Data Visualization using Python

welcome to DHV LAb

```
import numpy as np
import pandas as pd
```

welcome to DHV Lab

Using Colab

What is Google Colab?

Colab is Google's implementation of Jupyter Notebook. Like all Jupyter Notebooks, a Colab consists of two kinds of components:

- **Text cells**, which contain explanations. You are currently reading a text cell.
- **Code cells**, which contain Python code for you to run. Code cells have a light gray background.

You *read* the text cells and *run* the code cells.

Running code cells

You must run code cells in order. In other words, you may only run a code cell once all the code cells preceding it have already been run.

To run a code cell press Shift + Enter or place the cursor anywhere inside the [ ] area at the top left of a code cell. The area inside the [ ] will display an arrow. Click it to run a cell. You can also run all cells by clicking Runtime -> Run all

If you see errors...

The most common reasons for seeing code cell errors are as follows:

- You didn't run *all* of the code cells preceding the current code cell.

- If the code cell is labeled as a **Task**, then:
    - You haven't yet written the code that implements the task.
    - You did write the code, but the code contained errors.

Basic plotting in matplotlib

```
import numpy as np
import pandas as pd
```

```
!pip install matplotlib==3.4
```

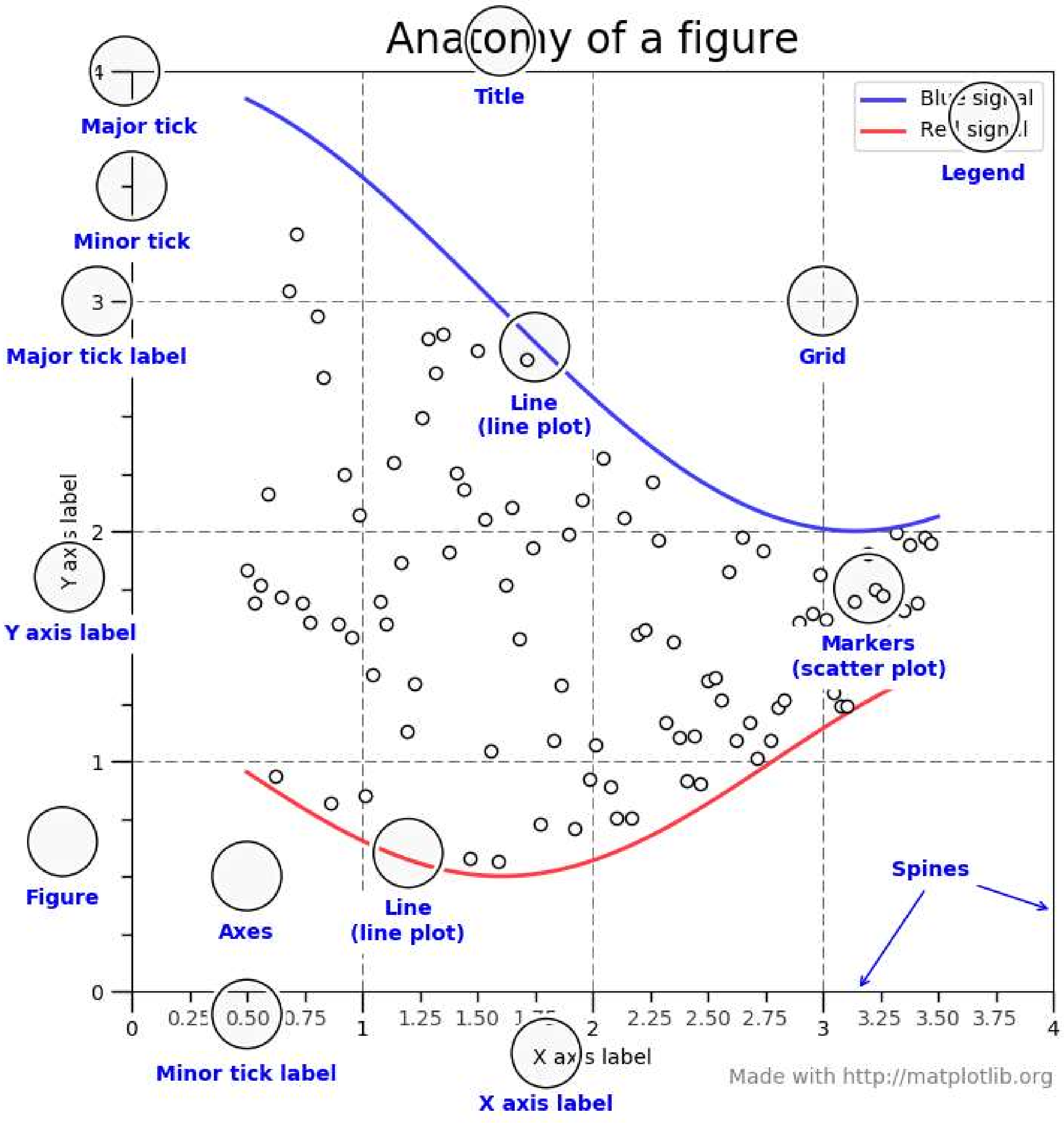
```
600+700
```

```
from matplotlib import pyplot as plt
plt.style.use('seaborn-whitegrid')
import pandas as pd
import numpy as np
```

For all Matplotlib plots, we start by creating a figure and an axes. In their simplest form, a figure and axes can be created as follows:

```
fig = plt.figure()
ax = plt.axes()
# ax.grid()
```

In Matplotlib, the figure (an instance of the class plt.Figure) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels. The axes (an instance of the class plt.Axes) is what we see above: a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization.



Line Plots

We'll commonly use the variable name `fig` to refer to a figure instance, and `ax` to refer to an axes instance or group of axes instances.

Once we have created an axes, we can use the a `x.plot` function to plot some data. Let's start with a simple sinusoid:

```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```

```
# Lets add a title and labels to the plot

fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x))
ax.set_title(' First Simple Plot graph')   # Add a title
ax.set_xlabel('x label values')          # Add x label
ax.set_ylabel('y label values');         # Add y label
```

Multiple lines on same plot

```
# Lets add a title to the plot above
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x))
ax.plot(x, np.cos(x))
#ax.plot(x, np.tan(x))
#ax.plot(x, np.cot(x))
ax.set_title('Multiple Lines  graph');
ax.set_xlabel('x label')
ax.set_ylabel('y label')
plt.show()
```

Adding a legend

```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x), label = 'sin')
ax.plot(x, np.cos(x), label = 'cos')
ax.set_title('Multiple Lines');
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.legend()
# ax.legend(loc=1)
plt.show()
```

Line colors

```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
```

```
ax.plot(x, np.sin(x), label = 'sin', color = 'm')   # specify color by name
ax.plot(x, np.cos(x), label = 'cos', color = 'c')    # short color code (rgbcmyk)
#ax.plot(x, np.tan(x), label = 'tan', color = 'y')
ax.set_title('Multiple Lines');
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.legend();
```

Line Styles

```
fig = plt.figure()
ax = plt.axes()
# ax.grid(linestyle = '--')

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x), label = 'sin', linestyle = 'dashed', color = 'c')
ax.plot(x, np.cos(x), label = 'cos', linestyle = 'dotted', color = 'b')
ax.plot(x, np.sin(x+1), label = 'sin(x+1)', linestyle = 'dashdot', color = 'r')
ax.set_title('Multiple Lines');
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.legend();
```

Axes Limits

```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x), label = 'sin', linestyle = 'dashed', color = 'r')

ax.set_xlim(1, 5)
ax.set_ylim(-1,2 );
ax.set_title('Multiple Lines');
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.legend();
```

Subplots

```
fig, ax = plt.subplots()        # a figure with a single Axes
```

Multiple subplots

```
from matplotlib.pyplot import *
```

```
fig, axs = plt.subplots(2, 2, figsize=(10,10)) # a figure with a 2x2 grid of Axes
x = np.linspace(0, 10, 1000)

axs[0,0].plot(x, np.sin(x), label = 'sin')
#axs[0,0].legend(loc='upper left')
axs[0,1].plot(x, np.cos(x), label = 'cos', color = 'maroon', linestyle = 'dashed')

axs[1,0].plot(x, np.sin(x+2), label = 'sin(x+2)', color = 'blue', linestyle = 'dottec

axs[1,1].plot(x, np.sin(x+4),label = 'sin(x+4)', color = 'green', linestyle = 'dashdc

#ax.set_title('Multiple subplots');
fig.suptitle('Multiple subplots\
from a all TM Fuctions', fontsize=25)
labels = ["sin", "cos", "sin(x+2)", "sin(x+4)" ]
ax.set_xlabel('x label')
ax.set_ylabel('y label')
fig.legend(axs[0,0], axs[0,1], axs[1,0], axs[1,1], labels=labels, loc="upper right")
#ax.legend();
plt.show()
```

Bar Plots

Simple bar chart

```
data = {'Apple': 10, 'Orange': 15, 'Lemon': 5, 'Lime': 20}
names = list(data.keys())
values = list(data.values())

fig = plt.figure()
ax = plt.axes()
# ax.grid(linestyle='--', color='0.85')  # Color: grayscale between 0 and 1

ax.bar(names, values);
```

```
data = {'Apple': 10, 'Orange': 15, 'Lemon': 5, 'Lime': 20, 'Bannan': 30, 'WM': 18, 'mango'
names = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize=(10,7))
ax = plt.axes()

ax.bar(names, values, color='r', width = 0.40);
ax.set_xlabel('Fruit')
ax.set_ylabel('Quantity')
ax.set_title('Fruit Quantities');
```

Horizontal Bar Chart

```
data = {'Apple': 10,
        'Orange': 15,
        'Banana': 13,
        'Pear': 9,
        'Pomegrante': 11,
        'Lemon': 5,
        'Lime': 20}
names = list(data.keys())
values = list(data.values())

ax = plt.axes()

ax.barh(names, values, color='c', height=0.80)            # Horizontal bar-chart
ax.set_xlabel('Quantity')
ax.set_ylabel('Fruit')
ax.set_title('Fruit Quantities');
```

Grouped Bar Chart

```
labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 34, 30, 35, 27]
women_means = [25, 32, 34, 20, 25]

x = np.arange(len(labels))  # the label locations
width = 0.40  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, men_means, width, label='Men')
rects2 = ax.bar(x + width/2, women_means, width, label='Women')

# # Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_xlabel('Groups')
ax.set_title('Scores by group and gender')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend();
plt.show()
```

Adding Bar Labels

```
#fig, ax = plt.subplots()
fig = plt.figure()
ax = plt.axes()
rects1 = ax.bar(x - width/2, men_means, width, label='Men')
rects2 = ax.bar(x + width/2, women_means, width, label='Women')

# # Add some text for labels, title and custom x-axis tick labels, etc.
```

```
ax.set_ylabel('Scores')
ax.set_xlabel('Groups')
ax.set_title('Scores by group and gender')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
# Add bar labels
ax.bar_label(rects1, padding=3) # bar_label exists in matplotlib versions > 3.4
ax.bar_label(rects2, padding=3)
ax.set_ylim(0,40);
```

Stacked bar chart

```
fig, ax = plt.subplots()
ax.grid(linestyle='--', color='0.95', axis = 'x');

ax.set_axisbelow(True)    # Grid behind bars

p1 = ax.bar(labels, men_means, width, label='Men')
p2 = ax.bar(labels, women_means, width, bottom=men_means,
       label='Women')

ax.set_ylabel('Scores')
ax.set_xlabel('Groups')
ax.set_title('Scores by group and gender')
ax.legend()

# Label with label_type 'center' instead of the default 'edge'
ax.bar_label(p1, label_type='center', color='y')
ax.bar_label(p2, label_type='center', color='k')
ax.bar_label(p2)
ax.set_ylim(0,100);
plt.show()
```

Scatter Plots

```
x = np.random.randn(20)
y = np.random.randn(20)
```

```
x
```

```
y
```

```
fig, ax = plt.subplots()
ax.scatter(x, y);
```

Markers

```
fig, ax = plt.subplots()        # a figure with a single Axes
ax.scatter(x, y, marker = "v");
```

```
fig, axs = plt.subplots(2, 3, sharex=True, sharey=True, figsize=(16,12));

# marker symbol
axs[0, 0].scatter(x, y, s=80, marker=">")
axs[0, 0].set_title("Right Triangle ")
# marker from TeX
axs[0, 1].scatter(x, y, s=80, marker=r'$\alpha$')
axs[0, 1].set_title(r"marker=r'\$\alpha\$'")

# marker from path
verts = [[-1, -1], [1, -1], [1, 1], [-1, -1]]
axs[0, 2].scatter(x, y, s=80, marker=verts)
axs[0, 2].set_title("marker=verts")

# regular polygon marker
axs[1, 0].scatter(x, y, s=80, marker=(5, 0))
axs[1, 0].set_title("polygon")

# regular star marker
axs[1, 1].scatter(x, y, s=80, marker=(5, 1))
axs[1, 1].set_title("star")

# regular asterisk marker
axs[1, 2].scatter(x, y, s=80, marker=(5, 2))
axs[1, 2].set_title("asterisk");
```

https://matplotlib.org/stable/api/markers_api.html#module-matplotlib.markers

x

y

Colors

```
plt.style.use('seaborn-darkgrid')

z1 = np.sqrt(x**2 + y**2)

fig, ax = plt.subplots()
pos = ax.scatter(x, y, c=z1, cmap='hot', marker='3')

fig.colorbar(pos);
```

https://matplotlib.org/stable/gallery/color/colormap_reference.html

```
x = np.random.randn(100)
y = np.random.randn(100)

z1 = np.sqrt(x**2 + y**2)
z2 = np.random.randint(100, 200, size=len(x))
```

```
x
```

```
y
```

```
z1
```

```
z2
```

```
fig, ax = plt.subplots()
pos = ax.scatter(x, y, c=z1, s=z2, alpha = 0.55, cmap='viridis')
fig.colorbar(pos);
```

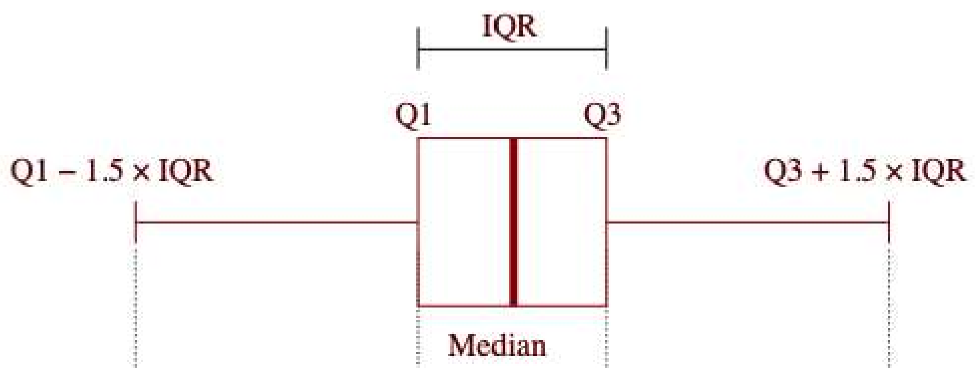
```
x = np.linspace(0, 10, 45)
y = np.cos(x)

plt.plot(x, y, '>', color='r');
```

```
plt.plot(x, y, 'o-', color='maroon');
```

Note: `plt.plot` should be preferred over `plt.scatter` for large datasets.

Box Plots

A box plot displays a summary of a set of data containing the minimum, first quartile, median, third quartile, and maximum. In a box plot, we draw a box from the **first quartile** to the **third quartile**. A vertical line goes through the box at the **median**. The whiskers go from each quartile to the **minimum** or **maximum**.



```
# Create some data

spread = np.random.rand(50) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low))



data


# Basic boxplot

fig, ax = plt.subplots()
ax.boxplot(data)
ax.set_title('Basic Box Plot');


# Basic boxplot

fig, ax = plt.subplots()
ax.boxplot(data,1, '')
ax.set_title('Basic Box Plot');


# Notched boxplot

fig, ax = plt.subplots()
ax.boxplot(data, 1)
ax.set_title('Notched Box Plot');


# Notched boxplot without outliers

fig, ax = plt.subplots()
ax.boxplot(data, 1, '')
ax.set_title('Notched Box Plot without Outliers');


# Notched boxplot with outliers

fig, ax = plt.subplots()
ax.boxplot(data, 0, 'r^')
ax.set_title('Boxplot with different symbol for outliers');


# Horizontal box plot
fig, ax = plt.subplots()
ax.boxplot(data, 1, 'bs', 0)
ax.set_title('Horizontal box plot')


collectn_1 = np.random.normal(100, 10, 200)
```

```
collectn_2 = np.random.normal(80, 30, 200)
collectn_3 = np.random.normal(90, 20, 200)
collectn_4 = np.random.normal(100, 10, 200)
collectn_5 = np.random.normal(50, 50, 200)

## combine these different collections into a list
data_to_plot = [collectn_1, collectn_2, collectn_3, collectn_4, collectn_5]
```

```
data_to_plot
```

```
fig, ax = plt.subplots()
ax.boxplot(data_to_plot);
```

```
Even=[2,4,6,8,10]
Odd=[1,3,5,7,9]
primenum=[2,3,5,7,11,13,17,19,23,31,37]
square=[4,9,16,25,36]
cube=[1,8,27,64]

## combine these different collections into a list
dataEO = [Even, Odd, primenum, square, cube]
```

```
dataEO
```

```
fig, ax = plt.subplots()
ax.boxplot(dataEO);
```

Violin plot

Violin plots are similar to box plots, except that they also show the probability density of the data at different values. These plots include a marker for the median of the data and a box indicating the interquartile range, as in the standard box plots. Overlaid on this box plot is a kernel density estimation. Like box plots, violin plots are used to represent comparison of a variable distribution (or sample distribution) across different "categories".

```
Even=[2,4,6,8,10]
Odd=[1,3,5,7,9]
primenum=[2,3,5,7,11,13,17,19,23,31,37]
square=[4,9,16,25,36]
cube=[1,8,27,64]

## combine these different collections into a list
dataEO = [Even, Odd, primenum, square, cube]
```

```
dataEO



fig, ax = plt.subplots()
ax.violinplot(dataEO)

plt.show()


fig, ax = plt.subplots()
ax.violinplot(dataEO,vert=False);


fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(nrows=1, ncols=5)

# Plot violin plot on axes 1
ax1.violinplot(Even)
ax1.set_title('Even')

# Plot violin plot on axes 2
ax2.violinplot(Odd)
ax2.set_title('Odd')

# Plot violin plot on axes 3
ax3.violinplot( primenum)
ax3.set_title('primenum')

# Plot violin plot on axes 4
ax4.violinplot( square)
ax4.set_title('square')

# Plot violin plot on axes 5
ax5.violinplot( cube)
ax5.set_title('cube')

plt.show()
```

Histograms

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. It is a kind of bar graph.

To construct a histogram, follow these steps –

- Bin the range of values.
- Count how many values fall into each interval.
- Divide the entire range of values into a series of intervals.

Following example plots a histogram of marks obtained by students in a class. Four bins, 0-25, 26-50, 51-75, and 76-100 are defined. The Histogram shows number of students falling in this range.

```
fig,ax = plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100])
ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```

```
fig,ax = plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,20,40,60,80,100])
ax.set_title("histogram of result")
ax.set_xticks([0,20,40,60,80,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```

```
data = np.random.randn(1000)
```

```
data
```

```
plt.hist(data);
```

```
data
```

The hist() function has many options to tune both the calculation and the display.

```
plt.hist(data, bins=90,
         alpha=1,
         histtype='stepfilled',
         color='green',
         edgecolor='red');

         #'bar', 'barstacked', 'step', 'stepfilled'
```

Comparing histograms of several distributions

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
```

```
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)


kwargs = dict(histtype='stepfilled',
              alpha=0.3,
              density=True,
              bins=40)

plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```

Pie Chart

A Pie Chart can only display one series of data. Pie charts show the size of items (called wedge) in one data series, proportional to the sum of the items. The data points in a pie chart are shown as a percentage of the whole pie.

Matplotlib API has a pie() function that generates a pie diagram representing data in an array. The fractional area of each wedge is given by x/sum(x). If sum(x)< 1, then the values of x give the fractional area directly and the array will not be normalized. Theresulting pie will have an empty wedge of size 1 - sum(x).

The pie chart looks best if the figure and axes are square, or the Axes aspect is equal.

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
ax.pie(students, labels = langs,autopct='%1.2f%%')
plt.show()
```

```
labels='Apple','Bannana','Mango','lemen'
Sizes= [6, 10, 15, 9]
fig,ax=plt.subplots()
ax.pie (Sizes, labels=labels, autopct='%.1f %%')
plt.show ()
```

Visualizing some datasets with seaborn

Iris dataset

```
import sklearn
```

```python
import pandas as pd
import numpy as np

from sklearn import datasets

# Load the iris dataset
iris = datasets.load_iris()

# Load it to a pandas dataframe:
iris_df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['label'])

iris_df['label_names'] = iris_df['label'].apply(lambda x: iris.target_names[int(x)])

# Let's have a look at how the data looks like
display(iris_df)

from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from array import array
data = load_iris()
data.target[[10, 25, 50]]
a = np.array([0, 0, 1])  #array([0, 0, 1])
list(data.target_names)
['setosa', 'versicolor', 'virginica']


# Visualize the data using Matplotlib
import matplotlib.pyplot as plt

colours = ['orange', 'blue', 'green']
species = iris.target_names

f = plt.figure(figsize=(15,10))

for i in range(0, 3):
    species_df = iris_df[iris_df['label'] == i]
    plt.scatter(
        species_df['petal length (cm)'],
        species_df['sepal length (cm)'],
        color=colours[i],
        alpha=0.5,
        label=species[i]
    )
plt.xlabel('petal length (cm)')
plt.ylabel('sepal length (cm)')
plt.title('Iris dataset: petal length vs sepal length')
plt.legend(loc='lower right')
plt.show()


# Visualize the data using Seaborn
# Matrix scatter plot.
```

```
import seaborn as sns

sns.set_style("darkgrid")
sns.pairplot(iris_df[iris.feature_names + ['label_names']], hue="label_names",height=3);
```

Cars dataset

```
from google.colab import drive
# drive.mount('/content/drive')

# Loaad the csv file into a pandas dataframe

cars_df = pd.read_csv('/content/drive/My Drive/data/cars.csv', index_col=0)
```

```
# Let's explore some rows.
cars_df.head(20)
```

Let's look at the distribution of cars by the state.

```
fig, ax = plt.subplots(figsize=(16,10))

graph = sns.countplot(x='state',
              data=cars_df,
              ax=ax,
              order=cars_df.state.value_counts().index)
graph.set_xticklabels(graph.get_xticklabels(),rotation=90)

for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")

plt.show()
```

```
fig, ax = plt.subplots(figsize=(10,8))
sns.violinplot(x ='title_status', y='price', data=cars_df)
plt.show()
```

```
cars_df.loc[cars_df.groupby('color')['color'].transform('size') < 40, 'color'] = 'other'
palette = ['white', 'gray', 'darkgray', 'silver', 'indianred',  'khaki', 'royalblue','oran

fig, ax = plt.subplots(figsize=(14,8))
sns.boxplot(x ='price', y='color', data=cars_df, order = cars_df.color.value_counts().inde
plt.show()
```