csc177-p1-datapreprocessing

October 11, 2023

1 CSC177- Project 1

2 Data Preprocessing Project (Fall 2023)

2.1 Team Challengers:

1. Srujay Reddy Vangoor
2. Vaibhav Jain
3. Bashar Allwza
4. Varun Bailapudi
5. Uddayankith Chodagam

2.2 1.0 Introduction

[1]:
```
import os
import io

import pandas as pd
import numpy as np

import matplotlib as pltLib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
data = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/
 ↪CSC177-DataAnalyticsAndMining/Project1-DataPrepocessing/
 ↪Acoustic_Extinguisher_Fire_Dataset.xlsx')
data.columns = ['SIZE', 'FUEL', 'DISTANCE', 'DESIBEL', 'AIRFLOW', 'FREQUENCY',
```

```
       'STATUS']
```

[4]:
```
pip install pandoc
```

```
Collecting pandoc
  Downloading pandoc-2.3.tar.gz (33 kB)
  Preparing metadata (setup.py) … done
Collecting plumbum (from pandoc)
  Downloading plumbum-1.8.2-py3-none-any.whl (127 kB)
                           127.0/127.0

kB 5.3 MB/s eta 0:00:00
Collecting ply (from pandoc)
  Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
                           49.6/49.6 kB
4.5 MB/s eta 0:00:00
Building wheels for collected packages: pandoc
  Building wheel for pandoc (setup.py) … done
  Created wheel for pandoc: filename=pandoc-2.3-py3-none-any.whl size=33261
sha256=9033b22aff17f72e0036693defa6f7703da71ab4ce7959b60f79f1f8c3c61916
  Stored in directory: /root/.cache/pip/wheels/76/27/c2/c26175310aadcb8741b77657
a1bb49c50cc7d4cdbf9eee0005
Successfully built pandoc
Installing collected packages: ply, plumbum, pandoc
Successfully installed pandoc-2.3 plumbum-1.8.2 ply-3.11
```

[5]:
```
data.head(10)
```

[5]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
0	1	gasoline	10.0	96.0	0.0	75.0	0
1	1	gasoline	10.0	96.0	0.0	72.0	1
2	1	gasoline	10.0	96.0	2.6	70.0	1
3	1	gasoline	10.0	96.0	3.2	68.0	1
4	1	gasoline	10.0	109.0	4.5	67.0	1
5	1	gasoline	10.0	109.0	7.8	66.0	1
6	1	gasoline	10.0	103.0	9.7	65.0	1
7	1	gasoline	10.0	95.0	12.0	60.0	1
8	1	gasoline	10.0	102.0	13.3	55.0	1
9	1	gasoline	NaN	93.0	15.4	52.0	1

We will check total number of rows and columns in the dataset.

[6]:
```
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
```

```
Number of instances = 17442
Number of attributes = 7
```

We will check number of empty NA data fields for each attribute/column in the dataset. (We have

7 attributes).

[7]:
```
for col in data.columns:
    print('%s :  %d' % (col,data[col].isna().sum()))
    print()
```

```
SIZE :  0

FUEL :  0

DISTANCE :  37

DESIBEL :  11

AIRFLOW :  23

FREQUENCY :  29

STATUS :  0
```

We see that there are lot of NaN values for DISTANCE, DESIBEL, AIRFLOW and FREQUENCY atrributes.

[8]:
```
data.DISTANCE.value_counts()
```

[8]:
```
100.0    918
110.0    918
180.0    918
170.0    918
160.0    918
150.0    918
140.0    918
130.0    918
120.0    918
190.0    918
80.0     918
70.0     918
60.0     918
50.0     918
40.0     918
90.0     916
10.0     914
30.0     903
20.0     902
Name: DISTANCE, dtype: int64
```

[9]:
```
# As distance is a ordinal attribute, we cannot decide on how to fill the NA
↪values with the mean, median, or mode
# So we are just dropping the NA rows
data = data[data['DISTANCE'].notna()]
```

[10]:
```
#We see that shape of data changed before and after the removal from 17442 to
↪17405
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
```

```
Number of instances = 17405
Number of attributes = 7
```

[11]:
```
#For airflow we will fill with mean values as
data_1 = data['AIRFLOW']

print('Before replacing missing values:')
print(data_1.iloc[30:40])
data_1 = data_1.fillna(data_1.mean())
print('\nAfter replacing missing values:')
print(data_1.iloc[30:40])
```

```
Before replacing missing values:
34      NaN
35     15.2
36     15.1
37     14.5
38     13.8
39     14.4
40      NaN
41      NaN
42     11.9
43      NaN
Name: AIRFLOW, dtype: float64

After replacing missing values:
34      6.96171
35     15.20000
36     15.10000
37     14.50000
38     13.80000
39     14.40000
40      6.96171
41      6.96171
42     11.90000
43      6.96171
Name: AIRFLOW, dtype: float64
```

[12]:
```python
data['AIRFLOW'] = data_1
```

[13]:
```python
data[30:40]
```

[13]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
34	1	gasoline	10.0	108.0	6.96171	20.0	1
35	1	gasoline	10.0	108.0	15.20000	19.0	1
36	1	gasoline	10.0	106.0	15.10000	18.0	1
37	1	gasoline	10.0	105.0	14.50000	17.0	1
38	1	gasoline	10.0	105.0	13.80000	16.0	1
39	1	gasoline	10.0	106.0	14.40000	15.0	1
40	1	gasoline	10.0	106.0	6.96171	14.0	1
41	1	gasoline	10.0	93.0	6.96171	13.0	1
42	1	gasoline	10.0	90.0	11.90000	12.0	1
43	1	gasoline	10.0	92.0	6.96171	11.0	1

[14]:
```python
data.DESIBEL.value_counts()
#For DESIBEL we will fill with mean values as
data_1 = data['DESIBEL']

print('Before replacing missing values:')
print(data['DESIBEL'].iloc[20:30])
data_1 = data_1.fillna(data_1.mean())
print('\nAfter replacing missing values:')
print(data_1.iloc[20:30])
```

```
Before replacing missing values:
24      NaN
25    109.0
26    108.0
27      NaN
28    107.0
29    109.0
30    108.0
31    108.0
32    108.0
33    109.0
Name: DESIBEL, dtype: float64

After replacing missing values:
24     96.370043
25    109.000000
26    108.000000
27     96.370043
28    107.000000
29    109.000000
30    108.000000
```

```
31     108.000000
32     108.000000
33     109.000000
Name: DESIBEL, dtype: float64
```

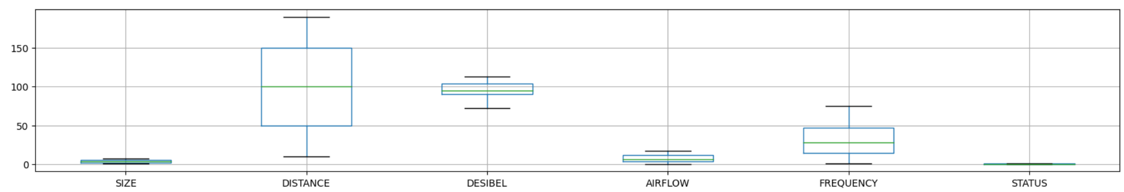
[15]:
```python
data['DESIBEL']  = data_1
```

[16]:
```python
#Dropping NA data items for frequency attribute
data = data[data['FREQUENCY'].notna()]
```

2.3 2.0 Check for outliers

[17]:
```python
data.boxplot(figsize=(20,3))
```

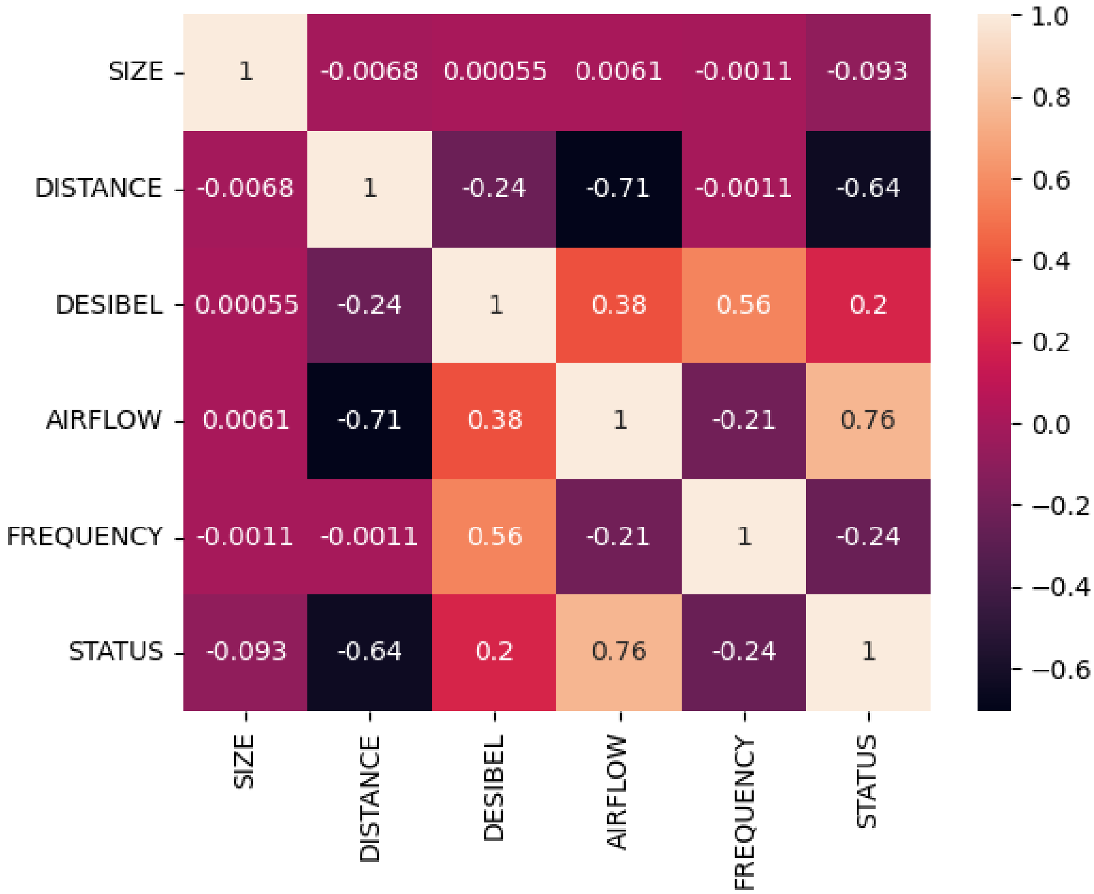
[17]:
```
<Axes: >
```



2.4 3.0 Generate Heatmap

[18]:
```python
corr = data.corr(numeric_only = True)
sns.heatmap(corr, annot = True)
```

[18]:
```
<Axes: >
```



As we see there are no outliers for any attribute

2.5 4.0 Perform Standardizing

[19]:
```
Z = (data-data.mean(numeric_only=True))/data.std(numeric_only=True)
```

[20]:
```
print('Number of rows before discarding outliers = %d' % (Z.shape[0]))

Z2 = Z.loc[((Z > -3).sum(axis=1)==9) & ((Z <= 3).sum(axis=1)==9),:]
print('Number of rows after discarding missing values = %d' % (Z2.shape[0]))
```

```
Number of rows before discarding outliers = 17379
Number of rows after discarding missing values = 0
```

[21]:
```
Z['FUEL'] = data['FUEL']
```

2.6 5.0 Check for duplicates

[22]:
```
dups = data.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
```

```
Number of duplicate rows = 0
```

2.7 6.0 Shuffle the dataset

[23]:
```
data = data.sample(frac=1).reset_index(drop=True)
```

[24]:
```
data.head()
```

[24]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
0	3	gasoline	10.0	105.0	13.8	16.0	1
1	5	kerosene	80.0	107.0	12.2	42.0	1
2	4	gasoline	10.0	92.0	12.5	11.0	1
3	6	lpg	130.0	92.0	8.3	32.0	1
4	2	gasoline	110.0	106.0	10.6	30.0	1

