PythonExercise 2 - Data Preprocessing in Python

Author: Sanjay Bhargav Siddi

This document contains Python examples for data preprocessing. Data preprocessing consists of a broad set of techniques for cleaning, selecting, and transforming data to improve data mining analysis.

1. Data Quality Issues

1.1. Loading the Wisconsin's breast cancer dataset

In [1]:
```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wis
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
                'Normal Nucleoli', 'Mitoses','Class']

data = data.drop(['Sample code'],axis=1)
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
data.head()
```

```
Number of instances = 699
Number of attributes = 10
```

Out[1]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2

1.2. Missing Values

In [2]:
```
import numpy as np

# convert the missing values to NaNs
data = data.replace('?',np.NaN)

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

# count the number of missing values in each column of the data
print('Number of missing values:')
for col in data.columns:
    print('\t%s: %d' % (col,data[col].isna().sum()))
```

```
Number of instances = 699
Number of attributes = 10
Number of missing values:
        Clump Thickness: 0
        Uniformity of Cell Size: 0
        Uniformity of Cell Shape: 0
        Marginal Adhesion: 0
        Single Epithelial Cell Size: 0
        Bare Nuclei: 16
        Bland Chromatin: 0
        Normal Nucleoli: 0
        Mitoses: 0
        Class: 0
```

In [3]:
```
print('Number of rows in original data = %d' % (data.shape[0]))

# discarding rows with missing values
data2 = data.dropna()
print('Number of rows after discarding missing values = %d' % (data2.shape[0]))
```

```
Number of rows in original data = 699
Number of rows after discarding missing values = 683
```

1.3. Outliers

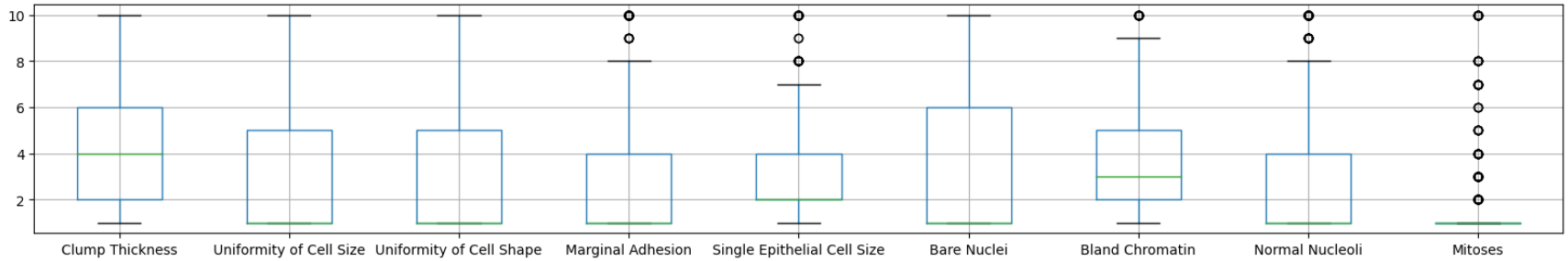
In [4]:
```
%matplotlib inline

# convert the column into numeric values
data2 = data.drop(['Class'],axis=1)
```

```
data2['Bare Nuclei'] = pd.to_numeric(data2['Bare Nuclei'])
data2.boxplot(figsize=(20,3))
```

Out[4]:

```
<Axes: >
```



In [5]:

```
# standardizing the columns of the data
Z = (data2-data2.mean())/data2.std()
Z[20:25]
```

Out[5]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
20	0.917080	-0.044070	-0.406284	2.519152	0.805662	1.771569	0.640688	0.371049	1.405526
21	1.982519	0.611354	0.603167	0.067638	1.257272	0.948266	1.460910	2.335921	-0.343666
22	-0.503505	-0.699494	-0.742767	-0.632794	-0.549168	-0.698341	-0.589645	-0.611387	-0.343666
23	1.272227	0.283642	0.603167	-0.632794	-0.549168	NaN	1.460910	0.043570	-0.343666
24	-1.213798	-0.699494	-0.742767	-0.632794	-0.549168	-0.698341	-0.179534	-0.611387	-0.343666

In [6]:

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

# discarding columns with Z > 3 or Z <= -3
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 = 699
Number of rows after discarding missing values = 632
```

1.4. Duplicate Data

In [7]:

```
# check for duplicate instances in the breast cancer dataset
dups = data.duplicated()
```

```
print('Number of duplicate rows = %d' % (dups.sum()))
data.loc[[11,28]]
```

```
Number of duplicate rows = 236
```

Out[7]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
11	2	1	1	1	2	1	2	1	1	2
28	2	1	1	1	2	1	2	1	1	2

In [8]:

```
print('Number of rows before discarding duplicates = %d' % (data.shape[0]))

# discarding duplicates
data2 = data.drop_duplicates()
print('Number of rows after discarding duplicates = %d' % (data2.shape[0]))
```

```
Number of rows before discarding duplicates = 699
Number of rows after discarding duplicates = 463
```

2. Aggregation

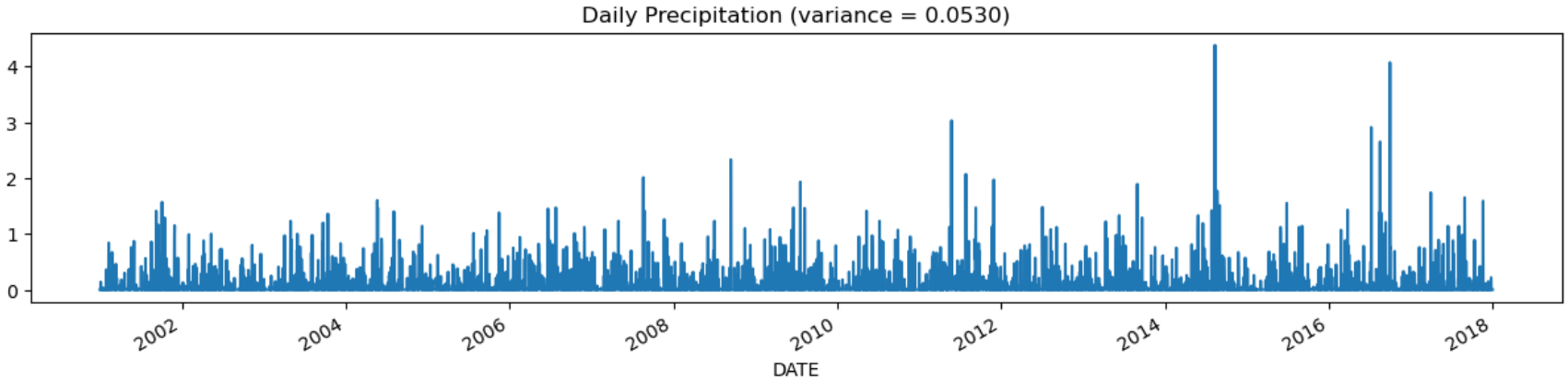
2.1. Loading the precipitation time series data

In [9]:

```
daily = pd.read_csv('C:/Users/sanja/OneDrive/Desktop/University of Arizona Classes/INFO 523 - Data Mining/HW/r-python-e
daily.index = pd.to_datetime(daily['DATE'])
daily = daily['PRCP']
ax = daily.plot(kind='line',figsize=(15,3))
ax.set_title('Daily Precipitation (variance = %.4f)' % (daily.var()))
```

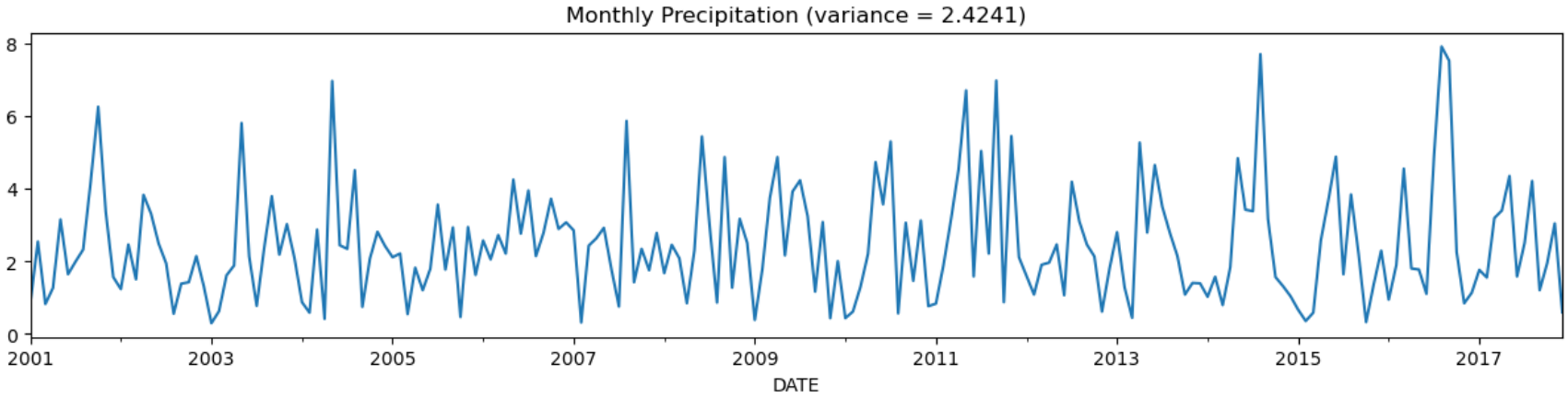
Out[9]:

```
Text(0.5, 1.0, 'Daily Precipitation (variance = 0.0530)')
```



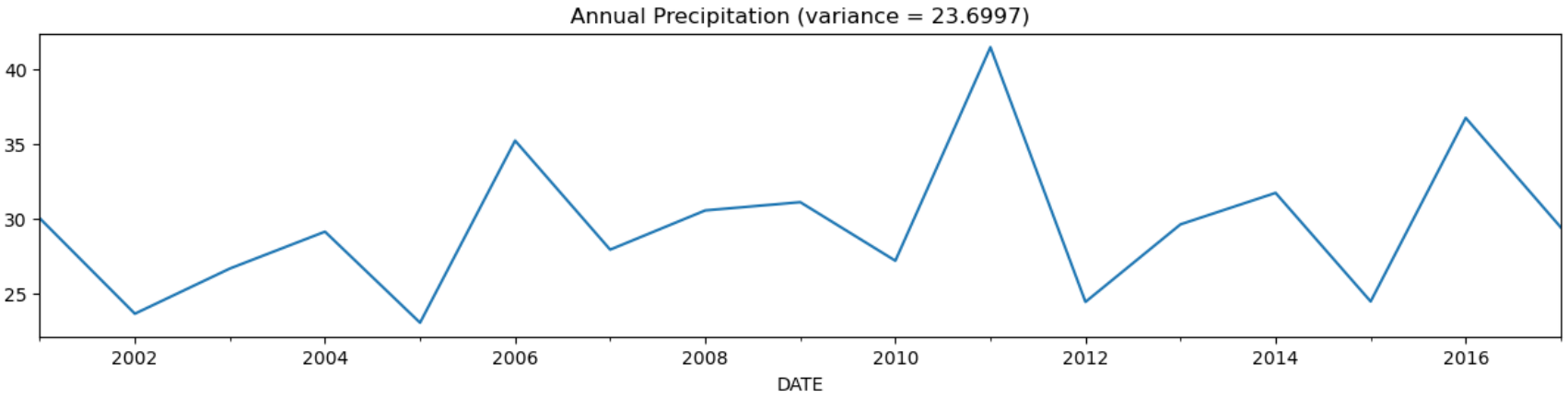
In [10]:
```
# The time series can be grouped and aggregated by month to obtain the total monthly precipitation values.
monthly = daily.groupby(pd.Grouper(freq='M')).sum()
ax = monthly.plot(kind='line',figsize=(15,3))
ax.set_title('Monthly Precipitation (variance = %.4f)' % (monthly.var()))
```

Out[10]:
```
Text(0.5, 1.0, 'Monthly Precipitation (variance = 2.4241)')
```



In [11]:
```
# The daily precipitation time series are grouped and aggregated by year to obtain the annual precipitation values
annual = daily.groupby(pd.Grouper(freq='Y')).sum()
ax = annual.plot(kind='line',figsize=(15,3))
ax.set_title('Annual Precipitation (variance = %.4f)' % (annual.var()))
```

Out[11]:
```
Text(0.5, 1.0, 'Annual Precipitation (variance = 23.6997)')
```



3. Sampling

3.1. Loading the Wisconsin's breast cancer data

In [12]:
```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wis
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
                'Normal Nucleoli', 'Mitoses','Class']
data.head()
```

Out[12]:

	Sample code	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

In [13]:
```
# sample of size 3 is randomly selected (without replacement) from the original data
sample = data.sample(n=3)
sample
```

Out[13]:

	Sample code	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
491	1119189	5	8	9	4	3	10	7	1	1	4
589	1272166	5	1	1	1	2	1	1	1	1	2
687	566346	3	1	1	1	2	1	2	3	1	2

In [14]:
```
# randomly select 1% of the data (without replacement) and display the selected samples
sample = data.sample(frac=0.01, random_state=1)
sample
```

Out[14]:

	Sample code	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
584	1217717	5	1	1	6	3	1	1	1	1	2
417	1239967	1	1	1	1	2	1	2	1	1	2
606	353098	4	1	1	2	2	1	1	1	1	2
349	832567	4	2	3	5	3	8	7	6	1	4
134	1180831	3	1	1	1	3	1	2	1	1	2
502	1253917	4	1	1	2	2	1	2	1	1	2
117	1173509	4	5	5	10	4	10	7	5	8	4

In [15]:
```
# Perform a sampling with replacement to create a sample whose size is equal to 1% of the entire data
sample = data.sample(frac=0.01, replace=True, random_state=1)
sample
```

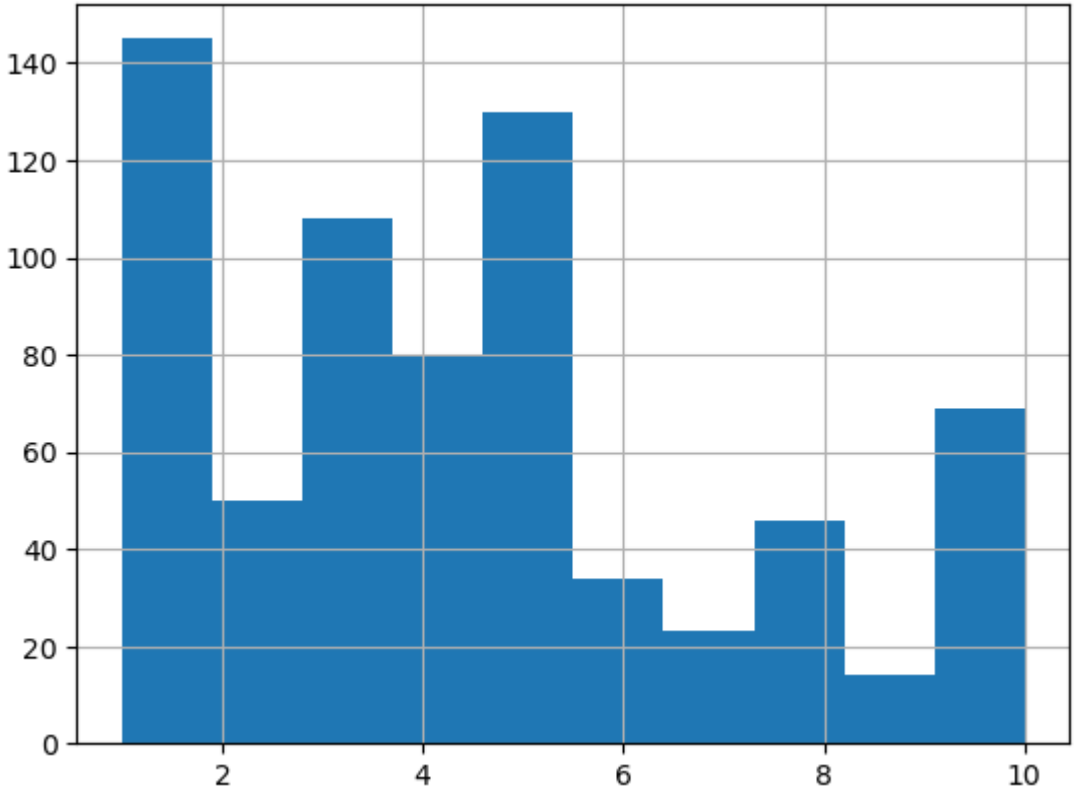
Out[15]:

	Sample code	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
37	1081791	6	2	1	1	1	1	7	1	1	2
235	1241232	3	1	4	1	2	?	3	1	1	2
72	1124651	1	3	3	2	2	1	7	2	1	2
645	1303489	3	1	1	1	2	1	2	1	1	2
144	1184241	2	1	1	1	2	1	2	1	1	2
129	1177512	1	1	1	1	10	1	1	1	1	2
583	1115762	3	1	1	1	2	1	1	1	1	2

4. Discretization

In [16]:
```
# we plot a histogram that shows the distribution of the attribute values
data['Clump Thickness'].hist(bins=10)
data['Clump Thickness'].value_counts(sort=False)
```

Out[16]:
```
5     130
3     108
6      34
4      80
8      46
1     145
2      50
7      23
10     69
9      14
Name: Clump Thickness, dtype: int64
```



In [17]:
```
# For the equal width method, we can apply the cut() function to discretize the attribute into 4 bins of similar interv
bins = pd.cut(data['Clump Thickness'],4)
bins.value_counts(sort=False)
```

Out[17]:
```
(0.991, 3.25]    303
(3.25, 5.5]      210
(5.5, 7.75]       57
(7.75, 10.0]     129
Name: Clump Thickness, dtype: int64
```

In [18]:
```
# For the equal frequency method, the qcut() function can be used to partition the values into 4 bins such that each bi
bins = pd.qcut(data['Clump Thickness'],4)
bins.value_counts(sort=False)
```

Out[18]:
```
(0.999, 2.0]    195
(2.0, 4.0]      188
(4.0, 6.0]      164
(6.0, 10.0]     152
Name: Clump Thickness, dtype: int64
```

5. Principal Component Analysis

In [19]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

numImages = 16
fig = plt.figure(figsize=(7,7))
imgData = np.zeros(shape=(numImages,36963))

# application of PCA to an image dataset
for i in range(1,numImages+1):
    filename = 'C:/Users/sanja/OneDrive/Desktop/University of Arizona Classes/INFO 523 - Data Mining/HW/r-python-exerci
    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])
```

In [20]:

```
import pandas as pd
from sklearn.decomposition import PCA

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

# Using PCA, the data matrix is projected to its first two principal components
```

```
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
```

Out[20]:

	pc1	pc2	food
1	-1576.637320	6641.099187	burger
2	-493.869833	6397.494042	burger
3	990.147898	7236.095006	burger
4	2189.875303	9051.220651	burger
5	-7842.906733	-1059.453103	drink
6	-8498.327622	-5437.391925	drink
7	-11181.806255	-5319.789127	drink
8	-6852.147014	1122.388582	drink
9	7635.242947	-5042.729453	pasta
10	-708.109660	-529.242327	pasta
11	7236.120687	-5302.947611	pasta
12	4417.242886	-4660.768445	pasta
13	11864.528182	1472.857851	chicken
14	76.508169	1366.710336	chicken
15	-7505.703721	-1164.681234	chicken
16	10249.842086	-4770.862430	chicken

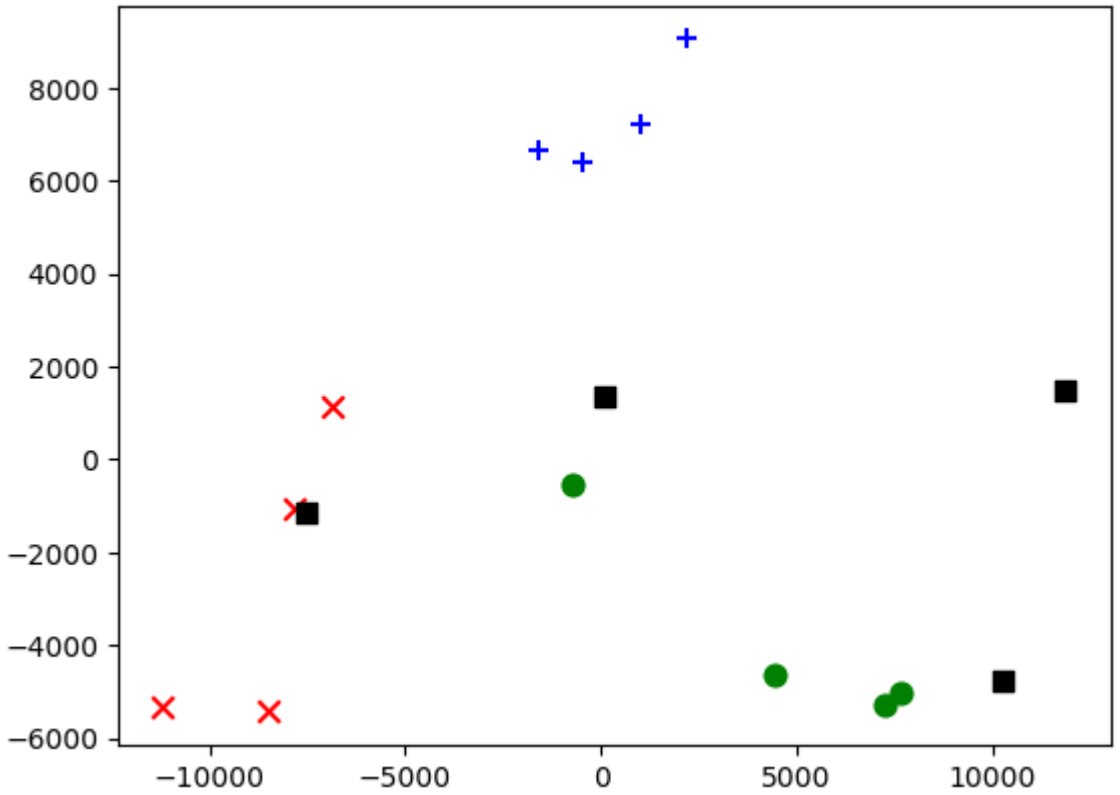
In [21]:

```
import matplotlib.pyplot as plt

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

# scatter plot to display the projected values
for foodType in markerTypes:
```

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



In [ ]: