3030&7130ICT
Data Analytics

Lab 02 – Data Exploration and Visualisation with Python

Trimester 1 - 2023

Table of Contents

I. Matplotlib 4
1. Importing 4
2. Basic example 4
3. Basic Matplotlib Commands 4
4. Creating Multiplots on Same Canvas 5
5. Object Oriented Method 5
6. Suplot() 6
7. Figure size, aspect ratio and DPI 8
8. Saving figures 8
9. Figure titles 8
10. Axis labels 9
11. Legends 9
12. Plot range 9

II. Seaborn 10
1. Load testing dataset 10
2. Scatter plot 10
3. Categorical functions 11

III. Dimensionality Reduction 12
Principal Component Analysis (PCA) 12

IV. Exercises 14
1. Job Market 14
2. Data Correlation 14

I. Matplotlib

Matplotlib is the "grandfather" library of data visualization with Python. It is an excellent 2D and 3D graphics library for generating scientific figures.
Matplotlib allows you to create reproducible figures programmatically. Please explore the official Matplotlib web page: http://matplotlib.org/ for your reference

1. Importing

Import the matplotlib.pyplot module under the name plt (the tidy way):

```
In [1]: import matplotlib.pyplot as plt
```

You'll also need to use this line to see plots in the notebook:

```
In [2]: %matplotlib inline
```

That line is only for jupyter notebooks, if you are using another editor, you'll use: **plt.show()** at the end of all your plotting commands to have the figure pop up in another window.

2. Basic example

Let's walk through a very simple example using two numpy arrays. You can also use lists, but most likely you'll be passing numpy arrays or pandas columns (which essentially also behave like arrays).

```
In [3]: import numpy as np
        x = np.linspace(0, 5, 11)
        y = x ** 2
```

```
In [4]: x
```

```
Out[4]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

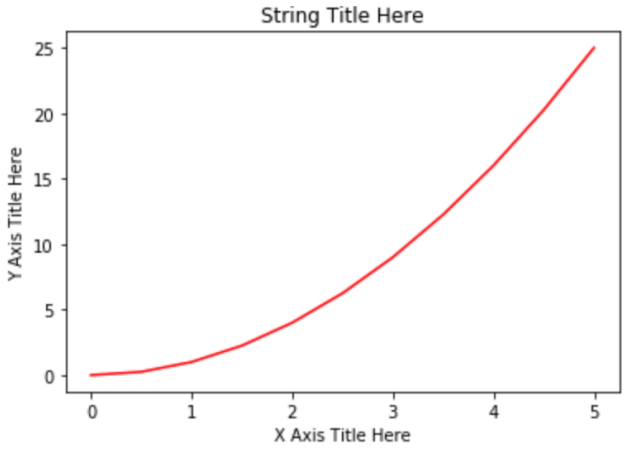
```
In [5]: y
```

```
Out[5]: array([ 0.  ,  0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  , 12.25, 16.  ,
               20.25, 25.  ])
```

3. Basic Matplotlib Commands

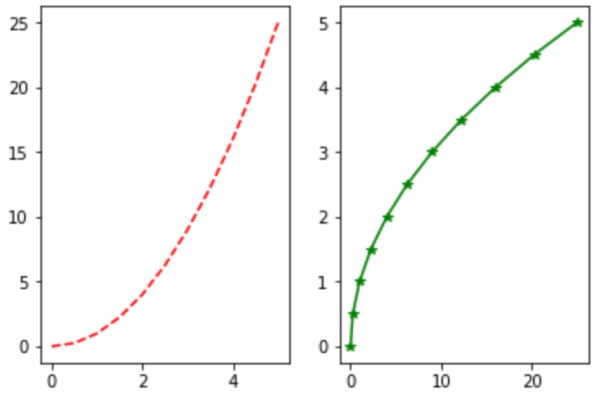
We can create a very simple line plot using the following.

```
In [6]: plt.plot(x, y, 'r') # 'r' is the color red
        plt.xlabel('X Axis Title Here')
        plt.ylabel('Y Axis Title Here')
        plt.title('String Title Here')
        plt.show()
```



4. Creating Multiplots on Same Canvas

In [7]:
```
# plt.subplot(nrows, ncols, plot_number)
plt.subplot(1,2,1)
plt.plot(x, y, 'r--') # More on color options later
plt.subplot(1,2,2)
plt.plot(y, x, 'g*-');
```

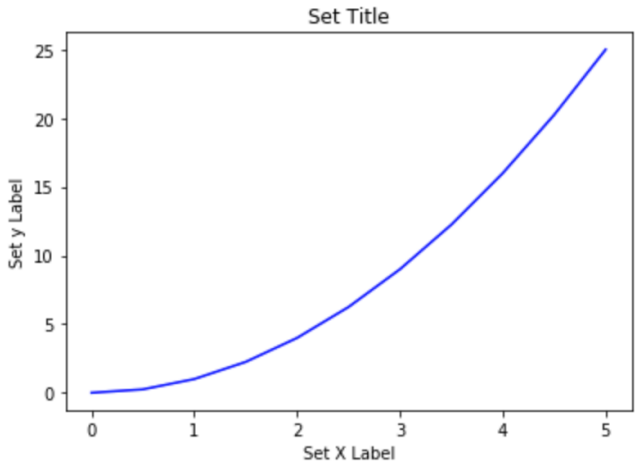


5. Object Oriented Method

The main idea is to create figure objects and then just call methods or attributes off of that object. This approach is nicer when dealing with a canvas that has multiple plots on it.
To begin we create a figure instance. Then we can add axes to that figure:

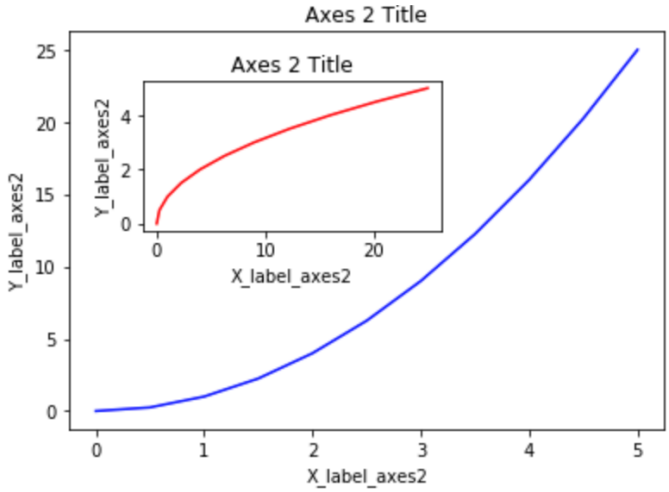
In [8]:
```
# Create Figure (empty canvas)
fig = plt.figure()

# Add set of axes to figure
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (ra

# Plot on that set of axes
axes.plot(x, y, 'b')
axes.set_xlabel('Set X Label') # Notice the use of set_ to begin methods
axes.set_ylabel('Set y Label')
axes.set_title('Set Title')
```

Out[8]:
```
Text(0.5,1,'Set Title')
```



Code is a little more complicated, but the advantage is that we now have full control of where the plot axes are placed, and we can easily add more than one axis to the figure:

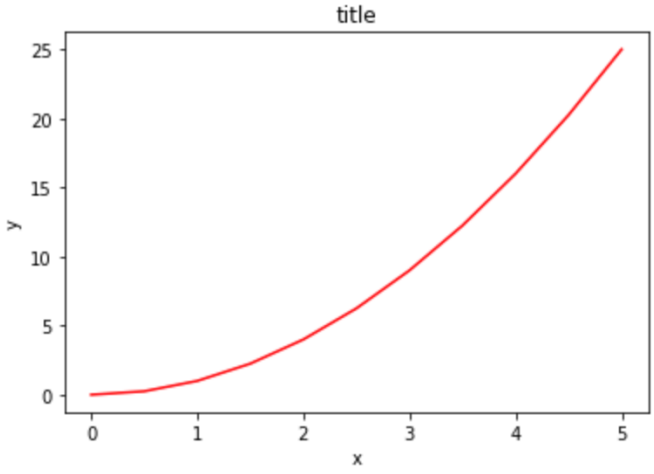
In [9]:
```
# Creates blank canvas
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # inset axes

# Larger Figure Axes 1
axes1.plot(x, y, 'b')
axes1.set_xlabel('X_label_axes2')
axes1.set_ylabel('Y_label_axes2')
axes1.set_title('Axes 2 Title')

# Insert Figure Axes 2
axes2.plot(y, x, 'r')
axes2.set_xlabel('X_label_axes2')
axes2.set_ylabel('Y_label_axes2')
axes2.set_title('Axes 2 Title');
```



6. Suplot()

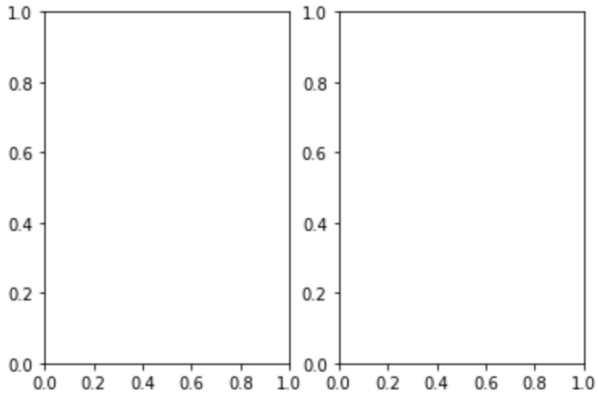
The plt.subplots() object will act as a more automatic axis manager.

In [10]:
```
# Use similar to plt.figure() except use tuple unpacking to grab fig and ax
fig, axes = plt.subplots()

# Now use the axes object to add stuff to plot
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



Then you can specify the number of rows and columns when creating the subplots() object:

In [11]:
```
# Empty canvas of 1 by 2 subplots
fig, axes = plt.subplots(nrows=1, ncols=2)
```



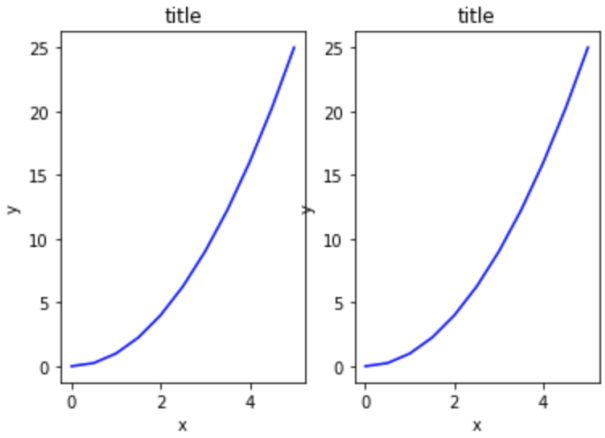
In [12]:
```
# Axes is an array of axes to plot on
axes
```

Out[12]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x10f010a90>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x10f175278>],
      dtype=object)
```

We can iterate through this array:

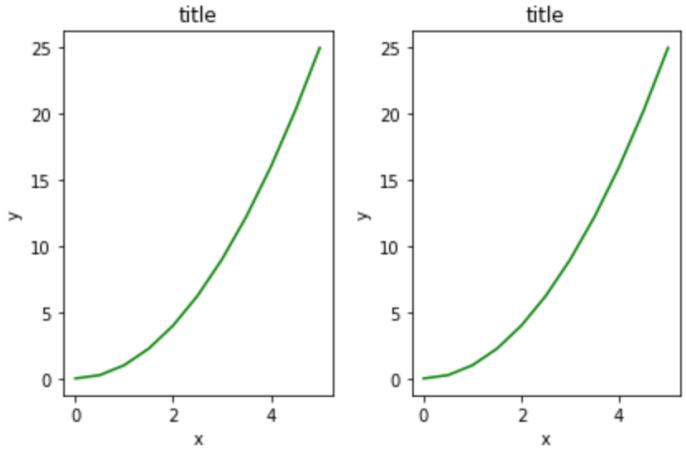
In [13]:
```
for ax in axes:
    ax.plot(x, y, 'b')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

# Display the figure object
fig
```

Out[13]:



A common issue with matplolib is overlapping subplots or figures. We can use **fig.tight_layout()** or **plt.tight_layout()** method, which automatically adjusts the positions of the axes on the figure canvas so that there is no overlapping content:

In [14]:
```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'g')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

fig
plt.tight_layout()
```



7. Figure size, aspect ratio and DPI

Matplotlib allows the aspect ratio, DPI and figure size to be specified when the Figure object is created. You can use the figsize and dpi keyword arguments.

- `figsize` is a tuple of the width and height of the figure in inches
- `dpi` is the dots-per-inch (pixel per inch).

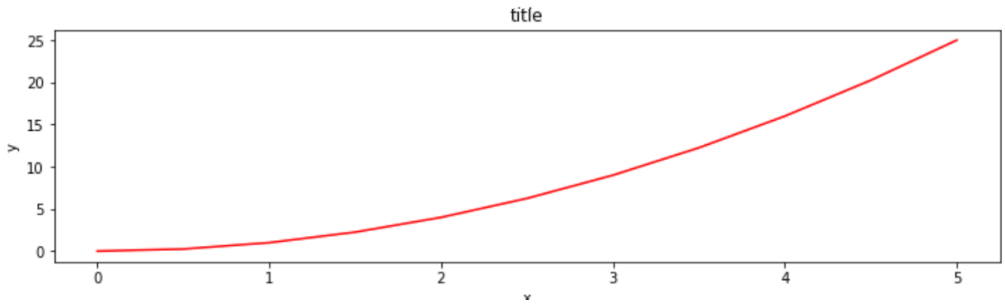
For example:

```
In [15]: fig = plt.figure(figsize=(8,4), dpi=100)
```

```
<matplotlib.figure.Figure at 0x10ed62240>
```

The same arguments can also be passed to layout managers, such as the subplots function:

```
In [16]: fig, axes = plt.subplots(figsize=(12,3))

         axes.plot(x, y, 'r')
         axes.set_xlabel('x')
         axes.set_ylabel('y')
         axes.set_title('title');
```



8. Saving figures

Matplotlib can generate high-quality output in a number formats, including PNG, JPG, EPS, SVG, PGF and PDF.
To save a figure to a file we can use the savefig method in the Figure class:

```
In [17]: fig.savefig("filename.png")
```

Here we can also optionally specify the DPI and choose between different output formats:

```
In [18]: fig.savefig("filename.png", dpi=200)
```

9. Figure titles

A title can be added to each axis instance in a figure. To set the title, use the set_title method in the axes instance:

```
In [19]: ax.set_title("title");
```

10. Axis labels

Similarly, with the methods set_xlabel and set_ylabel, we can set the labels of the X and Y axes:

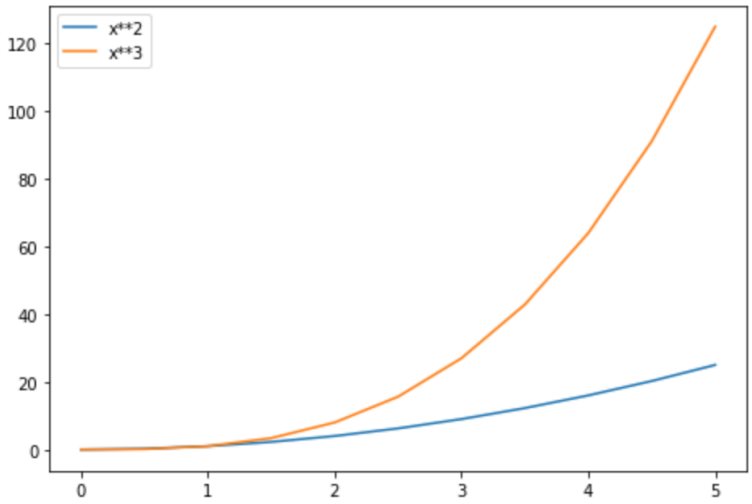
In [20]:
```
ax.set_xlabel("x")
ax.set_ylabel("y");
```

11. Legends

You can use the **label="label text"** keyword argument when plots or other objects are added to the figure, and then using the **legend** method without arguments to add the legend to the figure:

In [21]:
```
fig = plt.figure()

ax = fig.add_axes([0,0,1,1])

ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend()
```

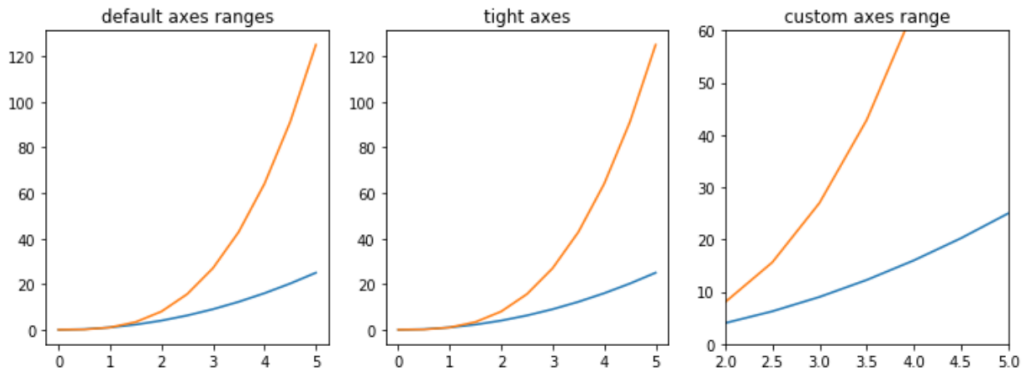
Out[21]:
```
<matplotlib.legend.Legend at 0x10fc97080>
```



12. Plot range

We can configure the ranges of the axes using the set_ylim and set_xlim methods in the axis object, or axis('tight') for automatically getting "tightly fitted" axes ranges:

In [26]:
```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title("tight axes")

axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim([0, 60])
axes[2].set_xlim([2, 5])
axes[2].set_title("custom axes range");
```



II. Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Full reference can be found here: https://seaborn.pydata.org/

1. Load testing dataset

Seaborn makes a small list of datasets easily available:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline

sns.get_dataset_names()
```

Out[1]:
```
['anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'tips',
 'titanic']
```

This makes it easier to learn how to use the library using standard data. The first time you load a dataset you'll need an internet connection so that the data can be downloaded from github.
The **load_dataset** function returns a pandas data frame.

In [3]:
```
tips = sns.load_dataset("tips")
tips.head()
```

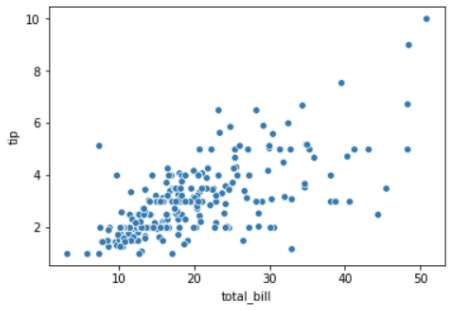
Out[3]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

2. Scatter plot

Seaborn is designed to facilitate the use of data frames. Making an attractive scatter plot is as simple as:

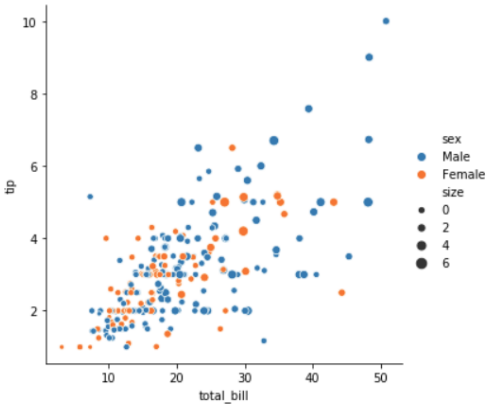
In [4]:
```
ax = sns.scatterplot(x="total_bill", y="tip", data=tips)
```



Making more complex figures is also very simple.

In [5]:
```
sns.relplot(x="total_bill", y="tip", data=tips, kind="scatter",
            hue="sex", size="size",
            )
```

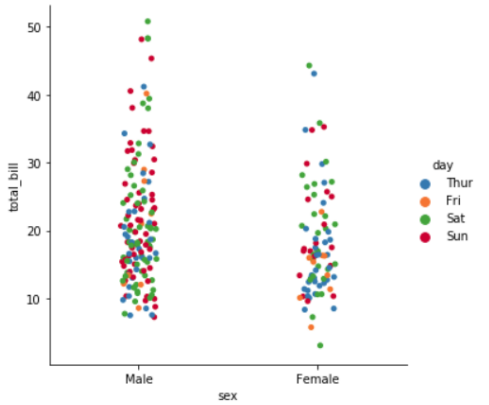
Out[5]: <seaborn.axisgrid.FacetGrid at 0x1f6b9a01048>



3. Categorical functions

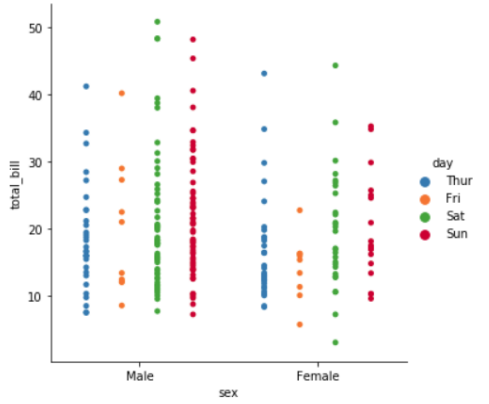
In [6]:
```
sns.catplot(x="sex", y="total_bill", hue="day", data=tips, kind="strip")
```

Out[6]: <seaborn.axisgrid.FacetGrid at 0x1f6b9a4c828>



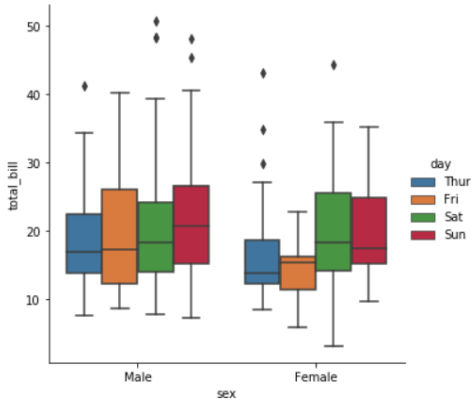
In [7]:
```
sns.catplot(x="sex", y="total_bill", hue="day", data=tips, kind="strip",
            jitter=False, dodge=True)
```

Out[7]: <seaborn.axisgrid.FacetGrid at 0x1f6a4c4b438>



In [8]:
```
sns.catplot(x="sex", y="total_bill", hue="day", data=tips, kind="box")
```

Out[8]: <seaborn.axisgrid.FacetGrid at 0x1f6b9a55828>



III. Dimensionality Reduction

Principal Component Analysis (PCA)

Apply PCA to reduce the dimension of Iris dataset (the dataset has 150 samples individual flowers and three distinct Iris species that each flower is classified into).

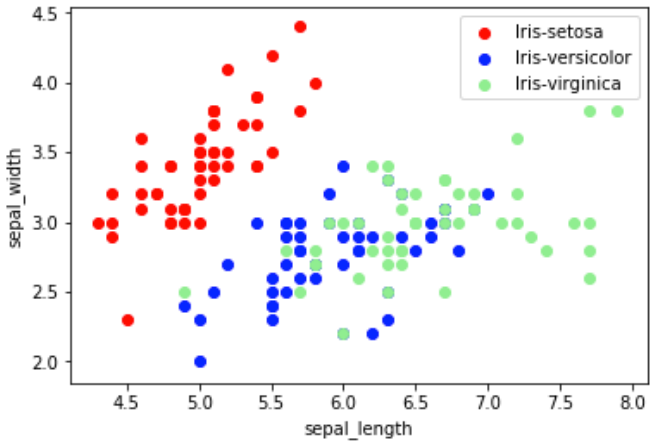
- ❖ Load the dataset

In [68]:
```
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.decomposition import PCA as sklearnPCA
```

In [69]:
```
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
data = pd.read_csv(url,header=None)

y = data[4] # Split off classifications
X = data.iloc[:,0:4] # Split off features
```

- ❖ Visualize the dataset into scatter series.

In [75]:
```
# three different scatter series so the class labels in the legend are distinct
plt.scatter(X[y=='Iris-setosa'].iloc[:,0], X[y=='Iris-setosa'].iloc[:,1], label='Iris-setosa', c='red')
plt.scatter(X[y=='Iris-versicolor'].iloc[:,0], X[y=='Iris-versicolor'].iloc[:,1], label='Iris-versicolor', c='blue')
plt.scatter(X[y=='Iris-virginica'].iloc[:,0], X[y=='Iris-virginica'].iloc[:,1], label='Iris-virginica', c='lightgreen')

# Prettify the graph
plt.legend()
plt.xlabel('sepal_length')
plt.ylabel('sepal_width')

# display
plt.show()
```

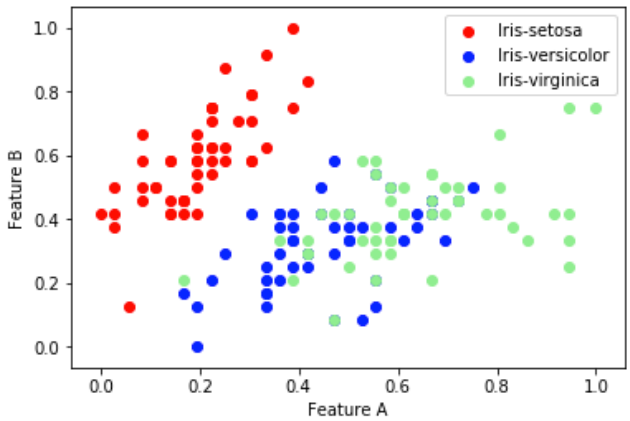


❖ Rescale the data to a [0,1] range.

```
In [71]: X_norm = (X - X.min())/(X.max() - X.min())
```

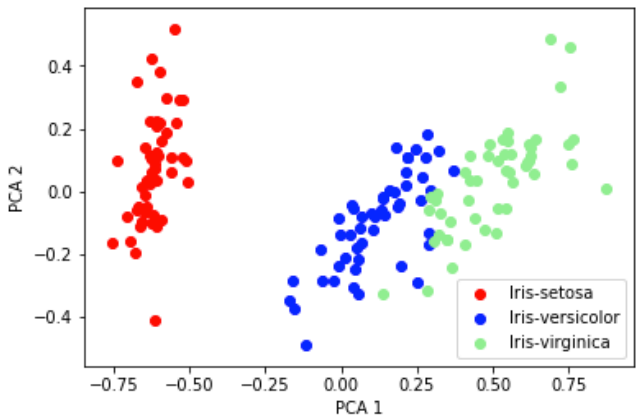
❖ Plot the X_norm again:

```
In [72]: # three different scatter series so the class labels in the legend are distinct
plt.scatter(X_norm[y=='Iris-setosa'].iloc[:,0], X_norm[y=='Iris-setosa'].iloc[:,1], label='Iris-setosa', c='red')
plt.scatter(X_norm[y=='Iris-versicolor'].iloc[:,0], X_norm[y=='Iris-versicolor'].iloc[:,1], label='Iris-versicolor', c='blue')
plt.scatter(X_norm[y=='Iris-virginica'].iloc[:,0], X_norm[y=='Iris-virginica'].iloc[:,1], label='Iris-virginica', c='lightgreen')

# Prettify the graph
plt.legend()
plt.xlabel('Feature A')
plt.ylabel('Feature B')

# display
plt.show()
```



❖ Use sklearn PCA object to the normalized dataset and visualize it again in the plot.

```
In [73]: pca = sklearnPCA(n_components=2) #2-dimensional PCA
transformed = pd.DataFrame(pca.fit_transform(X_norm))
```

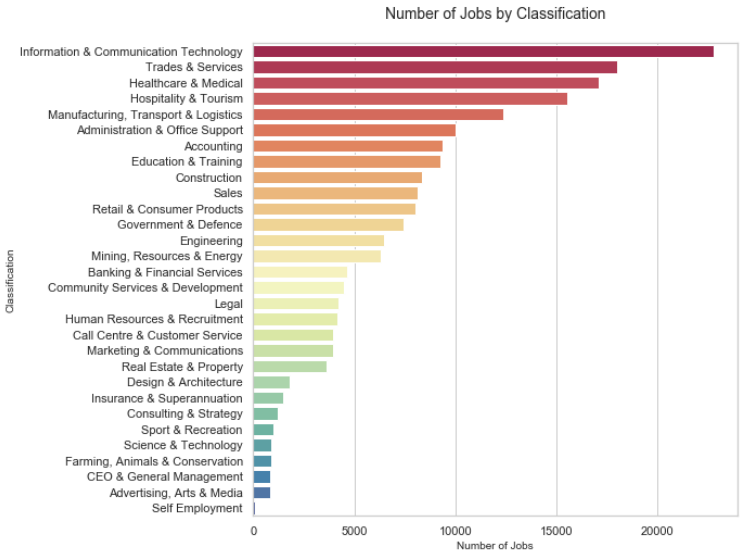


IV. Exercises

1. Job Market

Continuing with the job market data, use Matplotlib/Seaborn to visualize:

- ✓ Job by location. (Below image is just a sample)



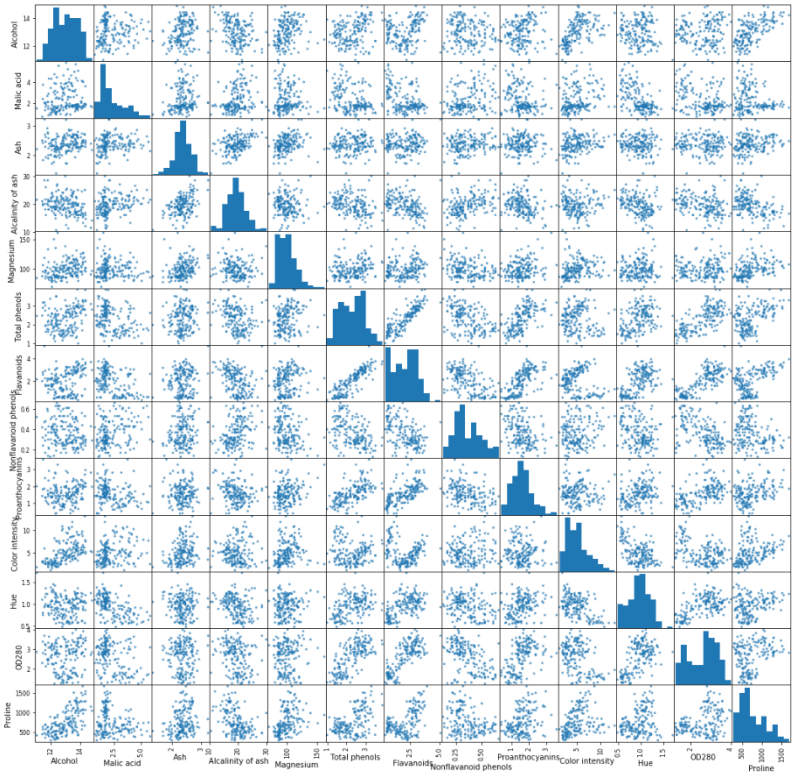
- ✓ Job posts by salary range. (Below image is just a sample)

- ✓ Explore other aspects of the dataset.

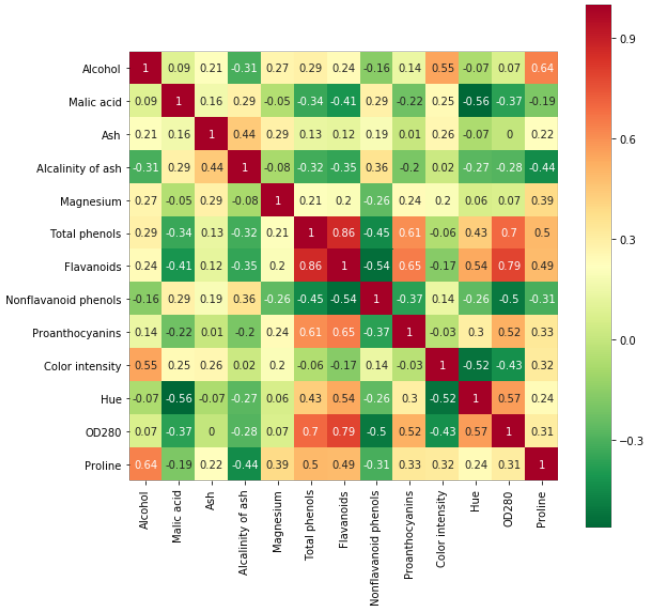
2. Data Correlation

- ❖ Load data from wine.data.csv file. Keep 1st column into a separate variable (label) and remove it from DataFrame.

❖ Use Scatter plot to learn attributes of data. What is your conclusion?



❖ Try to visualize data with correlation heatmap? List three pairs of attributes which have the largest correlation?



❖ Use kMeans with k=3 to cluster the normalized. Use pairplot to visualize the wine attributes with their cluster.

Hint:
kMeansClustering = KMeans(n_clusters = 3, random_state=0)
res = kMeansClustering.fit_predict(wine)

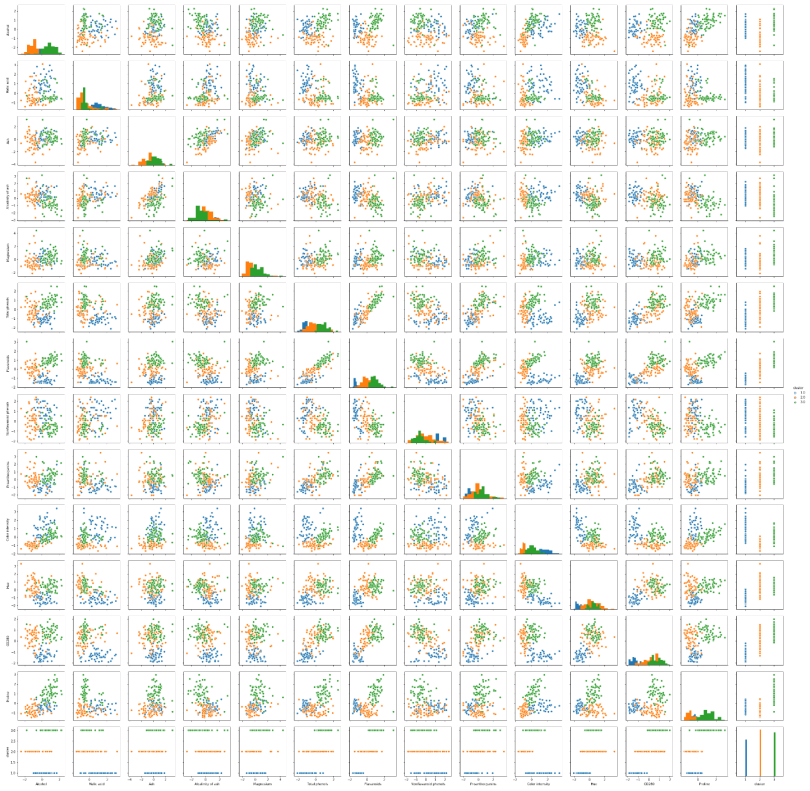
```
wine ["cluster"] = label_pred_KM.astype('float64')
sns_plot = sns.pairplot(wine, hue = "cluster",diag_kind="hist")
```

- Normalize data by removing the mean and scaling to unit variance using `preprocessing.StandardScaler` and do kMeans again.

Hint:

```
standardScaler = preprocessing.StandardScaler()
standardScaler.fit(wine)
X_scaled_array = standardScaler.transform(wine)
normalizedData = pd.DataFrame(X_scaled_array, columns = wine.columns)
```

- (OPTIONAL) By using explained_variance_ratio_ attribute, we know that first 6 PCs explain 85.1% of variance if we reduce the dimension using PCA. You need to apply PCA with 6 components for the above normalized data. Then applying kMeans (3 clusters) to cluster the data after dimensionality reduction.



- (OPTIONAL) Use adjusted_rand_score in sklearn.metrics.cluster to calculate the scores of original kMeans and kMeans after PCA. What is your conclusion?

Hint:

adjusted_rand_score(label, label_pred_KM_PCA) with label is kept in 1st step.