3a

March 28, 2021

0.1 CSc 84020 Neural Networks and Deep Learning, Spring 2021

Homework 2 (3a)

Andrea Ceres and Shao Liu

Load library

[ ]:
```
import glob
import os
import numpy as np
import keras
from keras import layers
from urllib import request
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
import pylab
from pandas.plotting import scatter_matrix
from pandas import set_option
import time
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
%matplotlib inline
```

```
print(tf.__version__)
```

```
2.4.1
```

Load Data from Google Drive

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
# critters (6 classes)
critters = ['ant', 'bee', 'butterfly', 'mosquito', 'scorpion',
            'spider']

# birds (6 classes)
birds = ['bird', 'duck', 'flamingo', 'owl', 'parrot',
            'penguin']

# ocean animals (8 classes)
ocean_animals = ['crab', 'dolphin', 'fish', 'lobster', 'octopus',
            'sea%20turtle', 'shark', 'whale']

# land mammals (22 classes)
land_mammals = ['bear', 'camel', 'cat', 'cow', 'dog',
            'elephant', 'giraffe', 'hedgehog', 'horse', 'kangaroo',
            'lion', 'monkey', 'mouse', 'panda', 'pig',
            'rabbit', 'raccoon', 'rhinoceros', 'sheep', 'squirrel',
            'tiger', 'zebra']

# all animals (47 classes)
all_animals = ['ant', 'bat', 'bear', 'bee', 'bird',
            'butterfly', 'camel', 'cat', 'cow', 'crab',
            'crocodile', 'dog', 'dolphin', 'duck', 'elephant',
            'fish', 'flamingo', 'frog', 'giraffe', 'hedgehog',
            'horse', 'kangaroo', 'lion', 'lobster', 'monkey',
            'mosquito', 'mouse', 'octopus', 'owl', 'panda',
            'parrot', 'penguin', 'pig', 'rabbit', 'raccoon',
            'rhinoceros', 'scorpion', 'sea_20turtle', 'shark', 'sheep',
            'snail', 'snake', 'spider', 'squirrel',  'tiger',
            'whale', 'zebra']
```

[ ]:
```
classes = critters
CLASS_SAMPLE_MAX = 25000
DATA_DIR = '/content/drive/My Drive/Colab Notebooks/NNDL/HW2/data/'
```

[ ]:
```
def load_bitmaps(data_dir=DATA_DIR, class_sample_max=CLASS_SAMPLE_MAX):
    class_files = []
    for c in classes:
        print(c, end=' ')
        class_files.append(os.path.join(DATA_DIR, c + '.npy'))
    class_files.sort()

    X = np.empty([0,784])
    y = np.empty([0])

    print()
    for id, class_file in enumerate(class_files):
        print(id, end=' ')
        loaded_data = np.load(class_file)
        loaded_data = loaded_data[0:CLASS_SAMPLE_MAX, :]
        labels = np.full(loaded_data.shape[0],id)

        X = np.concatenate((X, loaded_data), axis = 0)
        y = np.append(y, labels)

    return X, y
```

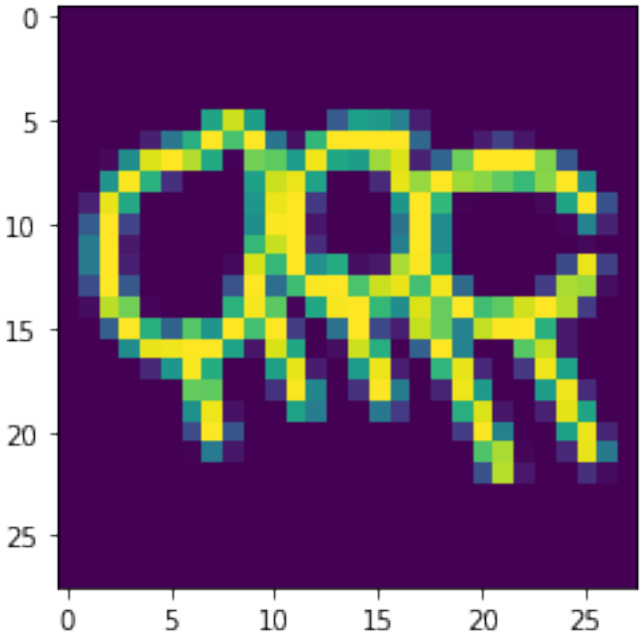
[ ]:
```
start = time.time()
X, y = load_bitmaps()
print(f'\nLoading time: %.3f' % int(time.time() - start), 'seconds')
print(X.shape)
print(y.shape)
print(y[-CLASS_SAMPLE_MAX-5:-CLASS_SAMPLE_MAX+5])
```

```
ant bee butterfly mosquito scorpion spider
0 1 2 3 4 5
Loading time: 1.000 seconds
(150000, 784)
(150000,)
[4. 4. 4. 4. 4. 5. 5. 5. 5. 5.]
```

Data sample

[ ]:
```
id = 20000
plt.imshow(X[id].reshape(28,28))
print(classes[int(y[id].item())])
```

```
ant
```



[ ]:
```
set_option('display.max_columns', 64)
# plt.style.use('seaborn-white')
```

1. Printing the shape of the data

[ ]:
```
# Create pandas dataframe
Y = y.reshape(150000,1)
df = np.append(X,Y,1)
df = pd.DataFrame(df)
# 1. Printing the shape of the data
print(df.shape)
```

```
(150000, 785)
```

2. Printing the data

[ ]:
```
print(df.head(20))
```

```
     0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   \
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  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2   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
3   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
4   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
5   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
6   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
```

7	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
8	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
9	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
10	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
11	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
12	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
13	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
14	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
15	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
16	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
17	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
18	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
19	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

	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	\
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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	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	
3	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	
4	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	
5	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	
6	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	
7	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	
8	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	
9	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	
10	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	
11	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	
12	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	
13	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	
14	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	
15	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	
16	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	
17	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	
18	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	
19	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	

	30	31	…	753	754	755	756	757	758	759	760	761	762	763	764	\
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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	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	
3	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	
4	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	
5	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	
6	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	
7	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	
8	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	
9	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	
10	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	

11	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
12	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
13	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
14	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
15	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
16	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
17	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
18	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
19	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

	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	\
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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	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	
3	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	
4	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	
5	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	
6	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	
7	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	
8	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	
9	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	
10	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	
11	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	
12	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	
13	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	
14	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	
15	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	
16	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	
17	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	
18	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	
19	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	

	780	781	782	783	784
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0

```
15  0.0  0.0  0.0  0.0  0.0
16  0.0  0.0  0.0  0.0  0.0
17  0.0  0.0  0.0  0.0  0.0
18  0.0  0.0  0.0  0.0  0.0
19  0.0  0.0  0.0  0.0  0.0

[20 rows x 785 columns]
```

[ ]:
```
# Only show the 28 pixels in center because most of the pixels are zeros in␣
 ↪other areas.
features = list(df.columns[391:419])
df_mid = df[features]
print(df_mid)
```

```
        391  392    393    394    395    396    397    398    399    400  \
0       0.0  0.0    0.0    0.0  249.0  133.0   75.0  253.0  255.0  237.0
1       0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2       0.0  0.0   22.0  247.0  140.0    0.0    0.0    0.0    0.0    0.0
3       0.0  0.0    0.0  123.0  255.0  245.0  238.0  238.0  246.0  233.0
4       0.0  0.0    0.0  167.0  230.0    4.0    0.0    0.0    0.0    0.0
…     …  …    …    …    …    …    …    …    …    …
149995  0.0  0.0    0.0   63.0   15.0    0.0    0.0    0.0   22.0  252.0
149996  0.0  0.0    0.0    0.0    0.0    0.0   69.0  255.0   60.0    0.0
149997  0.0  0.0  102.0  255.0   73.0   59.0  255.0  210.0  239.0  255.0
149998  0.0  0.0   21.0  253.0  255.0  255.0  255.0  255.0  173.0    5.0
149999  0.0  0.0    0.0  219.0  227.0  255.0  238.0  178.0  255.0   94.0

          401    402    403    404    405    406    407    408    409    410  \
0         0.0    0.0    0.0    0.0    0.0    0.0    1.0  243.0  255.0  255.0
1         0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2         7.0  240.0  202.0  254.0  107.0    0.0    0.0    0.0   49.0  252.0
3       235.0  136.0    9.0   72.0   42.0  118.0  103.0    0.0  116.0  252.0
4         2.0  226.0  255.0  103.0    0.0    0.0  185.0  254.0  210.0  245.0
…       …    …    …    …    …    …    …    …    …    …
149995  130.0    0.0    0.0    0.0    0.0   11.0  239.0  242.0  204.0  245.0
149996    0.0    1.0  195.0  253.0  254.0  255.0   45.0    0.0    0.0  134.0
149997  246.0  211.0  255.0  255.0  254.0  255.0  255.0  241.0  198.0  239.0
149998    0.0    0.0    0.0    0.0    0.0  155.0  254.0  249.0  209.0  167.0
149999  251.0  228.0  151.0   25.0    0.0    2.0   92.0  242.0  239.0  245.0

          411    412    413    414    415    416    417  418
0       144.0    0.0    0.0    0.0    0.0  128.0  255.0  4.0
1         0.0    0.0    0.0    0.0    0.0    0.0    0.0  0.0
2       255.0   30.0    0.0    0.0    9.0  219.0  211.0  5.0
3       243.0  255.0  231.0   35.0    0.0    0.0    0.0  0.0
4       205.0  210.0  249.0  199.0    2.0    0.0    0.0  0.0
…       …    …    …    …    …    …    …  …
149995  239.0   82.0    0.0    0.0    0.0    0.0    0.0  0.0
```

```
149996  249.0   18.0    0.0   12.0  248.0  130.0    0.0  0.0
149997  226.0  103.0  255.0   97.0    0.0    0.0    0.0  0.0
149998  122.0   80.0   31.0    0.0    0.0    0.0    0.0  0.0
149999   59.0    0.0    0.0  242.0  253.0  255.0  191.0  0.0

[150000 rows x 28 columns]
```

3. Printing the descriptive statistics

[ ]:
```
# 3. Printing the descriptive statistics
print(df.describe())
```

```
                0         1         2         3         4         5         6    \
count  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0
mean        0.0       0.0       0.0       0.0       0.0       0.0       0.0
std         0.0       0.0       0.0       0.0       0.0       0.0       0.0
min         0.0       0.0       0.0       0.0       0.0       0.0       0.0
25%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
50%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
75%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
max         0.0       0.0       0.0       0.0       0.0       0.0       0.0

                7         8         9         10        11        12        13   \
count  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0
mean        0.0       0.0       0.0       0.0       0.0       0.0       0.0
std         0.0       0.0       0.0       0.0       0.0       0.0       0.0
min         0.0       0.0       0.0       0.0       0.0       0.0       0.0
25%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
50%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
75%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
max         0.0       0.0       0.0       0.0       0.0       0.0       0.0

                14        15        16        17        18        19        20   \
count  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0
mean        0.0       0.0       0.0       0.0       0.0       0.0       0.0
std         0.0       0.0       0.0       0.0       0.0       0.0       0.0
min         0.0       0.0       0.0       0.0       0.0       0.0       0.0
25%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
50%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
75%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
max         0.0       0.0       0.0       0.0       0.0       0.0       0.0

                21        22        23        24        25        26        27   \
count  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0
mean        0.0       0.0       0.0       0.0       0.0       0.0       0.0
std         0.0       0.0       0.0       0.0       0.0       0.0       0.0
min         0.0       0.0       0.0       0.0       0.0       0.0       0.0
25%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
```

```
50%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
75%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
max         0.0       0.0       0.0       0.0       0.0       0.0       0.0

             28             29             30             31   … \
count  150000.0  150000.000000  150000.000000  150000.000000   …
mean        0.0       0.005780       0.104113       0.288120   …
std         0.0       0.319313       2.632717       4.587792   …
min         0.0       0.000000       0.000000       0.000000   …
25%         0.0       0.000000       0.000000       0.000000   …
50%         0.0       0.000000       0.000000       0.000000   …
75%         0.0       0.000000       0.000000       0.000000   …
max         0.0      28.000000     127.000000     164.000000   …

                753            754       755       756       757       758  \
count  150000.000000  150000.000000  150000.0  150000.0  150000.0  150000.0
mean        0.157387       0.011920       0.0       0.0       0.0       0.0
std         3.340784       0.456895       0.0       0.0       0.0       0.0
min         0.000000       0.000000       0.0       0.0       0.0       0.0
25%         0.000000       0.000000       0.0       0.0       0.0       0.0
50%         0.000000       0.000000       0.0       0.0       0.0       0.0
75%         0.000000       0.000000       0.0       0.0       0.0       0.0
max       117.000000      28.000000       0.0       0.0       0.0       0.0

             759       760       761       762       763       764       765  \
count  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0
mean        0.0       0.0       0.0       0.0       0.0       0.0       0.0
std         0.0       0.0       0.0       0.0       0.0       0.0       0.0
min         0.0       0.0       0.0       0.0       0.0       0.0       0.0
25%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
50%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
75%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
max         0.0       0.0       0.0       0.0       0.0       0.0       0.0

             766       767       768       769       770       771       772  \
count  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0
mean        0.0       0.0       0.0       0.0       0.0       0.0       0.0
std         0.0       0.0       0.0       0.0       0.0       0.0       0.0
min         0.0       0.0       0.0       0.0       0.0       0.0       0.0
25%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
50%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
75%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
max         0.0       0.0       0.0       0.0       0.0       0.0       0.0

             773       774       775       776       777       778       779  \
count  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0  150000.0
mean        0.0       0.0       0.0       0.0       0.0       0.0       0.0
std         0.0       0.0       0.0       0.0       0.0       0.0       0.0
```

```
min         0.0       0.0       0.0       0.0       0.0       0.0       0.0
25%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
50%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
75%         0.0       0.0       0.0       0.0       0.0       0.0       0.0
max         0.0       0.0       0.0       0.0       0.0       0.0       0.0

            780       781       782       783           784
count  150000.0  150000.0  150000.0  150000.0  150000.000000
mean        0.0       0.0       0.0       0.0       2.500000
std         0.0       0.0       0.0       0.0       1.707831
min         0.0       0.0       0.0       0.0       0.000000
25%         0.0       0.0       0.0       0.0       1.000000
50%         0.0       0.0       0.0       0.0       2.500000
75%         0.0       0.0       0.0       0.0       4.000000
max         0.0       0.0       0.0       0.0       5.000000

[8 rows x 785 columns]
```

[ ]:
```
# Show the 28 pixels in center
print(df_mid.describe())
```

```
            391       392           393           394           395  \
count  150000.0  150000.0  150000.000000  150000.000000  150000.000000
mean        0.0       0.0      16.071487      77.873900      89.022967
std         0.0       0.0      33.096175     104.470091     100.659358
min         0.0       0.0       0.000000       0.000000       0.000000
25%         0.0       0.0       0.000000       0.000000       0.000000
50%         0.0       0.0       0.000000       0.000000      34.000000
75%         0.0       0.0       5.000000     187.000000     193.000000
max         0.0       0.0     233.000000     255.000000     255.000000

                396           397           398           399  \
count  150000.000000  150000.000000  150000.000000  150000.000000
mean       93.113167      99.674453     105.749780     111.392773
std       103.005679     104.218451     104.863114     105.675783
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%        38.000000      57.000000      77.000000      93.000000
75%       205.000000     215.000000     223.000000     230.000000
max       255.000000     255.000000     255.000000     255.000000

                400           401           402           403  \
count  150000.000000  150000.000000  150000.000000  150000.000000
mean       116.316193     119.133680     120.280240     120.820567
std        106.187348     106.646262     106.512676     106.450905
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%       107.000000     115.000000     118.000000     119.000000
```

```
75%       236.000000     239.000000     239.000000     240.000000
max       255.000000     255.000000     255.000000     255.000000

                404            405            406            407  \
count  150000.000000  150000.000000  150000.000000  150000.000000
mean      119.324933     116.869247     117.036580     118.894807
std       106.778142     106.826461     106.703533     106.320279
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%       115.000000     108.000000     108.000000     114.000000
75%       240.000000     238.000000     238.000000     239.000000
max       255.000000     255.000000     255.000000     255.000000

                408            409            410           411  \
count  150000.000000  150000.000000  150000.000000  150000.00000
mean      119.458227     118.670393     117.435033     114.36366
std       106.146854     106.150995     106.093541     105.81381
min         0.000000       0.000000       0.000000       0.00000
25%         0.000000       0.000000       0.000000       0.00000
50%       117.000000     114.000000     111.000000     102.00000
75%       238.000000     238.000000     237.000000     233.00000
max       255.000000     255.000000     255.000000     255.00000

                412            413            414            415  \
count  150000.000000  150000.000000  150000.000000  150000.000000
mean      109.872207     104.477713      97.911473      90.461853
std       105.244956     104.594383     103.666818     102.332311
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%        89.000000      74.000000      53.000000      31.000000
75%       227.000000     221.000000     212.000000     200.000000
max       255.000000     255.000000     255.000000     255.000000

                416            417            418
count  150000.000000  150000.000000  150000.000000
mean       85.214113      73.174820      15.327413
std       100.050587     102.419959      32.700069
min         0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000
50%        24.000000       0.000000       0.000000
75%       187.000000     168.000000       2.000000
max       255.000000     255.000000     229.000000
```

4. Printing the class distribution

[ ]:
```
# 4. Printing the class distribution
print(df.groupby(784).size())
```

```
784
0.0    25000
1.0    25000
2.0    25000
3.0    25000
4.0    25000
5.0    25000
dtype: int64
```

5. Printing the data types of the features.

[ ]:
```
#  5. Printing the data types of the features.
print(df.dtypes)
```

```
0      float64
1      float64
2      float64
3      float64
4      float64
        …
780    float64
781    float64
782    float64
783    float64
784    float64
Length: 785, dtype: object
```

6. Printing the Pearson Correlation.

[ ]:
```
# 6. Printing the Pearson Correlation.
# Only use the 28 pixels in center because most of the pixels are zeros in␣
 ↪other areas.
features = list(df.columns[391:419])
df_mid = df[features]
print(df_mid.corr(method='pearson'))
```

```
     391  392       393       394       395       396       397       398  \
391  NaN  NaN       NaN       NaN       NaN       NaN       NaN       NaN
392  NaN  NaN       NaN       NaN       NaN       NaN       NaN       NaN
393  NaN  NaN  1.000000  0.781378  0.135300 -0.040004 -0.057305 -0.060787
394  NaN  NaN  0.781378  1.000000  0.522337  0.061733 -0.036313 -0.058031
395  NaN  NaN  0.135300  0.522337  1.000000  0.599488  0.203512  0.077573
396  NaN  NaN -0.040004  0.061733  0.599488  1.000000  0.659972  0.285539
397  NaN  NaN -0.057305 -0.036313  0.203512  0.659972  1.000000  0.674497
398  NaN  NaN -0.060787 -0.058031  0.077573  0.285539  0.674497  1.000000
399  NaN  NaN -0.043471 -0.046485  0.037201  0.145972  0.307702  0.674939
400  NaN  NaN -0.015608 -0.021281  0.024619  0.084455  0.163029  0.309816
401  NaN  NaN  0.005347 -0.004068  0.015873  0.049172  0.095983  0.165679
```

402	NaN	NaN	0.011984	0.004523	0.014296	0.031313	0.059169	0.101694
403	NaN	NaN	0.004557	-0.000502	0.012509	0.027334	0.045168	0.068630
404	NaN	NaN	-0.004541	-0.013299	0.000840	0.020048	0.037880	0.054262
405	NaN	NaN	-0.004188	-0.011816	-0.001696	0.012717	0.032804	0.050775
406	NaN	NaN	0.001304	-0.005839	0.003107	0.015890	0.031095	0.047172
407	NaN	NaN	0.010586	0.005444	0.009674	0.016208	0.030056	0.045335
408	NaN	NaN	0.018443	0.013842	0.008917	0.012828	0.027680	0.044647
409	NaN	NaN	0.021249	0.014732	0.007043	0.012893	0.027599	0.050805
410	NaN	NaN	0.017322	0.011369	0.008762	0.016549	0.035947	0.064832
411	NaN	NaN	0.006987	0.002177	0.009323	0.022942	0.046283	0.076498
412	NaN	NaN	0.001964	-0.000627	0.015717	0.032878	0.053811	0.079506
413	NaN	NaN	0.009484	0.009860	0.030696	0.051136	0.067076	0.076305
414	NaN	NaN	0.028782	0.037650	0.057324	0.074598	0.078324	0.063146
415	NaN	NaN	0.049520	0.070600	0.094954	0.096135	0.070059	0.046246
416	NaN	NaN	0.089831	0.132866	0.136903	0.090120	0.049445	0.026079
417	NaN	NaN	0.139525	0.180374	0.125060	0.055849	0.026371	0.008880
418	NaN	NaN	0.122562	0.138775	0.082855	0.034672	0.019617	0.009768

	399	400	401	402	403	404	405	\
391	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
392	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
393	-0.043471	-0.015608	0.005347	0.011984	0.004557	-0.004541	-0.004188	
394	-0.046485	-0.021281	-0.004068	0.004523	-0.000502	-0.013299	-0.011816	
395	0.037201	0.024619	0.015873	0.014296	0.012509	0.000840	-0.001696	
396	0.145972	0.084455	0.049172	0.031313	0.027334	0.020048	0.012717	
397	0.307702	0.163029	0.095983	0.059169	0.045168	0.037880	0.032804	
398	0.674939	0.309816	0.165679	0.101694	0.068630	0.054262	0.050775	
399	1.000000	0.660315	0.287654	0.153707	0.094955	0.070188	0.063540	
400	0.660315	1.000000	0.636886	0.262199	0.138373	0.094361	0.082472	
401	0.287654	0.636886	1.000000	0.617134	0.237931	0.145780	0.125457	
402	0.153707	0.262199	0.617134	1.000000	0.595088	0.235326	0.186698	
403	0.094955	0.138373	0.237931	0.595088	1.000000	0.592881	0.277422	
404	0.070188	0.094361	0.145780	0.235326	0.592881	1.000000	0.614103	
405	0.063540	0.082472	0.125457	0.186698	0.277422	0.614103	1.000000	
406	0.061979	0.083056	0.118756	0.181011	0.255860	0.324401	0.625143	
407	0.061686	0.087589	0.122588	0.167838	0.228864	0.273983	0.308149	
408	0.067713	0.104442	0.141396	0.168317	0.188925	0.213787	0.230621	
409	0.083119	0.124542	0.160728	0.170247	0.155558	0.149616	0.161949	
410	0.098925	0.131967	0.153990	0.149690	0.122593	0.103001	0.107995	
411	0.105182	0.123777	0.127354	0.113501	0.090499	0.077695	0.080857	
412	0.096826	0.102186	0.093395	0.078937	0.067671	0.064130	0.067961	
413	0.074837	0.071896	0.059193	0.050118	0.047885	0.050573	0.055195	
414	0.050325	0.045780	0.037604	0.030437	0.032508	0.036445	0.037521	
415	0.033479	0.028864	0.024679	0.020712	0.022621	0.024332	0.021334	
416	0.015384	0.015845	0.017857	0.018913	0.016860	0.012465	0.003810	
417	0.005012	0.009024	0.018767	0.022429	0.017215	0.002967	-0.007549	
418	0.010446	0.015228	0.024989	0.026101	0.020760	0.003983	-0.003865	

	406	407	408	409	410	411	412	\
391	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
392	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
393	0.001304	0.010586	0.018443	0.021249	0.017322	0.006987	0.001964	
394	-0.005839	0.005444	0.013842	0.014732	0.011369	0.002177	-0.000627	
395	0.003107	0.009674	0.008917	0.007043	0.008762	0.009323	0.015717	
396	0.015890	0.016208	0.012828	0.012893	0.016549	0.022942	0.032878	
397	0.031095	0.030056	0.027680	0.027599	0.035947	0.046283	0.053811	
398	0.047172	0.045335	0.044647	0.050805	0.064832	0.076498	0.079506	
399	0.061979	0.061686	0.067713	0.083119	0.098925	0.105182	0.096826	
400	0.083056	0.087589	0.104442	0.124542	0.131967	0.123777	0.102186	
401	0.118756	0.122588	0.141396	0.160728	0.153990	0.127354	0.093395	
402	0.181011	0.167838	0.168317	0.170247	0.149690	0.113501	0.078937	
403	0.255860	0.228864	0.188925	0.155558	0.122593	0.090499	0.067671	
404	0.324401	0.273983	0.213787	0.149616	0.103001	0.077695	0.064130	
405	0.625143	0.308149	0.230621	0.161949	0.107995	0.080857	0.067961	
406	1.000000	0.600271	0.255546	0.176347	0.126974	0.091388	0.077342	
407	0.600271	1.000000	0.587279	0.232267	0.149397	0.107336	0.085997	
408	0.255546	0.587279	1.000000	0.596879	0.246284	0.154300	0.111453	
409	0.176347	0.232267	0.596879	1.000000	0.627358	0.277146	0.169640	
410	0.126974	0.149397	0.246284	0.627358	1.000000	0.645196	0.302831	
411	0.091388	0.107336	0.154300	0.277146	0.645196	1.000000	0.666816	
412	0.077342	0.085997	0.111453	0.169640	0.302831	0.666816	1.000000	
413	0.061398	0.068611	0.081141	0.112302	0.174216	0.315628	0.676569	
414	0.040056	0.047927	0.057247	0.072859	0.106392	0.170864	0.313839	
415	0.018480	0.025246	0.035923	0.046235	0.061609	0.091618	0.150523	
416	-0.002864	0.000948	0.008288	0.014973	0.020490	0.027001	0.039722	
417	-0.013952	-0.015224	-0.012090	-0.011809	-0.011757	-0.024338	-0.044544	
418	-0.004257	-0.005874	-0.003856	-0.004539	-0.003906	-0.015763	-0.036810	

	413	414	415	416	417	418
391	NaN	NaN	NaN	NaN	NaN	NaN
392	NaN	NaN	NaN	NaN	NaN	NaN
393	0.009484	0.028782	0.049520	0.089831	0.139525	0.122562
394	0.009860	0.037650	0.070600	0.132866	0.180374	0.138775
395	0.030696	0.057324	0.094954	0.136903	0.125060	0.082855
396	0.051136	0.074598	0.096135	0.090120	0.055849	0.034672
397	0.067076	0.078324	0.070059	0.049445	0.026371	0.019617
398	0.076305	0.063146	0.046246	0.026079	0.008880	0.009768
399	0.074837	0.050325	0.033479	0.015384	0.005012	0.010446
400	0.071896	0.045780	0.028864	0.015845	0.009024	0.015228
401	0.059193	0.037604	0.024679	0.017857	0.018767	0.024989
402	0.050118	0.030437	0.020712	0.018913	0.022429	0.026101
403	0.047885	0.032508	0.022621	0.016860	0.017215	0.020760
404	0.050573	0.036445	0.024332	0.012465	0.002967	0.003983
405	0.055195	0.037521	0.021334	0.003810	-0.007549	-0.003865
406	0.061398	0.040056	0.018480	-0.002864	-0.013952	-0.004257
407	0.068611	0.047927	0.025246	0.000948	-0.015224	-0.005874

```
408  0.081141  0.057247  0.035923  0.008288 -0.012090 -0.003856
409  0.112302  0.072859  0.046235  0.014973 -0.011809 -0.004539
410  0.174216  0.106392  0.061609  0.020490 -0.011757 -0.003906
411  0.315628  0.170864  0.091618  0.027001 -0.024338 -0.015763
412  0.676569  0.313839  0.150523  0.039722 -0.044544 -0.036810
413  1.000000  0.679485  0.290926  0.082488 -0.047394 -0.045730
414  0.679485  1.000000  0.661845  0.215011 -0.016251 -0.035634
415  0.290926  0.661845  1.000000  0.611004  0.088438 -0.013074
416  0.082488  0.215011  0.611004  1.000000  0.538662  0.159435
417 -0.047394 -0.016251  0.088438  0.538662  1.000000  0.786495
418 -0.045730 -0.035634 -0.013074  0.159435  0.786495  1.000000
```

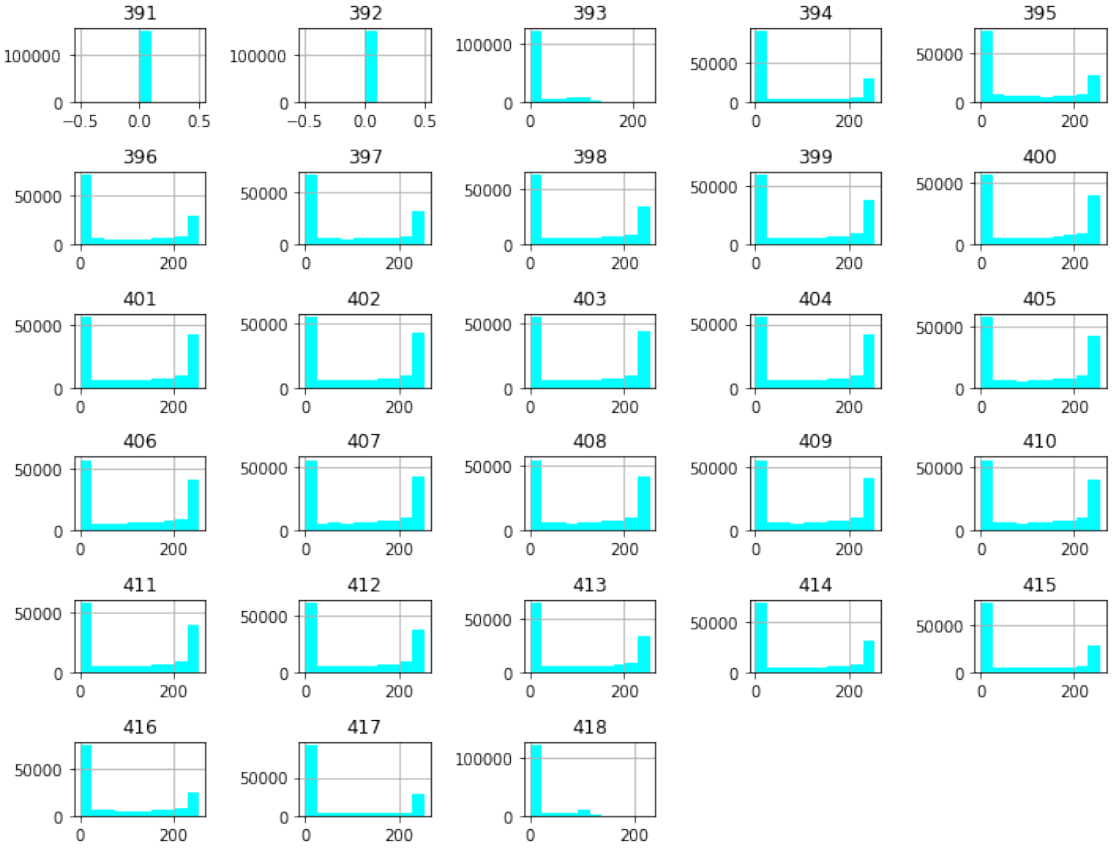
7. Printing the skewness of the dataset.

[ ]:
```
# 7. Printing the skewness of the dataset (28 attributes in center).
print(df_mid.skew())
```

```
391    0.000000
392    0.000000
393    1.958852
394    0.823587
395    0.581306
396    0.504271
397    0.392541
398    0.292173
399    0.202713
400    0.123766
401    0.079506
402    0.063172
403    0.056665
404    0.080748
405    0.119656
406    0.117515
407    0.086738
408    0.076377
409    0.087730
410    0.106146
411    0.155359
412    0.225863
413    0.311649
414    0.421201
415    0.550212
416    0.648557
417    0.918921
418    2.051123
dtype: float64
```

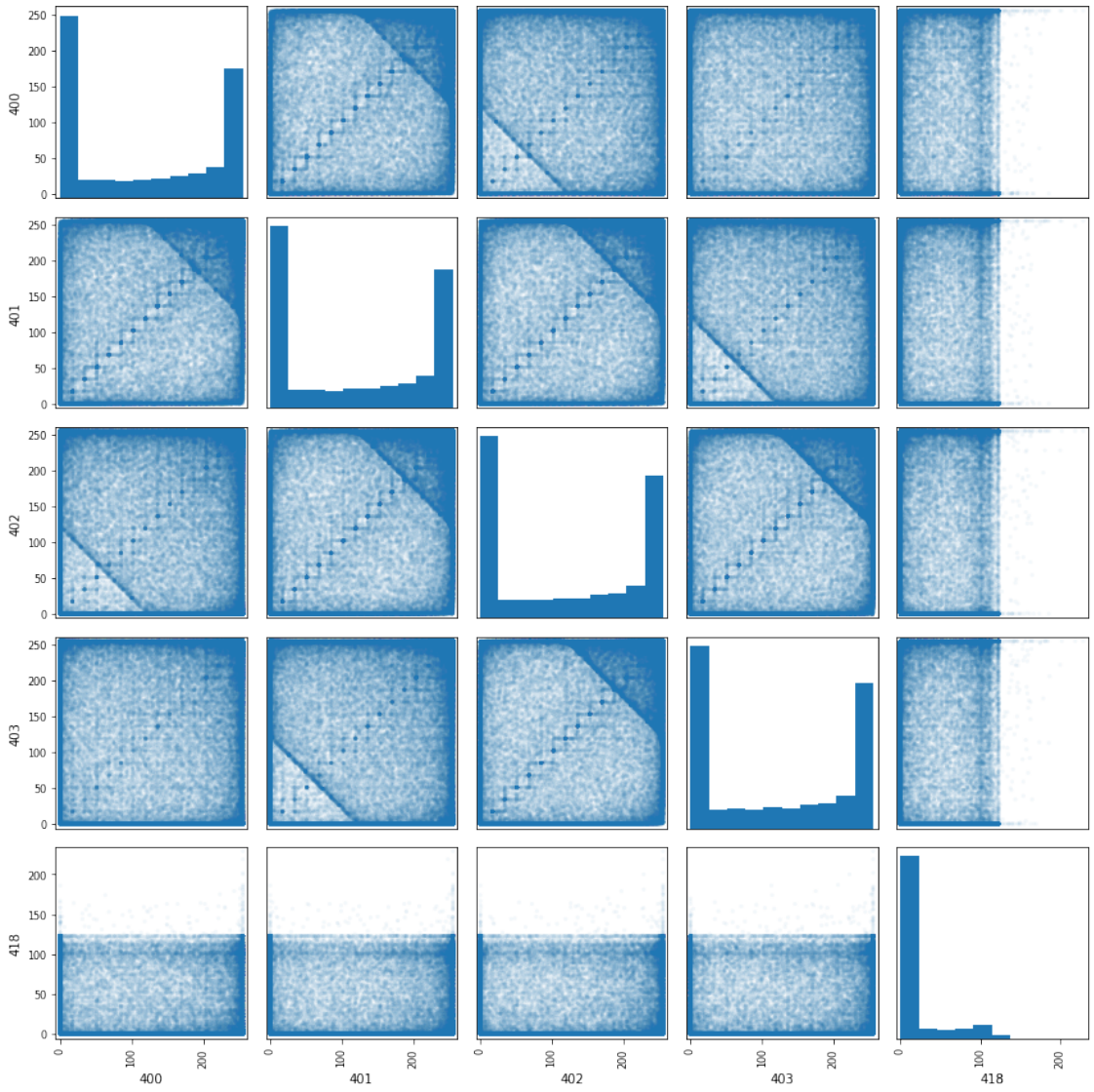
8. Histograms of all features

```
[ ]: # 8. Histograms of all features
pylab.rcParams['figure.figsize'] = (10,20)
df_mid.hist(color='cyan', layout=(16,5))
plt.tight_layout()
plt.show()
```



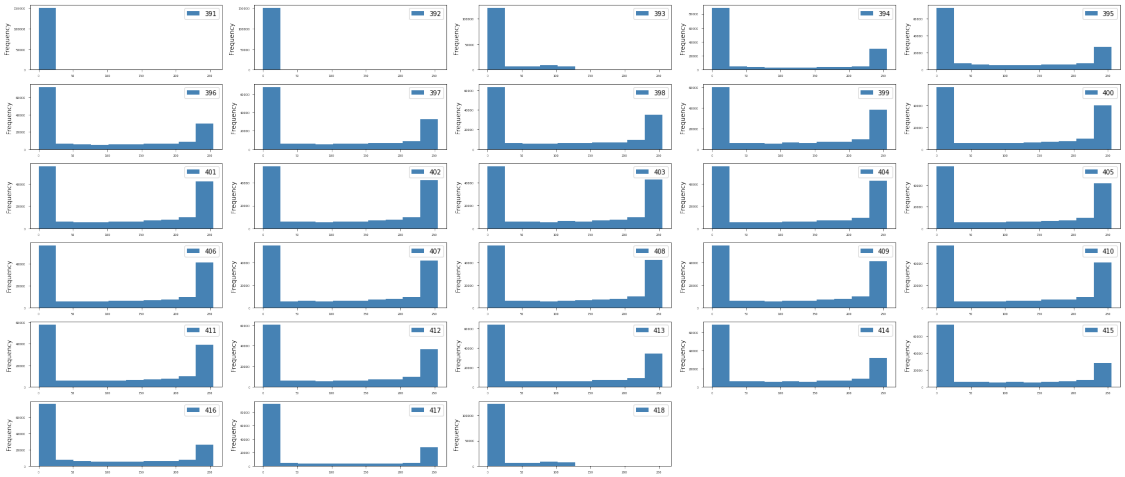
9. Scatter Matrix

```
[ ]: # 9. Scatter Matrix
pylab.rcParams['figure.figsize'] = (12,12)
scatter_matrix(df_mid[[400,401,402,403,418]],alpha=0.05)
plt.tight_layout()
plt.show()
```



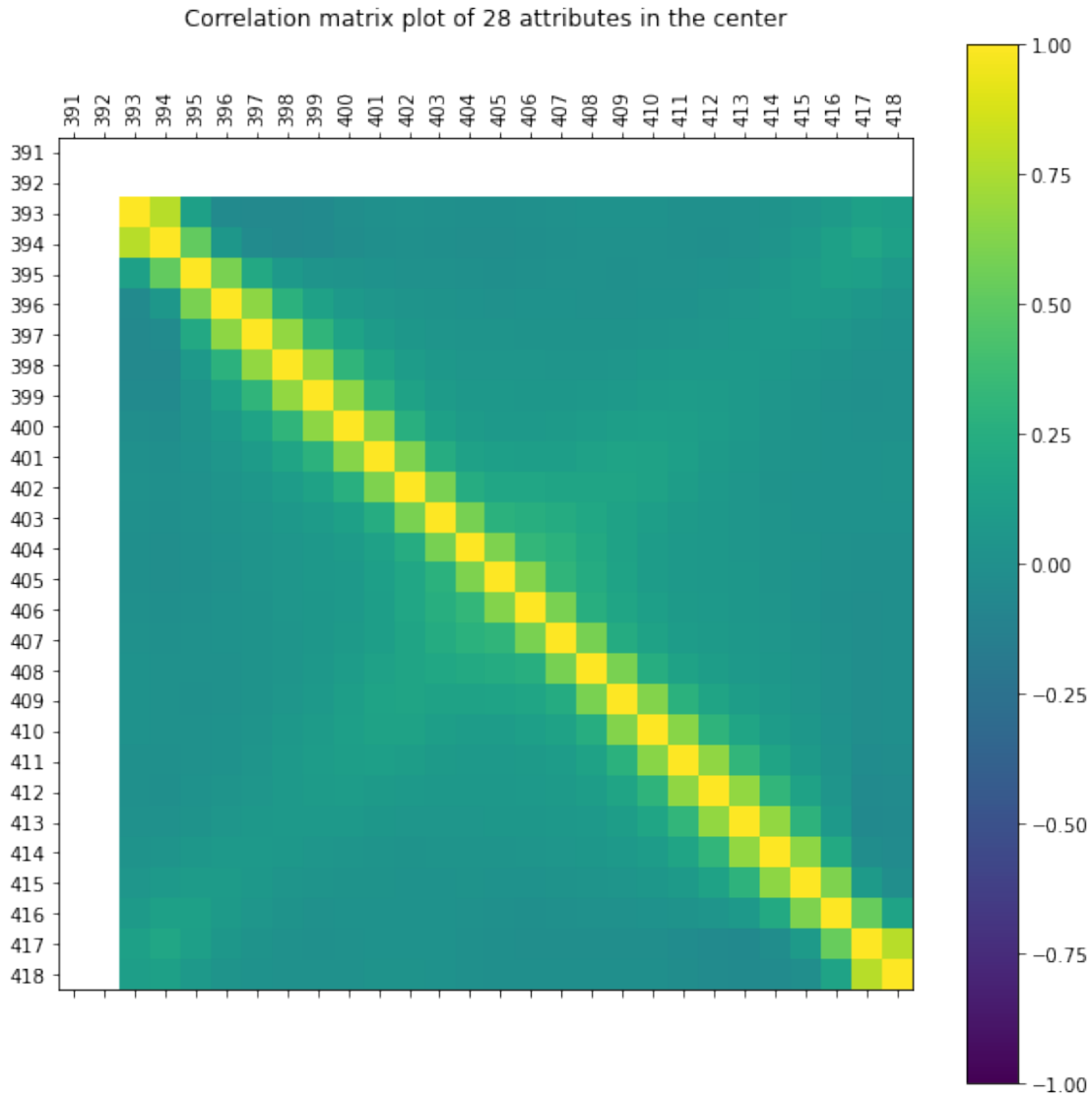
10. Density plot for each feature.

[ ]:
```
# 10. Density plot for each feature.
# pylab.rcParams['figure.figsize'] = (10,25)
# print(df_mid)
df_mid.select_dtypes('float').plot(kind='hist', subplots=True, layout=(20,5),␣
 ↪figsize=(25,35), sharex=False, sharey = False, fontsize=5, color='steelblue')
plt.tight_layout()
plt.show()
```



11. Correlation Matrix Plot

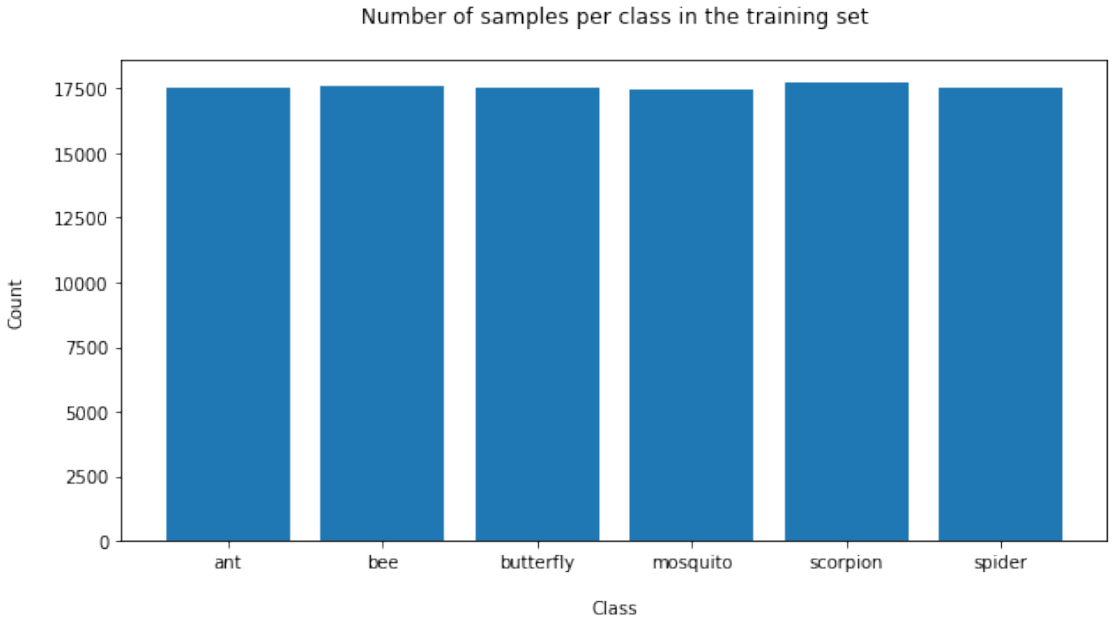
[ ]:
```
# 11. Correlation Matrix Plot
pylab.rcParams['figure.figsize'] = (10,10)
correlations = df_mid.corr()
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin = -1, vmax = 1, cmap='viridis')
fig.colorbar(cax)
ticks = np.arange(0,len(df_mid.columns),1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(df_mid.columns, rotation=90)
ax.set_yticklabels(df_mid.columns, rotation=0)
plt.title('Correlation matrix plot of 28 attributes in the center\n\n')
plt.show()
```



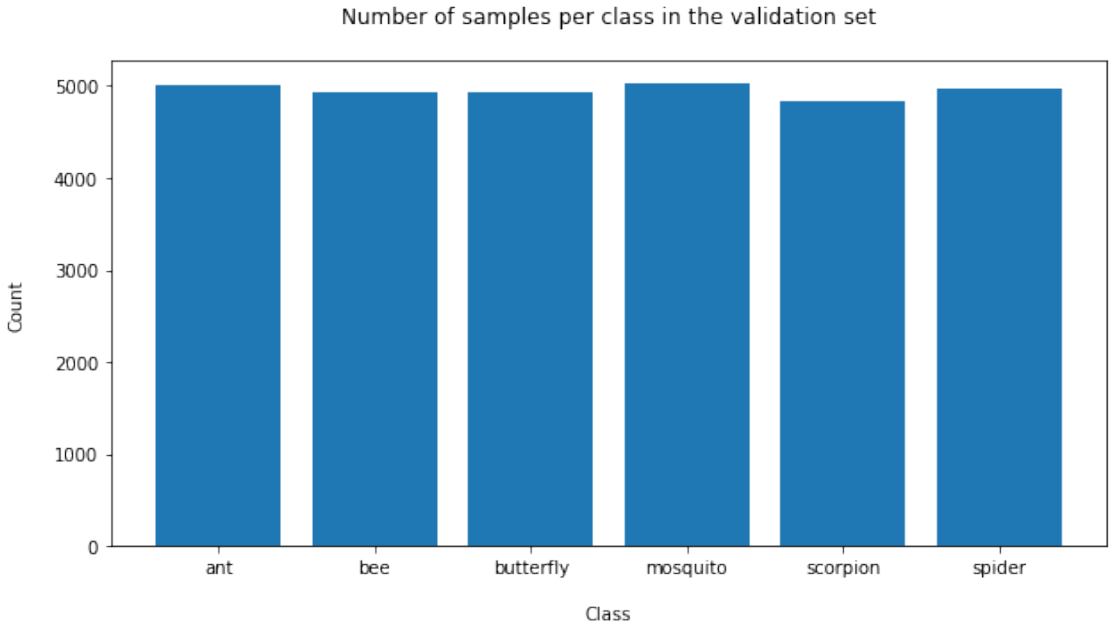
12. Classification: split the dataset into train, validate and test

[ ]:
```
# Classification: split the dataset into train, validate and test
test_size = 0.10
val_size = 0.22
seed = 84020
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=test_size,␣
↪random_state=seed)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.
↪22, random_state=seed)
```

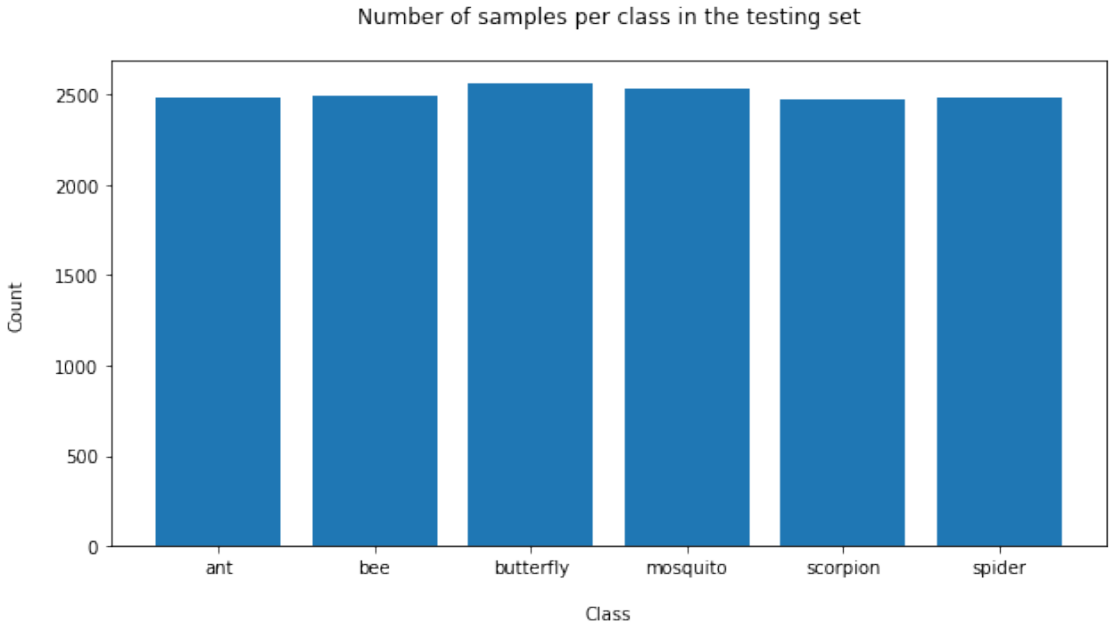
[ ]:
```
# Training data
unique, counts = np.unique(Y_train, return_counts=True)
pylab.rcParams['figure.figsize'] = (10,5)
plt.bar(critters, counts)
plt.title('Number of samples per class in the training set\n')
plt.ylabel('Count\n')
plt.xlabel('\nClass')
plt.show()
```



[ ]:
```
# Validation data
unique, counts = np.unique(Y_val, return_counts=True)
pylab.rcParams['figure.figsize'] = (10,5)
plt.bar(critters, counts)
plt.title('Number of samples per class in the validation set\n')
plt.ylabel('Count\n')
plt.xlabel('\nClass')
plt.show()
```



[ ]:
```
# Testing data
unique, counts = np.unique(Y_test, return_counts=True)
pylab.rcParams['figure.figsize'] = (10,5)
plt.bar(critters, counts)
plt.title('Number of samples per class in the testing set\n')
plt.ylabel('Count\n')
plt.xlabel('\nClass')
plt.show()
```



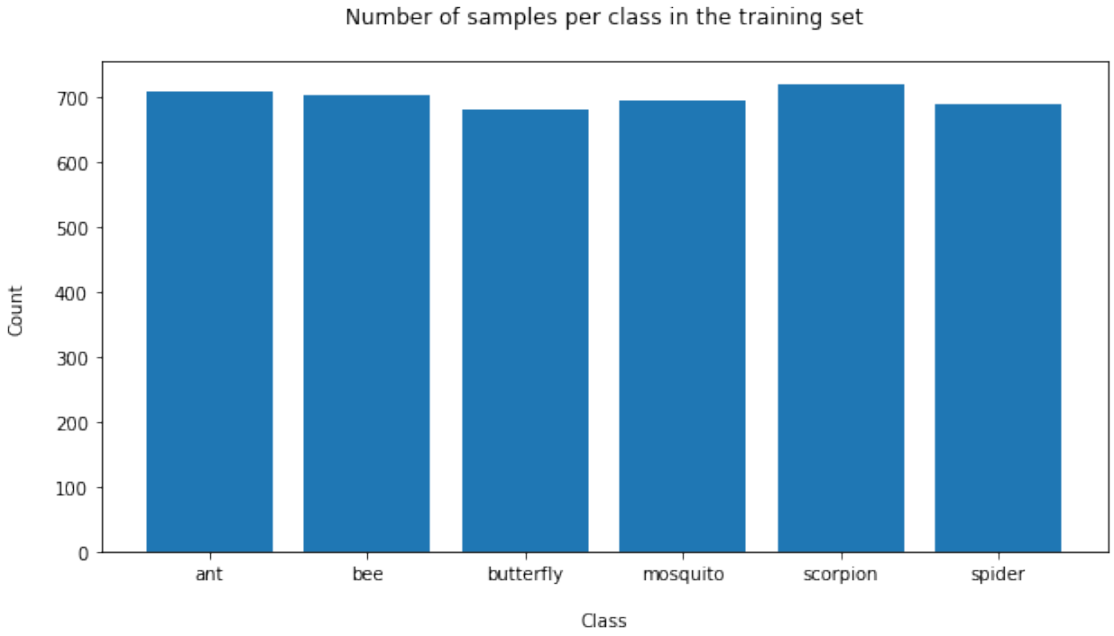
[ ]:
```
# Use small set of 6 classes for Spot Check Algorithms
classes = critters
CLASS_SAMPLE_MAX = 1000
DATA_DIR = '/content/drive/My Drive/Colab Notebooks/NNDL/HW2/data/'
```

[ ]:
```
start = time.time()
X, y = load_bitmaps()
print(f'\nLoading time: %.3f' % int(time.time() - start), 'seconds')
print(X.shape)
print(y.shape)
print(y[-CLASS_SAMPLE_MAX-5:-CLASS_SAMPLE_MAX+5])
```

```
ant bee butterfly mosquito scorpion spider
0 1 2 3 4 5
Loading time: 0.000 seconds
(6000, 784)
(6000,)
[4. 4. 4. 4. 4. 5. 5. 5. 5. 5.]
```

[ ]:
```
test_size = 0.10
val_size = 0.2
seed = 46
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=test_size,␣
 ↪random_state=seed)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train,␣
 ↪test_size=val_size/(1-test_size), random_state=seed)
```

[ ]:
```
unique, counts = np.unique(Y_train, return_counts=True)
pylab.rcParams['figure.figsize'] = (10,5)
plt.bar(critters, counts)
plt.title('Number of samples per class in the training set\n')
plt.ylabel('Count\n')
plt.xlabel('\nClass')
plt.show()
```



13. Spot-Check Algorithms

[ ]:
```
# Spot-Check Algorithms
start = time.time()
models = []
models.append(('LR', LogisticRegression(C=0.01,solver='sag')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=seed)))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
for name, model in models:
  kfold = KFold(n_splits=10,random_state=seed, shuffle=True)
  # cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,␣
 ↪scoring='f1_micro')
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,␣
 ↪scoring='accuracy')
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

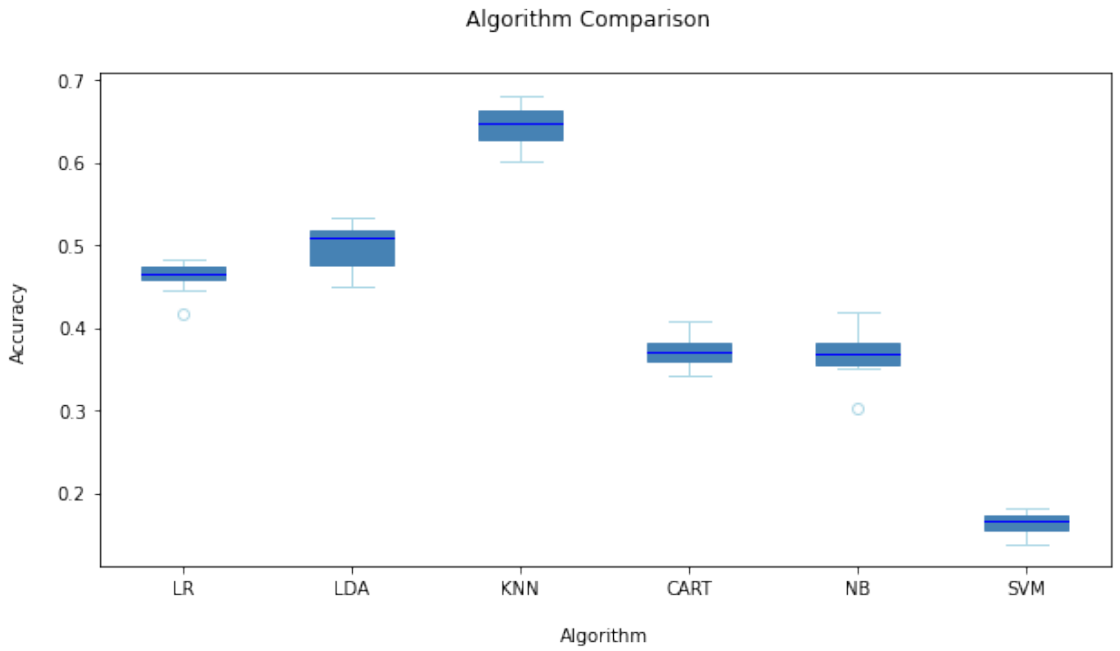
```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
```

```
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)

LR: 0.461667 (0.018302)
LDA: 0.498571 (0.025648)
KNN: 0.645238 (0.023738)
CART: 0.370952 (0.018158)
NB: 0.367381 (0.028652)
SVM: 0.163333 (0.012381)
```

14. Compare Algorithms

[ ]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
c1 = 'steelblue'
c2 = 'blue'
c3 = 'lightblue'
box = plt.boxplot(results, patch_artist=True,
                  boxprops=dict(facecolor=c1, color=c1),
                  capprops=dict(color=c3),
                  whiskerprops=dict(color=c3),
                  flierprops=dict(color=c3, markeredgecolor=c3),
                  medianprops=dict(color=c2))
ax.set_xticklabels(names)
plt.ylabel('Accuracy\n')
plt.xlabel('\nAlgorithm')
plt.show()
plt.show()
```



15. Make predictions on validation dataset

[ ]:
```
# Make predictions on validation dataset
knn = KNeighborsClassifier()
knn.fit(X_train,Y_train)
```

```
pred_knn = knn.predict(X_val)
print('accuracy:',accuracy_score(Y_val, pred_knn))
print('confusion matrix:\n',confusion_matrix(Y_val, pred_knn))
```

```
accuracy: 0.6716666666666666
confusion matrix:
 [[139   8   1  21  15  11]
 [ 32 125   9  11  14   8]
 [ 11  17 174   5   4   4]
 [ 47  15   5 105   9  23]
 [ 37  14   0  15 115   5]
 [ 21   9   2  17   4 148]]
```

16. Classification report

[ ]:
```
print('classification report:\n', classification_report(Y_val, pred_knn))
```

```
classification report:
               precision    recall  f1-score   support

         0.0       0.48      0.71      0.58       195
         1.0       0.66      0.63      0.65       199
         2.0       0.91      0.81      0.86       215
         3.0       0.60      0.51      0.56       204
         4.0       0.71      0.62      0.66       186
         5.0       0.74      0.74      0.74       201

    accuracy                           0.67      1200
   macro avg       0.69      0.67      0.67      1200
weighted avg       0.69      0.67      0.68      1200
```