

# Midterm #1

# A business case

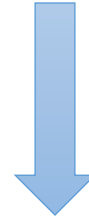
Audiobooks

# The dataset

A	B	C	D	E	F	G	H	I	J	K	L
ID	Book length (mins)_overall	Book length (mins)_avg	Price_overall	Price_avg	Review	Review 10/10	Minutes listened	Completion	Support Requests	Last visited minus Purchase date	Targets
00994	1620	1620	7	19.73	1	10	1603.8	0.99	5	92	0
01143	2160	2160	5.33	5.33	0		0	0	0	0	0
02059	2160	2160	5.33	5.33	0		0	0	0	388	0
02882	1620	1620	5.96	5.96	0		680.4	0.42	1	129	0
03342	2160	2160	5.33	5.33	0		475.2	0.22	0	361	0
03416	2160	2160	4.61	4.61	0		0	0	0	0	0
04949	2160	2160	5.33	5.33	0		86.4	0.04	0	366	0
09011	648	648	5.33	5.33	0		0	0	0	0	1
09282	2160	2160	5.33	5.33	0		561.6	0.26	0	33	0
10500	2160	2160	5.33	5.33	1	10	583.2	0.27	0	366	0
12898	540	540	5.33	5.33	0		151.2	0.28	0	34	0
16370	2160	2160	5.33	5.33	0		0	0	0	0	1
16587	2160	2160	13.33	13.33	0		1296	0.6	0	98	0
18065	648	648	50.66	50.66	0		0	0	0	0	0
19313	2160	2160	7.03	7.03	0		0	0	0	0	0
19996	1188	594	58.67	29.33	1	10	0	0	0	0	1
20124	2160	2160	5.33	5.33	0		0	0	0	29	0
23153	1620	1620	11.68	11.68	1	8	1150.2	0.71	1	25	0
23372	2160	2160	5.33	5.33	0		0	0	0	1	0
23757	2160	2160	5.33	5.33	0		0	0	0	9	0
24187	1620	1620	14.29	14.29	1	7	696.6	0.43	0	66	0
25850	1620	1620	5.33	5.33	0		129.6	0.08	0	342	0
27030	4752	1188	23.99	6	1	10	0	0	0	176	1
27703	1620	1620	5.33	5.33	0		0	0	0	0	0
29095	2808	1404	55.99	27.99	0		0	0	0	0	1
31062	2160	2160	5.33	5.33	0		1296	0.6	0	228	0
33068	2160	2160	5.96	5.96	0		0	0	0	0	0

# The dataset


Missing Data!



A	B	C	D	E	F	G	H	I	J	K	L
ID	Book length (mins)_overall	Book length (mins)_avg	Price_overall	Price_avg	Review	Review 10/10	Minutes listened	Completion	Support Requests	Last visited minus Purchase date	Targets
00994	1620	1620	7	19.73	1	10	1603.8	0.99	5	92	0
01143	2160	2160	5.33	5.33	0		0	0	0	0	0
02059	2160	2160	5.33	5.33	0		0	0	0	388	0
02882	1620	1620	5.96	5.96	0		680.4	0.42	1	129	0
03342	2160	2160	5.33	5.33	0		475.2	0.22	0	361	0
03416	2160	2160	4.61	4.61	0		0	0	0	0	0
04949	2160	2160	5.33	5.33	0		86.4	0.04	0	366	0
09011	648	648	5.33	5.33	0		0	0	0	0	1
09282	2160	2160	5.33	5.33	0		561.6	0.26	0	33	0
10500	2160	2160	5.33	5.33	1	10	583.2	0.27	0	366	0
12898	540	540	5.33	5.33	0		151.2	0.28	0	34	0
16370	2160	2160	5.33	5.33	0		0	0	0	0	1
16587	2160	2160	13.33	13.33	0		1296	0.6	0	98	0
18065	648	648	50.66	50.66	0		0	0	0	0	0
19313	2160	2160	7.03	7.03	0		0	0	0	0	0
19996	1188	594	58.67	29.33	1	10	0	0	0	0	1
20124	2160	2160	5.33	5.33	0		0	0	0	29	0
23153	1620	1620	11.68	11.68	1	8	1150.2	0.71	1	25	0
23372	2160	2160	5.33	5.33	0		0	0	0	1	0
23757	2160	2160	5.33	5.33	0		0	0	0	9	0
24187	1620	1620	14.29	14.29	1	7	696.6	0.43	0	66	0
25850	1620	1620	5.33	5.33	0		129.6	0.08	0	342	0
27030	4752	1188	23.99	6	1	10	0	0	0	176	1
27703	1620	1620	5.33	5.33	0		0	0	0	0	0
29095	2808	1404	55.99	27.99	0		0	0	0	0	1
31062	2160	2160	5.33	5.33	0		1296	0.6	0	228	0
33068	2160	2160	5.96	5.96	0		0	0	0	0	0

# The dataset

Missing Data!  
We add the average



ID	Book length (mins)_overall	Book length (mins)_avg	Price_overall	Price_avg	Review	Review 10/10	Minutes listened	Completion	Support Requests	Last visited minus Purchase date	Targets
00994	1620	1620	7	19.73	1	10	1603.8	0.99	5	92	0
01143	2160	2160	5.33	5.33	0	8.91	0	0	0	0	0
02059	2160	2160	5.33	5.33	0	8.91	0	0	0	388	0
02882	1620	1620	5.96	5.96	0	8.91	680.4	0.42	1	129	0
03342	2160	2160	5.33	5.33	0	8.91	475.2	0.22	0	361	0
03416	2160	2160	4.61	4.61	0	8.91	0	0	0	0	0
04949	2160	2160	5.33	5.33	0	8.91	86.4	0.04	0	366	0
09011	648	648	5.33	5.33	0	8.91	0	0	0	0	1
09282	2160	2160	5.33	5.33	0	8.91	561.6	0.26	0	33	0
10500	2160	2160	5.33	5.33	1	10	583.2	0.27	0	366	0
12898	540	540	5.33	5.33	0	8.91	151.2	0.28	0	34	0
16370	2160	2160	5.33	5.33	0	8.91	0	0	0	0	1
16587	2160	2160	13.33	13.33	0	8.91	1296	0.6	0	98	0
18065	648	648	50.66	50.66	0	8.91	0	0	0	0	0
19313	2160	2160	7.03	7.03	0	8.91	0	0	0	0	0
19996	1188	594	58.67	29.33	1	10	0	0	0	0	1
20124	2160	2160	5.33	5.33	0	8.91	0	0	0	29	0
23153	1620	1620	11.68	11.68	1	8	1150.2	0.71	1	25	0
23372	2160	2160	5.33	5.33	0	8.91	0	0	0	1	0

# The dataset

Total minutes listened / book\_length\_overall

ID	Book length (mins)_overall	Book length (mins)_avg	Price_overall	Price_avg	Review	Review 10/10	Minutes listened	Completion	Support Requests	Last visited minus Purchase date	Targets
00994	1620	1620	7	19.73	1	10	1603.8	0.99	5	92	0
01143	2160	2160	5.33	5.33	0	8.91	0	0	0	0	0
02059	2160	2160	5.33	5.33	0	8.91	0	0	0	388	0
02882	1620	1620	5.96	5.96	0	8.91	680.4	0.42	1	129	0
03342	2160	2160	5.33	5.33	0	8.91	475.2	0.22	0	361	0
03416	2160	2160	4.61	4.61	0	8.91	0	0	0	0	0
04949	2160	2160	5.33	5.33	0	8.91	86.4	0.04	0	366	0
09011	648	648	5.33	5.33	0	8.91	0	0	0	0	1
09282	2160	2160	5.33	5.33	0	8.91	561.6	0.26	0	33	0
10500	2160	2160	5.33	5.33	1	10	583.2	0.27	0	366	0
12898	540	540	5.33	5.33	0	8.91	151.2	0.28	0	34	0
16370	2160	2160	5.33	5.33	0	8.91	0	0	0	0	1
16587	2160	2160	13.33	13.33	0	8.91	1296	0.6	0	98	0
18065	648	648	50.66	50.66	0	8.91	0	0	0	0	0
19313	2160	2160	7.03	7.03	0	8.91	0	0	0	0	0
19996	1188	594	58.67	29.33	1	10	0	0	0	0	1
20124	2160	2160	5.33	5.33	0	8.91	0	0	0	29	0
23153	1620	1620	11.68	11.68	1	8	1150.2	0.71	1	25	0
23372	2160	2160	5.33	5.33	0	8.91	0	0	0	1	0

# The dataset

The difference between the last time a person interacted with the platform and their first purchase date

ID	Book length (mins)_overall	Book length (mins)_avg	Price_overall	Price_avg	Review	Review 10/10	Minutes listened	Completion	Support Requests	Last visited minus Purchase date	Targets
00994	1620	1620	7	19.73	1	10	1603.8	0.99	5	92	0
01143	2160	2160	5.33	5.33	0	8.91	0	0	0	0	0
02059	2160	2160	5.33	5.33	0	8.91	0	0	0	388	0
02882	1620	1620	5.96	5.96	0	8.91	680.4	0.42	1	129	0
03342	2160	2160	5.33	5.33	0	8.91	475.2	0.22	0	361	0
03416	2160	2160	4.61	4.61	0	8.91	0	0	0	0	0
04949	2160	2160	5.33	5.33	0	8.91	86.4	0.04	0	366	0
09011	648	648	5.33	5.33	0	8.91	0	0	0	0	1
09282	2160	2160	5.33	5.33	0	8.91	561.6	0.26	0	33	0
10500	2160	2160	5.33	5.33	1	10	583.2	0.27	0	366	0
12898	540	540	5.33	5.33	0	8.91	151.2	0.28	0	34	0
16370	2160	2160	5.33	5.33	0	8.91	0	0	0	0	1
16587	2160	2160	13.33	13.33	0	8.91	1296	0.6	0	98	0
18065	648	648	50.66	50.66	0	8.91	0	0	0	0	0
19313	2160	2160	7.03	7.03	0	8.91	0	0	0	0	0
19996	1188	594	58.67	29.33	1	10	0	0	0	0	1
20124	2160	2160	5.33	5.33	0	8.91	0	0	0	29	0
23153	1620	1620	11.68	11.68	1	8	1150.2	0.71	1	25	0
23372	2160	2160	5.33	5.33	0	8.91	0	0	0	1	0



# The dataset

After 2 years period (data collected), this Boolean tells if the user bought another book in the following 6 months

ID	Book length (mins)_overall	Book length (mins)_avg	Price_overall	Price_avg	Review	Review 10/10	Minutes listened	Completion	Support Requests	Last visited minus Purchase date	Targets
00994	1620	1620	7	19.73	1	10	1603.8	0.99	5	92	0
01143	2160	2160	5.33	5.33	0	8.91	0	0	0	0	0
02059	2160	2160	5.33	5.33	0	8.91	0	0	0	388	0
02882	1620	1620	5.96	5.96	0	8.91	680.4	0.42	1	129	0
03342	2160	2160	5.33	5.33	0	8.91	475.2	0.22	0	361	0
03416	2160	2160	4.61	4.61	0	8.91	0	0	0	0	0
04949	2160	2160	5.33	5.33	0	8.91	86.4	0.04	0	366	0
09011	648	648	5.33	5.33	0	8.91	0	0	0	0	1
09282	2160	2160	5.33	5.33	0	8.91	561.6	0.26	0	33	0
10500	2160	2160	5.33	5.33	1	10	583.2	0.27	0	366	0
12898	540	540	5.33	5.33	0	8.91	151.2	0.28	0	34	0
16370	2160	2160	5.33	5.33	0	8.91	0	0	0	0	1
16587	2160	2160	13.33	13.33	0	8.91	1296	0.6	0	98	0
18065	648	648	50.66	50.66	0	8.91	0	0	0	0	0
19313	2160	2160	7.03	7.03	0	8.91	0	0	0	0	0
19996	1188	594	58.67	29.33	1	10	0	0	0	0	1
20124	2160	2160	5.33	5.33	0	8.91	0	0	0	29	0
23153	1620	1620	11.68	11.68	1	8	1150.2	0.71	1	25	0
23372	2160	2160	5.33	5.33	0	8.91	0	0	0	1	0



# Goal

- Create a machine learning model that predicts if a customer will buy again from the platform

# Steps

## 1. Preprocess the data

### a. Balance the dataset

- This is crucial! You want the “targets” to be evenly distributed

### b. Divide the dataset in Training, Validation and Testing sets

- We want to prevent overfitting

### c. Save the dataset in tensor format [.npz]

## 2. Create the machine learning algorithm

# Preprocessing example

- We will use the **sklearn** preprocessing library, as it will be easier to standardize the data

You can install it with `pip install sklearn`

# Preprocessing example

```
import numpy as np
from sklearn import preprocessing

raw_csv_data = np.loadtxt('Audiobooks_data.csv', delimiter=',')
unscaled_inputs_all = raw_csv_data[:, 1:-1]
targets_all = raw_csv_data[:, -1]
```

[ :, 1 :-1 ]

- The first part [ :, indicates that we want to read all the rows (we don't specify any upper and lower bound before and after the : symbol)
- Then, we want the inputs from all columns in the csv, except for the first one 1: (indexed 0) which is just the arbitrary customer IDs that bear no useful information, and the last one :,-1] (which is our targets)

➤ The targets are in the last column

- That's how datasets are conventionally organized.

[ :, -1 ] , all rows and all columns but the last one

# Balancing the dataset

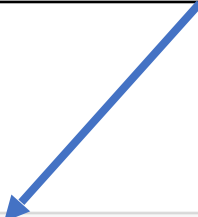
```
num_one_targets = int(np.sum(targets_all))
zero_targets_counter = 0

indices_to_remove = []

for i in range(targets_all.shape[0]):
    if targets_all[i] == 0:
        zero_targets_counter += 1
        if zero_targets_counter > num_one_targets:
            indices_to_remove.append(i)

unscaled_inputs_equal_priors = np.delete(unscaled_inputs_all, indices_to_remove, axis=0)
targets_equal_priors = np.delete(targets_all, indices_to_remove, axis=0)
```

- We count how many targets are 1 (meaning that the customer did buy again)
- And we set a counter for targets that are 0 (meaning that the customer did not buy again)



```
num_one_targets = int(np.sum(targets_all))  
zero_targets_counter = 0
```

```
indices_to_remove = []
```

```
for i in range(targets_all.shape[0]):  
    if targets_all[i] == 0:  
        zero_targets_counter += 1  
        if zero_targets_counter > num_one_targets:  
            indices_to_remove.append(i)
```

```
unscaled_inputs_equal_priors = np.delete(unscaled_inputs_all, indices_to_remove, axis=0)  
targets_equal_priors = np.delete(targets_all, indices_to_remove, axis=0)
```



- We want to create a "balanced" dataset, so we will have to remove some input/target pairs, and we declare a variable that will do that

```
num_one_targets = int(np.sum(targets_all))  
zero_targets_counter = 0
```

```
indices_to_remove = []
```

```
for i in range(targets_all.shape[0]):  
    if targets_all[i] == 0:  
        zero_targets_counter += 1  
        if zero_targets_counter > num_one_targets:  
            indices_to_remove.append(i)
```

```
unscaled_inputs_equal_priors = np.delete(unscaled_inputs_all, indices_to_remove, axis=0)  
targets_equal_priors = np.delete(targets_all, indices_to_remove, axis=0)
```

- We count the number of targets that are 0
- And once there are as many 0s as 1s, we mark entries where the target is 0

```
num_one_targets = int(np.sum(targets_all))  
zero_targets_counter = 0
```

```
indices_to_remove = []
```

```
for i in range(targets_all.shape[0]):  
    if targets_all[i] == 0:  
        zero_targets_counter += 1  
        if zero_targets_counter > num_one_targets:  
            indices_to_remove.append(i)
```


```
unscaled_inputs_equal_priors = np.delete(unscaled_inputs_all, indices_to_remove, axis=0)  
targets_equal_priors = np.delete(targets_all, indices_to_remove, axis=0)
```

```
num_one_targets = int(np.sum(targets_all))
zero_targets_counter = 0

indices_to_remove = []

for i in range(targets_all.shape[0]):
    if targets_all[i] == 0:
        zero_targets_counter += 1
        if zero_targets_counter > num_one_targets:
            indices_to_remove.append(i)
```

```
unscaled_inputs_equal_priors = np.delete(unscaled_inputs_all, indices_to_remove, axis=0)
targets_equal_priors = np.delete(targets_all, indices_to_remove, axis=0)
```

- 
- We create two new variables, one that will contain the inputs, and one that will contain the targets
  - And we delete all indices that we marked "to remove" in the loop above

# Standardizing the inputs

- That's the only place we use **sklearn** functionality  
We will take advantage of its preprocessing capabilities

```
scaled_inputs = preprocessing.scale(unscaled_inputs_equal_priors)
```

- It's a simple line of code, which standardizes the inputs
- At the end of your business case, you can try to run the algorithm **WITHOUT** this line of code  
The result will be interesting...

# Shuffle the data

- When the data was collected it was actually **arranged by date**
- We want to **shuffle the indices of the data**, so the data is not arranged in any way when we feed it

Since we will be batching, we want the data to be as randomly spread out as possible

```
shuffled_indices = np.arange(scaled_inputs.shape[0])  
np.random.shuffle(shuffled_indices)  
  
shuffled_inputs = scaled_inputs[shuffled_indices]  
shuffled_targets = targets_equal_priors[shuffled_indices]
```

We use the shuffled indices to shuffle the inputs and targets

NOTE: It also makes sense to shuffle the indices **prior** to balancing the dataset

# Splitting the data

```
samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)

test_samples_count = samples_count - train_samples_count - validation_samples_count

train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]

validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]

test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]

print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)
```



- We count the total number of samples

- We count the samples in each subset, assuming we want 80-10-10 distribution of training, validation, and test.

```
samples_count = shuffled_inputs.shape[0]
train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)
```

```
test_samples_count = samples_count - train_samples_count - validation_samples_count
```

```
train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]
```

```
validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]
```

```
test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]
```

```
print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)
```

- The 'test' dataset contains all remaining data

```
samples_count = shuffled_inputs.shape[0]
```

```
train_samples_count = int(0.8 * samples_count)
```

```
validation_samples_count = int(0.1 * samples_count)
```

```
test_samples_count = samples_count - train_samples_count - validation_samples_count
```

```
train_inputs = shuffled_inputs[:train_samples_count]
```

```
train_targets = shuffled_targets[:train_samples_count]
```

```
validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
```

```
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]
```

```
test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
```

```
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]
```

```
print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
```

```
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
```

```
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)
```

- We create variables that record the inputs and targets for training
- In our shuffled dataset, they are the first "train\_samples\_count" observations

```
samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)
```

```
test_samples_count = samples_count - train_samples_count - validation_samples_count
```

```
train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]
```

```
validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]
```

```
test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]
```

```
print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)
```

- We create variables that record the inputs and targets for validation
- They are the next "validation\_samples\_count" observations, following the "train\_samples\_count" we already assigned

```
samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)
```

```
test_samples_count = samples_count - train_samples_count - validation_samples_count
```

```
train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]
```

```
validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]
```

```
test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]
```

```
print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)
```

- We create variables that record the inputs and targets for test
- They are everything that is remaining

```
samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)
```

```
test_samples_count = samples_count - train_samples_count - validation_samples_count
```

```
train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]
```

```
validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]
```

```
test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]
```

```
print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)
```



- We print the number of targets that are 1s, the total number of samples, and the proportion for training, validation, and test.

```
samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)

test_samples_count = samples_count - train_samples_count - validation_samples_count

train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]

validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]

test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]

print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)
```

- The output should be close to 50% for all three



```

samples_count = shuffled_inputs.shape[0]

train_samples_count = int(0.8 * samples_count)
validation_samples_count = int(0.1 * samples_count)

test_samples_count = samples_count - train_samples_count - validation_samples_count

train_inputs = shuffled_inputs[:train_samples_count]
train_targets = shuffled_targets[:train_samples_count]

validation_inputs = shuffled_inputs[train_samples_count:train_samples_count+validation_samples_count]
validation_targets = shuffled_targets[train_samples_count:train_samples_count+validation_samples_count]

test_inputs = shuffled_inputs[train_samples_count+validation_samples_count:]
test_targets = shuffled_targets[train_samples_count+validation_samples_count:]

print(np.sum(train_targets), train_samples_count, np.sum(train_targets) / train_samples_count)
print(np.sum(validation_targets), validation_samples_count, np.sum(validation_targets) / validation_samples_count)
print(np.sum(test_targets), test_samples_count, np.sum(test_targets) / test_samples_count)

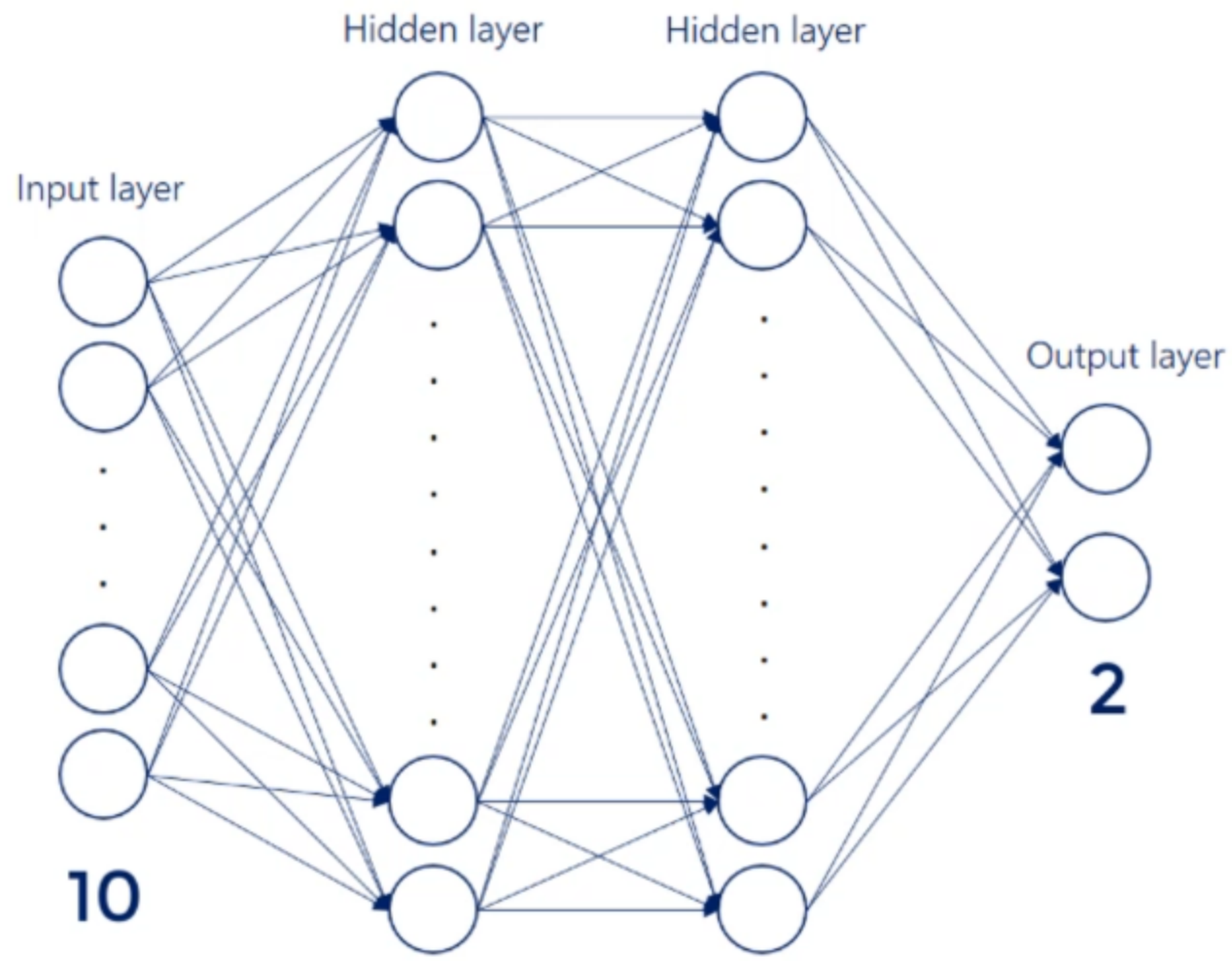
```

- We balanced our dataset to be 50-50 (for targets 0 and 1), but the training, validation, and test were taken from a shuffled dataset. Check if they are balanced, too
- Note that each time you rerun this code, you will get different values, as each time they are shuffled randomly
  - Normally you preprocess ONCE, so you need not rerun this code once it is done
  - If you rerun this whole sheet, the .npzs will be overwritten with your newly preprocessed data

# Saving in .npz format

- Finally, we save the three datasets in \*.npz format

```
np.savez('Audiobooks_data_train', inputs=train_inputs, targets=train_targets)
np.savez('Audiobooks_data_validation', inputs=validation_inputs, targets=validation_targets)
np.savez('Audiobooks_data_test', inputs=test_inputs, targets=test_targets)
```



In a separate file, we work on the model:

```
import numpy as np
import tensorflow as tf
```

```
npz = np.load('Audiobooks_data_train.npz')
```

```
train_inputs = npz['inputs'].astype(np.float)
```

```
train_targets = npz['targets'].astype(np.int)
```

```
npz = np.load('Audiobooks_data_validation.npz')
```

```
validation_inputs, validation_targets = npz['inputs'].astype(np.float), npz['targets'].astype(np.int)
```

```
npz = np.load('Audiobooks_data_test.npz')
```

```
test_inputs, test_targets = npz['inputs'].astype(np.float), npz['targets'].astype(np.int)
```

- For the model, we don't need "flatten", because the data has been processed already
- The model will very likely overfit!

You need to set up an early stopping mechanism

```
early_stopping = tf.keras.callbacks.EarlyStopping(patience=2)
```

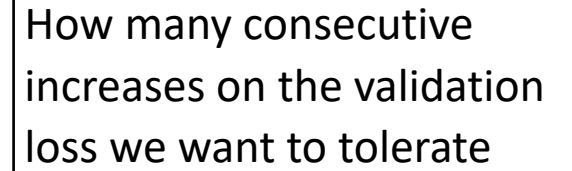
- There are a lot of interesting “**callbacks**” in TensorFlow, but here we need **EarlyStopping**

You can check here for other interesting features:

[https://www.tensorflow.org/api\\_docs/python/tf/keras/callbacks](https://www.tensorflow.org/api_docs/python/tf/keras/callbacks)



How many consecutive increases on the validation loss we want to tolerate



```
early_stopping = tf.keras.callbacks.EarlyStopping(patience=2)
```

```
model.fit(train_inputs,  
          train_targets,  
          batch_size=batch_size,  
          epochs=max_epochs,  
          callbacks=[early_stopping],  
          validation_data=(validation_inputs, validation_targets),  
          validation_steps=10,  
          verbose = 2  
          )
```

- Following these slides, you need to build a machine learning model to predict future audiobooks purchases. Furthermore, fiddling with the parameters, you will try to reach the higher accuracy possible.  
You can find the dataset for the business case on Canvas.
- Remember that the dataset needs to be processed as shown in class. However, you can try other ways if you want. Be sure to shuffle the data, though. Also, don't overfit the model!

### Submission:

- In "Submission - Abstract", you will submit the maximum accuracy reached against the test set, plus a brief description of the hyperparameters used (hidden layers, number of nodes, activation functions, etc..).
- In "Submission - Paper", you will submit an essay where you explain your reasoning and the performed tests.