: (268, 128) (47113, 128)
shape: (146, 128) (47259, 128)
shape: (698, 128) (47957, 128)
shape: (841, 128) (48798, 128)
shape: (733, 128) (49531, 128)
shape: (886, 128) (50417, 128)
shape: (1559, 128) (51976, 128)
shape: (403, 128) (52379, 128)
shape: (359, 128) (52738, 128)
shape: (507, 128) (53245, 128)
shape: (219, 128) (53464, 128)
shape: (1072, 128) (54536, 128)
shape: (1522, 128) (56058, 128)
```

```
shape: (117, 128) (56175, 128)
shape: (374, 128) (56549, 128)
shape: (781, 128) (57330, 128)
shape: (1285, 128) (58615, 128)
shape: (780, 128) (59395, 128)
shape: (679, 128) (60074, 128)
shape: (484, 128) (60558, 128)
shape: (671, 128) (61229, 128)
shape: (624, 128) (61853, 128)
shape: (657, 128) (62510, 128)
shape: (150, 128) (62660, 128)
shape: (224, 128) (62884, 128)
shape: (917, 128) (63801, 128)
shape: (911, 128) (64712, 128)
shape: (1152, 128) (65864, 128)
shape: (167, 128) (66031, 128)
shape: (672, 128) (66703, 128)
shape: (196, 128) (66899, 128)
shape: (576, 128) (67475, 128)
shape: (679, 128) (68154, 128)
shape: (503, 128) (68657, 128)
shape: (431, 128) (69088, 128)
shape: (114, 128) (69202, 128)
shape: (435, 128) (69637, 128)
shape: (244, 128) (69881, 128)
shape: (303, 128) (70184, 128)
shape: (402, 128) (70586, 128)
shape: (327, 128) (70913, 128)
shape: (423, 128) (71336, 128)
shape: (314, 128) (71650, 128)
shape: (641, 128) (72291, 128)
shape: (201, 128) (72492, 128)
shape: (366, 128) (72858, 128)
shape: (541, 128) (73399, 128)
shape: (1870, 128) (75269, 128)
shape: (439, 128) (75708, 128)
shape: (324, 128) (76032, 128)
shape: (383, 128) (76415, 128)
shape: (84, 128) (76499, 128)
shape: (269, 128) (76768, 128)
shape: (294, 128) (77062, 128)
shape: (778, 128) (77840, 128)
shape: (777, 128) (78617, 128)
shape: (176, 128) (78793, 128)
```

```
shape: (925, 128) (79718, 128)
shape: (1736, 128) (81454, 128)
shape: (434, 128) (81888, 128)
shape: (661, 128) (82549, 128)
shape: (820, 128) (83369, 128)
shape: (690, 128) (84059, 128)
shape: (276, 128) (84335, 128)
shape: (817, 128) (85152, 128)
shape: (779, 128) (85931, 128)
shape: (787, 128) (86718, 128)
shape: (554, 128) (87272, 128)
shape: (2510, 128) (89782, 128)
shape: (413, 128) (90195, 128)
shape: (1166, 128) (91361, 128)
shape: (423, 128) (91784, 128)
shape: (245, 128) (92029, 128)
shape: (137, 128) (92166, 128)
shape: (331, 128) (92497, 128)
shape: (904, 128) (93401, 128)
shape: (216, 128) (93617, 128)
shape: (521, 128) (94138, 128)
shape: (634, 128) (94772, 128)
shape: (3667, 128) (98439, 128)
shape: (726, 128) (99165, 128)
shape: (1654, 128) (100819, 128)
shape: (839, 128) (101658, 128)
shape: (1048, 128) (102706, 128)
shape: (369, 128) (103075, 128)
shape: (605, 128) (103680, 128)
shape: (432, 128) (104112, 128)
shape: (885, 128) (104997, 128)
shape: (296, 128) (105293, 128)
shape: (2254, 128) (107547, 128)
shape: (702, 128) (108249, 128)
shape: (244, 128) (108493, 128)
shape: (1004, 128) (109497, 128)
shape: (386, 128) (109883, 128)
shape: (819, 128) (110702, 128)
shape: (246, 128) (110948, 128)
shape: (709, 128) (111657, 128)
shape: (877, 128) (112534, 128)
shape: (945, 128) (113479, 128)
shape: (443, 128) (113922, 128)
shape: (143, 128) (114065, 128)
```

```
shape: (430, 128) (114495, 128)
shape: (643, 128) (115138, 128)
shape: (971, 128) (116109, 128)
shape: (1139, 128) (117248, 128)
shape: (879, 128) (118127, 128)
shape: (1134, 128) (119261, 128)
shape: (523, 128) (119784, 128)
shape: (295, 128) (120079, 128)
shape: (484, 128) (120563, 128)
shape: (1222, 128) (121785, 128)
shape: (780, 128) (122565, 128)
shape: (191, 128) (122756, 128)
shape: (283, 128) (123039, 128)
shape: (487, 128) (123526, 128)
shape: (177, 128) (123703, 128)
shape: (411, 128) (124114, 128)
shape: (600, 128) (124714, 128)
shape: (364, 128) (125078, 128)
shape: (373, 128) (125451, 128)
shape: (358, 128) (125809, 128)
shape: (1555, 128) (127364, 128)
shape: (206, 128) (127570, 128)
shape: (758, 128) (128328, 128)
shape: (3967, 128) (132295, 128)
shape: (493, 128) (132788, 128)
shape: (862, 128) (133650, 128)
shape: (511, 128) (134161, 128)
shape: (1072, 128) (135233, 128)
shape: (262, 128) (135495, 128)
shape: (425, 128) (135920, 128)
shape: (286, 128) (136206, 128)
shape: (1832, 128) (138038, 128)
shape: (304, 128) (138342, 128)
shape: (48, 128) (138390, 128)
shape: (436, 128) (138826, 128)
shape: (603, 128) (139429, 128)
shape: (919, 128) (140348, 128)
shape: (1823, 128) (142171, 128)
shape: (553, 128) (142724, 128)
shape: (1185, 128) (143909, 128)
shape: (332, 128) (144241, 128)
shape: (328, 128) (144569, 128)
shape: (895, 128) (145464, 128)
shape: (733, 128) (146197, 128)
```

```
shape: (666, 128) (146863, 128)
shape: (413, 128) (147276, 128)
shape: (279, 128) (147555, 128)
shape: (934, 128) (148489, 128)
shape: (454, 128) (148943, 128)
shape: (191, 128) (149134, 128)
shape: (1342, 128) (150476, 128)
shape: (471, 128) (150947, 128)
shape: (1409, 128) (152356, 128)
shape: (333, 128) (152689, 128)
shape: (473, 128) (153162, 128)
shape: (391, 128) (153553, 128)
shape: (2384, 128) (155937, 128)
shape: (296, 128) (156233, 128)
shape: (925, 128) (157158, 128)
shape: (392, 128) (157550, 128)
shape: (1755, 128) (159305, 128)
shape: (1133, 128) (160438, 128)
shape: (478, 128) (160916, 128)
shape: (497, 128) (161413, 128)
shape: (1112, 128) (162525, 128)
shape: (1853, 128) (164378, 128)
shape: (292, 128) (164670, 128)
shape: (421, 128) (165091, 128)
shape: (209, 128) (165300, 128)
shape: (1287, 128) (166587, 128)
shape: (1408, 128) (167995, 128)
shape: (244, 128) (168239, 128)
shape: (149, 128) (168388, 128)
shape: (131, 128) (168519, 128)
shape: (281, 128) (168800, 128)
shape: (879, 128) (169679, 128)
shape: (637, 128) (170316, 128)
shape: (142, 128) (170458, 128)
shape: (1066, 128) (171524, 128)
shape: (274, 128) (171798, 128)
shape: (783, 128) (172581, 128)
shape: (1486, 128) (174067, 128)
shape: (797, 128) (174864, 128)
shape: (383, 128) (175247, 128)
shape: (227, 128) (175474, 128)
shape: (241, 128) (175715, 128)
shape: (351, 128) (176066, 128)
shape: (795, 128) (176861, 128)
```

```
shape: (261, 128) (177122, 128)
BOFs :  [[ 92  33  49 137  28]
 [640 284 388 511 188]
 [269 110 129 239 166]
 ...
 [ 80  63  19 156  33]
 [287  88 187 139  94]
 [ 69  31  30  95  36]]
(250, 5)
```

Out[ ]:

Build a basic model

There are much more interesting things in the dataset and I'm going to explore them, but for now let's build a simple model as a baseline.

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

categorical_preprocessor = OneHotEncoder(handle_unknown="ignore")
numerical_preprocessor = StandardScaler()
text_preprocessor = CountVectorizer()
image_preprocessor = BOF_extractor(nb_img_features=3, verbose=False)

preprocessor = ColumnTransformer([
    ("categorical encoding", categorical_preprocessor, cat_cols),
    ("numerical encoding", numerical_preprocessor, num_cols),
    ("text encoding", text_preprocessor, 'Description'),
    ("image encoding", image_preprocessor, 'Images'),
])

classifier = LogisticRegression()

model = make_pipeline(preprocessor, classifier)
```

In [ ]:

```
model.fit(X_train, y_train)
y_pred = model.predict(X_train)
```

```
accuracy_score(y_train, y_pred)
```

```
100%|██████████| 10/10 [00:00<00:00, 15.84it/s]
1.0
```

Out[ ]:

In [ ]:

```
# Save the model
from joblib import dump, load

dump(model, 'michel.joblib') # Put your name as a model name
```

Out[ ]:

```
['michel.joblib']
```

Evaluation of the model

We will only execute the following cells.

In [ ]:

```
test = pd.read_csv(path+"test.csv")

y_test = test['target']
X_test = test.drop(['target'], axis=1)

img_dir = "test_images/"
X_test['Images'] = [path+img_dir+img for img in test['Images']]
print("Test size:", len(X_test))

model = load('michel.joblib')
y_pred = model.predict(X_train)
print("ACC on train", accuracy_score(y_train, y_pred))
y_pred = model.predict(X_test)
print("ACC on test", accuracy_score(y_test, y_pred))
```

```
  0%|          | 0/250 [00:00<?, ?it/s]
Test size: 250
ACC on train 1.0
100%|██████████| 250/250 [00:19<00:00, 12.52it/s]
ACC on test 0.524
```

In [ ]:

In [ ]: