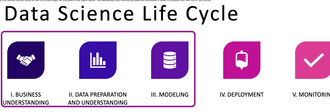
Developing AI tools – Supervised Classification

The second module of our **AI-Ready** course describes how to develop AI solutions for granular data. This tutorial leads the practice of development process with a **supervised classification** example in Python. The Data Science Life Cycle consists of the following five steps:

1. Business Understanding
2. Data Preparation & Understanding
3. Modeling
4. Deployment
5. Monitoring

In this tutorial, we focus on the first three steps, as stated in the figure below. The deployment and monitoring steps are discussed briefly in the conclusion section of this tutorial.



1. Business Understanding

Every A project should begin with a **business opportunity** to create value. Imagine that you are a cruise line company, operating a fleet of ships with which we market cruises to the general public. Unfortunately, cruise ships sometimes run into troubles that can ruin vacations. Remember the cruise ship Costa Concordia that capsized, capsized and sank after colliding with a rock?

Cruise companies want to avoid such negative publicity and want to ensure their customers have a memorable sailing. The price for this safe and no-incident travel is the same for everyone. We are now considering whether this is the right strategy or whether we can diversify the prices based on personal characteristics. Taking a data-driven approach, we decide to investigate the hypothesis that survival probability for passengers with different personal characteristics varies.

Our goal, stated as a predictive task, is therefore the following: **for each of cruise shooter, predict the survival probability of passengers based on their personal characteristics**. This allows us to spot whether price diversification would make sense or not, and how to diversify.

2. Data Preparation & Understanding

After defining the business goal, we can now start our data wrangling towards a solution. The first important step is to collect, prepare and understand **data** that is needed to solve our problem. We therefore need information from a cruise company that contains personal passenger information and whether they survived the disaster or not. The most famous cruise ship disaster in history is the sinking of the **Titanic**, so let's start from there.

Obtaining data

The first step is to actually find relevant data. The `Seaborn` package provides instant access to several datasets within Python. Luckily for us, there is a dataset available with information on the Titanic disaster in this package. The following steps are executed to obtain the data:

1. Install the package `seaborn` in your environment (`pip install seaborn`)
2. Load the function to read data from a source into Python (`sns`)
3. Assign the dataset to an object called (here `df`)

```
# load the library seaborn
(base) C:\Users\gmaat>
(base) C:\Users\gmaat>conda activate base
(base) C:\Users\gmaat>ipy
Python 3.7.6 (default, Jan 8, 2020, 19:59:22) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]: import seaborn as sns

In [2]: df = sns.load_dataset("titanic")
Downloading https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv
ssl.SSLCertificate
In [3]: df.shape
Out[3]: (891, 15)

In [4]: df.dtypes
Out[4]:
survived     int64
pclass       int64
sex         object
age        float64
sibsp        int64
parch        int64
fare       float64
embarked    object
class       object
who         object
adult_male    bool
deck        object
embark_town object
alive       object
alone        bool
dtype: object

In [5]: df.head(5)
Out[5]:
   survived  pclass     sex   age  sibsp  parch     fare embarked  class
0         0       3    male  22.0      1      0     7.25        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0    7.925        S  Third
3         1       1  female  35.0      1      0     53.1        S  First
4         0       3    male  35.0      0      0     8.05        S  Third

    who  adult_male deck  embark_town alive  alone
0   man        True  NaN  Southampton    no  False
1 woman       False    C    Cherbourg   yes  False
2 woman       False  NaN  Southampton   yes   True
3 woman       False    C  Southampton   yes  False
4   man        True  NaN  Southampton    no   True

In [6]: df.class.value_counts()
Out[6]:
Third    491
First    216
Second   184
Name: class, dtype: int64

In [7]: df.who.value_counts()
Out[7]:
man    537
woman  271
child  83
Name: who, dtype: int64

In [8]: df.deck.value_counts()
Out[8]:
C    59
B    47
D    33
E    32
A    15
F    13
G     4
Name: deck, dtype: int64

In [9]: df.embark_town.value_counts()
Out[9]:
Southampton    644
Cherbourg      168
Queenstown      77
Name: embark_town, dtype: int64

In [10]: df.alive.value_counts()
Out[10]:
no    549
yes   342
Name: alive, dtype: int64

In [11]: df.describe()
Out[11]:
         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 [12]: df.describe(include=['object'])
Out[12]:
        sex embarked  class  who deck  embark_town alive
count   891      889    891  891  203          889   891
unique    2        3      3    3    7            3     2
top    male        S  Third  man    C  Southampton    no
freq    577      644    491  537   59          644   549

In [13]: df.describe(include=['bool'])
Out[13]:
       adult_male  alone
count         891    891
unique          2      2
top         False   True
freq          537    537

In [14]: df.describe(include=['category'])
Out[14]:
Empty DataFrame
Columns: []
Index: []

In [15]: df.describe(include=['object','bool'])
Out[15]:
        sex embarked  class  who  adult_male deck  embark_town alive  alone
count   891      889    891  891         891  203          889   891    891
unique    2        3      3    3           2    7            3     2      2
top    male        S  Third  man       False    C  Southampton    no   True
freq    577      644    491  537         537   59          644   549    537

In [16]: df.describe(include=['int64'])
Out[16]:
       survived      pclass       sibsp       parch
count  891.000000  891.000000  891.000000  891.000000
mean     0.383838    2.308642    0.523008    0.381594
std      0.486592    0.836071    1.102743    0.806057
min      0.000000    1.000000    0.000000    0.000000
25%      0.000000    2.000000    0.000000    0.000000
50%      0.000000    3.000000    0.000000    0.000000
75%      1.000000    3.000000    1.000000    0.000000
max      1.000000    3.000000    8.000000    6.000000

In [17]: df.describe(include=['float64'])
Out[17]:
              age       fare
count  714.000000  891.000000
mean    29.699118   32.204208
std     14.526497   49.693429
min      0.420000    0.000000
25%     20.125000    7.910400
50%     28.000000   14.454200
75%     38.000000   31.000000
max     80.000000  512.329200

In [18]: df.describe(include=['int64','float64'])
Out[18]:
       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
```

Find each column and how they fail are how the data, we can have a first look at it by inspecting the head object.

```
In [19]: df.head(5)
Out[19]:
   survived  pclass     sex   age  sibsp  parch     fare embarked  class
0         0       3    male  22.0      1      0     7.25        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0    7.925        S  Third
3         1       1  female  35.0      1      0     53.1        S  First
4         0       3    male  35.0      0      0     8.05        S  Third

    who  adult_male deck  embark_town alive  alone
0   man        True  NaN  Southampton    no  False
1 woman       False    C    Cherbourg   yes  False
2 woman       False  NaN  Southampton   yes   True
3 woman       False    C  Southampton   yes  False
4   man        True  NaN  Southampton    no   True
```

Find each column and how they fail are how the data, we can have a first look at it by inspecting the head object.

```
In [20]: df.dtypes
Out[20]:
survived     int64
pclass       int64
sex         object
age        float64
sibsp        int64
parch        int64
fare       float64
embarked    object
class       object
who         object
adult_male    bool
deck        object
embark_town object
alive       object
alone        bool
dtype: object
```

Finally, the last step is to inspect the column types. Pandas distinguishes (now) with four characteristics (column), class, type, and subclass. We can already guess what these means, but let's have a closer look at them anyway.

Inspect the data

We first want to know the number of unique values that each of the column contains. The output below indicates that class contains three unique values, while the other three (sex, who and survived) contain two unique values.

```
In [21]: df.sex.value_counts
Out[21]:
male    577
female  314
Name: sex, dtype: int64

In [22]: df.who.value_counts
Out[22]:
man    537
woman  271
child  83
Name: who, dtype: int64

In [23]: df.survived.value_counts
Out[23]:
0    549
1    342
Name: survived, dtype: int64
```

Knowing this, we would like to figure out what these unique values actually represent. The output below shows the unique values for each of the columns.

```
In [24]: df.class.value_counts
Out[24]:
Third    491
First    216
Second   184
Name: class, dtype: int64

In [25]: df.pclass.value_counts
Out[25]:
3    491
1    216
2    184
Name: pclass, dtype: int64

In [26]: df.embarked.value_counts
Out[26]:
S    644
C    168
Q     77
Name: embarked, dtype: int64

In [27]: df.embark_town.value_counts
Out[27]:
Southampton    644
Cherbourg      168
Queenstown      77
Name: embark_town, dtype: int64

In [28]: df.deck.value_counts
Out[28]:
C    59
B    47
D    33
E    32
A    15
F    13
G     4
Name: deck, dtype: int64
```

Remove and high quality data, that's great! Let's gain some insights in the data first.

Univariate exploration

The univariate exploration focuses on the distribution of the different columns. Frequency tables count the number of occurrences for each of the unique values in a column. Below we show the frequency table for each of the columns in the Titanic data. There seems for example 708 passengers in Southampton, 577 male and 891 in total. Also passengers seem young (714 values), about 100 children and more males (577) than females (314). We furthermore observe that only 1886 passengers survived while 549 did not, showing the tragedy of the Titanic's sinking.

```
In [29]: df.class.value_counts
Out[29]:
Third    491
First    216
Second   184
Name: class, dtype: int64

In [30]: df.who.value_counts
Out[30]:
man    537
woman  271
child  83
Name: who, dtype: int64

In [31]: df.deck.value_counts
Out[31]:
C    59
B    47
D    33
E    32
A    15
F    13
G     4
Name: deck, dtype: int64

In [32]: df.embark_town.value_counts
Out[32]:
Southampton    644
Cherbourg      168
Queenstown      77
Name: embark_town, dtype: int64

In [33]: df.alive.value_counts
Out[33]:
no    549
yes   342
Name: alive, dtype: int64

In [34]: df.describe(include=['object'])
Out[34]:
        sex embarked  class  who deck  embark_town alive
count   891      889    891  891  203          889   891
unique    2        3      3    3    7            3     2
top    male        S  Third  man    C  Southampton    no
freq    577      644    491  537   59          644   549

In [35]: df.describe(include=['bool'])
Out[35]:
       adult_male  alone
count         891    891
unique          2      2
top         False   True
freq          537    537

In [36]: df.describe(include=['category'])
Out[36]:
Empty DataFrame
Columns: []
Index: []

In [37]: df.describe(include=['object','bool'])
Out[37]:
        sex embarked  class  who  adult_male deck  embark_town alive  alone
count   891      889    891  891         891  203          889   891    891
unique    2        3      3    3           2    7            3     2      2
top    male        S  Third  man       False    C  Southampton    no   True
freq    577      644    491  537         537   59          644   549    537

In [38]: df.describe(include=['int64'])
Out[38]:
       survived      pclass       sibsp       parch
count  891.000000  891.000000  891.000000  891.000000
mean     0.383838    2.308642    0.523008    0.381594
std      0.486592    0.836071    1.102743    0.806057
min      0.000000    1.000000    0.000000    0.000000
25%      0.000000    2.000000    0.000000    0.000000
50%      0.000000    3.000000    0.000000    0.000000
75%      1.000000    3.000000    1.000000    0.000000
max      1.000000    3.000000    8.000000    6.000000

In [39]: df.describe(include=['float64'])
Out[39]:
              age       fare
count  714.000000  891.000000
mean    29.699118   32.204208
std     14.526497   49.693429
min      0.420000    0.000000
25%     20.125000    7.910400
50%     28.000000   14.454200
75%     38.000000   31.000000
max     80.000000  512.329200

In [40]: df.describe(include=['int64','float64'])
Out[40]:
       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
```

Below we show how the bar chart looks like for the variable `survived`. Remember that the variable survived contains dummy variables with 1 for survival and 0 for not survived. The mean of this variable therefore shows the proportion of survival. The bar chart shows that more people did not survive than those who did survive.

```
In [41]: df.survived.mean
Out[41]:
0.3838383838383838

In [42]: df.survived.plot.bar()
Out[42]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [43]: df.survived.plot.hist()
Out[43]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [44]: df.age.plot.hist()
Out[44]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [45]: df.fare.plot.hist()
Out[45]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [46]: df.sex.value_counts.plot.bar()
Out[46]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [47]: df.class.value_counts.plot.bar()
Out[47]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [48]: df.who.value_counts.plot.bar()
Out[48]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [49]: df.deck.value_counts.plot.bar()
Out[49]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [50]: df.embark_town.value_counts.plot.bar()
Out[50]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [51]: df.alive.value_counts.plot.bar()
Out[51]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

The univariate analysis could also create insights into the distribution of the different variables. Frequency tables count the number of occurrences for each of the unique values in a column. Below we show the frequency table for each of the columns in the Titanic data. There seems for example 708 passengers in Southampton, 577 male and 891 in total. Also passengers seem young (714 values), about 100 children and more males (577) than females (314). We furthermore observe that only 1886 passengers survived while 549 did not, showing the tragedy of the Titanic's sinking.

```
In [52]: df.survived.mean
Out[52]:
0.3838383838383838

In [53]: df.survived.plot.bar()
Out[53]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [54]: df.survived.plot.hist()
Out[54]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [55]: df.age.plot.hist()
Out[55]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [56]: df.fare.plot.hist()
Out[56]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [57]: df.sex.value_counts.plot.bar()
Out[57]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [58]: df.class.value_counts.plot.bar()
Out[58]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [59]: df.who.value_counts.plot.bar()
Out[59]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [60]: df.deck.value_counts.plot.bar()
Out[60]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [61]: df.embark_town.value_counts.plot.bar()
Out[61]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [62]: df.alive.value_counts.plot.bar()
Out[62]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

Multivariate exploration

So far we focused on univariate exploration of the data, which means only considering one column at a time. It can also be interesting to look at multiple columns simultaneously in a multivariate way. The very popular `pandas` package for data analysis in Python comes in handy here.

```
In [63]: df.survived.mean
Out[63]:
0.3838383838383838

In [64]: df.survived.plot.bar()
Out[64]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [65]: df.survived.plot.hist()
Out[65]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [66]: df.age.plot.hist()
Out[66]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [67]: df.fare.plot.hist()
Out[67]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [68]: df.sex.value_counts.plot.bar()
Out[68]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [69]: df.class.value_counts.plot.bar()
Out[69]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [70]: df.who.value_counts.plot.bar()
Out[70]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [71]: df.deck.value_counts.plot.bar()
Out[71]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [72]: df.embark_town.value_counts.plot.bar()
Out[72]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [73]: df.alive.value_counts.plot.bar()
Out[73]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

Below we provide a few examples of such a cross table. The first one shows the survival rate for each class. We see that survival rate is 63% for the first class passengers, 47% for second class passengers and only 24% for third class passengers. This shows that survival rate is not equally distributed across the different classes. The second cross table shows the survival rate for different combinations of class and gender. This indicates that the difference in male versus female passengers was much smaller in first class than in third class.

```
In [74]: df.class.value_counts.plot.bar()
Out[74]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [75]: df.who.value_counts.plot.bar()
Out[75]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [76]: df.deck.value_counts.plot.bar()
Out[76]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [77]: df.embark_town.value_counts.plot.bar()
Out[77]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [78]: df.alive.value_counts.plot.bar()
Out[78]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>

In [79]: df.sex.value_counts.plot.bar()
Out[79]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [80]: df.class.value_counts.plot.bar()
Out[80]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [81]: df.who.value_counts.plot.bar()
Out[81]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [82]: df.deck.value_counts.plot.bar()
Out[82]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [83]: df.embark_town.value_counts.plot.bar()
Out[83]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [84]: df.alive.value_counts.plot.bar()
Out[84]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

Empirical relation of the features to the target

So far we focused on univariate and multivariate analysis of the target and features. Where it really interesting is to discover the empirical relation between the features and the target. This is the pattern that you want to capture in an AI. model, so let's have a look!

The target is a binary variable, indicating whether the passenger survived or not. The proportion of "yes" over the whole data therefore measures the overall survival probability (empirical probability). Survival probability appears to be only 37.87% on average. But we already know this from before, remember?

```
In [85]: df.survived.mean
Out[85]:
0.3838383838383838

In [86]: df.survived.plot.bar()
Out[86]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [87]: df.survived.plot.hist()
Out[87]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [88]: df.age.plot.hist()
Out[88]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [89]: df.fare.plot.hist()
Out[89]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [90]: df.sex.value_counts.plot.bar()
Out[90]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [91]: df.class.value_counts.plot.bar()
Out[91]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [92]: df.who.value_counts.plot.bar()
Out[92]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [93]: df.deck.value_counts.plot.bar()
Out[93]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [94]: df.embark_town.value_counts.plot.bar()
Out[94]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [95]: df.alive.value_counts.plot.bar()
Out[95]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

We therefore have extracted the target variable and stored it in the object `y`. This is a vector (pandas Series) of length 1046. The features are stored in object `X`. This is a matrix (pandas DataFrame) with 1046 rows and 3 columns, one for each feature.

```
In [96]: df.survived.mean
Out[96]:
0.3838383838383838

In [97]: df.survived.plot.bar()
Out[97]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [98]: df.survived.plot.hist()
Out[98]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [99]: df.age.plot.hist()
Out[99]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [100]: df.fare.plot.hist()
Out[100]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [101]: df.sex.value_counts.plot.bar()
Out[101]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [102]: df.class.value_counts.plot.bar()
Out[102]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [103]: df.who.value_counts.plot.bar()
Out[103]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [104]: df.deck.value_counts.plot.bar()
Out[104]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [105]: df.embark_town.value_counts.plot.bar()
Out[105]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [106]: df.alive.value_counts.plot.bar()
Out[106]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

We can now look and for this is more dimensions. Let's calculate the survival probability based on the combination of age and gender. The table below indicates for example that female adults had a survival probability of 73.85% and male children had 45.25% of surviving.

```
In [107]: df.survived.mean
Out[107]:
0.3838383838383838

In [108]: df.survived.plot.bar()
Out[108]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [109]: df.survived.plot.hist()
Out[109]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [110]: df.age.plot.hist()
Out[110]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [111]: df.fare.plot.hist()
Out[111]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [112]: df.sex.value_counts.plot.bar()
Out[112]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [113]: df.class.value_counts.plot.bar()
Out[113]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [114]: df.who.value_counts.plot.bar()
Out[114]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [115]: df.deck.value_counts.plot.bar()
Out[115]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [116]: df.embark_town.value_counts.plot.bar()
Out[116]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [117]: df.alive.value_counts.plot.bar()
Out[117]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

3. Modeling

Data, both e.g. cruises as an AI. completely done with the data yet. Some preparations for our modeling phase is still necessary.

Target and features

We first separate the target and features. The target is stored in the object `y`. This is a vector (pandas Series) of length 1046. The features are stored in object `X`. This is a matrix (pandas DataFrame) with 1046 rows and 3 columns, one for each feature.

```
In [118]: df.survived.mean
Out[118]:
0.3838383838383838

In [119]: df.survived.plot.bar()
Out[119]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [120]: df.survived.plot.hist()
Out[120]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [121]: df.age.plot.hist()
Out[121]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [122]: df.fare.plot.hist()
Out[122]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [123]: df.sex.value_counts.plot.bar()
Out[123]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [124]: df.class.value_counts.plot.bar()
Out[124]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [125]: df.who.value_counts.plot.bar()
Out[125]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [126]: df.deck.value_counts.plot.bar()
Out[126]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [127]: df.embark_town.value_counts.plot.bar()
Out[127]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [128]: df.alive.value_counts.plot.bar()
Out[128]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

Feature engineering

We start with the target vector `y`. In supervised binary classification problems, it is customary to indicate the positive class that you want to predict as 1 and the negative class as 0. In our survival example this means transforming `yes` to `1` and `no` to `0`. We save the result in the object `y_binary`.

```
In [129]: df.survived.mean
Out[129]:
0.3838383838383838

In [130]: df.survived.plot.bar()
Out[130]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [131]: df.survived.plot.hist()
Out[131]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [132]: df.age.plot.hist()
Out[132]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [133]: df.fare.plot.hist()
Out[133]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [134]: df.sex.value_counts.plot.bar()
Out[134]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [135]: df.class.value_counts.plot.bar()
Out[135]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [136]: df.who.value_counts.plot.bar()
Out[136]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [137]: df.deck.value_counts.plot.bar()
Out[137]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [138]: df.embark_town.value_counts.plot.bar()
Out[138]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [139]: df.alive.value_counts.plot.bar()
Out[139]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

We furthermore know what when we represent class as a numerical feature, just like age and sex. Mathematic operations may not be meaningful of features to do (e.g. average) (sex) the mean (male) (female) and sex (male) (female) is meaningful. Nominal features, like sex, can be represented by dummy variables, or one-hot encoding. Ordinal features, like class, can be represented by integers, such as 1 for first class, 2 for second class, and 3 for third class. Mathematic operations may not be meaningful of features to do (e.g. average) (class) the mean (first class) (second class) and class (third class) is meaningful. Ordinal features, like class, can also be represented by dummy variables, or one-hot encoding. Continuous features, like age, can be represented by integers, such as 1 for 1 year old, 2 for 2 years old, and 3 for 3 years old. Mathematic operations may not be meaningful of features to do (e.g. average) (age) the mean (1 year old) (2 years old) and age (3 years old) is meaningful.

```
In [140]: df.survived.mean
Out[140]:
0.3838383838383838

In [141]: df.survived.plot.bar()
Out[141]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [142]: df.survived.plot.hist()
Out[142]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [143]: df.age.plot.hist()
Out[143]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [144]: df.fare.plot.hist()
Out[144]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [145]: df.sex.value_counts.plot.bar()
Out[145]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [146]: df.class.value_counts.plot.bar()
Out[146]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [147]: df.who.value_counts.plot.bar()
Out[147]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [148]: df.deck.value_counts.plot.bar()
Out[148]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [149]: df.embark_town.value_counts.plot.bar()
Out[149]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [150]: df.alive.value_counts.plot.bar()
Out[150]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

Some of the features that we have, contains 100 observations (sex) with four characteristics (column), class, age, sex and survived. We can already guess what these means, but let's have a closer look at them anyway.

Inspect the data

We first want to know the number of unique values that each of the column contains. The output below indicates that class contains three unique values, while the other three (age, sex and survived) contain two unique values.

```
In [151]: df.survived.mean
Out[151]:
0.3838383838383838

In [152]: df.survived.plot.bar()
Out[152]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [153]: df.survived.plot.hist()
Out[153]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [154]: df.age.plot.hist()
Out[154]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [155]: df.fare.plot.hist()
Out[155]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [156]: df.sex.value_counts.plot.bar()
Out[156]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [157]: df.class.value_counts.plot.bar()
Out[157]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [158]: df.who.value_counts.plot.bar()
Out[158]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [159]: df.deck.value_counts.plot.bar()
Out[159]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [160]: df.embark_town.value_counts.plot.bar()
Out[160]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [161]: df.alive.value_counts.plot.bar()
Out[161]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

Knowing this, we would like to figure out what these unique values actually represent. The output below shows the unique values for each of the columns.

```
In [162]: df.survived.mean
Out[162]:
0.3838383838383838

In [163]: df.survived.plot.bar()
Out[163]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [164]: df.survived.plot.hist()
Out[164]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [165]: df.age.plot.hist()
Out[165]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [166]: df.fare.plot.hist()
Out[166]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [167]: df.sex.value_counts.plot.bar()
Out[167]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [168]: df.class.value_counts.plot.bar()
Out[168]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [169]: df.who.value_counts.plot.bar()
Out[169]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3aa0>

In [170]: df.deck.value_counts.plot.bar()
Out[170]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa4ab0>

In [171]: df.embark_town.value_counts.plot.bar()
Out[171]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa5ac0>

In [172]: df.alive.value_counts.plot.bar()
Out[172]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa6ad0>
```

Remove and high quality data, that's great! Let's gain some insights in the data first.

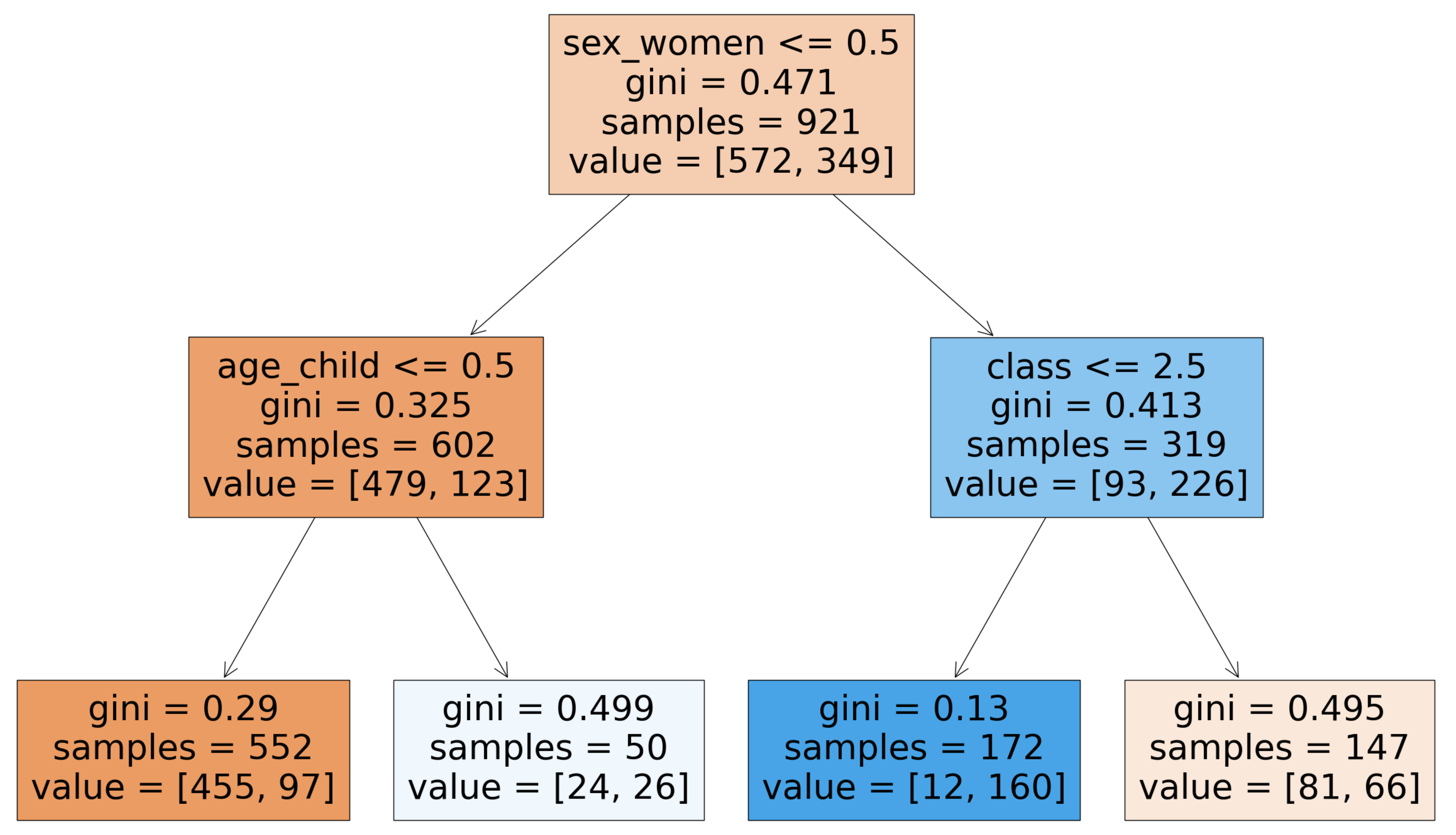
Univariate exploration

The univariate exploration focuses on the distribution of the different columns. Frequency tables count the number of occurrences for each of the unique values in a column. Below we show the frequency table for each of the columns in the Titanic data. There seems for example 708 passengers in Southampton, 577 male and 891 in total. Also passengers seem young (714 values), about 100 children and more males (577) than females (314). We furthermore observe that only 1886 passengers survived while 549 did not, showing the tragedy of the Titanic's sinking.

```
In [173]: df.survived.mean
Out[173]:
0.3838383838383838

In [174]: df.survived.plot.bar()
Out[174]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a7a0>

In [175]: df.survived.plot.hist()
Out[175]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f8a8d0>

In [176]: df.age.plot.hist()
Out[176]:
<matplotlib.axes._subplots.AxesSubplot at 0x1f9a9e0>

In [177]: df.fare.plot.hist()
Out[177]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa0a70>

In [178]: df.sex.value_counts.plot.bar()
Out[178]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa1a80>

In [179]: df.class.value_counts.plot.bar()
Out[179]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2a90>

In [180]: df.who.value_counts.plot.bar()
Out[180]:
<matplotlib.axes._subplots.AxesSubplot at 0x1fa3a
```

The tree structure is very easily visualized to show the underlying decision process.

In [30]:
```
# Visualize the simple decicion tree
# Import the 'tree' module from the package 'sklearn'
from sklearn import tree
# Plot the tree structure using the 'plot_tree()' function
plt.figure(figsize = (30,20))
tree.plot_tree(simple_tree, feature_names = Xo_train.columns, filled = True)
plt.show()
```



The first split checks whether sex_woman <= 0.5 is true (go left) or false (go right). This means that males (sex_woman = 0) go to the left of the tree and females (sex_woman = 1) go to the right of the tree.

The next split on the left side of the tree checks whether age_child <= 0.5 is true (go left) or false (go right). This means that male adults (age_child = 0) go to the left of the node and male children (age_child = 1) go to the right of the node. The leftmost bottom node therefore collects all 552 male adults in our data of which 455 did not survive and 97 did. We therefore predict that male adults have a survival probability of 97/552, approximately equal to 18%. The same reasoning allows us to conclude that male children have a survival probability of 26/50 or 52%.

Following the same process on the right side of the tree therefore shows four possible survival probabilities:

- IF man \& adult THEN 18\%
- IF man \& child THEN 52\%
- IF woman \& 1st or 2nd class THEN 93\%
- IF woman \& 3rd class THEN 45\%

We can also assess each feature's predictive power by calculating the so-called feature importance metric. Below we show how 'sex' explains 69% of the predictive power, 'class' 24% and 'age' only 7%. For this tree this is not super informative, but this measure becomes really helpful when working with big complex black box models.

In [31]:
```
# Inspect feature importance values
pd.DataFrame(data = simple_tree.feature_importances_,
             index = Xo_train.columns,
             columns = ['importance']).sort_values('importance', ascending = False)
```

Out[31]:

	importance
sex_women	0.690166
class	0.239066
age_child	0.070768

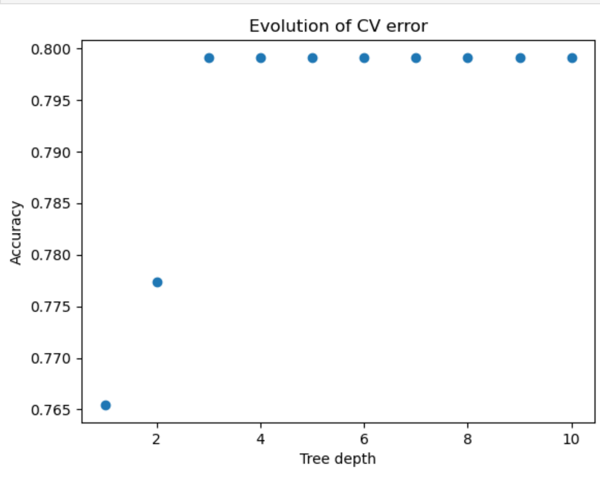
Optimal regression tree

Let's now fit a slightly more complicated decision tree. We allow the tree to take a maximum depth of 10 and will search for the optimal tree depth. How do we do that? This is where the **validation** data comes in. We use part of the training data to evaluate all depth options and choose the depth that leads to the best performance. For now we define performance as the classification **accuracy** but we come back to this later. *Sidenote: we use a more complicated setup of cross-validation (instead of one simple train-validation split) which has some advantages, but the implementation details are not a priority here.*

In [32]:
```
# Perform cross-validation to assess different tree depth options
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score

cv_grid = GridSearchCV(DecisionTreeClassifier(random_state = 1),
                       param_grid = {'max_depth':(1,2,3,4,5,6,7,8,9,10)},
                       scoring = make_scorer(accuracy_score, greater_is_better = True),
                       cv = 5, refit = True).fit(Xo_train, y_train)
```

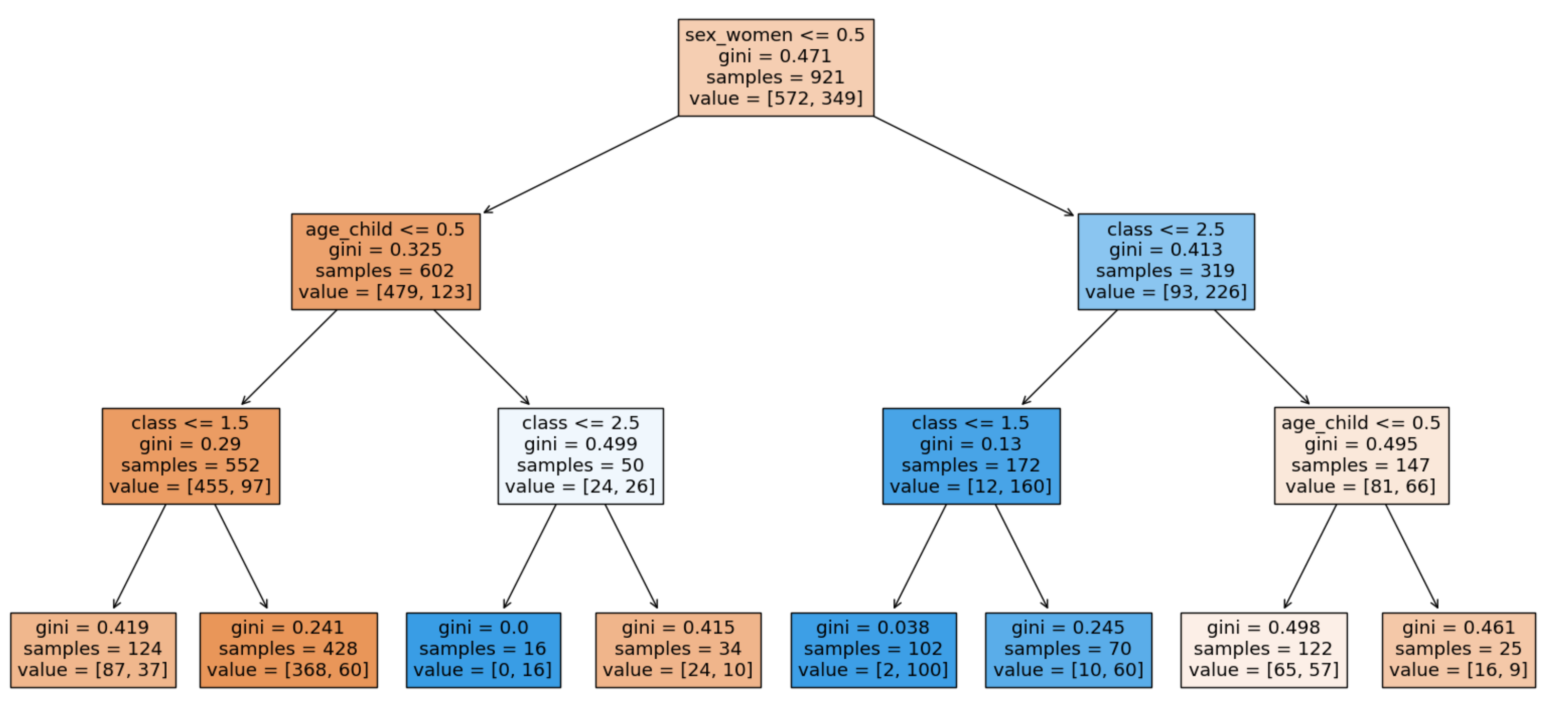
Below we visualize the accuracy performance on our validation data for each tree depth. It is clear that accuracy improves from depth 1 to 2 and from depth 2 to 3. The accuracy stays constant for depths higher than 3 because of our small limited dataset. Typically this accuracy would drop after a while because the model starts overfitting. However, in our case, we can now safely choose a tree of depth 3 as optimal model.

In [33]:
```
# Visualize the accuracy performance for each tree depth
plt.scatter(x = [1,2,3,4,5,6,7,8,9,10],
            y = cv_grid.cv_results_.get('mean_test_score'))
plt.title('Evolution of CV error')
plt.xlabel('Tree depth')
plt.ylabel('Accuracy')
plt.show()
```



The optimal tree of depth 3 is visualized below. The underlying decision process can be unraveled in the exact same way as before.

In [34]:
```
# Visualize the optimal decicion tree
plt.figure(figsize = (20,10))
best_tree = cv_grid.best_estimator_
tree.plot_tree(best_tree, feature_names = Xo_train.columns, filled = True)
plt.show()
```



This optimal tree shows eight possible survival probabilities:

- IF man \& adult \& 1st class THEN 30\%
- IF man \& adult \& not 1st class THEN 14\%
- IF man \& child \& not 3rd class THEN 100\%
- IF man \& child \& 3rd class THEN 29\%
- IF woman \& 1st class THEN 98\%
- IF woman \& 2nd class THEN 86\%
- IF woman \& 3rd class & child THEN 47\%
- IF woman \& 3rd class & adult THEN 36\%

The feature importance scores show that 'class' has gained some predictive performance in favor of 'sex' (which is still by far the most important feature).

In [35]:
```
# Inspect feature importance values
pd.DataFrame(data = best_tree.feature_importances_,
             index = Xo_train.columns,
             columns = ['importance']).sort_values('importance', ascending = False)
```

Out[35]:

	importance
sex_women	0.619953
class	0.313688
age_child	0.066359

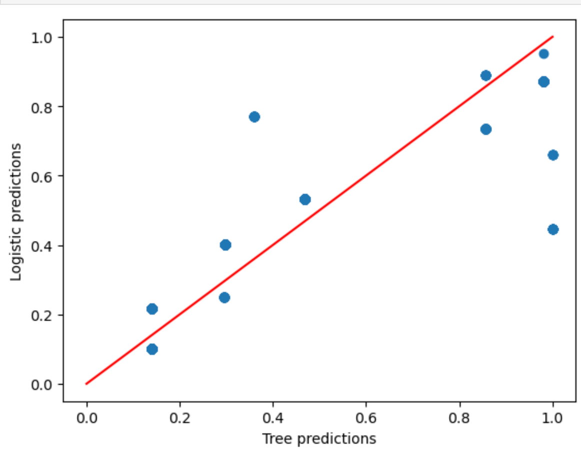
Model comparison

We now have two models which we can compare, namely the logistic regression model and the optimal decision tree of depth 3. For both models we calculate the survival probabilities for each possible passenger profile. This allows us to spot differences in the underlying decision process of each model.

In [36]:
```
# Calculate the survival probabilities for each model
proba_tree = best_tree.predict_proba(Xo_train)[:,1]
proba_logr = log_reg.predict(sm.add_constant(Xd_train))
```

Below we show a scatterplot of the survival probabilities with those of the decision tree on the x-axis and those of the logistic regression on the y-axis. Each blue point is therefore plotted at the coordinates (tree prediction, logistic prediction) and the red line indicates locations where the tree and logistic predictions are equal. In total there are 12 (3 x 2 x 2) possible passenger profiles, each represented by a blue dot. Most blue points are situated around the red line, indicating that predictions are very similar in general. There are however some profiles where differences are substantial.

In [37]:
```
# Scatterplot of the probabilities
plt.scatter(proba_tree, proba_logr)
plt.plot([0,1], [0,1], 'r')
plt.xlabel('Tree predictions')
plt.ylabel('Logistic predictions')
plt.show()
```



The table below details those differences between the tree and logistic predictions for the 12 passenger profiles. We only highlight a couple:

- The tree assigns a survival probability of 14% to adult males in 2nd and 3rd class, while the logistic regression diversifies further (2nd class: 22% and 3rd class: 10%).
- The tree assigns a survival probability of 36% to female children in 3rd class, while the logistic regression estimates survival chances of 77%.
- The tree assigns a survival probability of 100% to male children in 1st and 2nd class, while the logistic regression predicts survival at only 66% and 45% respectively. This is because all these profiles in the training data survived the disaster, leading the decision tree to a very optimistic estimate.

This shows that there are some differences to be found between both models, so which one performs best?

In [38]:
```
# Profiles with predictions
Xo_train.assign(proba_tree = proba_tree, proba_logr = proba_logr).drop_duplicates().sort_values('proba_logr')
```

Out[38]:

	age_child	sex_women	class	proba_tree	proba_logr
688	0	0	3	0.140187	0.101985
365	0	0	2	0.140187	0.217063
1274	1	0	3	0.294118	0.248881
50	0	0	1	0.298387	0.401387
592	1	0	2	1.000000	0.447174
1164	0	1	3	0.467213	0.533417
324	1	0	1	1.000000	0.661745
556	0	1	2	0.857143	0.736210
1294	1	1	3	0.360000	0.769349
287	0	1	1	0.980392	0.870967
603	1	1	2	0.857143	0.890624
325	1	1	1	0.980392	0.951676

Model evaluation

Let's now evaluate these two models on the **test** data to assess the predictive generalization performance towards unseen data. We start by calculating the predicted survival classes (survived yes or no) on the test data.

In [39]:
```
# Calculate survival classes
class_tree = best_tree.predict(Xo_test)
class_logr = (log_reg.predict(sm.add_constant(Xd_test)) > 0.5).astype(int)
```

We now have the predicted survival class for each model on the test data and we know the actual observed target for the test data. Let's first calculate the **accuracy** for both models, defined as the number of correct classifications relative to all predictions: $accuracy = \frac{\#correct}{\#total}$. The difference in accuracy is very small, with the decision tree achieving 79.49% and the logistic regression 79.24%, indicating that both models perform similarly well.

In [40]:
```
# Calculate accuracy
from sklearn.metrics import accuracy_score
print('Test accuracy decision tree:', accuracy_score(y_test, class_tree))
print('Test accuracy logistic regression:', accuracy_score(y_test, class_logr))
print('\nManual checks:')
print('DT: ', sum(y_test == class_tree) / len(y_test))
print('LR: ', sum(y_test == class_logr) / len(y_test))
```

```
Test accuracy decision tree: 0.7949367088607595
Test accuracy logistic regression: 0.7924050632911392

Manual checks:
DT:  0.7949367088607595
LR:  0.7924050632911392
```

Accuracy is however not always the best measure to use, especially when your target variable is highly skewed to mainly having either positive or negative classes. Our example contains a 60-40 distribution which is fairly balanced. But let's have a look at the **confusion matrix** which has the following format in scikit-learn with true/false (T/P) negatives/positives (N/P) respectively:

conf.	matrix
TN	FP
FN	TP

From this we observe that the decision tree has more TNs while logistic regression has more TPs. Or in other words, the decision tree has more FNs while logistic regression has more FPs. The decision tree produces more negative predictions while logistic regression produces more positive predictions.

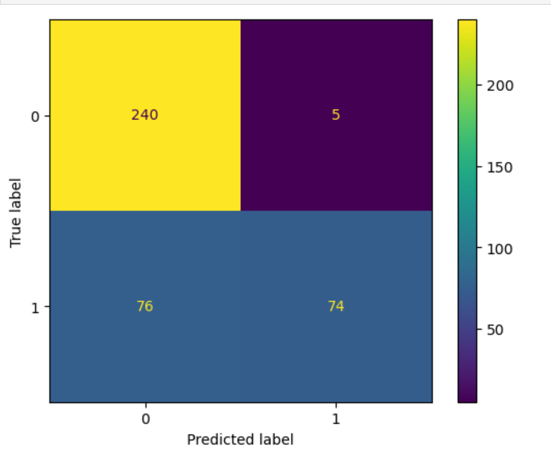
Which of these you should prefer depends on the use case at hand. What is more costly, predicting a positive case as negative or the other way around? Think about medical diagnosis where you really want to avoid a "no disease" prediction for someone who is sick because this will result in no treatment and possibly to the patient's death.

In [41]:
```
# Calculate confusion matrices
from sklearn.metrics import confusion_matrix
print('Confusion matrix decision tree:\n', confusion_matrix(y_test, class_tree))
print('Confusion matrix logistic regression:\n', confusion_matrix(y_test, class_logr))
```

```
Confusion matrix decision tree:
 [[240   5]
 [ 76  74]]
Confusion matrix logistic regression:
 [[215  30]
 [ 52  98]]
```

We can also visualize the confusion matrix to make the output more appealing to the human eye.

In [42]:
```
# Plot DT confusion matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(best_tree, Xo_test, y_test)
plt.show()
```



From the confusion matrix we can calculate several metrics:

- **recall**: proportion of correctly predicted positive instances among all positive instances, TP / (TP + FN)
- **precision**: proportion of correctly predicted positive instances among all instances predicted as positive, TP / (TP + FP)
- **F-score**: combines recall and precision in one number, 2 x (P x R) / (P + R)

The logistic regression model attains a higher recall, indicating that it is better at finding the positive cases. The decision tree attains a higher precision, indicating that such a positive prediction can be trusted more. The F-score prefers the logistic regression model over the decision tree.

In [43]:
```
# Calculate recall, precision and F-score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
print('Recall:\n')
print('DT:', recall_score(y_test, class_tree))
print('LR :', recall_score(y_test, class_logr))
print('\n')
print('Precision:\n')
print('DT:', precision_score(y_test, class_tree))
print('LR :', precision_score(y_test, class_logr))
print('\n')
print('F-score:\n')
print('DT:', f1_score(y_test, class_tree))
print('LR :', f1_score(y_test, class_logr))
```

```
Recall:

DT: 0.49333333333333335
LR : 0.6533333333333333


Precision:

DT: 0.9367088607594937
LR : 0.765625


F-score:

DT: 0.646288209606987
LR : 0.7050359712230216
```

This analysis shows that different metrics can prefer different models in the end, so it is important to clearly define your business goals in detail and align the proper metrics to optimize your AI solution.

4. Deployment & 5. Monitoring

Once you picked a model it is time to leave the data scientist's playground and put that model in production where it can start delivering actual business value. We discuss deployment and monitoring in modules 4 and 5 of the AI4Business course, so stay tuned!

PROPHECY LABS   ctg   nannyML

AI4Business

ESF INVESTEERT IN JOUW TOEKOMST   Europese Unie   Vlaanderen is werk