

Practical Notebook 2

Pandas

In this course, we will use pandas to import the data into DataFrame objects. Pandas is a commonly used library working with and manipulating data in various formats, such as txt, csv, excel format, and more.

You can read more about pandas [here](#), or by searching online.

```
In [ ]: # The first thing we need to do is to import pandas
import pandas as pd

# We will also change how the floating point numbers are displayed
pd.set_option("display.float_format", lambda x: f"{x:.5f}")
```

Creating our own dataset to file

We will start by creating our own data set, but later on we will import the data from a file.

```
In [ ]: names = ['Alice', 'Bob', 'Charlie']
animals = ['Dog', 'Cat', None]
age = [27, 12, 43]
sex = ['Female', 'Male', 'Male']
```

We will then merge the lists together using the *zip* function.

```
In [ ]: people = list(zip(names, animals, age, sex))
print(people)

[('Alice', 'Dog', 27, 'Female'), ('Bob', 'Cat', 12, 'Male'), ('Charlie', None, 43, 'Male')]
```

Now we can make our merged list into a DataFrame object by using pandas.

```
In [ ]: df = pd.DataFrame(data=people, columns=['Names', 'Animals', 'Age', 'Sex'])
print(df)
```

	Names	Animals	Age	Sex
0	Alice	Dog	27	Female
1	Bob	Cat	12	Male
2	Charlie	None	43	Male

You can also export the dataframe to a csv file, where we use the function *to_csv* to export the file. You will find the file you created in the folder you are in. (In colab you will find the folder to the left.) The index parameter is set to *False*, i.e. we won't write

the row names to the new file (in this case the row names are 0, 1, 2). The header parameter is set to *True*, i.e. we will write the column names to the file (in this case the column names are *Names, Animals, Age, Sex*). You can change these parameters yourself to see the difference.

```
In [ ]: df.to_csv('test_people.csv', index=False, header=True)
```

Read a dataset from file

To read the data from a csv file we will use the function *read_csv*.

```
In [ ]: df = pd.read_csv('test_people.csv')
print(df)
```

	Names	Animals	Age	Sex
0	Alice	Dog	27	Female
1	Bob	Cat	12	Male
2	Charlie	NaN	43	Male

We can inspect the numerical values in the data using the function *describe*.

```
In [ ]: print(df.describe())
```

	Age
count	3.00000
mean	27.33333
std	15.50269
min	12.00000
25%	19.50000
50%	27.00000
75%	35.00000
max	43.00000

And look at one specific column by using the names of the header.

```
In [ ]: print(f"Here you will see the names: \n{df['Names']}")
print(f"\nHere you will see the animals: \n{df['Animals']}")
print(f"\nHere you will see the ages: \n{df['Age']}")
print(f"\nHere you will see the sex: \n{df['Sex']}")
```

Here you will see the names:

0 Alice

1 Bob

2 Charlie

Name: Names, dtype: object

Here you will see the animals:

0 Dog

1 Cat

2 NaN

Name: Animals, dtype: object

Here you will see the ages:

0 27

1 12

2 43

Name: Age, dtype: int64

Here you will see the sex:

0 Female

1 Male

2 Male

Name: Sex, dtype: object

You can also divide the groups into females and males.

```
In [ ]: male, female = df['Sex'].value_counts()
print(f"Here we have {male} male(s) and {female} female(s).")
```

Here we have 2 male(s) and 1 female(s).

By looking only at one column, as we did before, we can find some interesting data about it as well.

```
In [ ]: # finding the mean value of the ages (with 2 decimals)
print(f"mean: {df['Age'].mean():.2f}")
# and the standard deviation (with 2 decimals)
print(f"std: {df['Age'].std():.2f}")
```

mean: 27.33

std: 15.50

Titanic

Now we will download and use a larger dataset, to get a better understanding about the pandas library. The dataset contains passenger data from Titanic, and later on we will predict "what sort of people were most likely to survive?". The passenger data has 7 features: Name, Sex, Socio-economic class, Siblings/Spouses Aboard, Parents/Children Aboard and Fare and a binary response variable "survived".

```
In [ ]: # Downloading the titanic dataset
#!wget https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/tita
```

Assignment a)

```
In [ ]: # ASSIGNMENT:
# Load the data and get familiar with it
# Use the .describe() method to inspect numerical values

df = pd.read_csv('titanic.csv')
print(df)
df.describe()
```

	Survived	Pclass	Name \
0	0	3	Mr. Owen Harris Braund
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...
2	1	3	Miss. Laina Heikkinen
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle
4	0	3	Mr. William Henry Allen
..
882	0	2	Rev. Juozas Montvila
883	1	1	Miss. Margaret Edith Graham
884	0	3	Miss. Catherine Helen Johnston
885	1	1	Mr. Karl Howell Behr
886	0	3	Mr. Patrick Dooley

	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard \
0	male	22.00000	1	0
1	female	38.00000	1	0
2	female	26.00000	0	0
3	female	35.00000	1	0
4	male	35.00000	0	0
..
882	male	27.00000	0	0
883	female	19.00000	0	0
884	female	7.00000	1	2
885	male	26.00000	0	0
886	male	32.00000	0	0

	Fare
0	7.25000
1	71.28330
2	7.92500
3	53.10000
4	8.05000
..	...
882	13.00000
883	30.00000
884	23.45000
885	30.00000
886	7.75000

[887 rows x 8 columns]

Out[]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.00000	887.00000	887.00000	887.00000	887.00000	887.00000
mean	0.38557	2.30552	29.47144	0.52537	0.38331	32.30542
std	0.48700	0.83666	14.12191	1.10467	0.80747	49.78204
min	0.00000	1.00000	0.42000	0.00000	0.00000	0.00000
25%	0.00000	2.00000	20.25000	0.00000	0.00000	7.92500
50%	0.00000	3.00000	28.00000	0.00000	0.00000	14.45420
75%	1.00000	3.00000	38.00000	1.00000	0.00000	31.13750
max	1.00000	3.00000	80.00000	8.00000	6.00000	512.32920



Assignment b)

```
In [ ]: # ASSIGNMENT:
# Count the number of males and females
male, female = df['Sex'].value_counts()
print(f"Males: {male}\nFemales: {female}")
```

Males: 573

Females: 314

Assignment c)

```
In [ ]: # ASSIGNMENT:
# Find the mean fare and display with 2 floating point numbers

mean_fare = df['Fare'].mean()
print(f"Fare mean: {mean_fare:.2f}")

# Find the standard deviation of the fare and display with 2 floating point

std_fare = df['Fare'].std()
print(f"Fare std dev: {std_fare:.2f}")
```

Fare mean: 32.31

Fare std dev: 49.78

Assignment d)

```
In [ ]: # ASSIGNMENT:
# Count how many survived (1) and how many died (0)

# YOUR CODE HERE

died, survived = df['Survived'].value_counts()
print(f"Died: {died}\nSurvived: {survived}")
```

Died: 545

Survived: 342

Assignment e)

```
In [ ]: # ASSIGNMENT:
# count and display the number of women who survived
# and the number of men who survived

# YOUR CODE HERE
female_survived, male_survived = df[df["Survived"].isin([1])]["Sex"].value_counts()
print(f"Female survivors: {female_survived}\nMale survivors: {male_survived}")
```

Female survivors: 233

Male survivors: 109

Assignment f)

```
In [ ]: # ASSIGNMENT:
# Separate the dataset from Titanic into X and y,
# where y is the column Survived, and X is the rest.
# Inspect the data. Look at for instance the function "describe" in pandas

# YOUR CODE HERE
X = df.loc[:, 'Pclass':'Fare'] # as Survived is the 1st column, we can just ignore it
y = df['Survived']

x_describe = X.describe()
y_describe = y.describe()

print(x_describe, "\n", y_describe)
```

	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard
\				
count	887.00000	887.00000	887.00000	887.00000
mean	2.30552	29.47144	0.52537	0.38331
std	0.83666	14.12191	1.10467	0.80747
min	1.00000	0.42000	0.00000	0.00000
25%	2.00000	20.25000	0.00000	0.00000
50%	3.00000	28.00000	0.00000	0.00000
75%	3.00000	38.00000	1.00000	0.00000
max	3.00000	80.00000	8.00000	6.00000

Fare

count	887.00000
mean	32.30542
std	49.78204
min	0.00000
25%	7.92500
50%	14.45420
75%	31.13750
max	512.32920

count	887.00000
-------	-----------

mean	0.38557
std	0.48700
min	0.00000
25%	0.00000
50%	0.00000
75%	1.00000
max	1.00000

Name: Survived, dtype: float64

Assignment g)

```
In [ ]: # ASSIGNMENT:
# Standardize the data by subtracting the mean and dividing by the standard
# Inspect the data again to see that the mean is (close to) zero and the sta

# YOUR CODE HERE
X = X.select_dtypes(include='number') # There was an error telling me to only

X_new = (X - X.mean())/X.std()
y_new = (y - y.mean())/y.std()

# Inspecting the data again:
X_new_describe = X_new.describe()
y_new_describe = y_new.describe()

print(X_new_describe, "\n", y_new_describe)
```

	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard
\				
count	887.00000	887.00000	887.00000	887.00000
mean	-0.00000	0.00000	-0.00000	-0.00000
std	1.00000	1.00000	1.00000	1.00000
min	-1.56040	-2.05719	-0.47559	-0.47471
25%	-0.36517	-0.65299	-0.47559	-0.47471
50%	0.83006	-0.10420	-0.47559	-0.47471
75%	0.83006	0.60392	0.42966	-0.47471
max	0.83006	3.57803	6.76640	6.95594

	Fare
count	887.00000
mean	0.00000
std	1.00000
min	-0.64894
25%	-0.48974
50%	-0.35859
75%	-0.02346
max	9.64251

count	887.00000
mean	0.00000
std	1.00000
min	-0.79172
25%	-0.79172
50%	-0.79172
75%	1.26165
max	1.26165

Name: Survived, dtype: float64

Matplotlib

Matplotlib is a commonly used library for visualizing data in Python. Other visualization libraries exist for Python, such as seaborn, plotly, and more. Beyond the first practical notebook, we do not enforce any particular plotting library, but strongly encourage the use of Matplotlib. Below we will use the plotting functions inside of *matplotlib.pyplot*. You can read more about matplotlib [here](#) and pyplot [here](#).

Examples

```
In [ ]: # import the relevant libraries
import matplotlib.pyplot as plt
import numpy as np
```

We will start by looking at some small lists.

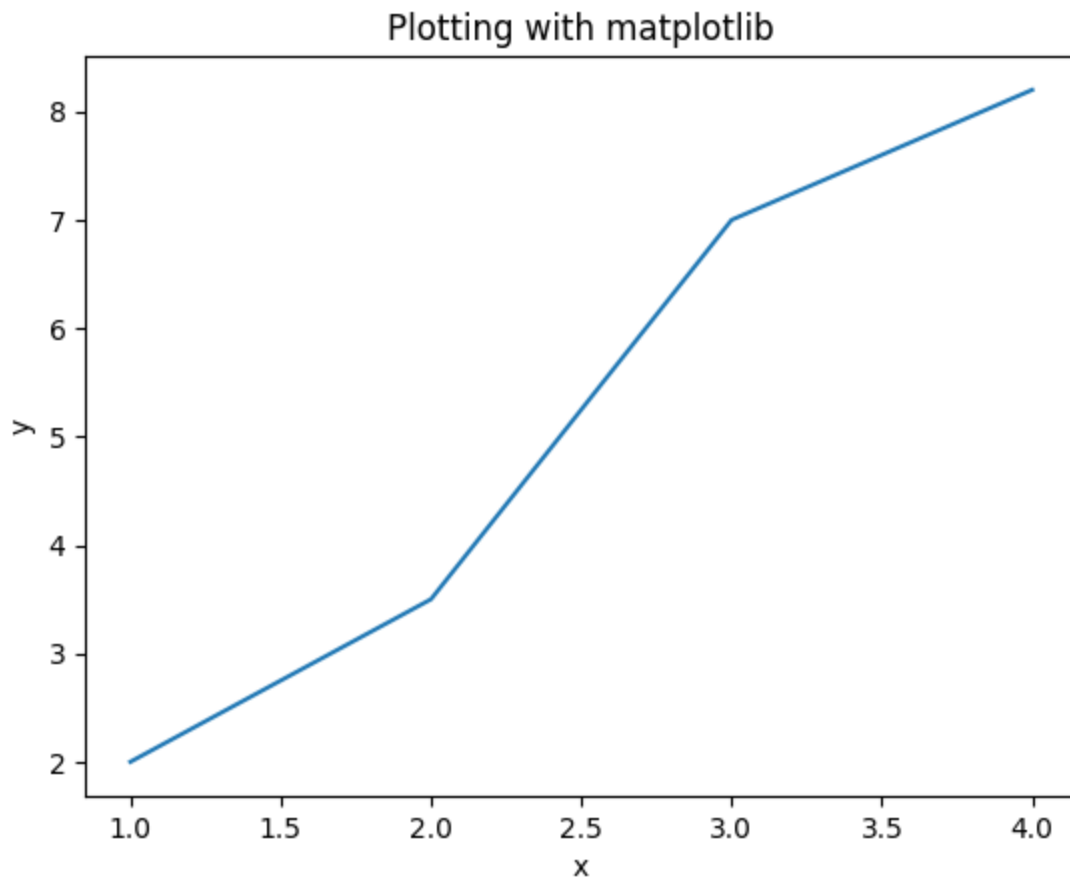
```
In [ ]: # examples of some datapoint
x = [1,2,3,4]
y = [2,3.5,7,8.2]

# plotting the data using matplotlib.pyplot.plot
```



```
plt.plot(x, y)

# It is important to add labels for the axes and a title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plotting with matplotlib")
# and always end with show(), which will show you the plot.
plt.show()
```



Plots can also be below each other, or side by side by using `subplot`.

```
In [ ]: # Vertical subplot

plt.style.use('bmh')

t = np.arange(0.0, 1.0, 0.01)
sin = np.sin(2*np.pi*t)
cos = np.cos(2*np.pi*t)

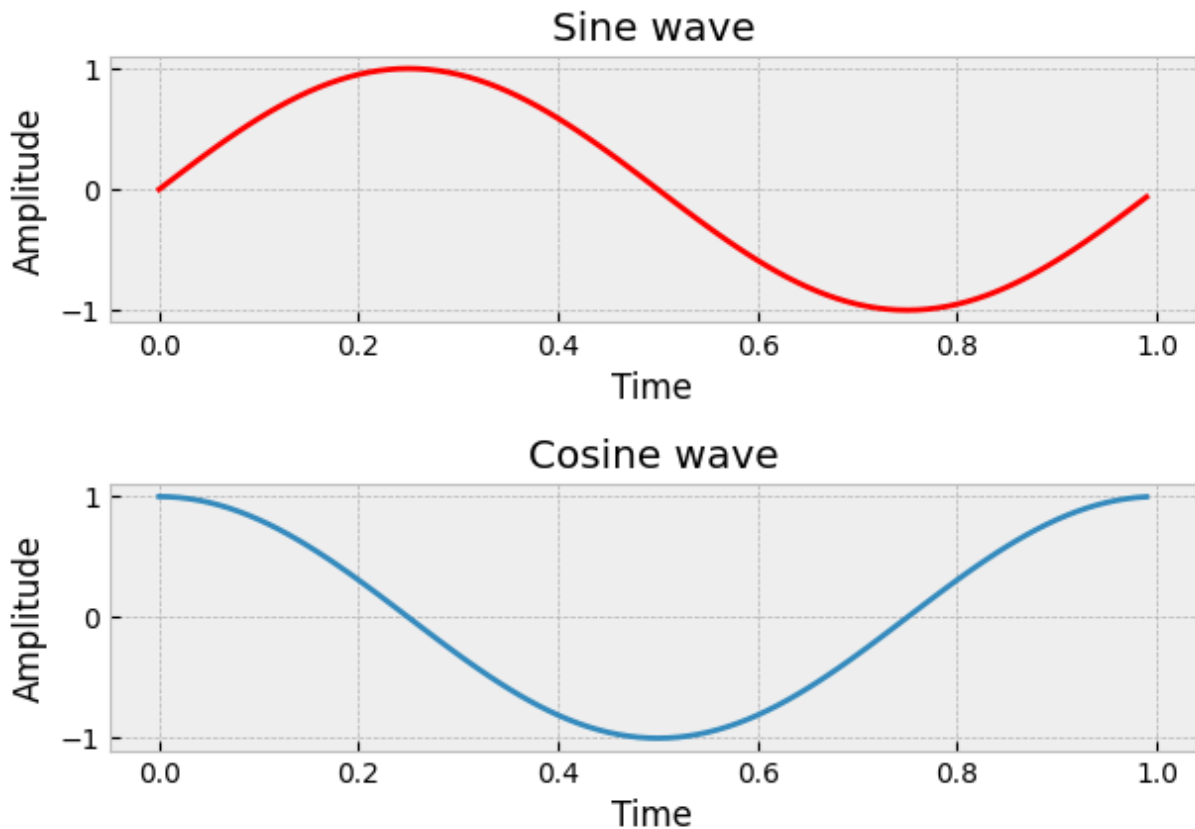
fig = plt.figure()
fig.suptitle("Sine and cosine for different t", fontsize=18)

ax1 = fig.add_subplot(2,1,1)
ax1.plot(t, sin, color='red', lw=2)
ax1.set_ylabel('Amplitude')
ax1.set_xlabel('Time')
ax1.set_title('Sine wave')
```

```
ax2 = fig.add_subplot(2,1,2)
ax2.plot(t, cos)
ax2.set_ylabel('Amplitude')
ax2.set_xlabel('Time')
ax2.set_title('Cosine wave')

fig.tight_layout() # comment out this line to see the difference
fig.subplots_adjust(top=0.85)
plt.show()
```

Sine and cosine for different t



```
In [ ]: # Horizontal subplot

plt.style.use('bmh')

t = np.arange(0.0, 1.0, 0.01)
sin = np.sin(2*np.pi*t)
cos = np.cos(2*np.pi*t)

fig = plt.figure()
fig.suptitle("Sine and cosine for different t", fontsize=18)

ax1 = fig.add_subplot(1,2,1) # we have changed (2,1,1) to (1,2,1)
ax1.plot(t, sin, color='red', lw=2)
ax1.set_ylabel('Amplitude')
ax1.set_xlabel('Time')
ax1.set_title('Sine wave')
```

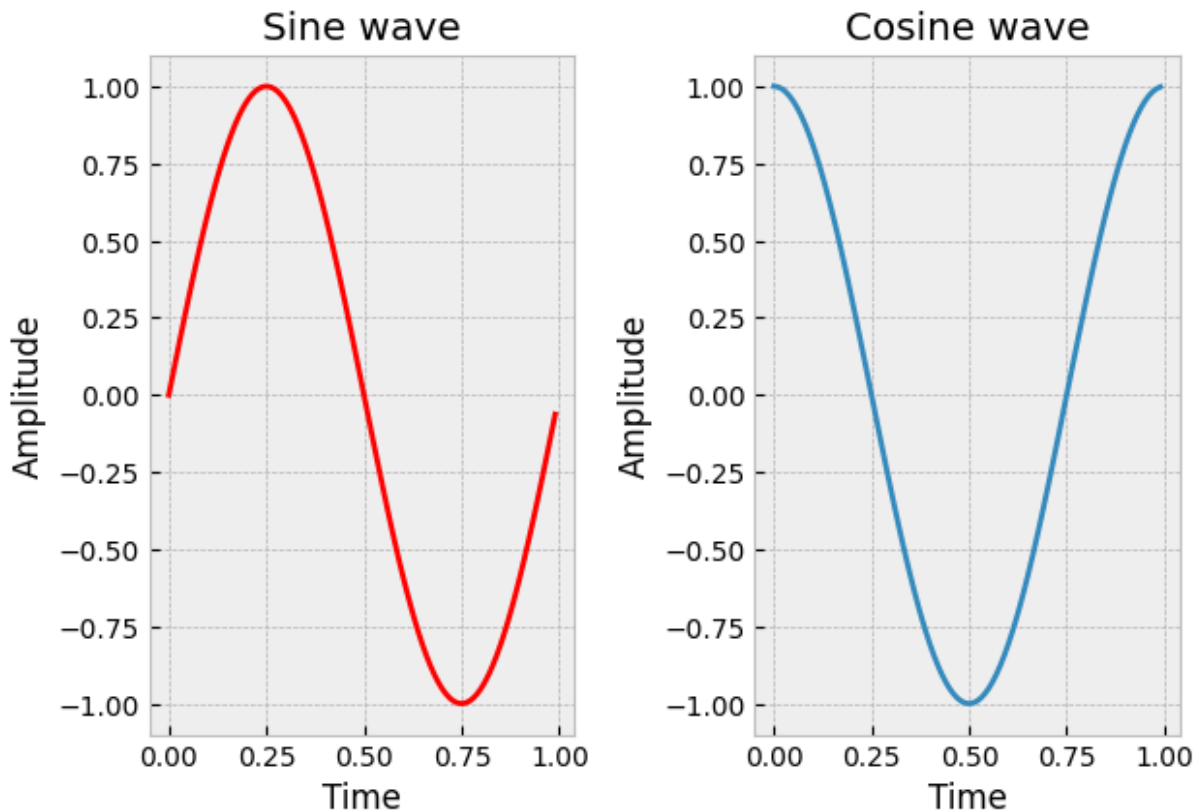
```

ax2 = fig.add_subplot(1,2,2) # we have changed (2,1,2) to (1,2,2)
ax2.plot(t, cos)
ax2.set_ylabel('Amplitude')
ax2.set_xlabel('Time')
ax2.set_title('Cosine wave')

fig.tight_layout() # comment out this line to see the difference
fig.subplots_adjust(top=0.85)
plt.show()

```

Sine and cosine for different t



And with different stylings

```

In [ ]: # Here are all the different "pre-configured" styles matplotlib supports
# https://matplotlib.org/tutorials/intermediate/artists.html#sphx-glr-tutori
plt.style.available

```

```
Out[ ]: ['Solarize_Light2',
         '_classic_test_patch',
         '_mpl-gallery',
         '_mpl-gallery-nogrid',
         'bmh',
         'classic',
         'dark_background',
         'fast',
         'fivethirtyeight',
         'ggplot',
         'grayscale',
         'seaborn-v0_8',
         'seaborn-v0_8-bright',
         'seaborn-v0_8-colorblind',
         'seaborn-v0_8-dark',
         'seaborn-v0_8-dark-palette',
         'seaborn-v0_8-darkgrid',
         'seaborn-v0_8-deep',
         'seaborn-v0_8-muted',
         'seaborn-v0_8-notebook',
         'seaborn-v0_8-paper',
         'seaborn-v0_8-pastel',
         'seaborn-v0_8-poster',
         'seaborn-v0_8-talk',
         'seaborn-v0_8-ticks',
         'seaborn-v0_8-white',
         'seaborn-v0_8-whitegrid',
         'tableau-colorblind10']
```

The plots can also be both below each other and side by side at the same time (as a matrix) as you can see below. Here we have also plotted two graphs together in every figure, and added a color and a label for each one of them.

```
In [ ]: # Matrix subplot

fig = plt.figure()
fig.suptitle("Sine and cosine for different t", fontsize=18)

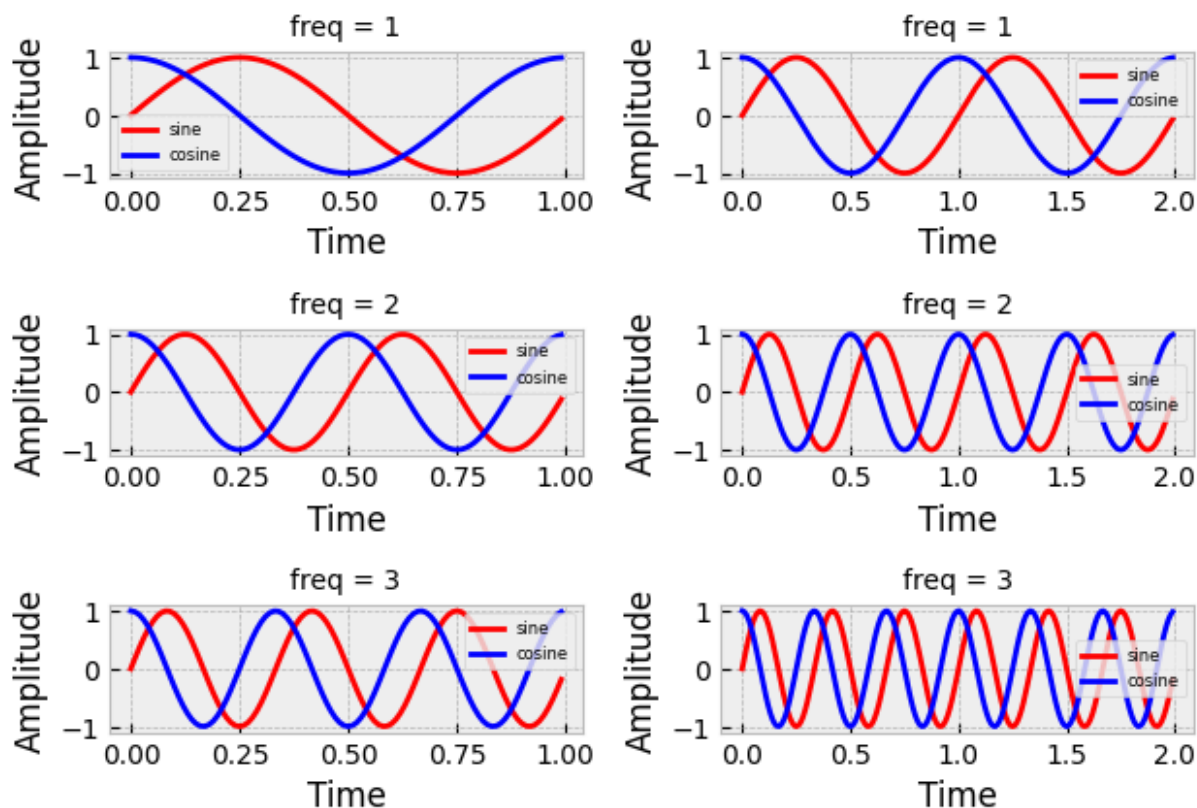
i = 1
for freq in [1, 2, 3]:
    for t_max in [1, 2]:
        t = np.arange(0.0, t_max, 0.01)
        sin = np.sin(2*freq*np.pi*t)
        cos = np.cos(2*freq*np.pi*t)

        ax = fig.add_subplot(3,2,i)
        ax.plot(t, sin, color='red', lw=2, label='sine')
        ax.plot(t, cos, color='blue', lw=2, label='cosine')
        ax.set_ylabel('Amplitude')
        ax.set_xlabel('Time')
        ax.legend(fontsize=6)
        ax.set_title(f'freq = {freq}', fontsize=10)
        i += 1

fig.tight_layout() # comment out this line to see the difference
```

```
fig.subplots_adjust(top=0.85)
plt.show()
```

Sine and cosine for different t



Plotting data from Pandas

Now we will plot some of the datapoints from the titanic dataset to visualize it.

```
In [ ]: # Downloading the titanic dataset
#!wget https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/tita
```

```
In [ ]: # Load the titanic dataset for plotting
import pandas as pd
df = pd.read_csv('titanic.csv')
```

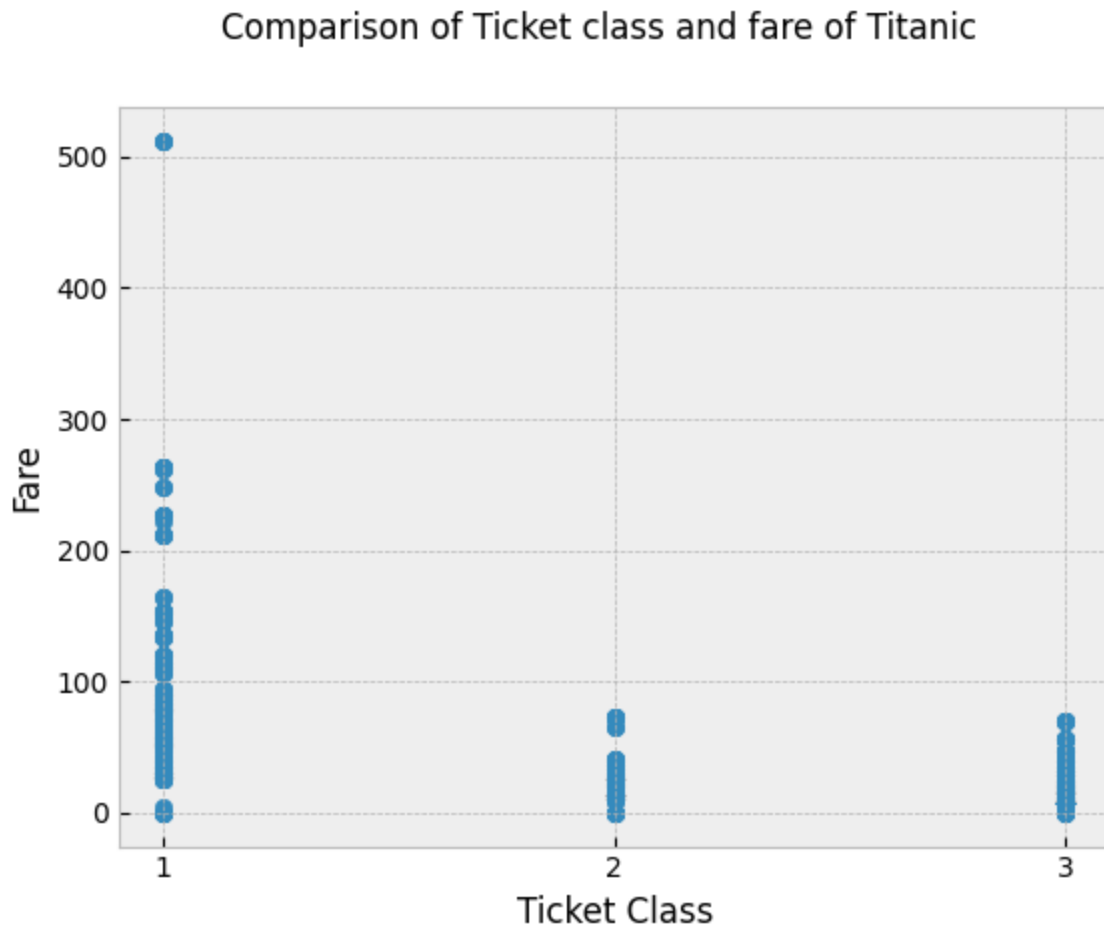
Assignment h)

```
In [ ]: # ASSIGNMENT:
# make a scatterplot of the class of ticket in the x-axis
# and the fare on the y-axis
# label the plot and the axes appropriately

# YOUR CODE HERE
tic_class = df["Pclass"]
fare = df["Fare"]

plt.scatter(tic_class, fare)
```

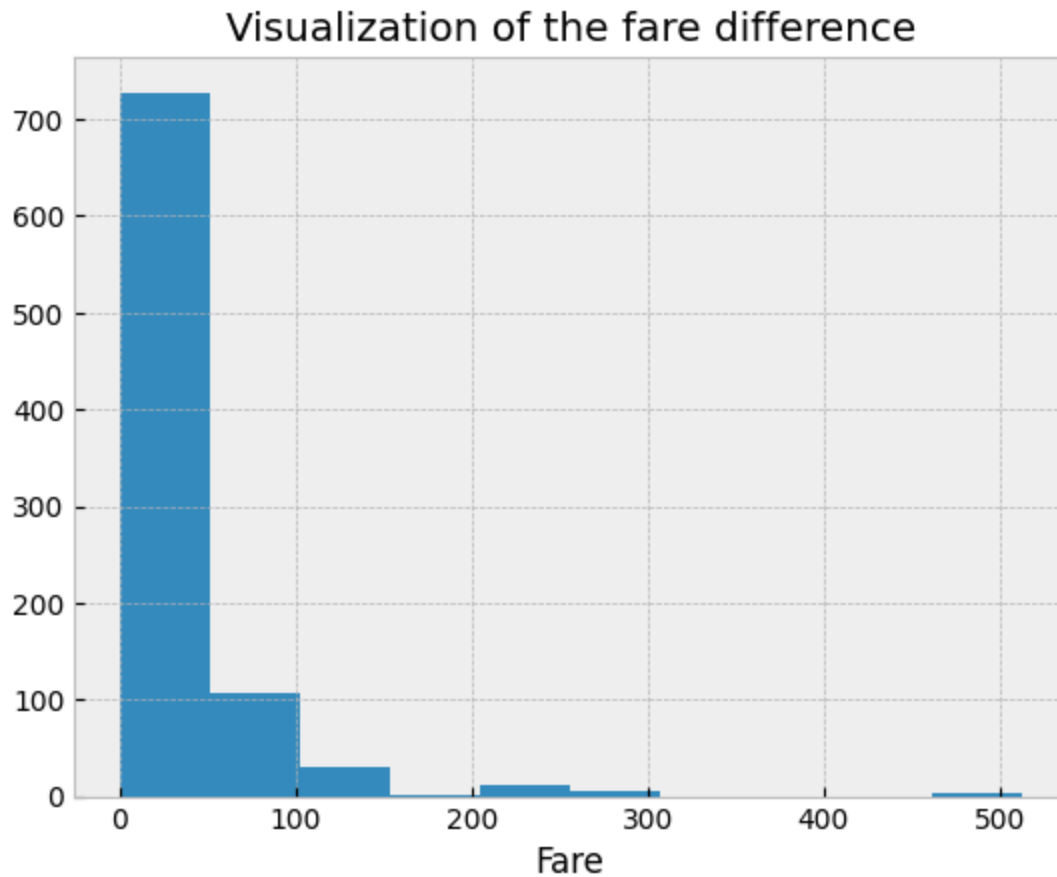
```
plt.xlabel("Ticket Class")
plt.ylabel("Fare")
plt.xticks((1,2,3)) # Since it makes no sense to have continuous values
plt.suptitle("Comparison of Ticket class and fare of Titanic")
plt.show()
```



Assignment i)

It might also be a good idea to plot a histogram over the data, to get a better understanding of how the data looks. This can be done using the function *hist* from matplotlib.

```
In [ ]: fare = df["Fare"]
plt.hist(fare)
plt.xlabel("Fare")
plt.title("Visualization of the fare difference")
plt.show()
```

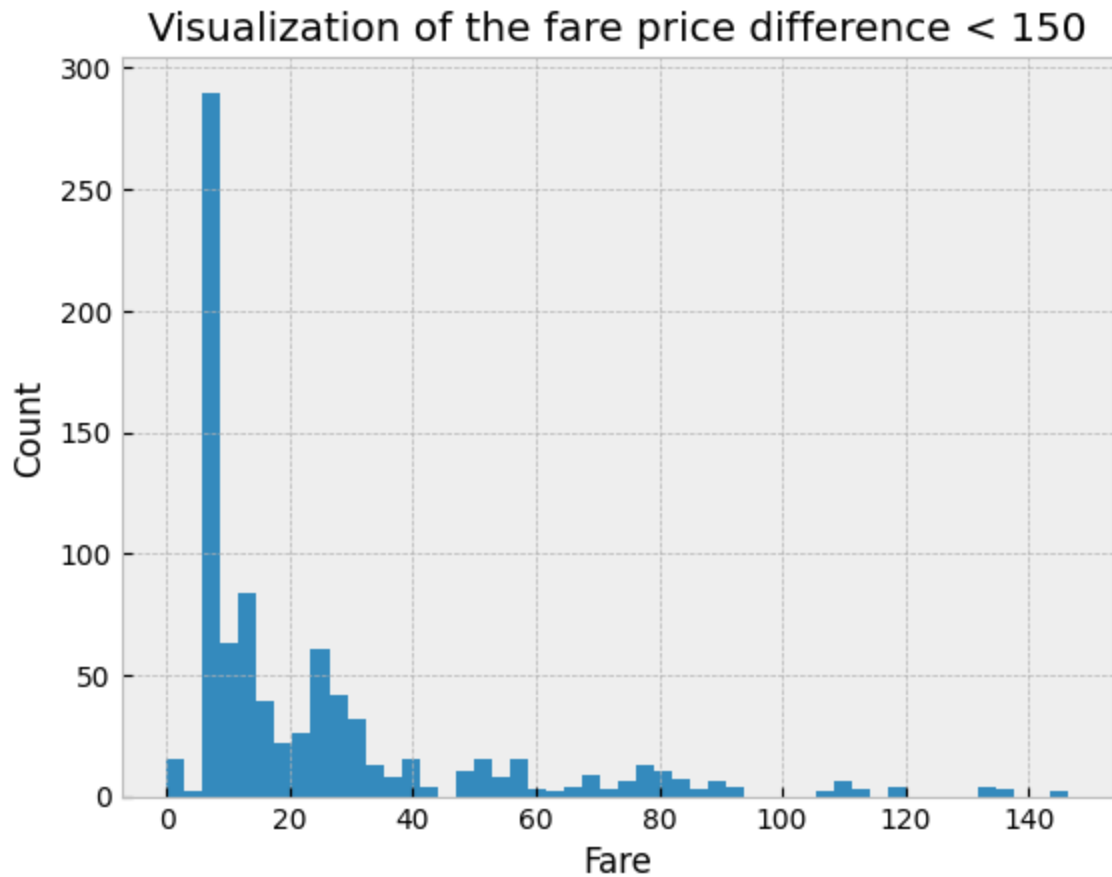


As you can see, most of the people paid less than 150 for the ticket.

```
In [ ]: # ASSIGNMENT:
# Plot a histogram over the people who paid less than, or equal to, 150.
# label the plot and the axes appropriately

# YOUR CODE HERE

fare = df["Fare"].where(df["Fare"] <= 150)
plt.hist(fare, bins=50) # I experimented a bit with number of bins, and found
plt.xlabel("Fare")
plt.ylabel("Count")
plt.title("Visualization of the fare price difference < 150")
plt.show()
```



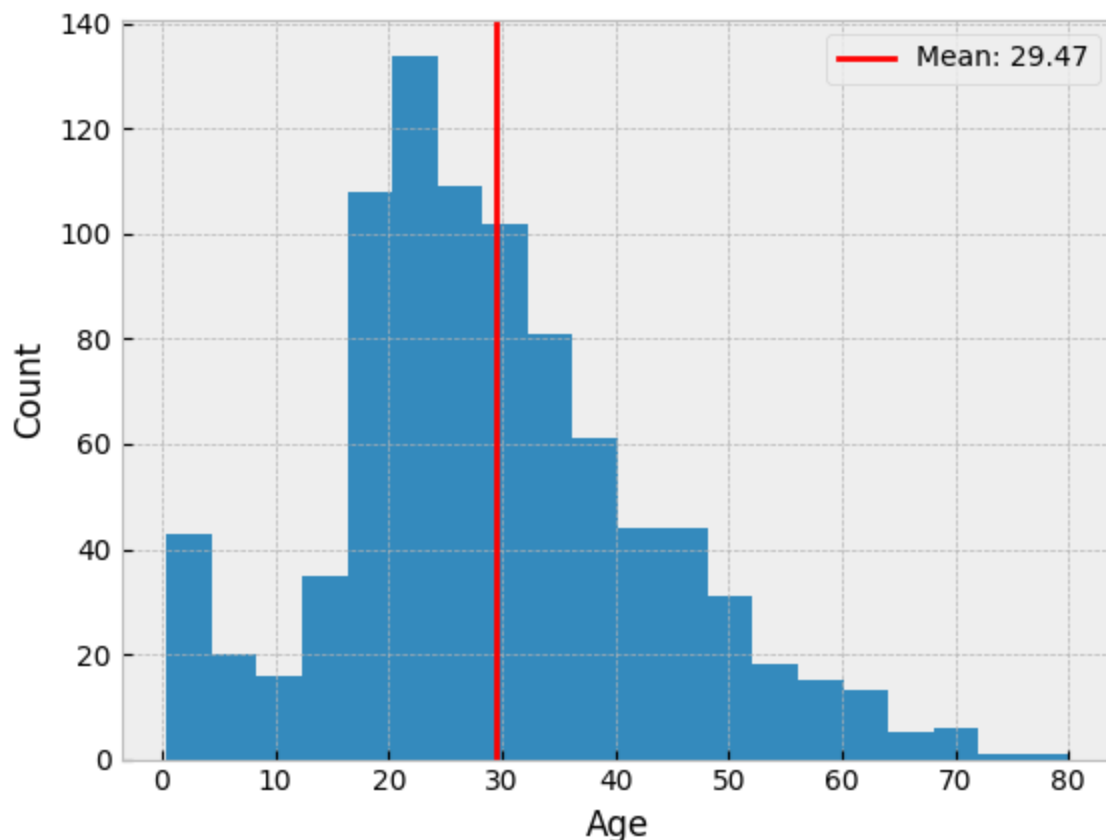
Assignment j)

```
In [ ]: # ASSIGNMENT:
# plot a histogram over all the ages with 20 bins. Draw a vertical line at t
# label the plot and the axes appropriately

# YOUR CODE HERE
ages = df.Age
mean = ages.mean()

plt.hist(ages, bins=20)
plt.axvline(mean, color='r', label=f"Mean: {mean:.2f}")
plt.xlabel("Age")
plt.ylabel("Count")
plt.suptitle("Age distribution of Titanic passengers")
plt.legend()
plt.show()
```


Age distribution of Titanic passengers



Assignment k)

Sometimes it is better to plot the figures together in one figure instead. This can be done with subplot, as shown in the examples above.

```
In [ ]: # ASSIGNMENT:
# Make a subplot over the Fare, Class, and Age
# label the plot and the axes appropriately

# YOUR CODE HERE
fig = plt.figure()
fig.suptitle("Fare, class and age of Titanic passengers", fontsize=18)

ax1 = fig.add_subplot(2,2,1)
fare = df.Fare
ax1.hist(fare, bins=50) # I experimented a bit with number of bins, and found
ax1.set_xlabel("Fare")
ax1.set_ylabel("Count")
ax1.set_title("Histogram of fares")

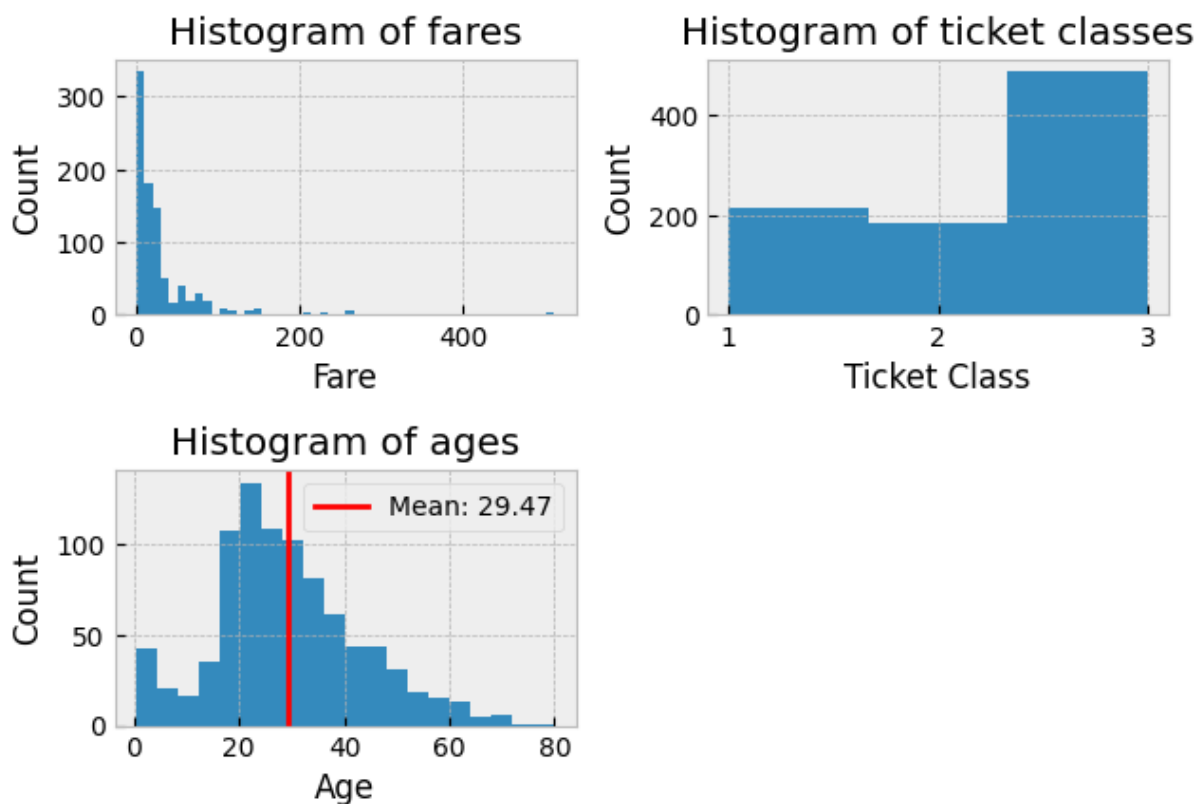
ax2 = fig.add_subplot(2,2,2)
ax2.hist(tic_class, bins=3)
ax2.set_xlabel("Ticket Class")
ax2.set_ylabel("Count")
ax2.set_xticks((1,2,3)) # Since it makes no sense to have continuous values
```

```
ax2.set_title("Histogram of ticket classes")

ax3 = fig.add_subplot(2,2,3)
ax3.hist(ages, bins=20)
ax3.axvline(mean,color='r',label=f"Mean: {mean:.2f}")
ax3.set_xlabel("Age")
ax3.set_ylabel("Count")
ax3.set_title("Histogram of ages")

fig.tight_layout()
fig.subplots_adjust(top=0.85)
plt.legend()
plt.show()
```

Fare, class and age of Titanic passengers



Assignment I)

Now we want to compare the fare and class, as we did before, but this time we want to divide them into two colors, depending on if they survived or not.

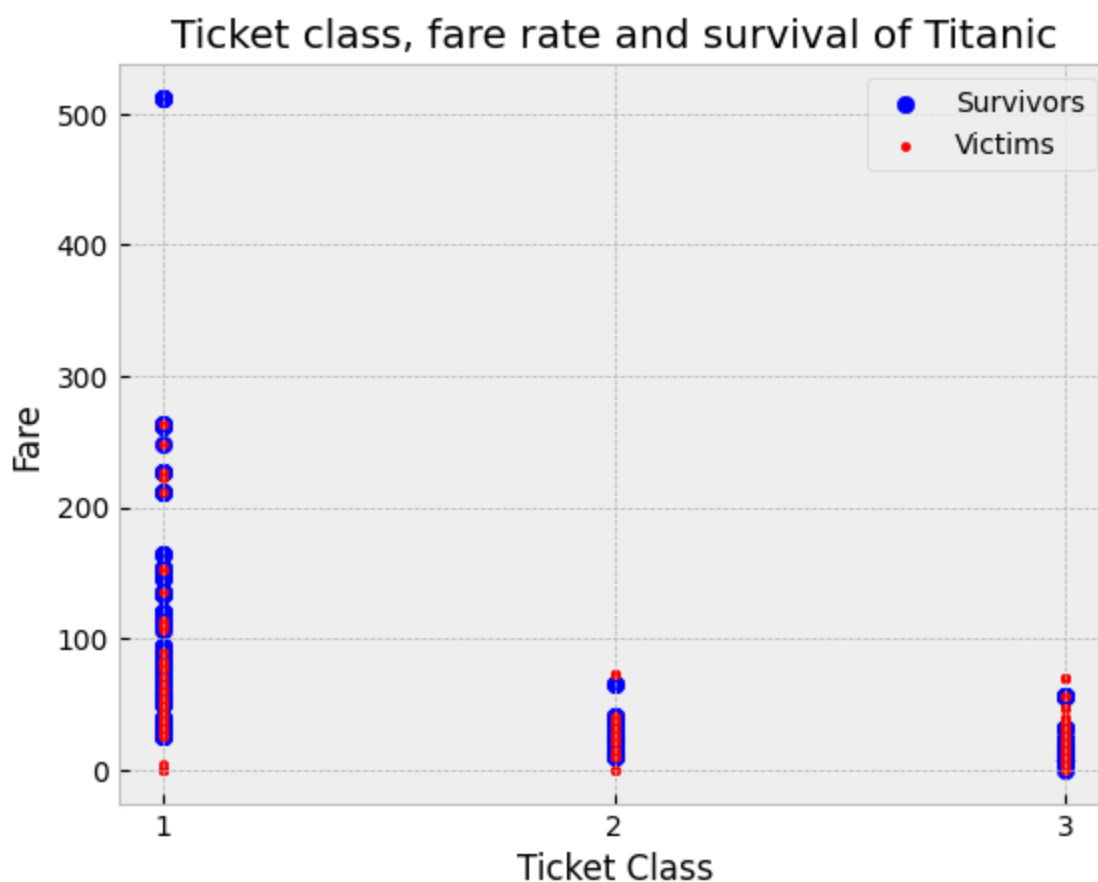
```
In [ ]: # ASSIGNMENT:
# Make a scatter plot with fare on the y-axis
# and class on the x-axis
# using red dots for all the people who died
# and blue dots for the people who survived.
# use different markers for the survived and died points
# label the plot and the axes appropriately
```

```
# YOUR CODE HERE
survivors = df[df["Survived"].isin([1])]
victims = df[df["Survived"].isin([0])]

plt.scatter(survivors.Pclass, survivors.Fare, c='blue', marker='o', label="Survivors")
plt.scatter(victims.Pclass, victims.Fare, c='red', marker='.', label="Victims")
plt.xticks((1,2,3))

plt.xlabel("Ticket Class")
plt.ylabel("Fare")
plt.xticks((1,2,3)) # Since it makes no sense to have continuous values
plt.title("Ticket class, fare rate and survival of Titanic")

plt.legend()
plt.show()
```



Assignment m)

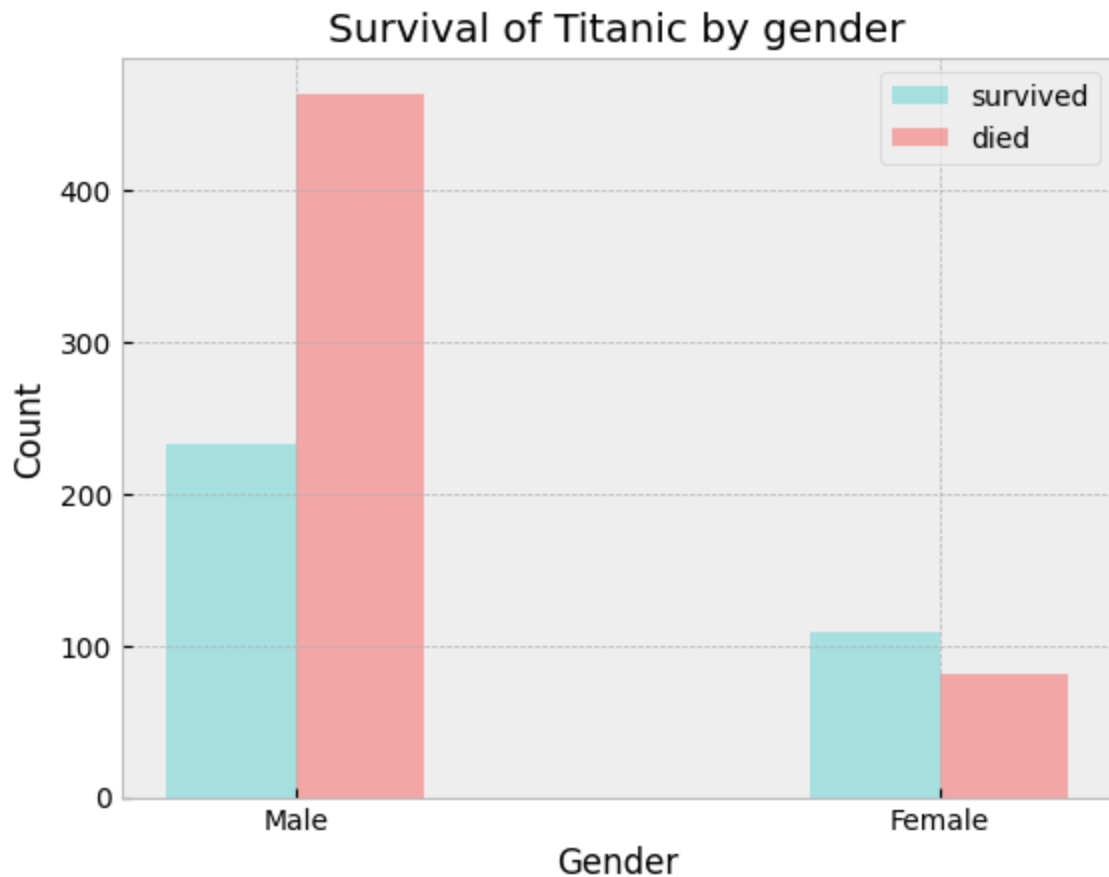
It might also be interesting to visualize how many of the men and women survived. This can be done with the bar function, which will be given to you.

```
In [ ]: # ASSIGNMENT:
# Calculate how many women and men died and survived.
# label the plot and the axes appropriately

# YOUR CODE HERE
female_survived, male_survived = df[df["Survived"].isin([1])]["Sex"].value_c
```

```
female_died, male_died = df[df["Survived"].isin([0])]["Sex"].value_counts()

plt.bar([0.9,1.9], [female_survived, male_survived] , color='c', label='survived')
plt.bar([1.1, 2.1], [female_died, male_died] , color='r', label='died', width=0.8)
plt.xticks([1,2], ['Male', 'Female']) # Switched the order, as the given code
plt.xlabel("Gender")
plt.ylabel("Count")
plt.title("Survival of Titanic by gender")
plt.legend()
plt.show()
```



OPTIONAL:

Plotting a Histogram of Random values

Your task is to generate 10000 random numbers that follows the normal distribution, with a mean, $\mu = 1$, and variance $\sigma^2 = 0.25$.

Plot the **normalized** histogram with 50 bars and a contour plot.

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt

plt.style.use('ggplot')
np.random.seed(42)

# OPTIONAL ASSIGNMENT:
# Draw 10000 random values from a normal distribution with:
```

```

# mu = 1, sigma2 = 0.25
#
# Plot the histogram and cumulative distribution
# label the plot and the axes appropriately

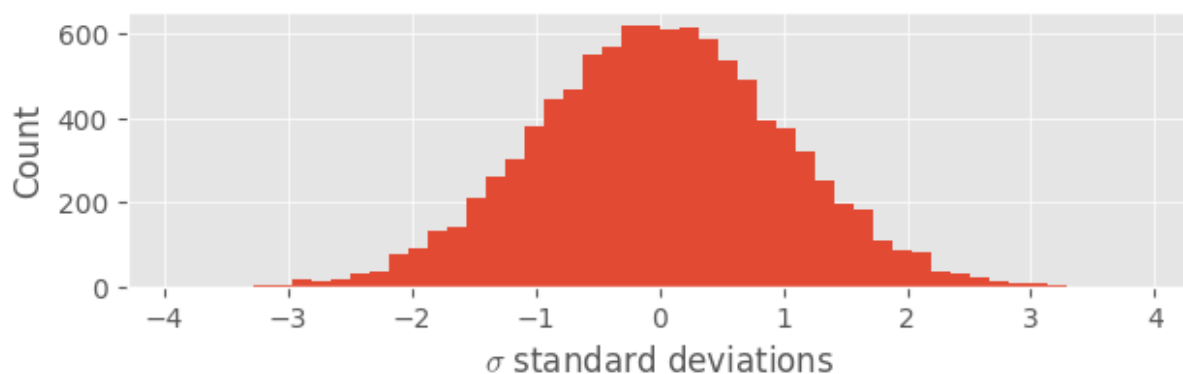
mu = 1
sigma = 0.5 # as sigma2 = 0.25
count = 10000
# YOUR CODE HERE
rs = pd.DataFrame(data=np.random.normal(loc=mu, scale=sigma, size=count))
rs_norm = (rs-rs.mean())/rs.std()
fig, (ax1,ax2) = plt.subplots(2,1)
fig.suptitle("Random numbers distribution")
ax1.hist(rs_norm,50)
ax1.set_xlabel("$\sigma$ standard deviations")
ax1.set_ylabel("Count")

ax2.plot(sorted(rs.to_numpy()))
ax2.set_title("Cumulative distribution")
ax2.set_xlabel("Sample number")
ax2.set_ylabel("Value")

plt.tight_layout()
plt.show()

```

Random numbers distribution



Cumulative distribution

