Exploratory Data Analysis

In [1]:
```
# lecture imports / dependencies
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import seaborn as sns
sns.set(style="ticks")
from sklearn.feature_extraction.text import CountVectorizer
from skimage.io import imread, imshow
```

Typical steps of ML/DM

1. Identify question / task
2. Collect data
3. Clean and preprocess data
4. Exploratory data anlysis (EDA)
5. Feature and model selection
6. Train model
7. Evaluate and communicate results
8. Deploy working system

(but not necessarily in this order...)

Today we'll discuss steps (3) and (4)

What does data look like?

Often, it is tabular (but certainly not always!).

In [6]:
```python
titanic = sns.load_dataset("titanic")
titanic.head()
```

Out[6]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [16]:
```python
titanic.dtypes
```

Out[16]:
```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck           category
embark_town      object
alive            object
alone              bool
dtype: object
```

In [17]:
```python
titanic.groupby("deck").size()
```

Out[17]:
```
deck
A    15
B    47
C    59
D    33
E    32
F    13
G     4
dtype: int64
```

- Each row is an **object** (or training example, or sample)
- Each column is a **feature** (or variable, covariate).

Types of features

- Categorical (e.g. `survived` , `embark_town`)
- Numerical (e.g. age, fare)
- Some are more ambiguous, like `pclass` : is this categorical or numerical?

Converting types:

- Many of our methods are meant to work with numerical features.
- We can convert categorical to numerical.

```
In [11]: pd.get_dummies(titanic, columns=["class"]).head()
```

Out[11]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	adult_male	deck	embark_town	alive	alone	class_First	class_Seco
0	0	3	male	22.0	1	0	7.2500	S	man	True	NaN	Southampton	no	False	0	
1	1	1	female	38.0	1	0	71.2833	C	woman	False	C	Cherbourg	yes	False	1	
2	1	3	female	26.0	0	0	7.9250	S	woman	False	NaN	Southampton	yes	True	0	
3	1	1	female	35.0	1	0	53.1000	S	woman	False	C	Southampton	yes	False	1	
4	0	3	male	35.0	0	0	8.0500	S	man	True	NaN	Southampton	no	True	0	

```
In [12]: titanic.shape
```

```
Out[12]: (891, 15)
```

```
In [111]: titanic.describe()
```

Out[111]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

If we do this for all our features, we can now interpret objects as points in space.

In [112]:
```
titanic_num = pd.get_dummies(titanic, columns=["sex","embarked","class","who","adult_male","deck","embark_town","alive
titanic_num.shape
```

Out[112]: (891, 33)

In [113]:
```
titanic_num.head()
```

Out[113]:

	survived	pclass	age	sibsp	parch	fare	sex_female	sex_male	embarked_C	embarked_Q	...	deck_E	deck_F	deck_G	embark_town_Ch
0	0	3	22.0	1	0	7.2500	0	1	0	0	...	0	0	0	
1	1	1	38.0	1	0	71.2833	1	0	1	0	...	0	0	0	
2	1	3	26.0	0	0	7.9250	1	0	0	0	...	0	0	0	
3	1	1	35.0	1	0	53.1000	1	0	0	0	...	0	0	0	
4	0	3	35.0	0	0	8.0500	0	1	0	0	...	0	0	0	

5 rows × 33 columns

In [114]:
```
titanic_num.dtypes
```

Out[114]:
```
survived                        int64
pclass                          int64
age                           float64
sibsp                           int64
parch                           int64
fare                          float64
sex_female                      uint8
sex_male                        uint8
embarked_C                      uint8
embarked_Q                      uint8
embarked_S                      uint8
class_First                     uint8
class_Second                    uint8
class_Third                     uint8
who_child                       uint8
who_man                         uint8
who_woman                       uint8
adult_male_False                uint8
adult_male_True                 uint8
deck_A                          uint8
deck_B                          uint8
deck_C                          uint8
deck_D                          uint8
deck_E                          uint8
deck_F                          uint8
deck_G                          uint8
embark_town_Cherbourg           uint8
embark_town_Queenstown          uint8
embark_town_Southampton         uint8
alive_no                        uint8
alive_yes                       uint8
alone_False                     uint8
alone_True                      uint8
dtype: object
```

- So we now have 891 objects and 33 features.
- In other words, each object is a point in 33-dimensional space.
- This is why multivariable calculus is a prerequisite.

Other feature types: text data

In [36]:
```
text = "The University of British Columbia (UBC) is a public research university with campuses and facilities in Briti
```

One approach: **bag of words** features.

```
# clustering of google documents
# checking if your mail is spam or not
# analyzing the reviews of customer on your product (positive or negative)
```

In [56]:
```
cv = CountVectorizer()
feat = cv.fit_transform([text])
```

In [57]:
```
for word, idx in cv.vocabulary_.items():
    print("%-14s%d" % (word, feat[0,idx]))
```

```
the           1
university    2
of            1
british       2
columbia      2
ubc           1
is            1
public        1
research      1
with          1
campuses      1
and           1
facilities    1
in            1
canada        1
```

- Bag of words ignores the order of words but still can work well.

Other feature types: images

In [ ]:

```
# facebook use images to tag friends and people (object recognition)
```

In [64]:

```
img = imread("Mufic.jpg")
plt.xticks([])
plt.yticks([])
imshow(img);
```

Photo credit: [Wikipedia: UBC (https://en.wikipedia.org/wiki/University_of_British_Columbia#/media/File:Irving_K._Barber_Library.jpg)](https://en.wikipedia.org/wiki/University_of_British_Columbia#/media/File:Irving_K._Barber_Library.jpg) by [CjayD (https://www.flickr.com/people/85424459@N08/)](https://www.flickr.com/people/85424459@N08/), [CC BY 2.0 (http://creativecommons.org/licenses/by/2.0)](http://creativecommons.org/licenses/by/2.0).

In [65]:

```
img.shape
```

Out[65]:

```
(1140, 2000, 3)
```

In [85]:
```
img[0:2,0:2,:]
```

Out[85]:
```
array([[[207, 213, 201],
        [210, 216, 204]],

       [[211, 217, 205],
        [216, 222, 210]]], dtype=uint8)
```

In [87]:
```
img.flatten().shape
```

Out[87]:
```
(6840000,)
```

- Now, again, the image is a point in space.
- But now the space is 6,840,000-dimensional!
- We'll talk about this towards the end of the course.

Data Cleaning

- ML+DM typically assume "clean" data.
- Ways that data might not be "clean":
  - noise (e.g., distortion on phone).
  - outliers (e.g., data entry or instrument error).
  - missing values (no value available or not applicable)
  - duplicated data (repetitions, or different storage formats).
- Any of these can lead to problems in analyses.
  - 1) want to fix these issues, apply data cleaning algorithms.
  - 2) some ML methods are robust to these.
  - often, ML is the best way to detect/fix these.

How much data do we need?

- A difficult if not impossible question to answer.
- Usual answer: "more is better".

- With the warning: "as long as the quality doesn't suffer".
- Another popular answer: "ten times the number of features".
    - I don't like this view. Features are not the enemy!

Feature aggregation

- Combine data to form new features
- Useful if there are few examples of a particular case

In [23]:
```
titanic['deck'].value_counts()
```

Out[23]:
```
C    59
B    47
D    33
E    32
A    15
F    13
G     4
Name: deck, dtype: int64
```

In [93]:
```
titanic_agg = titanic.copy()

# aggregate decks A and B into the "upper" deck category
titanic_agg["upper"] = titanic_agg['deck'].isin(("A","B"))
titanic_agg.tail()
```

Out[93]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	upper
886	0	2	male	27.0	0	0	13.00	S	Second	man	True	NaN	Southampton	no	True	False
887	1	1	female	19.0	0	0	30.00	S	First	woman	False	B	Southampton	yes	True	True
888	0	3	female	NaN	1	2	23.45	S	Third	woman	False	NaN	Southampton	no	False	False
889	1	1	male	26.0	0	0	30.00	C	First	man	True	C	Cherbourg	yes	True	False
890	0	3	male	32.0	0	0	7.75	Q	Third	man	True	NaN	Queenstown	no	True	False

(Not shown: we should still fix up the NaNs here!)

Feature selection

In [94]:
```
titanic_id = titanic.copy()

# Adding an irrelevant feature
titanic_id['id'] = titanic_id.index
titanic_id.head()
```

Out[94]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	id
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False	0
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False	1
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True	2
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False	3
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True	4

- Remove features that are not relevant to the task.
- `id` probably not relevant for prediction.

Feature transformation

Discretization (binning): turn numerical data into categorical

In [95]:
```
titanic['age'].head()
```

Out[95]:
```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: age, dtype: float64
```

In [108]:
```python
ages = pd.cut(titanic['age'], bins=(0,20,30,100))
ages.head()
```

Out[108]:
```
0     (20, 30]
1    (30, 100]
2     (20, 30]
3    (30, 100]
4    (30, 100]
Name: age, dtype: category
Categories (3, interval[int64, right]): [(0, 20] < (20, 30] < (30, 100]]
```

In [109]:
```python
ages_cat = pd.get_dummies(ages)
pd.concat([titanic['age'], ages_cat],axis=1).head()
```

Out[109]:

	age	(0, 20]	(20, 30]	(30, 100]
0	22.0	0	1	0
1	38.0	0	0	1
2	26.0	0	1	0
3	35.0	0	0	1
4	35.0	0	0	1

Mathematical transformsations

- e.g. log, exp, square, sqrt, etc.
- also, scaling/normalization

In [115]:
```
titanic.head()
```

Out[115]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [28]:
```
titanic_mod = titanic.copy()

# fare --> sqrt(fare)
titanic_mod['fare'] = np.sqrt(titanic_mod['fare'])
titanic_mod.head()
```

Out[28]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	2.692582	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	8.442944	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	2.815138	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	7.286975	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	2.837252	S	Third	man	True	NaN	Southampton	no	True

Example use case: something needs to be non-negative (exp) or shouldn't be non-negative (log).

Exploratory data analysis (EDA)

- You should always "look" at the data first.
- But how do you "look" at features and high-dimensional objects?
  - Summary statistics
  - Visualization

Categorical summary statistics

- Some summary statistics for a categorical variable:
  - **Frequencies** of different classes.
  - **Mode**: category that occurs most often.

In [120]:
```
titanic['deck'].value_counts(normalize=True) # frequencies
```

Out[120]:
```
C    0.290640
B    0.231527
D    0.162562
E    0.157635
A    0.073892
F    0.064039
G    0.019704
Name: deck, dtype: float64
```

In [121]:
```
titanic['deck'].mode()[0]
```

Out[121]:
```
'C'
```

In [122]:
```
titanic["survived"].mode()[0]
```

Out[122]:
```
0
```

In [123]:
```
titanic.groupby("survived").size()
```

Out[123]:
```
survived
0    549
1    342
dtype: int64
```

Continuous summary statistics

- Measures of location:

    - **Mean**: average value.
    - **Median**: value such that half points are larger/smaller.
    - **Quantiles**: value such that t fraction of points are smaller.
- Measures of spread:
    - **Range**: minimum and maximum values.
    - **Variance**: measures how far values are from mean.
        - Square root of variance is **standard deviation**.
    - **Intequantile ranges**: difference between quantiles

In [146]:
```
titanic.describe()
```

Out[146]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [137]:
```
titanic['fare'].var()
```

Out[137]:
```
2469.436845743117
```

Notice that the mean and std are sensitive to extreme values:

In [152]:
```python
data = [0,1,2,3,3,5,7,8,9,10,14,15,17,1000000] # the "1000000" is an outlier
print("Mean with outlier    :", np.mean(data))
print("Mean without outlier:", np.mean(data[:-1]))
```

```
Mean with outlier    : 71435.28571428571
Mean without outlier: 7.230769230769231
```

In [153]:
```python
print("Std with outlier    :", np.std(data))
print("Std without outlier:", np.std(data[:-1]))
```

```
Std with outlier    : 257537.51466268493
Std without outlier: 5.351546809515718
```

Whereas the median is not:

In [154]:
```python
print("Median with outlier    :", np.median(data))
print("Median without outlier:", np.median(data[:-1]))
```

```
Median with outlier    : 7.5
Median without outlier: 7.0
```

Distances and similarities

- There are also summary statistics between features.
  - Hamming distance:
    - Number of elements in the vectors that aren't equal.
  - Euclidean distance:
    - How far apart are the vectors?
  - Correlation:
    - Does one increase/decrease linearly as the other increases?
    - Between -1 and 1.

Limitations of summary statistics

- Summary statistics can be misleading
- A famous example is [Anscombe's quartet (https://en.wikipedia.org/wiki/Anscombe%27s_quartet)](https://en.wikipedia.org/wiki/Anscombe%27s_quartet), four datasets with:
  - Almost same means.
  - Almost same variances.
  - Almost same correlations.
  - Almost same linear fits.
  - Look completely different.

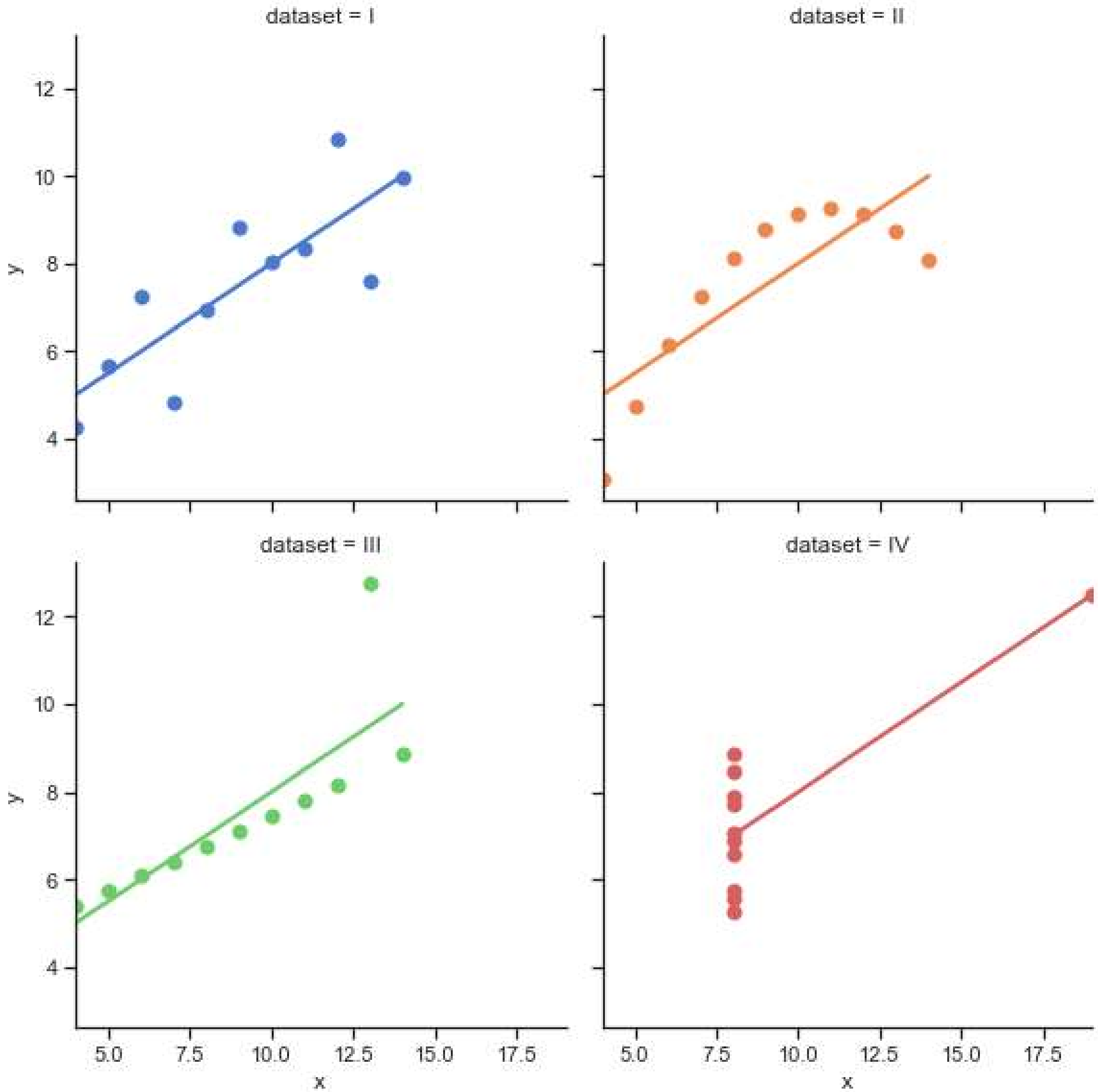
In [155]:

```
# Code below from seaborn documentation: https://seaborn.pydata.org/examples/anscombes_quartet.html

# Load the example dataset for Anscombe's quartet
anscombe = sns.load_dataset("anscombe")

# Show the results of a linear regression within each dataset
sns.lmplot(x="x", y="y", col="dataset", hue="dataset", data=anscombe,
           col_wrap=2, ci=None, palette="muted", size=4,
           scatter_kws={"s": 50, "alpha": 1});
```

```
C:\Users\User\anaconda3\lib\site-packages\seaborn\regression.py:580: UserWarning: The `size` parameter has been renam
ed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



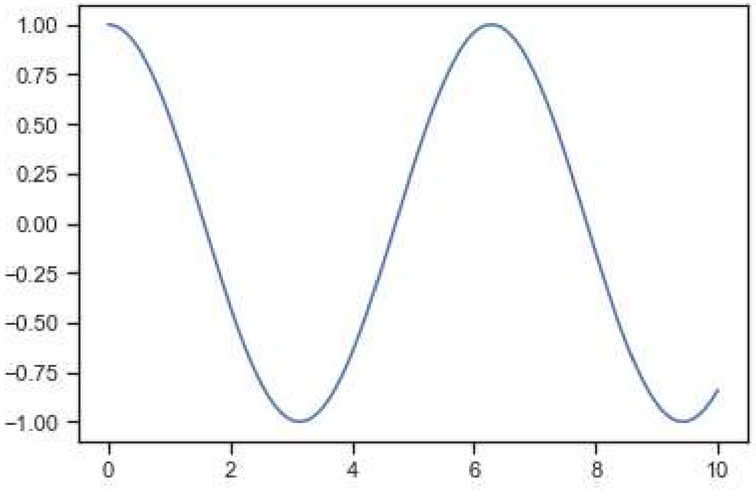
Visualization

- You can learn a lot from 2D plots of the data:
  - Patterns, trends, outliers, unusual patterns.

- We'll use the `matplotlib` library to do most of our basic plotting.
- For fancier plots, you can try `seaborn` .

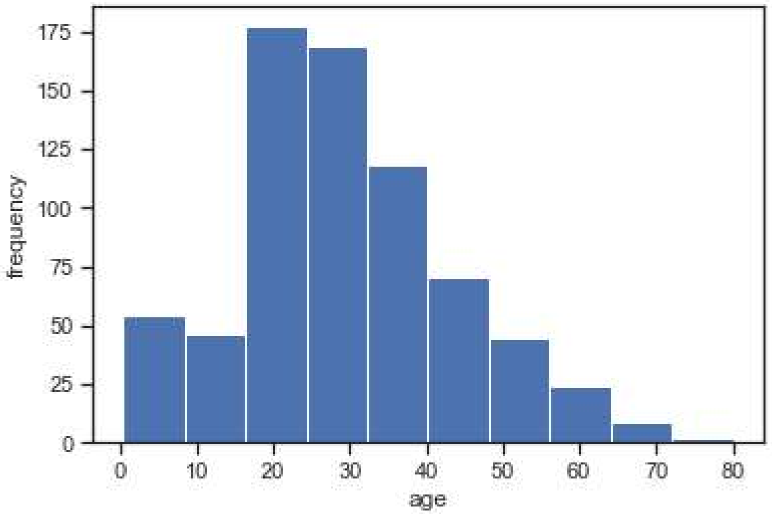
Basic plot

In [178]:
```
x = np.linspace(0,10,100)
plt.plot(x, np.cos(x));
```



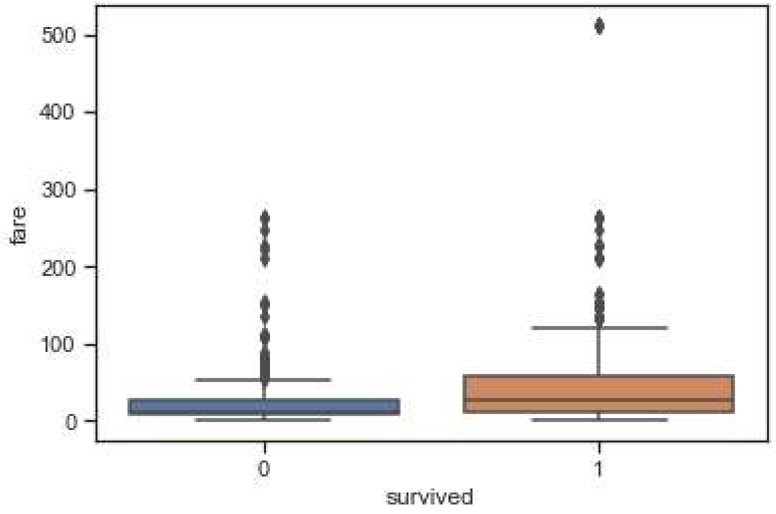
Histogram

In [190]:
```
plt.hist(titanic['age'])
plt.xlabel('age')
plt.ylabel('frequency');
# sns.distplot(iris["sepal_length"]);
```

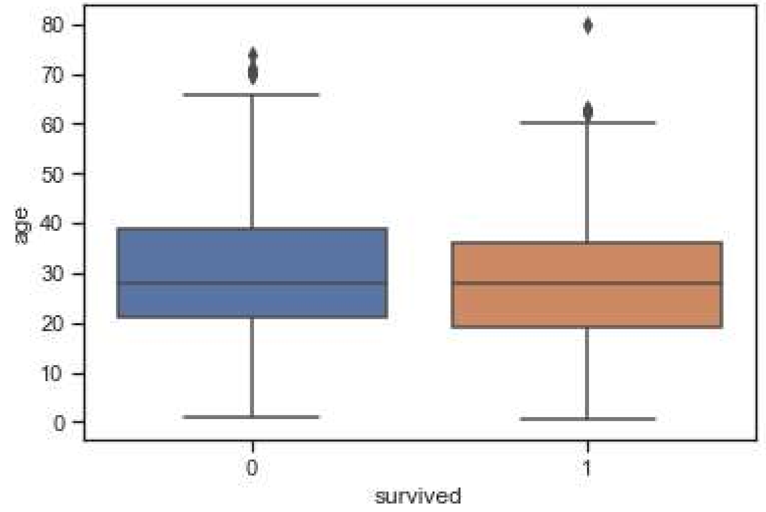


Box plot

In [195]:
```
sns.boxplot(x="survived", y="fare", data=titanic);
```

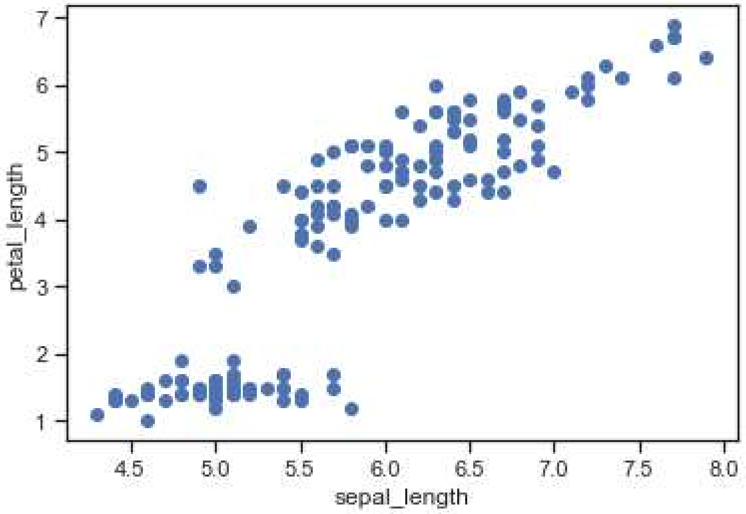


In [202]:
```
sns.boxplot(x="survived", y="age", data=titanic);
```



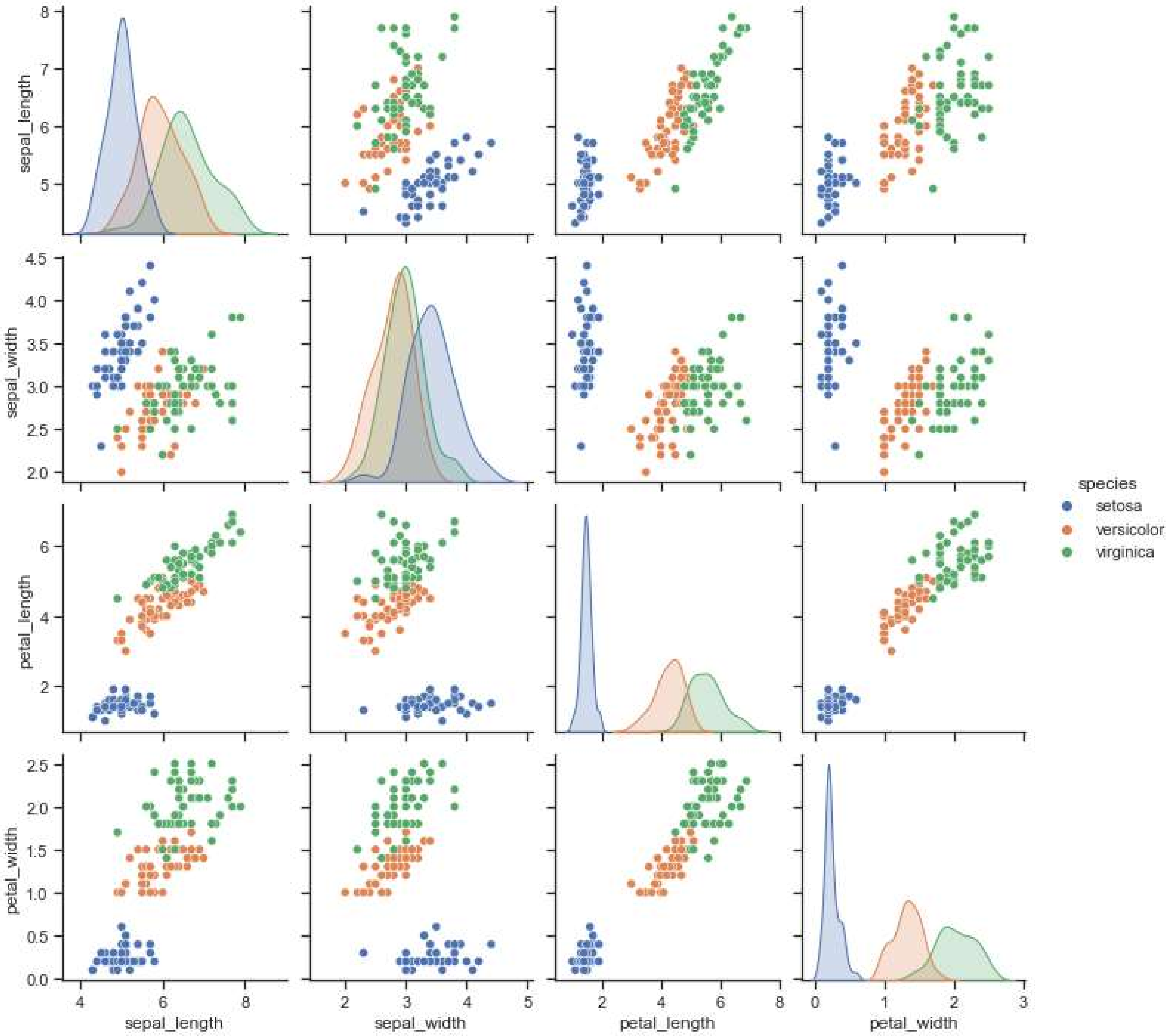
Scatterplot

In [208]:
```
plt.scatter(iris['sepal_length'], iris['petal_length'])
plt.xlabel('sepal_length')
plt.ylabel('petal_length');
```



Scatterplot array

In [206]:
```
sns.pairplot(iris, hue="species");
```


sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica

Summary

- Typical data mining steps:
  - Involves data collection, preprocessing, analysis, and evaluation.
- Object-feature representation and categorical/numerical features.
  - Transforming non-vector objects to vector representations.
- Feature transformations:
  - To address coupon collecting or simplify relationships between variables.
- Exploring data:
  - Summary statistics and data visualization.
- Post-lecture bonus slides: other visualization methods.