2.8 7.0 Sorting dataframe

[25]:
```
data = data.sort_values(by='DISTANCE',ascending=True)
```

[26]:
```
data
```

[26]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
0	3	gasoline	10.0	105.0	13.8	16.0	1
6540	1	gasoline	10.0	96.0	0.0	75.0	0
6528	4	thinner	10.0	103.0	14.9	35.0	1
6520	3	thinner	10.0	108.0	15.5	28.0	1
6434	2	thinner	10.0	92.0	12.5	11.0	1
…	…	…	…	…	…	…	
15456	1	gasoline	190.0	92.0	2.6	32.0	0
15869	3	kerosene	190.0	96.0	2.9	38.0	0
5940	3	thinner	190.0	100.0	3.2	27.0	0
1613	5	gasoline	190.0	94.0	2.2	30.0	0
6660	3	kerosene	190.0	92.0	2.7	17.0	0

[17379 rows x 7 columns]

2.9 8.0 Saving dataframe to disk as a CSV/Excel

[27]:
```
data = data.reindex(np.random.permutation(data.index))
data.to_csv('/content/drive/MyDrive/Colab Notebooks/
↪CSC177-DataAnalyticsAndMining/Project1-DataPrepocessing/Preprocessed_Dataset.
↪csv', index=False)
```

2.10 9.0 Dropping fields

[28]:
```
#We see from the correlation map that FREQUENCY is of less relevance
data.drop('FREQUENCY', axis=1, inplace=True)
```

[29]:
```
data
```

[29]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	STATUS
11595	3	gasoline	140.0	94.0	4.2	1
10083	1	kerosene	150.0	97.0	0.0	0
6173	2	gasoline	160.0	95.0	2.5	0
13348	3	kerosene	100.0	102.0	4.9	0
5375	5	gasoline	40.0	107.0	14.5	1
…	…	…	…	…	…	…
14651	4	thinner	140.0	109.0	0.0	0
1393	3	thinner	90.0	104.0	0.0	0
16745	2	kerosene	60.0	86.0	9.5	1
14946	5	kerosene	100.0	90.0	9.9	1
6727	2	thinner	160.0	94.0	6.0	0

[17379 rows x 6 columns]

2.11 10.0 Label encoding for categorical data

[30]:
```
le = preprocessing.LabelEncoder()
data['encoded_FUEL'] = le.fit_transform(data['FUEL'])
```

[31]:
```
data
```

[31]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	STATUS	encoded_FUEL
11595	3	gasoline	140.0	94.0	4.2	1	0
10083	1	kerosene	150.0	97.0	0.0	0	1
6173	2	gasoline	160.0	95.0	2.5	0	0
13348	3	kerosene	100.0	102.0	4.9	0	1
5375	5	gasoline	40.0	107.0	14.5	1	0
…	…	…	…	…	…	…	…
14651	4	thinner	140.0	109.0	0.0	0	3
1393	3	thinner	90.0	104.0	0.0	0	3
16745	2	kerosene	60.0	86.0	9.5	1	1
14946	5	kerosene	100.0	90.0	9.9	1	1
6727	2	thinner	160.0	94.0	6.0	0	3

[17379 rows x 7 columns]

2.12 11.0 Splitting the dataframe into training and testing datasets

[32]:
```
#We split the data into test and train set with trainset having 75% data and␣
↪test set having 25% data
x_train, x_test, y_train, y_test =␣
↪train_test_split(data[["SIZE","FUEL","DISTANCE","DESIBEL","AIRFLOW"]],␣
↪data["STATUS"], test_size=0.25, random_state=42)
```

[33]:
```
x_train, x_test, y_train, y_test
```

[33]:
```
(       SIZE      FUEL  DISTANCE  DESIBEL  AIRFLOW
 14613     5   thinner      80.0     87.0      7.7
 10064     1   thinner     170.0     91.0      2.1
 10661     3  gasoline     100.0    105.0      2.5
 12639     7       lpg      20.0    100.0      9.6
 1319      3  gasoline     150.0     93.0      4.5
 …       …         …         …        …
 7726      7       lpg      20.0     90.0     11.0
 14177     5  gasoline     110.0     92.0      6.5
 12517     1  gasoline      30.0    108.0     11.5
 15126     2   thinner      20.0    105.0     13.5
 3636      2   thinner      30.0     89.0     12.3

 [13034 rows x 5 columns],
        SIZE      FUEL  DISTANCE  DESIBEL  AIRFLOW
 140       5  kerosene     190.0     95.0      2.2
 6656      1  gasoline     140.0    106.0      4.3
 9128      3  gasoline      40.0    107.0     14.4
 494       4  kerosene     100.0    104.0      8.8
 13014     1   thinner      70.0    106.0      7.2
 …       …         …         …        …
 6437      3  kerosene      90.0    104.0     10.0
 1177      5   thinner     190.0     95.0      2.2
 12624     6       lpg     190.0    102.0      2.9
 4490      7       lpg     110.0     96.0      1.5
 13854     4   thinner      20.0    105.0     13.5

 [4345 rows x 5 columns],
 14613    0
 10064    0
 10661    0
 12639    1
 1319     0
         ..
 7726     1
 14177    1
 12517    1
```

```
15126    1
3636     1
Name: STATUS, Length: 13034, dtype: int64,
140      0
6656     0
9128     1
494      0
13014    1
        ..
6437     0
1177     0
12624    0
4490     0
13854    1
Name: STATUS, Length: 4345, dtype: int64)
```

[34]:
```python
x_train.shape
```

[34]:
```
(13034, 5)
```

[35]:
```python
x_test.shape
```

[35]:
```
(4345, 5)
```

[36]:
```python
y_train.shape
```

[36]:
```
(13034,)
```

[37]:
```python
y_test.shape
```

[37]:
```
(4345,)
```

2.13 11.0 Means and Standard Deviations on both partitions of the data

[ ]:
```python
# Mean and Standard Deviation for train data
```

[38]:
```python
x_train.mean(numeric_only=True)
```

[38]:
```
SIZE          3.416833
DISTANCE    100.438852
DESIBEL      96.361419
AIRFLOW       6.924028
dtype: float64
```

[39]:
```python
x_train.std(numeric_only=True)
```

[39]:
```
SIZE         1.749579
DISTANCE    54.660977
DESIBEL      8.146239
AIRFLOW      4.711610
dtype: float64
```

[ ]:
```
# Mean and Standard Deviation for test data
```

[40]:
```
x_test.mean(numeric_only=True)
```

[40]:
```
SIZE         3.431530
DISTANCE    99.751438
DESIBEL     96.419273
AIRFLOW      7.046717
dtype: float64
```

[41]:
```
x_test.std(numeric_only=True)
```

[41]:
```
SIZE         1.743832
DISTANCE    54.769170
DESIBEL      8.182814
AIRFLOW      4.768329
dtype: float64
```

2.14 12.0 Sampling

[42]:
```
sample = data.sample(frac=0.01, random_state=1)
```

[ ]:
```
#This shows 0.01 percent of the records of whole dataset
sample
```

[ ]:

