1.1. There are numerous examples of supervised machine learning in use today.

For example, fraud detection by credit card companies based on your prior usage behavior.
https://towardsdatascience.com/detecting-credit-card-fraud-using-machine-learning-a3d83423d3b8

Another example is precision medicine in healthcare, in which prognosis and treatment is customize to a particular patient.
https://towardsdatascience.com/precision-medicine-and-machine-learning-11060caa3065

1.2. Unsupervised learning has been applied to drug discovery by biomedical institutions and pharmaceutical companies. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6552674/

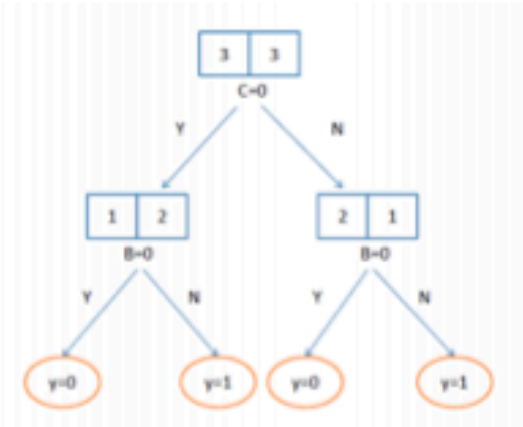
1.3. Reinforcement Learning has many applications including applications in trading and finance.
https://neptune.ai/blog/reinforcement-learning-applications

1.4. The code and output are below:

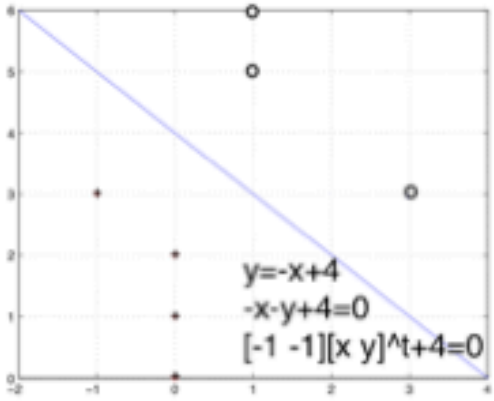
```
-------
def new_soln(sentence):
   return "".join([char for char in sentence if char not in "aeiouAEIOU"])

sentence = "The quick brown fox jumps over the lazy dog"
print("Filtered result: " + new_soln(sentence))
-------
Filtered result: Th qck brwn fx jmps vr th lzy dg
```

2.1. For n binary attributes, there are 2^n possible values. Each value can be mapped to 2 outputs/labels. So, in this scenario, there are 2^(2^n) possible decision trees.
This is the same as the number of Boolean functions. https://en.wikipedia.org/wiki/Boolean_function

2.2. (a) The entropy of the output Y is 1.
(b) If you split at A, then the output is partitioned as A=0: Y=0 0 1 and A=1: 0 1 1
So, H(y|A=0)=0.9183 and H(y|A=1)=0.9183 and
then H(Y|A)=Pr(A=0)* H(y|A=0)+Pr(A=1)* H(y|A=1)=0.9183
Similarly: H(Y|B)=1 and H(Y|C)=0.9183
So, we see that B is worse than A or C, either of which is good.
Randomly choose 1 and continue to build the tree.
(c) In the example below, I chose to split at C but you could also have chosen to get another decision tree. So, the decision tree in this case is not unique.



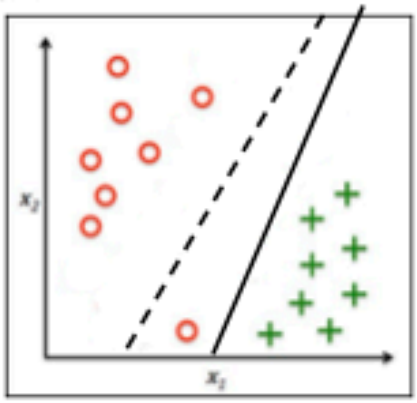
3.1. (a). The plot is as follows. Positive samples are plotted a '+' and negative samples are plotted as 'o'. The solid line is the linear svm / hard margin solution. The line is given by y = -x+4.



(b) Rewriting the equation of the line in the form (**w**^t)(**x**)+b=0, we get that **x**^t=[x1, x2], **w**^t=[-1 -1] and b=4. Then the classifier is given by $\hat{y}$=sign(**w**^t***x** + b) Any suitable scaling of the above function is also a valid solution.

(c) Note that all of the new positive points are in the interior of the positive class, so they actually do not affect the SVM linear boundary. Other linear classifiers such as the perceptron will change since they respond to the mean of positive class (which is shifted by the addition of new positive points).

3.2.



a) The solid line is the SVM linear classifier when C is large.
b) The dotted line is the SVM linear classifier when C is small.
c) The large C solution resembles the hard margin SVM.
d) C controls the penalty for misclassifications. So large C leads to not allowing misclassifications, which leads to overfitting to noisy samples and outliers. Small C leads to allowing misclassifications and enables ignoring outliers.

4.1. a) Accuracy is not a good metric for imbalanced datasets.
b) Precision in important when being right (positive prediction is correct) outweighs detecting all positives. For example, recommendation systems.
c) Recall is important when false negatives are catastrophic. For example, disease detection.

4.2. It is possible for classifier 1 to have higher accuracy but lower precision and recall than classifier 2. In general, whether a classifier is better than the other depends on the measure used to judge "better", so it is important to make sure that you choose the measure that matches your problem.

		Classifier 1		**Classifier 2**		
		actual class		**actual class**		Classifier 1: (4+0)/(4+1+5+0)=4/10 Classifier 2: (0+3)/(0+2+5+3)=3/10 < 4/10 Classifier 1: Precision=Recall=0 (since TP=0) Classifier 2: Precision=3/8 Recall=3/5
		Neg	**Pos**	**Neg**	**Pos**	
predicted class	**Neg**	**4** (*TN*)	**5** *(FN)*	**0** (*TN*)	**2** *(FN)*	
	Pos	**1** (*FP*)	**0** *(TP)*	**5** (*FP*)	**3** *(TP)*	

4.3. a) – 7.281x + 360.637, SSE_Train=2531.529
c) – 7.888x + 366.305, SSE_Train=1705.329, SSE_Test=1327.822
d) Sci-Kit Learn uses *gradient descent* to compute the solution to the optimization problem. In easy problems (with no local minima), the gradient descent solution will be the same as the exact solution. In other problems, it may not be the same, as gradient descent may get stuck in a local minimum.

5.1.

(i) Naïve Bayes and Logistic Regression both try to estimate the posterior probability P(y|x) and choose the outcome which maximizes that probability. They are different in that Logistic Regression estimates the posterior P(y|x) directly while Naïve Bayes uses Bayes Theorem to convert the problem into computing the likelihood P(x|y).

(ii) When you compute the posterior probability corresponding to the Gauassian Naïve Bayes approach, it turns out that you get the same posterior probability as Logistic Regression, which models the posterior with the logistic/sigmoid function.

(iii) In Logistic Regression, the idea is to model the posterior P(y|x) as a logistic function. When you do this, you get that the logit (or log odds) is a weighted linear combination of the features. This is similar to linear regression in that the dependent variable itself is a linear function of the features. In both cases, you can solve for the feature weight by regressing the appropriate function to fit the data. For Linear Regression, this solves the regression problem. For Logistic Regression, this solves the classification problem.

5.2. Recall the Play Tennis/No Play Tennis dataset

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

P(y=yes) = 9/14 P(y=no) = 5/14	P(sunny|yes) = P(S|yes) = 2/9 P(sunny|no) = P(S|no) = 3/5	P(windy|yes) = P(W|yes) = 3/9 P(windy|no) = P(W|no) = 3/5

→ You can look at the Sunny days and count the "yes": P(yes|S) = 2/5

→ Or apply Bayes/conditional probability rule:
P(yes|S)=P(S|yes)P(yes)/P(S)= (2/9)(9/14)/(5/14) = 2/5

→ P(S)=5/14 counting directly
→ Or using total probability:
P(S)=P(S|yes)P(yes)+P(S|no)P(no) = (2/9)(9/14) + (3/5)(5/14) = 5/14

→ Naïve Bayes: P(yes|S,W) = P(S,W|yes)P(yes)/P(S,W) = (6/81)(9/14)/(37/210) = 10/37
→ Naïve Bayes: P(S,W|yes) = P(S|yes)P(W|yes) = (2/9)(3/9) = 6/81
→ P(S,W)=P(S,W|yes)P(yes) + P(S,W|no)P(no)
= P(S|yes)P(W|yes)P(yes) + P(S|no)P(W|no)P(no)
= (2/9)(3/9)(9/14) + (3/5)(3/5)(5/14) = 6/126 + 9/70 = 37/210

→ Direct computation: P(yes|S,W) = 1/2 ← Only 2 data points, is there enough data?

There can be inconsistencies in computing probabilities! Need to be careful.