Reading Files and Categorical Graphs

January 28, 2024

1 Applied Data Science 1

1.0.1 Module Leader: Dr. William Cooper

1.1 Reading Files

There are many ways of reading files in python and many different types of file types you may be familiar with. We will focus in this module on csv (comma separated values) files, as this is one of the most commonly used file types across many fields. Data Handling and Visualisation will present further, more awkward data types. The first thing to do is to make sure your path to the file is always correct. The easiest way of doing this, especially for small data files, is to ensure the file is in the same directory as your python script. Let's look at some different methods of reading a simple csv.

[127]:

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv
```

[128]:

```python
# Manually (inbuilt python methods)
with open('Data/countries_top10.csv', 'r') as f:
    data = []
    for line in f.readlines():
        linelist = line.rstrip('\n').split(',')
        for i, value in enumerate(linelist):
            try:
                value = float(value)
                if value.is_integer():
                    value = int(value)
            except ValueError:
                continue
            linelist[i] = value
        data.append(linelist)
print(data)
print(type(data))
print(type(data[0]))
print(type(data[1][1]))
```

```
[['Country', 'Population', 'Area', 'GDP'], ['Bangladesh', 160996000, 147570,
195100000000], ['Brasil', 207848000, 8547404, 1775000000000], ['China',
1379113000, 9572419, 10866000000000], ['India', 1311051000, 3287263,
2047000000000], ['Indonesia', 257564000, 1912988, 861900000000], ['Mexico',
127017000, 1359162, 1144000000000], ['Nigeria', 182202000, 923768,
481100000000], ['Pakistan', 188925000, 796095, 270000000000], ['Russia',
144097000, 17075400, 1326000000000], ['USA', 321419000, 9809155,
17947000000000]]
<class 'list'>
<class 'list'>
<class 'int'>
```

[129]:

```
# CSV method (standard library)
with open('Data/countries_top10.csv', 'r') as f:
    csvreader = csv.reader(f)
    data = []
    for row in csvreader:
        for i, value in enumerate(row):
            try:
                value = float(value)
                if value.is_integer():
                    value = int(value)
            except ValueError:
                continue
            row[i] = value
        data.append(row)
print(data)
print(type(data))
print(type(data[0]))
print(type(data[1][1]))
```

```
[['Country', 'Population', 'Area', 'GDP'], ['Bangladesh', 160996000, 147570,
195100000000], ['Brasil', 207848000, 8547404, 1775000000000], ['China',
1379113000, 9572419, 10866000000000], ['India', 1311051000, 3287263,
2047000000000], ['Indonesia', 257564000, 1912988, 861900000000], ['Mexico',
127017000, 1359162, 1144000000000], ['Nigeria', 182202000, 923768,
481100000000], ['Pakistan', 188925000, 796095, 270000000000], ['Russia',
144097000, 17075400, 1326000000000], ['USA', 321419000, 9809155,
17947000000000]]
<class 'list'>
<class 'list'>
<class 'int'>
```

[130]:

```
# numpy method(s)
data = np.genfromtxt('Data/countries_top10.csv', dtype=[('Country', 'S10'),␣
  ↪('Population', int), ('Area', int), ('GDP', int)],
                     delimiter=',', skip_header=1)
```

```
# or data = np.loadtxt -- if you know there is no missing data
print(data)
print(type(data))
print(type(data[0]))
print(type(data[1][1]))
```

```
[(b'Bangladesh',  160996000,   147570,   195100000000)
 (b'Brasil',  207848000,  8547404,  1775000000000)
 (b'China', 1379113000,  9572419, 10866000000000)
 (b'India', 1311051000,  3287263,  2047000000000)
 (b'Indonesia',  257564000,  1912988,   861900000000)
 (b'Mexico',  127017000,  1359162,  1144000000000)
 (b'Nigeria',  182202000,   923768,   481100000000)
 (b'Pakistan',  188925000,   796095,   270000000000)
 (b'Russia',  144097000, 17075400,  1326000000000)
 (b'USA',  321419000,  9809155, 17947000000000)]
<class 'numpy.ndarray'>
<class 'numpy.void'>
<class 'numpy.int64'>
```

[131]:

```
# pandas
df = pd.read_csv('Data/countries_top10.csv')
print(df)
print(type(df))
print(type(df.iloc[0]))
print(type(df.iloc[1, 1]))
# or preferably
print(df['Population'][1])
```

```
      Country  Population      Area             GDP
0  Bangladesh   160996000    147570    195100000000
1      Brasil   207848000   8547404   1775000000000
2       China  1379113000   9572419  10866000000000
3       India  1311051000   3287263   2047000000000
4   Indonesia   257564000   1912988    861900000000
5      Mexico   127017000   1359162   1144000000000
6     Nigeria   182202000    923768    481100000000
7    Pakistan   188925000    796095    270000000000
8      Russia   144097000  17075400   1326000000000
9         USA   321419000   9809155  17947000000000
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
<class 'numpy.int64'>
207848000
```

So let's use the pandas method. What about making more data? Let's also reindex so the country is the index.

[132]:

```
df = pd.read_csv('Data/countries_top10.csv', index_col='Country')
df['GDP/head'] = df['GDP'] / df['Population']
df['Pop/km2'] = df['Population'] / df['Area']
df.head()
```

[132]:

```
            Population     Area            GDP     GDP/head      Pop/km2
Country
Bangladesh   160996000   147570   195100000000  1211.831350  1090.980552
Brasil       207848000  8547404  1775000000000  8539.894538    24.317091
China       1379113000  9572419 10866000000000  7878.977285   144.071525
India       1311051000  3287263  2047000000000  1561.342770   398.827535
Indonesia    257564000  1912988   861900000000  3346.352751   134.639632
```

Note the use of df.head. One caution is when in a jupyter notebook, be aware if you're modifying code and have already modified the dataframe, certain commands when re-run may break, e.g. if I had used set_index instead.

2 Question 1

2.1 Part 1

Load the file, GDP_2015dollars.csv, set the year as the index and plot the time series of the 4 countries GDP change, with a log scale (plt.yscale('log')). Note after setting the year as the index, you can access that data as df_gdp.index. Another hint, you can use a for loop across df_gdp.columns to save repeating similar lines of code.

[ ]:

```
df_gdp = pd.read_csv('', index_col='')
df_gdp.head()
```

[ ]:

```
def plot_yearly_gdp(df_gdp):
    """
    Plots the time series data for 4 countries GDP on a log scale
    """
    return
```

[ ]:

```
plot_yearly_gdp(df_gdp)
```

2.2 Part 2

Do the same as above but scale each GDP relative to the United States, as a percentage. This will be on a linear scale.

[ ]:

```
df_gdp =

for country in df_gdp.columns:
    df_gdp[country + '_rel'] =

df_gdp.head()
```

[ ]:

```
def plot_yearly_gdp_relative(df_gdp):
    """
    Plots the time series data for 3 countries GDP relative to the US, on a␣
 ↪linear scale
    """
    return
```

[ ]:

```
plot_yearly_gdp_relative(df_gdp)
```

2.3 End Question 1

Let's now look at some categorical data, continuing to work with pandas dataframes. # Categorical Data

[139]:

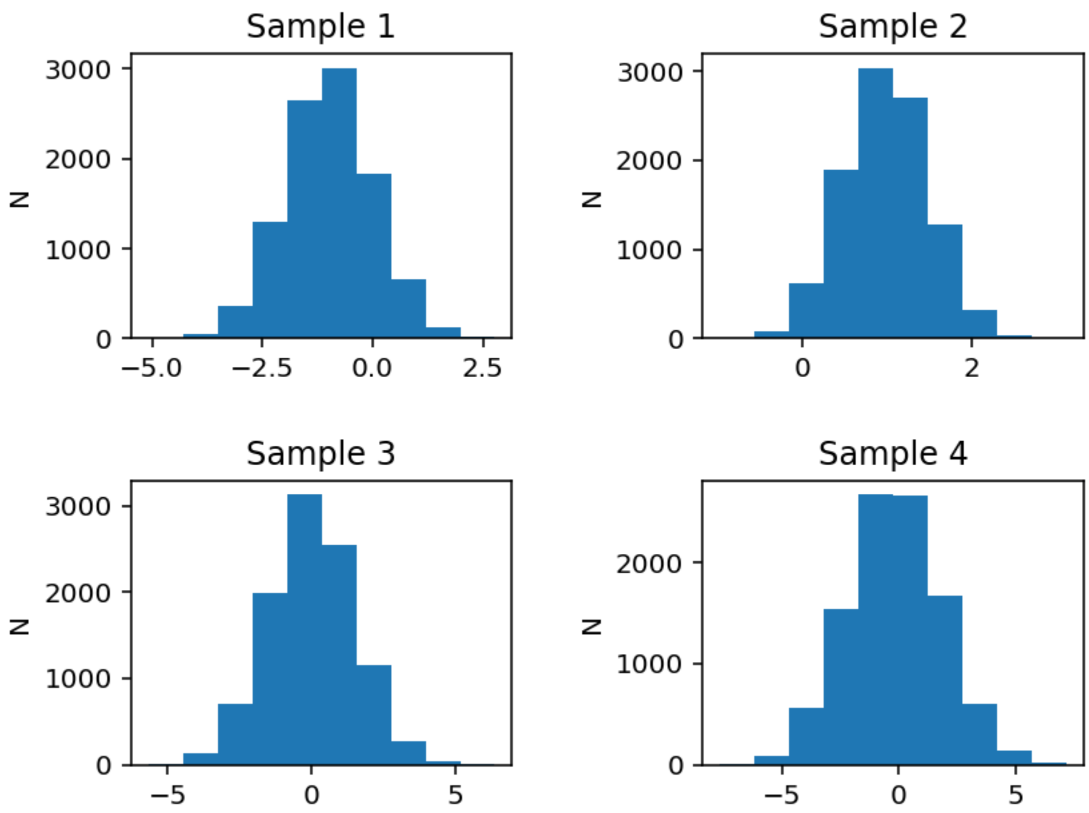
```
sample1 = np.random.normal(-1.0, 1.0, 10000)
sample2 = np.random.normal(1.0, 0.5, 10000)
sample3 = np.random.normal(0.0, 1.5, 10000)
sample4 = np.random.normal(-0.2, 2.0, 10000)
```

[140]:

```
def plot_subplotted_histograms(sample1, sample2, sample3, sample4):
    """
    Plots 4 histograms as subplots
    """
    fig, axs = plt.subplots(2, 2, dpi=144)
    axs = axs.flatten()

    axs[0].hist(sample1)
    axs[1].hist(sample2)
    axs[2].hist(sample3)
    axs[3].hist(sample4)

    for i, ax in enumerate(axs):
        ax.set_ylabel('N')
        ax.set_title('Sample ' + str(i + 1))

    fig.subplots_adjust(wspace=0.5, hspace=0.5)
    plt.show()
    return
```

[141]:

```
plot_subplotted_histograms(sample1, sample2, sample3, sample4)
```

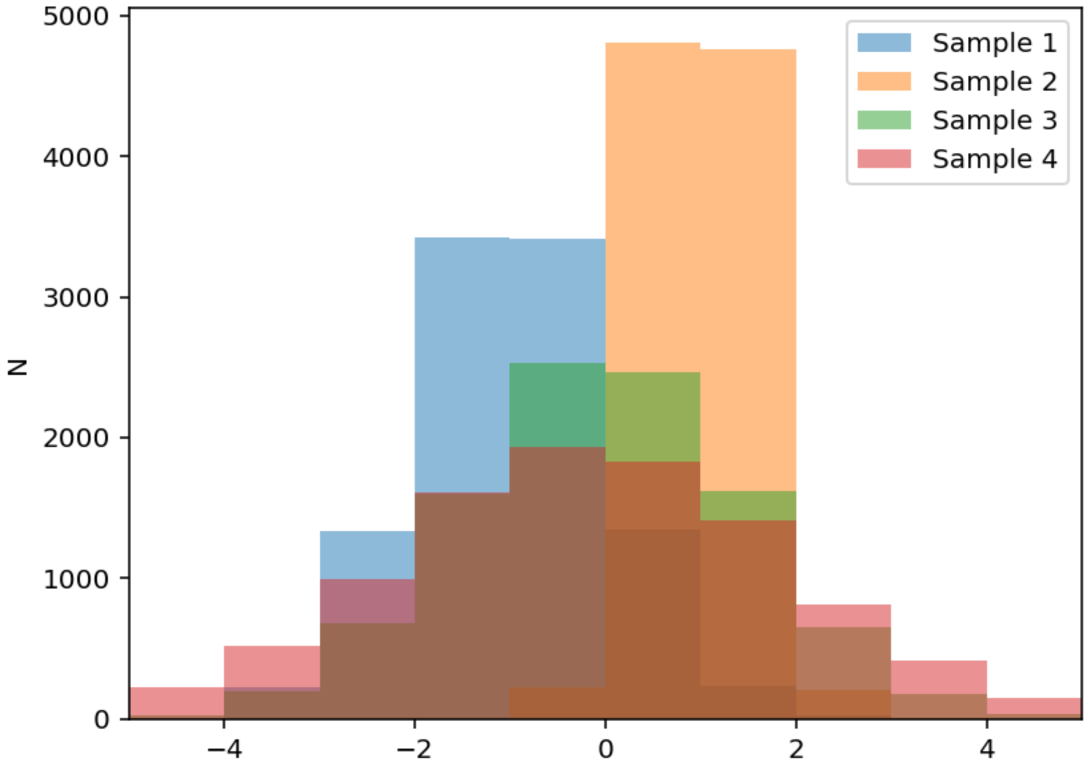


[142]:

```
def plot_overplotted_histograms(*samples):
    """
    Plots 4 histograms on top of each other
    """
    plt.figure(dpi=144)

    for i, sample in enumerate(samples):
        plt.hist(sample, label='Sample ' + str(i + 1), range=(-5, 5), bins=10,␣
 ↪alpha=0.5)

    plt.ylabel('N')
    plt.xlim(-5, 5)
    plt.legend()
    plt.show()
    return
```

[143]:

```
plot_overplotted_histograms(sample1, sample2, sample3, sample4)
```



Experiment with other histogram keywords in addition to the ones above, like density and cumulative. A histogram is just a special form of bar chart, where the values are on the x axis and a count in on the y axis. Feel free to further experiment with bar chart examples given online. # Question 2 Perform the same example histograms as above, so a subplots and overplotted (as a cumulative density) version, with the financial annual returns provided.

```
[ ]: df_tesco = pd.read_csv('', index_col='year')
     df_bp =
     df_barclays =
     df_vodaphone =
     print(df_tesco.head())
     print(df_bp.head())
     print(df_barclays.head())
     print(df_vodaphone.head())
```

```
[ ]: def plot_subplotted_annual_returns():
         """
         Plots 4 histograms as subplots
         """
         return
```

```
[ ]: plot_subplotted_annual_returns()
[ ]: def plot_overplotted_annual_returns():
         """
         Plots 4 histograms on top of each other
         """
         return
```

```
[ ]: plot_overplotted_annual_returns()
```

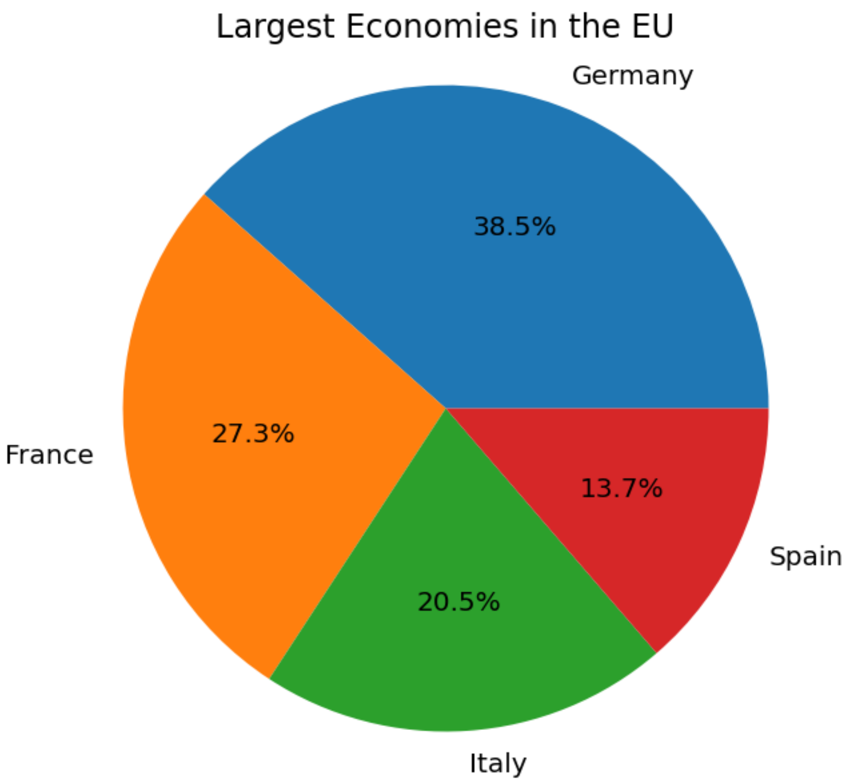
2.4 End Question 2

Let’s finish with some pie charts. For example, the largest countries in the EU, by GDP.

```
[149]: gdp = np.array([3132.670e9, 2225.260e9, 1672.438e9, 1113.851e9])
       countries = ["Germany", "France", "Italy", "Spain"]
```

```
[150]: def plot_gdp_pie(gdp, labels=countries):
           """
           Creates a pie chart of the GDPs of the 4 largest countries, by GDP
           """
           plt.figure(dpi=144)

           plt.pie(gdp, labels=labels, autopct='%1.1f%%')
           plt.title('Largest Economies in the EU')
           plt.axis('Equal')
           plt.show()
           return
```

```
[151]: plot_gdp_pie(gdp)
       # but this isn't all of the economies in the EU!
```



[152]:

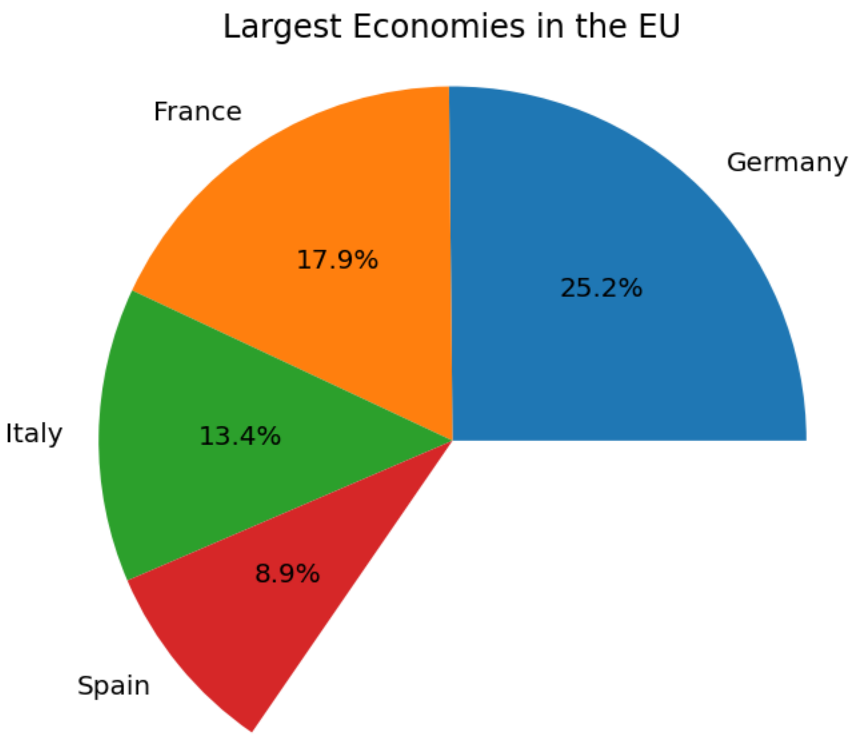
```
gdp = np.array([3132.670e9, 2225.260e9, 1672.438e9, 1113.851e9])
countries = ["Germany", "France", "Italy", "Spain"]

gdp_EU = 12451.987e9
gdp /= gdp_EU
```

[153]:

```
def plot_gdp_pie_unnormalised(gdp, labels=countries):
    """
    Creates a pie chart of the GDPs of the 4 largest countries, by GDP
    """
    plt.figure(dpi=144)

    plt.pie(gdp, labels=labels, autopct='%1.1f%%', normalize=False)
    plt.title('Largest Economies in the EU')
    plt.axis('Equal')
    plt.show()
    return
```

[154]:

```
plot_gdp_pie_unnormalised(gdp)
```



3 Question 3

Create these same style of pie charts, using the known market caps of the previously used companies, and relate it to the entire FTSE cap.

Tesco: 20,979

BP: 68,785

Barclays: 33,367

Vodaphone: 29,741

FTSE: 1,814,000

[ ]:

```
cap = np.array([])
ftse =
norm_cap =
```

[ ]:

```
def plot_market_cap_pie(cap):
    """
    Creates a pie chart of the market cap of 4 major companies
    """
    return
```

```
[ ]: plot_market_cap_pie()
```

```
[ ]: def plot_market_cap_pie_unnormalised(norm_cap):
         """
         Creates a pie chart of the market cap of 4 major companies,  normalised to␣
      ↪the FTSE
         """
         return
```

```
[ ]: plot_market_cap_pie_unnormalised()
```

3.1 End Question 3