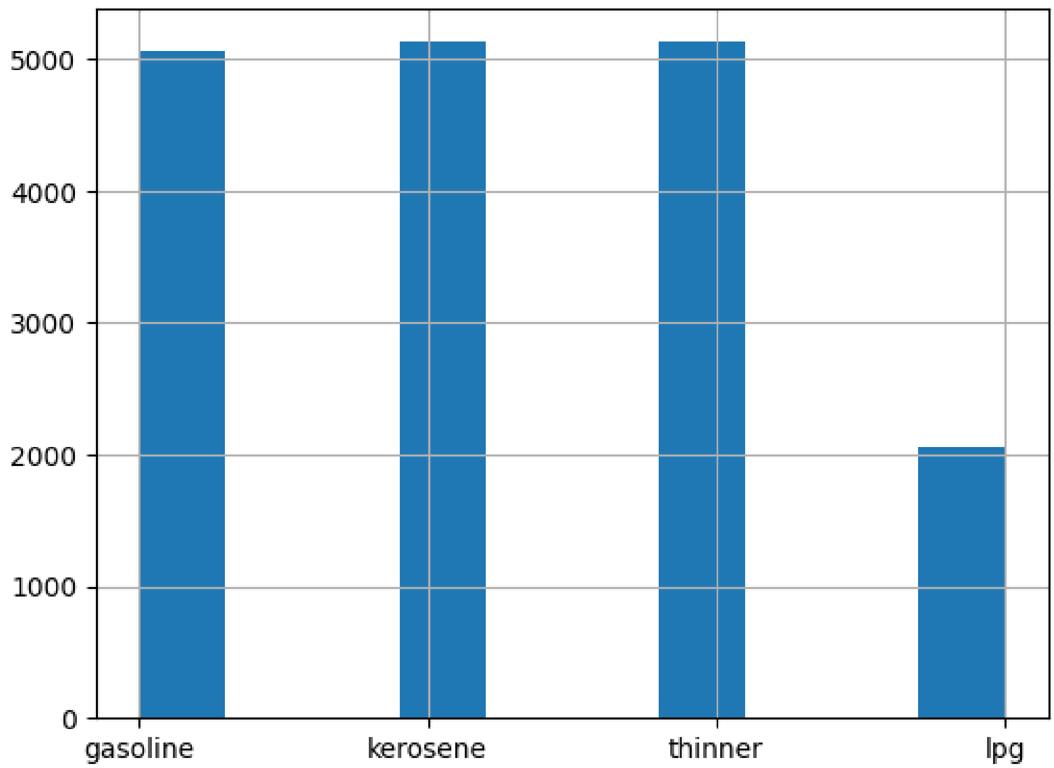
	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	STATUS
4268	4	thinner	70.0	107.0	13.6	1
3073	3	gasoline	170.0	93.0	2.2	0
12641	1	gasoline	180.0	102.0	3.2	0
5630	6	lpg	120.0	96.0	0.0	0
13940	2	thinner	100.0	86.0	6.5	0
…	…	…	…	…	…	…
8608	3	thinner	60.0	107.0	13.8	1
10678	4	gasoline	90.0	105.0	12.8	1
6076	2	thinner	20.0	103.0	12.0	1
226	1	kerosene	110.0	106.0	6.8	1
9822	4	kerosene	190.0	92.0	1.4	0

[174 rows x 6 columns]

2.15 13.0 Discretization

[43]:
```
data['FUEL'].hist(bins=10)
data['FUEL'].value_counts(sort=False)
```

[43]:
```
gasoline    5067
kerosene    5130
thinner     5130
lpg         2052
Name: FUEL, dtype: int64
```



[44]:
```
#Value counts gives how many unique data elements are present and count of each␣
↪of these unique elements
data['FUEL'].value_counts(sort=True)
```

[44]:
```
kerosene    5130
thinner     5130
gasoline    5067
lpg         2052
Name: FUEL, dtype: int64
```

[45]:
```
bins = pd.cut(data['DISTANCE'],4)
bins.value_counts(sort=False)
```

[45]:
```
(9.82, 55.0]      4529
(55.0, 100.0]     4588
(100.0, 145.0]    3672
(145.0, 190.0]    4590
Name: DISTANCE, dtype: int64
```

[46]:
```
bins = pd.qcut(data['DISTANCE'],4)
bins.value_counts(sort=False)
```

[46]:
```
(9.999, 50.0]     4529
(50.0, 100.0]     4588
(100.0, 150.0]    4590
(150.0, 190.0]    3672
Name: DISTANCE, dtype: int64
```

2.16 14.0 Performing principal component analysis

[47]:
```
numImages = 16
fig = plt.figure(figsize=(7,7))
imgData = np.zeros(shape=(numImages,36963))

for i in range(1,numImages+1):
    filename = '/content/drive/MyDrive/Colab Notebooks/
↪CSC177-DataAnalyticsAndMining/Project1-DataPrepocessing/pics/
↪Picture'+str(i)+'.jpg'

    img = mpimg.imread(filename)
    ax = fig.add_subplot(4,4,i)
    plt.imshow(img)
    plt.axis('off')
    ax.set_title(str(i))
    imgData[i-1] = np.array(img.flatten()).reshape(1,img.shape[0]*img.
↪shape[1]*img.shape[2])
```

[48]:
```python
import pandas as pd
from sklearn.decomposition import PCA

numComponents = 2
pca = PCA(n_components=numComponents)
pca.fit(imgData)

projected = pca.transform(imgData)
projected = pd.DataFrame(projected,columns=['pc1','pc2'],index=range(1,numImages+1))
projected['food'] = ['burger', 'burger','burger','burger','drink','drink','drink','drink',
```

```
                        'pasta', 'pasta', 'pasta', 'pasta', 'chicken', 'chicken',␣
 ↪'chicken', 'chicken']
projected
```

[48]:

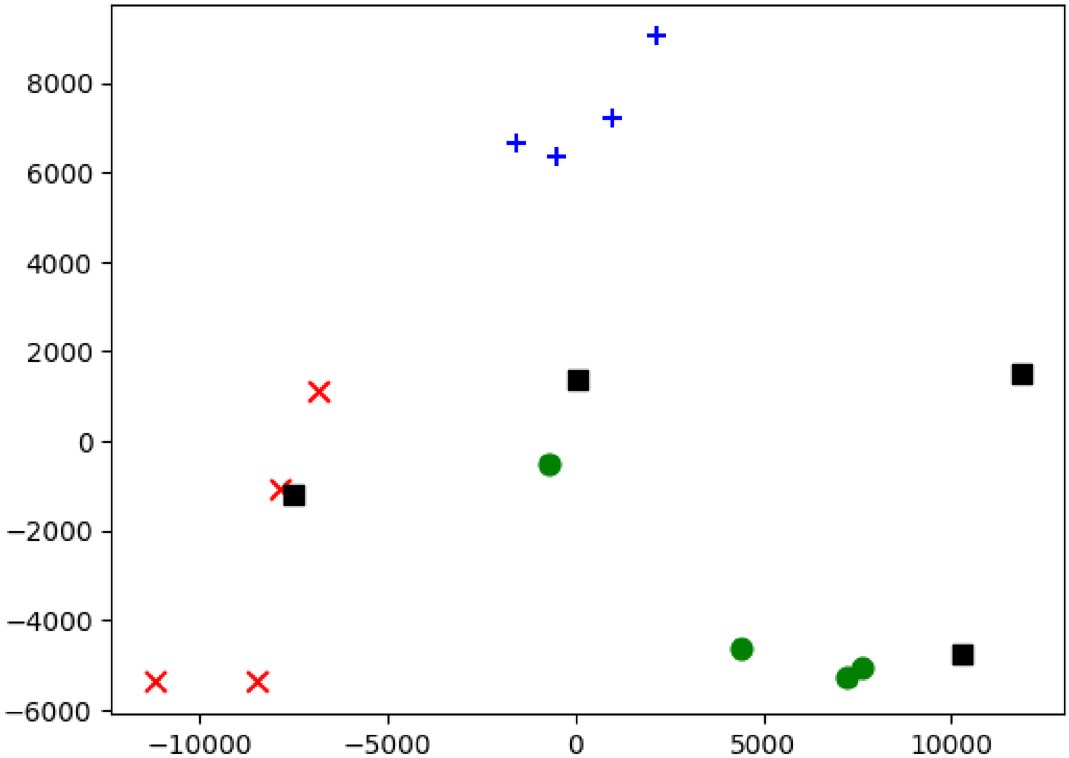
	pc1	pc2	food
1	-1592.892224	6653.034404	burger
2	-512.890107	6336.346855	burger
3	963.208524	7208.106298	burger
4	2164.964831	9034.639168	burger
5	-7842.487559	-1065.634625	drink
6	-8458.916770	-5386.376355	drink
7	-11181.805844	-5359.747024	drink
8	-6830.922505	1133.160740	drink
9	7639.860691	-5060.236026	pasta
10	-704.410521	-529.931496	pasta
11	7237.683422	-5284.931669	pasta
12	4426.728407	-4628.905223	pasta
13	11866.541449	1521.991999	chicken
14	73.990214	1381.115429	chicken
15	-7510.679444	-1191.966044	chicken
16	10262.027436	-4760.666433	chicken

[49]:

```
import matplotlib.pyplot as plt

colors = {'burger':'b', 'drink':'r', 'pasta':'g', 'chicken':'k'}
markerTypes = {'burger':'+', 'drink':'x', 'pasta':'o', 'chicken':'s'}

for foodType in markerTypes:
    d = projected[projected['food']==foodType]
    plt.
 ↪scatter(d['pc1'],d['pc2'],c=colors[foodType],s=60,marker=markerTypes[foodType])
```


8000
6000
4000
2000
0
−2000
−4000
−6000
−10000
−5000
0
5000
10000

[ ]: