Model selection and validation
1.1

Training set	Validation set
The training data is used to train the algorithm. This is also the first data set we give our algorithm.	The validation set is used for selecting the model and (hyper)parameters. It keeps track of how well the algorithm is doing as it learns.

"Why do we need validation sets?"

We want to stop the learning process before the algorithm overfits.
We must then know how well it generelises at each timesteps.
Cannot use training data, because it won't detect overfitting, and we cannot use test data, because it is used for the final test.
Therefore we need a validation set, to validate the learning.

1.2

Programming.... (Next side)

Decision Tree Learning
2.1

Node	A node is a point in a network and may be connected to another node which makes a path or a branch in a tree.
Leaf	A leaf is a node that at the end of the tree, also a result in a decision tree.
Root	Root (base) is the top of the tree, the starting node, progressing down to the leaves
Branch/split	A branch/split in a decision tree represents a possible decision
Entropy	Entropy is a measure of disorder. It measures average information content of a stochastic information source. Entropy of a random variable X is: $H(x) = -\sum_i P(x = x_i) log_2 P(x = x_i)$
Gini index	Gini index is a summary of income inequality. Gini index formula is given by: $G(x) = \sum_i P(x = x_i)(1 - P(x = x_i))$
Information gain	Information gain is increase in information after splitting the tree, with formula: $IG(x) = H(y) - H(y\|x)$

2.2

$$-P\left(\frac{13}{52}\right) * log_2 P\left(\frac{13}{52}\right) = 0.5$$

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.model_selection import cross_val_score # here is cross val. in python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris() # subset a part of this as test set for question 1.2.4
X_iris = iris["data"][:, :]
y_iris = (iris["target"])

bestLR = 0
bestKNN = 0

X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size = 0.2, random_state=10)

test_iteration = [(0.1, 11), (0.4, 10), (0.9, 3), (1.5, 25), (2.3, 40), (2, 20)]
for (c,k) in test_iteration:
    log_reg = LogisticRegression(multi_class="multinomial", solver="lbfgs", C=c)
    knn = KNeighborsClassifier(n_neighbors=k)

    scoresLR = cross_val_score(log_reg, X_train, y_train, cv=6, scoring="accuracy")
    scoresKNN = cross_val_score(knn, X_train, y_train, cv=6)

    print("Value for c: ", c)
    print("Value for k: ", k)

    meanLR = np.mean(scoresLR)
    print("LR avg: ", meanLR)
    if (meanLR > bestLR):
        bestLR = meanLR
        C = c

    meanKNN = np.mean(scoresKNN)
    print("KNN avg: ", meanKNN)
    if (meanKNN > bestKNN):
        bestKNN = meanKNN
        K = k

print("Best avg for LogisticRegression: ", bestLR, " with C: ", C)
print("Best avg for KNearestNeighbor: ", bestKNN, " with K: ", K)

mlo = LogisticRegression(multi_class="multinomial", solver="lbfgs", C=C)
mlo.fit(X_test, y_test)
mlo_score = mlo.score(X_test, y_test)
print("Best LogisticRegression score: ", mlo_score, " with C: ", C)

knn = KNeighborsClassifier(n_neighbors=K)
knn.fit(X_test, y_test)
knn_score = knn.score(X_test, y_test)
print("Best KNeighborsClassifier score: ", knn_score, " with K: ", K)

#Best avg for LogisticRegression:  0.9669856459330144  with C:  0.9
#Best avg for KNearestNeighbor:  0.9669856459330144  with K:  3
#Best LogisticRegression score:  0.9333333333333333  with C:  0.9
#Best KNeighborsClassifier score:  0.9666666666666667  with K:  3
```

2.3

A: entropy(E):

IG(solar system, distance) = E(solar system) - E(solar system|distance)

E(parent)

$$\text{planet} = P(\tfrac{6}{10}) \cdot \log_2 (P(\tfrac{6}{10})) \simeq -0{,}442$$

$$\text{star} = P(\tfrac{4}{10}) \cdot \log_2 (P(\tfrac{4}{10})) = -0{.}528$$

$$-(-0{,}442) + (-0{.}528) \approx 0{,}9709$$

E(child 1)

$$\text{planet} = -0{.}528$$

$$\text{star} = -0{,}442$$

$$= 0{.}9709$$

E(child 2)

$$\text{planet} = P(\tfrac{4}{5}) \cdot \log_2 (P(\tfrac{4}{5})) \simeq -0{,}257$$

$$\text{star} = P(\tfrac{1}{5}) \cdot \log_2 (P(\tfrac{1}{5})) \approx -0{,}464$$

$$-(-0{,}257) + (-0{.}464) \simeq 0{,}7219$$

IG =

$$0{,}9709 - (\tfrac{5}{10} \cdot 0{.}9709) - (\tfrac{5}{10} \cdot 0{,}7219)$$

$$= 0{,}1245$$

B: Gini index(G): $G(x) = \sum_i P(x = x_i)(1 - P(x = x_i))$

IG(solar system, distance) = G(solar system) - G(solar system|distance)

G(parent)

planet $= P(\frac{6}{10}) \cdot (1 - P(\frac{6}{10}))$

$= 0,24$

star $= P(\frac{4}{10}) \cdot (1 - P(\frac{4}{10}))$

$= 0,24$

$= 0.48$

G(child 1)

planet $= P(\frac{2}{5}) \cdot (1 - P(\frac{2}{5}))$

$= 0.24$

star $= P(\frac{3}{5}) \cdot (1 - P(\frac{3}{5})$

$= 0.24$

$= 0,48$

G(child 2)

planet $= P(\frac{4}{5}) \cdot (1 - P(\frac{4}{5}))$

$= 0.16$

star $= P(\frac{1}{5}) \cdot (1 - P(\frac{1}{5}))$

$= 0.16$

$= 0,32$

IG $= 0.08$

Pruning decision trees

A:

1. Choose a candidate for pruning
2. For a subtree S of the whole tree, if replacing S by a leaf does not increase the prediction errors on the pruning set than the original tree, replace S by a leaf
3. Repeat the last step again until performing pruning does not decrease the prediciton error

1. Pick a subtree S to performing pruning on
2. If (error of child > error of parent)
    a. Replace S by a leaf node //Pruning
3. Repeat step 2 until performing pruning does not decrease the prediction error

B: