Aleix Sicília
Practical case in (*Data science*)
Advanced data mining

Introduction

The aim of this report is to outline the importance of managing and comprehending data in real-world projects. In a very short term, it is highly plausible that many AI developments could influence in our day-to-day decisions. Hence, it seems far-fetched that we do not understand how these systems are designed per se.

In the upcoming future, it could give rise to plenty of hefty challenges. Be that as it may, it could sow the seeds of other cons. To name but a few, inequalities could rise, roles propagation could be perpetuated or social bipolarity could be extended.

Thus, it happens to be significant that we must interpret and understand their applications. My insight is that there are some easy-to-interpret algorithms such as decision trees. Albeit, we must give prominence to the fact that other methods such as Decision Gradient are a black box (they could run smoothly with a great precision. However, there is a setback in its difficult comprehension). Indeed, I dare say that the best way to democratize AI is reviewed it thoroughly.

In this practical case I want to show you how banks could concede a credit. In this small project we will observe the importance of obtaining complete data, how to preprocess it, how to manage different algorithms/methods, and how to interpret them. Two methods called Decision Tree and Gradient Boosting Classifier will be compared. Each dataset will be different, so we have to fit our solution to each case.

[1] This databased will be based on public data from Spanish Government. Firstly, we should install their required libraries. We will use a program called Python. This small project is inspired in a practical case of the subject "Minería de datos avanzados" from UOC.

In [1]:

```
# Import some libraries
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn import datasets
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.utils import check_random_state

# Decision tree visualization
from IPython.display import Image
import pydotplus
from six import StringIO

# UMAP dimensionality reduction
import umap

# Visualisation
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt

%matplotlib inline
```

1. Predicting whether a mortgage has been conceded

In this exercise a new dataset will deal about different criteria if the bank will concede a certain mortgage to a certain customer. This preliminary decision will be made by historical data. There is the following information:

- ingresos: family's income
- gastos_comunes: general monthly payments
- pago_coche: monthly spending in transportation
- gastos_otros: other monthly spending
- ahorros: total amount in savings
- vivienda: house price
- estado civil: 0-single, 1-married, 2-divorced
- fills: number of sons.
- treball: 0-without occupation, 1- Freelance, 2-Employee, 3-Employer, 4-Couple: freelance (x2), 5-couple: employee, 6-couple: freelance & employee, 7-couple- employer & freelance, 8-couple- employers (x2)
- **hipoteca: 0-It has been conceded, 1-Without concession. It will be our target variable**

Firstly, I would like to place emphasis on the fact that we will have to analyze data. If historical data is dirty, we have to realize we might fix it. Sometimes our society has an improper past. By illustration, whether we want our past is patriarchal, our system could concede more credit to males than females.

In [2]:

```
hipotecas = pd.read_csv("hipotecas.csv")
hipotecas.head(10)
```

Out[2]:

	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	estado_civil	hijos	trabajo	hipoteca
0	6000	1000	0	600	50000	400000	0	2	2	1
1	6745	944	123	429	43240	636897	1	3	6	0
2	6455	1033	98	795	57463	321779	2	1	8	1
3	7098	1278	15	254	54506	660933	0	0	3	0
4	6167	863	223	520	41512	348932	0	0	3	1
5	5692	911	11	325	50875	360863	1	4	5	1
6	6830	1298	345	309	46761	429812	1	1	5	1
7	6470	1035	39	782	57439	606291	0	0	1	0
8	6251	1250	209	571	50503	291010	0	0	3	1
9	6987	1258	252	245	40611	324098	2	1	7	1

2.1. Descriptive analysis

Given this dataset, an exploratory analysis has to be done. we Must to understand and interpret our dataset.

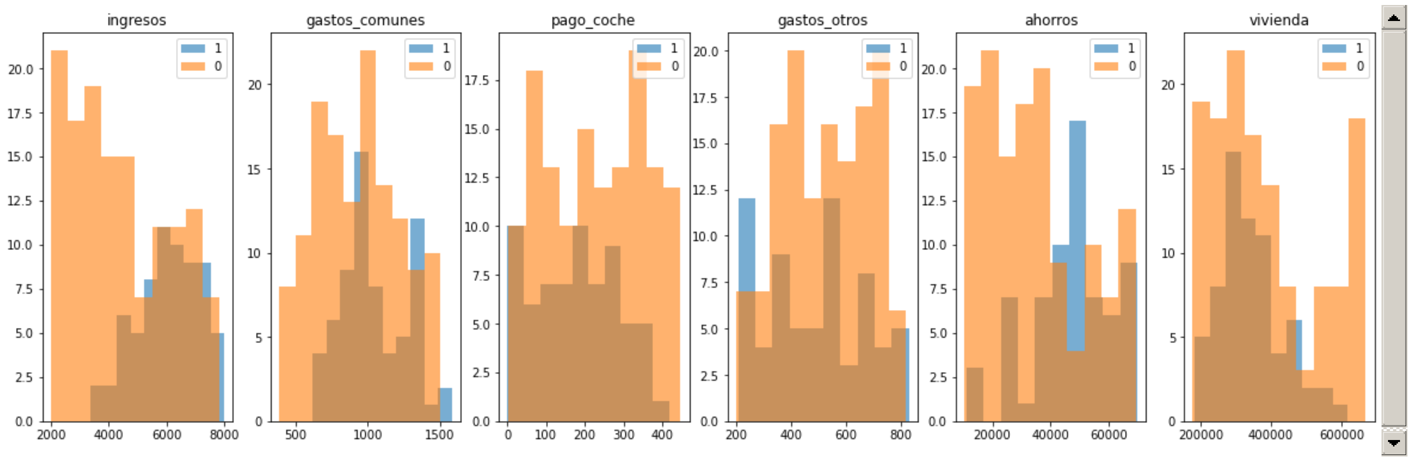
First and foremost, we have to calculate frequencies regarding the variable target (mortgage). Distribution of each categorical variable has to be analysed in bar graphs. Additionally, an histogram has to be visualized in numerical data.

In [3]:

```
# Firstly, numerical variables will be explored
feats_to_explore_num = ['ingresos', 'gastos_comunes', 'pago_coche', 'gastos_otros', 'ahorros','vivienda']
y = hipotecas["hipoteca"]
```

In [4]:

```
# Plot has to be prepared:
fig, axs = plt.subplots(nrows=1, ncols=len(feats_to_explore_num), figsize=(20,6))

# A graph for each feats_to_explore:
for i, featName in enumerate(feats_to_explore_num):
    # An histogram per each numerical variable:
    for diag in y.unique():
        axs[i].hist(hipotecas.loc[y==diag, featName], alpha=0.6, label=diag)
        axs[i].set_title(featName)
        axs[i].legend(loc='upper right')
```

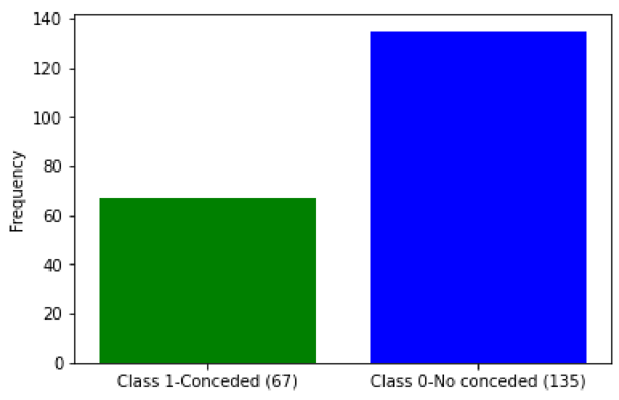


In [5]:

```
# A categorical has to be done
# Additionally
# We want to explore the number of concessions
y = hipotecas["hipoteca"]
counts = [y.value_counts()[1], y.value_counts()[0]]
place = np.arange(len(counts))
# Plot bar
plt.bar(place, counts, color=('green','blue'))

# We have to visualise each label
plt.xticks(place, ('Class 1-Conceded ('+str(y.value_counts()[1])+')',
                   'Class 0-No conceded ('+str(y.value_counts()[0])+')'))
plt.ylabel('Frequency')
```

Out[5]:

```
Text(0, 0.5, 'Frequency')
```

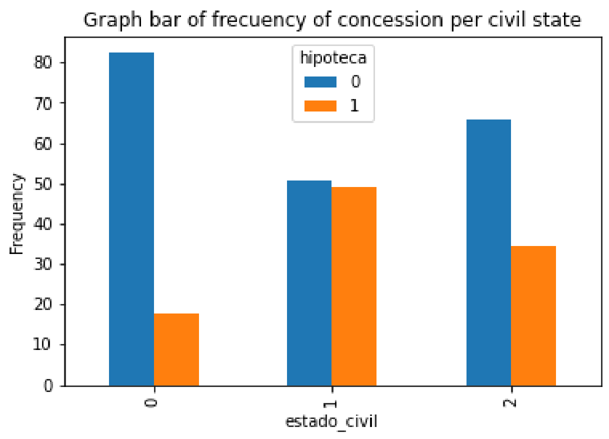


In [6]:

```
# A categorical analysis has to be done
# Graph bar in function of target

# Graph bar of frecuency of concession per civil state
plot = pd.crosstab(index=hipotecas['estado_civil'],
            columns=hipotecas['hipoteca']).apply(lambda r: r/r.sum() *100,
                                                  axis=1).plot(kind='bar')
plot.set_title('Graph bar of frecuency of concession per civil state')
plot.set_ylabel('Frequency')
```

Out[6]:

```
Text(0, 0.5, 'Frequency')
```

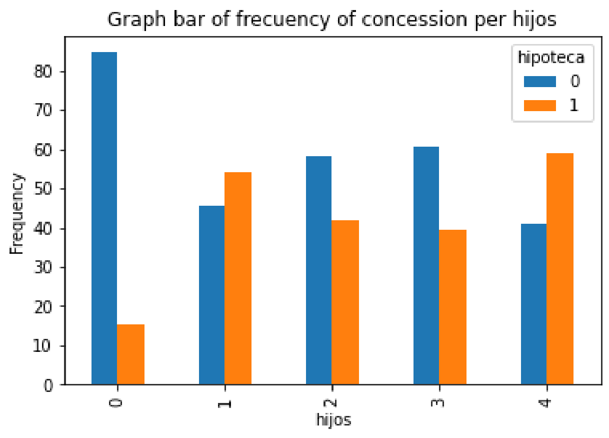


In [7]:

```
# A categorical analysis has to be done
# Graph bar in function of target

# Graph bar of frecuency of concession per number of children
plot = pd.crosstab(index=hipotecas['hijos'],
            columns=hipotecas['hipoteca']).apply(lambda r: r/r.sum() *100,
                                                  axis=1).plot(kind='bar')
plot.set_title('Graph bar of frecuency of concession per hijos')
plot.set_ylabel('Frequency')
```

Out[7]:

```
Text(0, 0.5, 'Frequency')
```

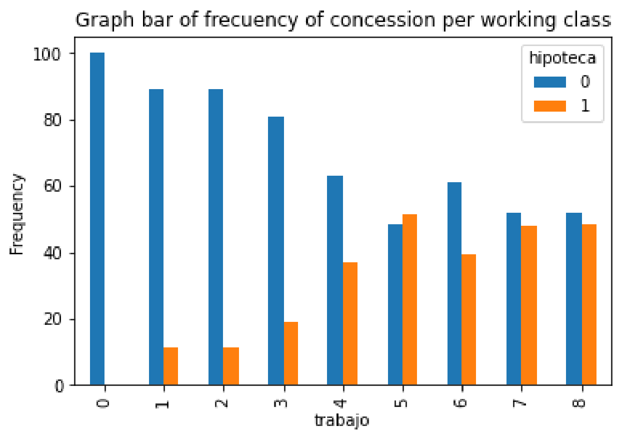


In [8]:

```
# A categorical analysis has to be done
# Graph bar in function of target

# Graph bar of frecuency of concession per type of work
plot = pd.crosstab(index=hipotecas['trabajo'],
            columns=hipotecas['hipoteca']).apply(lambda r: r/r.sum() *100,
                                                  axis=1).plot(kind='bar')
plot.set_title('Graph bar of frecuency of concession per working class')
plot.set_ylabel('Frequency')
```

Out[8]:

```
Text(0, 0.5, 'Frequency')
```



Interpretation of this step

In this histogram, we can observe whether a mortgage has been conceded in function of different variables such as expenditure or the cost of this mortgage. It is asserted that the variables that could influence the most in the project are expenditure and the cost of the house. Additionally, we also have to take into account that the professional profile could be an important factor. An unemployed has rarely a mortgage conceded. Stay tuned that a couple of employers has a high concession index (more than 50%).

On the bright side of having a marriage, there is an increasing number of percentages in the concessions (more than 30%). It could be related to the fact that the odds of being unemployed are less than having a lot of members.

Furthermore, on the upside of having a large number of children could influence on the concession of the mortgage. [2] According to INE, Spanish birth rate is 0,86 children per person. Generally, this survey asserts that there is a certain trend that shows that as more familiar resource gain, the number of children increase. Come what may, it is highly plausible that some unstructured families could have a large family, too. Hence, in a statistical analysis we could observe there is an inaccurate rate of variance's errors distributions.

2.2. Preprocessing data

The next step will have to be done is cleaning data. It is an essential step in the design of this model.

First step: Check null values.

In [9]:

```
# Check null values
hipotecas.isnull().sum()
# df_brcan.isna().any()
# There is not missing data
```

Out[9]:

```
ingresos          0
gastos_comunes    0
pago_coche        0
gastos_otros      0
ahorros           0
vivienda          0
estado_civil      0
hijos             0
trabajo           0
hipoteca          0
dtype: int64
```

Second step: Modify and fix the data - We will have to group by diferent expenditures such as pago_coche, gastos_comunes, gastos_otros in a new column called "total_cost" - Calculate the total amount to finance. Consequently, a new column will be called as "To_finance" - Drop all old varibles

In [10]:

```
# Total amount of expenditure
hipotecas['total_cost'] = hipotecas['pago_coche']+hipotecas['gastos_comunes']+hipotecas['gastos_otros']
# Total amount to finance
hipotecas['To_finance'] = hipotecas['vivienda']-hipotecas['ahorros']
# Drop all non necessary columns
new_hipotecas = hipotecas.drop(columns=['gastos_comunes','pago_coche','gastos_otros','ahorros','vivienda'
new_hipotecas
```

Out[10]:

	ingresos	estado_civil	hijos	trabajo	hipoteca	total_cost	To_finance
0	6000	0	2	2	1	1600	350000
1	6745	1	3	6	0	1496	593657
2	6455	2	1	8	1	1926	264316
3	7098	0	0	3	0	1547	606427
4	6167	0	0	3	1	1606	307420
...	...	...	...	...	...	...	...
197	3831	0	0	2	0	1530	352397
198	3961	2	3	8	0	1775	258541
199	3184	1	3	8	0	1915	352460
200	3334	1	2	5	0	1888	356907
201	3988	0	0	4	0	1644	245600

202 rows × 7 columns

Next step will be checking all correlations between all numerical feautres. Hence, we can observe if we have to combine some variables in a new column. Stay tuned, correlation does not imply causation.

> **Conclusion:** The following graph shows the correlation in all numerical features. If there are two variables with a correaltion greater than 80%, some of them could be dropped. Heatmap (seaborn) library will be used.

In [11]:

```
# Firstly, we will observe correlation in the old dataset
num_hipotecas_no_proc = hipotecas.drop(columns=['estado_civil','hijos','trabajo','hipoteca'])
corr_all1 = num_hipotecas_no_proc.corr()
corr_all1.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[11]:

	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	total_cost	To_finance
ingresos	1.00	0.56	-0.11	-0.12	0.71	0.61	0.36	0.56
gastos_comunes	0.56	1.00	-0.05	-0.10	0.21	0.20	0.77	0.19
pago_coche	-0.11	-0.05	1.00	0.01	-0.19	-0.09	0.34	-0.08
gastos_otros	-0.12	-0.10	0.01	1.00	-0.06	-0.05	0.44	-0.05
ahorros	0.71	0.21	-0.19	-0.06	1.00	0.61	0.07	0.52
vivienda	0.61	0.20	-0.09	-0.05	0.61	1.00	0.11	0.99
total_cost	0.36	0.77	0.34	0.44	0.07	0.11	1.00	0.11
To_finance	0.56	0.19	-0.08	-0.05	0.52	0.99	0.11	1.00

Interpretation

Firstly, I would like to draw attention to the fact that there are not any significant correlations. Without a shadow of doubt, forasmuch of increasing income, we can observe these families usually have more savings. Still futher, they usually tend to finance more amount, as their cost of life is greater. Assuming that the correct data set is the preprocessed, we will analyze it as well.

In [12]:

```
# Preprocessed dataset
num_hipotecas_proc = new_hipotecas.drop(columns=['estado_civil','hijos','trabajo','hipoteca'])
corr_all2 = num_hipotecas_proc.corr()
corr_all2.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[12]:

	ingresos	total_cost	To_finance
ingresos	1.00	0.36	0.56
total_cost	0.36	1.00	0.11
To_finance	0.56	0.11	1.00

Interpretation

In this graph, we can observe that there are not significiant correlation between variables. By the same token, the most correlated variables are income and the amount to "To_finance" (56% correlation).

In the next step we will check it in a numerical way.

In [13]:

```
# Select the upper triangle in correlation
upper = corr_all2.where(np.triu(np.ones(corr_all2.shape), k=1).astype(np.bool))

# We will check values with a correlation greater than 80%
highCorr = [column for column in upper.columns if any(upper[column] > 0.8)]
print("Correlation greater than 80%")
highCorr
```

```
Correlation greater than 80%
```

Out[13]:

```
[]
```

Interpretation

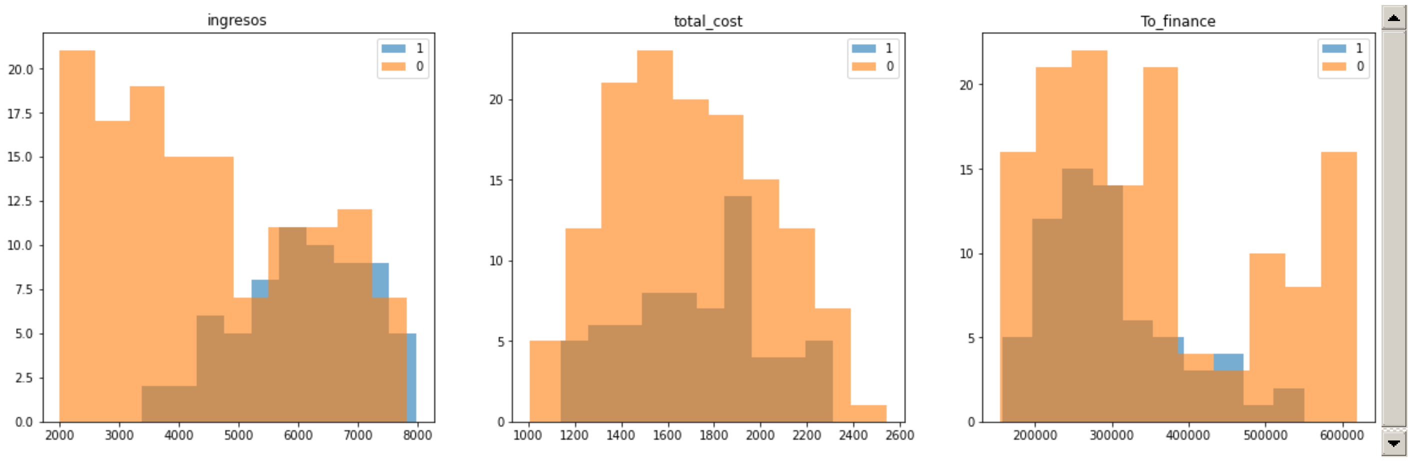
There is not any correlation greater than 80%. Hence, it is not necessary to drop any column.

In the next analysis we will check an exploratory analysis with the new processed dataset.

Case: We will have to plot the graph bar and histograms.

In [14]:

```
# Firstly, I will plot a histogram with numerical variables
feats_to_explore_num = ['ingresos', 'total_cost','To_finance']
y = new_hipotecas["hipoteca"]
# The next figure will be prepared
fig, axs = plt.subplots(nrows=1, ncols=len(feats_to_explore_num), figsize=(20,6))

# A graphic per each feats_to_explore:
for i, featName in enumerate(feats_to_explore_num):
    # An histogram per each numerical var:
    for diag in y.unique():
        axs[i].hist(new_hipotecas.loc[y==diag, featName], alpha=0.6, label=diag)
        axs[i].set_title(featName)
        axs[i].legend(loc='upper right')
```



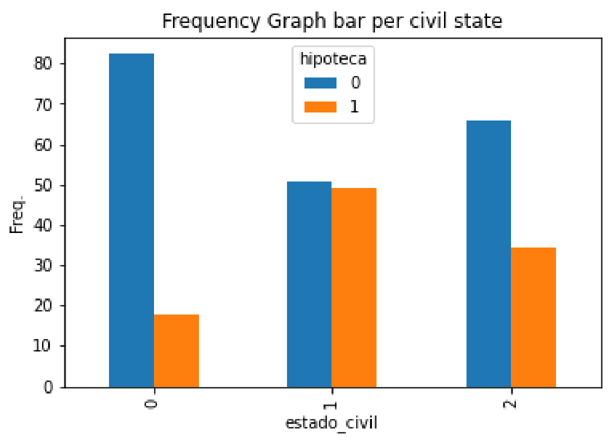
In [15]:

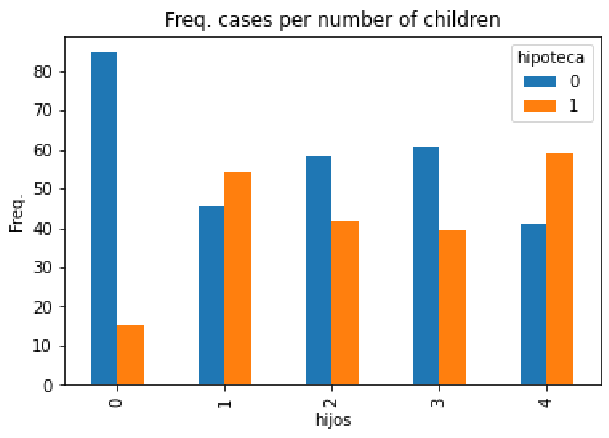
```
# Graph bar

# Graph bar per civil state
plot = pd.crosstab(index=new_hipotecas['estado_civil'],
            columns=new_hipotecas['hipoteca']).apply(lambda r: r/r.sum() *100,
                                                     axis=1).plot(kind='bar')
plot.set_title('Frequency Graph bar per civil state')
plot.set_ylabel('Freq.')

# Graph bar per number of children
plot = pd.crosstab(index=hipotecas['hijos'],
            columns=new_hipotecas['hipoteca']).apply(lambda r: r/r.sum() *100,
                                                     axis=1).plot(kind='bar')
```

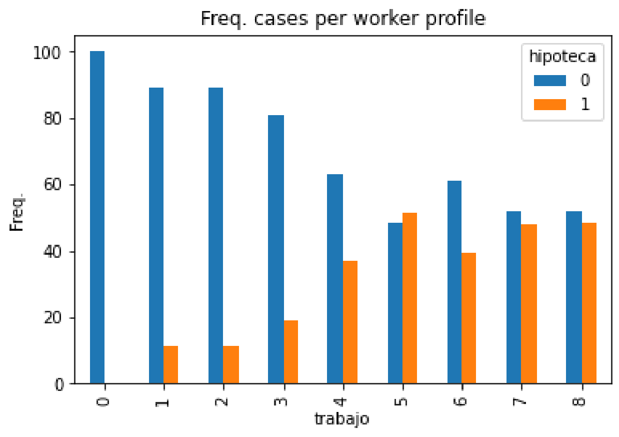
```
plot.set_title('Freq. cases per number of children')
plot.set_ylabel('Freq.')

# Graph bar per number per worker profile
plot = pd.crosstab(index=hipotecas['trabajo'],
            columns=new_hipotecas['hipoteca']).apply(lambda r: r/r.sum() *100,
                                                    axis=1).plot(kind='bar')
plot.set_title('Freq. cases per worker profile')
plot.set_ylabel('Freq.')
```

Out[15]:

```
Text(0, 0.5, 'Freq.')
```







Analysis

As it has been previously mencioned, we can observe the most incluential variables such as income and amount to finance. However, total expenditure does not seem to have a significant influence. According to the professional variable, we can observe that there is a certain trend regarding each employee profile.

Furthermore, in order to enhace our statistical analysis we will have to apply a one-hot-encoding in the required categorial data. A one hot encoding allows the representation of categorical data to be more expressive. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers.

> **Upcoming step:** Apply OneHotEncoder from sklearn.preprocessing.

In [16]:

```
# OneHotEncoder will be applied in estat_civil and trabajo
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
# creating instance of one-hot-encoder
encoder = OneHotEncoder()
# Seleccion of variables
enc_hipotecas = new_hipotecas[['estado_civil','trabajo']]
encoder.fit(enc_hipotecas)
# Transform a numpy array
resultat = encoder.transform(enc_hipotecas).toarray()
# New datafram
df = pd.DataFrame(resultat, columns = encoder.get_feature_names())
# Joining dataframe with key values
new_hipotecas = new_hipotecas.join(df)
new_hipotecas
```

Out[16]:

	ingresos	estado_civil	hijos	trabajo	hipoteca	total_cost	To_finance	x0_0	x0_1	x0_2	x1_0	x1_1	x1_2	x1_3	x1_4	x1_5	x1_6	x1_
0	6000	0	2	2	1	1600	350000	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.
1	6745	1	3	6	0	1496	593657	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.
2	6455	2	1	8	1	1926	264316	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	7098	0	0	3	0	1547	606427	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.
4	6167	0	0	3	1	1606	307420	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
197	3831	0	0	2	0	1530	352397	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.
198	3961	2	3	8	0	1775	258541	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
199	3184	1	3	8	0	1915	352460	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
200	3334	1	2	5	0	1888	356907	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.
201	3988	0	0	4	0	1644	245600	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.

202 rows × 19 columns

In [17]:

```
# We will have to drop non necessary columns
# It will be civil state and trabajo
new_hipotecas = new_hipotecas.drop(columns=['estado_civil','trabajo'])
new_hipotecas
```

Out[17]:

	ingresos	hijos	hipoteca	total_cost	To_finance	x0_0	x0_1	x0_2	x1_0	x1_1	x1_2	x1_3	x1_4	x1_5	x1_6	x1_7	x1_8
0	6000	2	1	1600	350000	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	6745	3	0	1496	593657	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	6455	1	1	1926	264316	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	7098	0	0	1547	606427	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	6167	0	1	1606	307420	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
197	3831	0	0	1530	352397	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
198	3961	3	0	1775	258541	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
199	3184	3	0	1915	352460	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
200	3334	2	0	1888	356907	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
201	3988	0	0	1644	245600	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

202 rows × 17 columns

Analyzing: Why do we use one hot encoding?

Interpretation

[3] One-hot-encoding has to be used in order to normalising data as some ML algorithms cannot work with categorical data directly.

By a way of illustration, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation). Be that as it may, many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.

This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

2.3. Training a model

In the upcoming step, a model will be trained.

> **Case:** We will have to split this dataset up *dataset* in two pars *train* (80% data) i *test* (20% data). It will have the following terms X_train, X_test, y_train, y_test. We will use random_state=24.

In [18]:

```
# New dataset
X = new_hipotecas.drop(columns=['hipoteca'])
Y = new_hipotecas['hipoteca']
X
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=24)
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(161, 16) (161,)
(41, 16) (41,)
```

In [19]:

```
columns_names = X.columns.values
columns_names_list = list(columns_names)
columns_names_list
Y_valors=['0','1']
```

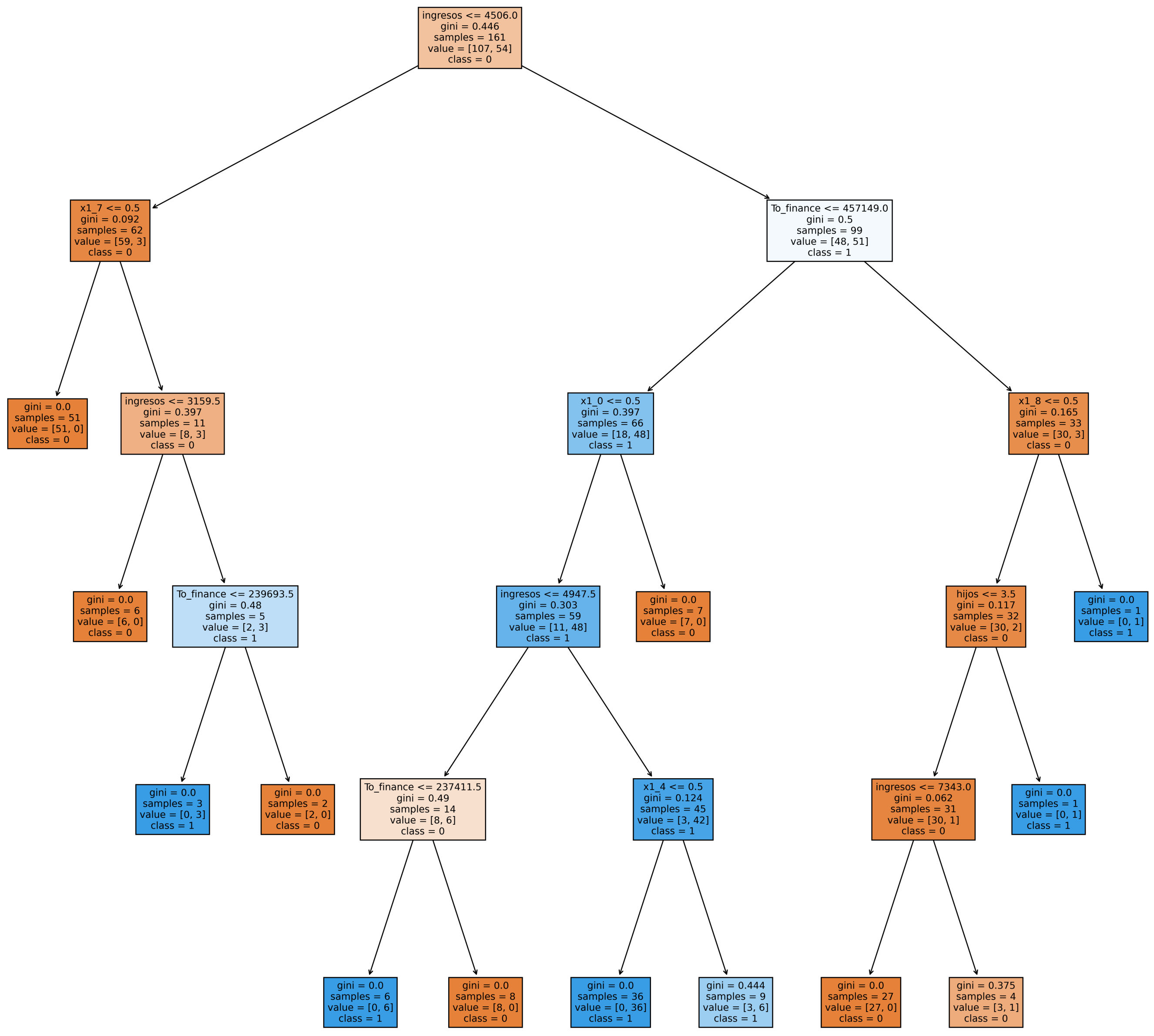
> **Implementation:** A new simple decision tree with max_depth=5 will be used to filter the dataset called "hipotecas" in the training set. Decision tree will be plotted. We will have to calculate accuracy in the confusion matrix.

In [20]:

```
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.tree import export_graphviz
import pydotplus
# Decision tree with max_depth=5
clf_DTC = DecisionTreeClassifier(max_depth=5)
model = clf_DTC.fit(X_train, y_train)
text_representation = tree.export_text(clf_DTC)
fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(20,20), dpi=300)
print(text_representation)
# DOT data
tree.plot_tree(clf_DTC,
                   feature_names=columns_names_list,
                   class_names=Y_valors,
                   filled=True
                            )
plt.show()
```

```
|--- feature_0 <= 4506.00
|   |--- feature_14 <= 0.50
|   |   |--- class: 0
|   |--- feature_14 >  0.50
|   |   |--- feature_0 <= 3159.50
|   |   |   |--- class: 0
|   |   |--- feature_0 >  3159.50
|   |   |   |--- feature_3 <= 239693.50
|   |   |   |   |--- class: 1
|   |   |   |--- feature_3 >  239693.50
|   |   |   |   |--- class: 0
|--- feature_0 >  4506.00
```

```
|   |--- feature_3 <= 457149.00
|   |   |--- feature_7 <= 0.50
|   |   |   |--- feature_0 <= 4947.50
|   |   |   |   |--- feature_3 <= 237411.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_3 >  237411.50
|   |   |   |   |   |--- class: 0
|   |   |   |--- feature_0 >  4947.50
|   |   |   |   |--- feature_11 <= 0.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_11 >  0.50
|   |   |   |   |   |--- class: 1
|   |   |--- feature_7 >  0.50
|   |   |   |--- class: 0
|   |--- feature_3 >  457149.00
|   |   |--- feature_15 <= 0.50
|   |   |   |--- feature_1 <= 3.50
|   |   |   |   |--- feature_0 <= 7343.00
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_0 >  7343.00
|   |   |   |   |   |--- class: 0
|   |   |   |--- feature_1 >  3.50
|   |   |   |   |--- class: 1
|   |   |--- feature_15 >  0.50
|   |   |   |--- class: 1
```

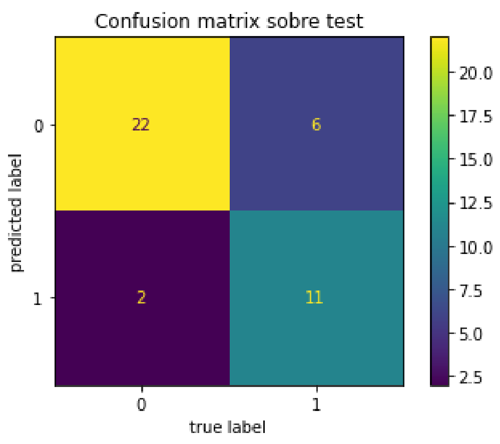


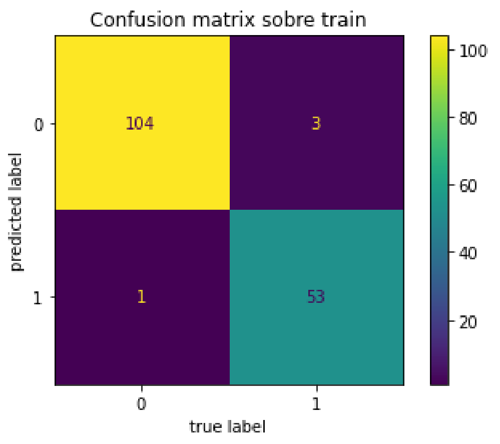
ingresos <= 4506.0 gini = 0.446 samples = 161 value = [107, 54] class = 0
x1_7 <= 0.5 gini = 0.092 samples = 62 value = [59, 3] class = 0
To_finance <= 457149.0 gini = 0.5 samples = 99 value = [48, 51] class = 1
gini = 0.0 samples = 51 value = [51, 0] class = 0
ingresos <= 3159.5 gini = 0.397 samples = 11 value = [8, 3] class = 0
x1_0 <= 0.5 gini = 0.397 samples = 66 value = [18, 48] class = 1
x1_8 <= 0.5 gini = 0.165 samples = 33 value = [30, 3] class = 0
gini = 0.0 samples = 6 value = [6, 0] class = 0
To_finance <= 239693.5 gini = 0.48 samples = 5 value = [2, 3] class = 1
ingresos <= 4947.5 gini = 0.303 samples = 59 value = [11, 48] class = 1
gini = 0.0 samples = 7 value = [7, 0] class = 0
hijos <= 3.5 gini = 0.117 samples = 32 value = [30, 2] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 3 value = [0, 3] class = 1
gini = 0.0 samples = 2 value = [2, 0] class = 0
To_finance <= 237411.5 gini = 0.49 samples = 14 value = [8, 6] class = 0
x1_4 <= 0.5 gini = 0.124 samples = 45 value = [3, 42] class = 1
ingresos <= 7343.0 gini = 0.062 samples = 31 value = [30, 1] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 6 value = [0, 6] class = 1
gini = 0.0 samples = 8 value = [8, 0] class = 0
gini = 0.0 samples = 36 value = [0, 36] class = 1
gini = 0.444 samples = 9 value = [3, 6] class = 1
gini = 0.0 samples = 27 value = [27, 0] class = 0
gini = 0.375 samples = 4 value = [3, 1] class = 0

In [21]:

```python
from sklearn.metrics import classification_report, confusion_matrix
#Import metrics class from sklearn
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix

predicted = clf_DTC.predict(X_test)

# Confusion matrix in testing set
metrics.accuracy_score(predicted, y_test)
plot_confusion_matrix(clf_DTC, X_test, y_test)

plt.title('Confusion matrix sobre test')
plt.xlabel('true label')
plt.ylabel('predicted label');

# Confusion matrix in training set
plot_confusion_matrix(clf_DTC, X_train, y_train)

plt.title('Confusion matrix sobre train')
plt.xlabel('true label')
plt.ylabel('predicted label');

# Precision in training model
score = clf_DTC.score(X_train,y_train)*100

print("Accuracy in training model {0:.2f} %".format(score))

# Precision in testing model
preds = clf_DTC.predict(X_test)

accuracy = np.true_divide(np.sum(preds == y_test), preds.shape[0])*100
cnf_matrix = confusion_matrix(y_test, preds)

print("Accuracy in testing model {0:.2f} %".format(accuracy))

# We will observe the recall values
print("Classification Report")
print(classification_report(y_test, preds))
```

```
Accuracy in training model 97.52 %
Accuracy in testing model 80.49 %
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.79      0.85        28
           1       0.65      0.85      0.73        13

    accuracy                           0.80        41
   macro avg       0.78      0.82      0.79        41
weighted avg       0.83      0.80      0.81        41
```





In [22]:

```python
# To prevent overfitting
# We will observe the optimal CV Grid Search

param_grid = {"max_depth": range(4, 10), "min_samples_split": [2, 10, 20, 50, 100]}

grid_search = GridSearchCV(clf_DTC, param_grid=param_grid, cv=4)
grid_search.fit(X_train, y_train)

means = grid_search.cv_results_["mean_test_score"]
stds = grid_search.cv_results_["std_test_score"]
params = grid_search.cv_results_['params']

for mean, std, pms in zip(means, stds, params):
    print("Average accuracy: {:.2f} +/- {:.2f} with parameters {}".format(mean*100, std*100, pms))
```

```
Average accuracy: 84.42 +/- 5.48 with parameters {'max_depth': 4, 'min_samples_split': 2}
Average accuracy: 83.17 +/- 7.63 with parameters {'max_depth': 4, 'min_samples_split': 10}
Average accuracy: 83.80 +/- 6.79 with parameters {'max_depth': 4, 'min_samples_split': 20}
Average accuracy: 80.11 +/- 3.15 with parameters {'max_depth': 4, 'min_samples_split': 50}
Average accuracy: 62.09 +/- 3.47 with parameters {'max_depth': 4, 'min_samples_split': 100}
Average accuracy: 82.58 +/- 4.76 with parameters {'max_depth': 5, 'min_samples_split': 2}
Average accuracy: 83.83 +/- 7.43 with parameters {'max_depth': 5, 'min_samples_split': 10}
Average accuracy: 83.80 +/- 6.79 with parameters {'max_depth': 5, 'min_samples_split': 20}
Average accuracy: 80.11 +/- 3.15 with parameters {'max_depth': 5, 'min_samples_split': 50}
Average accuracy: 62.09 +/- 3.47 with parameters {'max_depth': 5, 'min_samples_split': 100}
Average accuracy: 86.30 +/- 5.50 with parameters {'max_depth': 6, 'min_samples_split': 2}
Average accuracy: 85.66 +/- 8.94 with parameters {'max_depth': 6, 'min_samples_split': 10}
Average accuracy: 83.80 +/- 6.79 with parameters {'max_depth': 6, 'min_samples_split': 20}
Average accuracy: 80.11 +/- 3.15 with parameters {'max_depth': 6, 'min_samples_split': 50}
Average accuracy: 62.09 +/- 3.47 with parameters {'max_depth': 6, 'min_samples_split': 100}
Average accuracy: 86.95 +/- 2.74 with parameters {'max_depth': 7, 'min_samples_split': 2}
Average accuracy: 81.95 +/- 5.77 with parameters {'max_depth': 7, 'min_samples_split': 10}
Average accuracy: 83.80 +/- 6.79 with parameters {'max_depth': 7, 'min_samples_split': 20}
Average accuracy: 80.11 +/- 3.15 with parameters {'max_depth': 7, 'min_samples_split': 50}
Average accuracy: 62.09 +/- 3.47 with parameters {'max_depth': 7, 'min_samples_split': 100}
Average accuracy: 86.33 +/- 5.17 with parameters {'max_depth': 8, 'min_samples_split': 2}
Average accuracy: 83.78 +/- 8.05 with parameters {'max_depth': 8, 'min_samples_split': 10}
Average accuracy: 83.80 +/- 6.79 with parameters {'max_depth': 8, 'min_samples_split': 20}
Average accuracy: 80.11 +/- 3.15 with parameters {'max_depth': 8, 'min_samples_split': 50}
Average accuracy: 62.09 +/- 3.47 with parameters {'max_depth': 8, 'min_samples_split': 100}
Average accuracy: 85.67 +/- 5.47 with parameters {'max_depth': 9, 'min_samples_split': 2}
Average accuracy: 83.83 +/- 7.43 with parameters {'max_depth': 9, 'min_samples_split': 10}
Average accuracy: 83.80 +/- 6.79 with parameters {'max_depth': 9, 'min_samples_split': 20}
Average accuracy: 80.11 +/- 3.15 with parameters {'max_depth': 9, 'min_samples_split': 50}
Average accuracy: 62.09 +/- 3.47 with parameters {'max_depth': 9, 'min_samples_split': 100}
```

> **Analysis:** What are the most important variables? Have you got rather a good precision in testing set? Has overfitting been caused?

Interpretation

The main variable is income. If we have less than 4,5 k€, it is not common to have a mortgage concede. In case the income is minor than 4,5 k€ the following decision variable is the amount cost to finance. If this cost is less than 457k€, the next decision variable is the professional profile. If the customer is unemployed, whether the cost it is more difficult to get a credit. In this case, precision in testing set is 80,47%. Further to this, f1-score is 79%. As we have observed in the CV optimal Grid Search, we can minimise error in {'max_depth': 7, 'min_samples_split': 2} and average precision: 89.41% +/- 4.50%

Additionally, there is a certain overfitting as precision testing set is greater than training set (97,5%>80,4%). It could generate some hefty challenge if we predict new cases.

> **New model:** Random forest or gradient boosting will be used to filter the dataset "hipotecas". We will check for optimal CV Grid Search.

In [23]:

```python
# I will apply gradient boosting
# I will use 4 folds
from sklearn.ensemble import GradientBoostingClassifier

clf_GBC = GradientBoostingClassifier()

param_grid = {"max_depth": range(4, 12), "n_estimators": [50, 100, 200]}

grid_search = GridSearchCV(clf_GBC, param_grid=param_grid, cv=4)
grid_search.fit(X_train, y_train)

means = grid_search.cv_results_["mean_test_score"]
stds = grid_search.cv_results_["std_test_score"]
params = grid_search.cv_results_['params']

for mean, std, pms in zip(means, stds, params):
    print("Average accuracy: {:.2f} +/- {:.2f} with parameters {}".format(mean*100, std*100, pms))
```

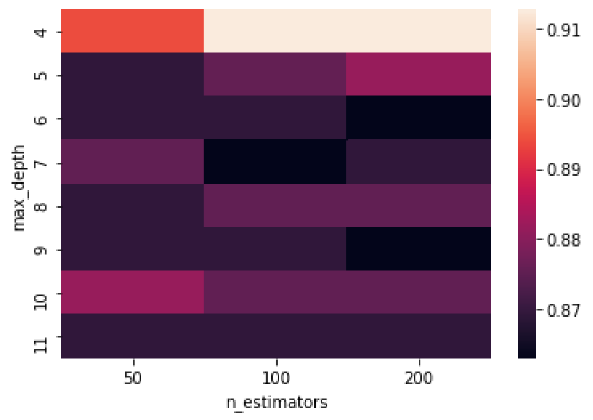
```
Average accuracy: 89.41 +/- 4.50 with parameters {'max_depth': 4, 'n_estimators': 50}
Average accuracy: 91.27 +/- 5.17 with parameters {'max_depth': 4, 'n_estimators': 100}
Average accuracy: 91.25 +/- 6.50 with parameters {'max_depth': 4, 'n_estimators': 200}
Average accuracy: 86.94 +/- 5.73 with parameters {'max_depth': 5, 'n_estimators': 50}
Average accuracy: 87.56 +/- 6.40 with parameters {'max_depth': 5, 'n_estimators': 100}
Average accuracy: 88.17 +/- 6.73 with parameters {'max_depth': 5, 'n_estimators': 200}
Average accuracy: 86.92 +/- 6.26 with parameters {'max_depth': 6, 'n_estimators': 50}
Average accuracy: 86.92 +/- 6.26 with parameters {'max_depth': 6, 'n_estimators': 100}
Average accuracy: 86.30 +/- 5.78 with parameters {'max_depth': 6, 'n_estimators': 200}
Average accuracy: 87.55 +/- 6.16 with parameters {'max_depth': 7, 'n_estimators': 50}
Average accuracy: 86.30 +/- 6.54 with parameters {'max_depth': 7, 'n_estimators': 100}
Average accuracy: 86.92 +/- 6.26 with parameters {'max_depth': 7, 'n_estimators': 200}
Average accuracy: 86.92 +/- 5.74 with parameters {'max_depth': 8, 'n_estimators': 50}
Average accuracy: 87.55 +/- 6.16 with parameters {'max_depth': 8, 'n_estimators': 100}
Average accuracy: 87.55 +/- 6.16 with parameters {'max_depth': 8, 'n_estimators': 200}
Average accuracy: 86.92 +/- 5.74 with parameters {'max_depth': 9, 'n_estimators': 50}
Average accuracy: 86.92 +/- 5.74 with parameters {'max_depth': 9, 'n_estimators': 100}
Average accuracy: 86.30 +/- 5.50 with parameters {'max_depth': 9, 'n_estimators': 200}
Average accuracy: 88.17 +/- 6.25 with parameters {'max_depth': 10, 'n_estimators': 50}
Average accuracy: 87.55 +/- 6.16 with parameters {'max_depth': 10, 'n_estimators': 100}
Average accuracy: 87.55 +/- 6.16 with parameters {'max_depth': 10, 'n_estimators': 200}
Average accuracy: 86.92 +/- 5.74 with parameters {'max_depth': 11, 'n_estimators': 50}
Average accuracy: 86.92 +/- 5.74 with parameters {'max_depth': 11, 'n_estimators': 100}
Average accuracy: 86.92 +/- 5.74 with parameters {'max_depth': 11, 'n_estimators': 200}
```

In [24]:

```
param1 = [x['max_depth'] for x in params]
param2 = [x['n_estimators'] for x in params]

precisions = pd.DataFrame(zip(param1, param2, means), columns=['max_depth', 'n_estimators', 'means'])
precisions = precisions.pivot('max_depth', 'n_estimators', 'means')
sns.heatmap(precisions)
```

Out[24]:

```
<AxesSubplot:xlabel='n_estimators', ylabel='max_depth'>
```



In [25]:

```
# The best arguments are max_depth=4, n_estimators=100
# Classificator will be trained with optimal parameters
clf_GBC = GradientBoostingClassifier(max_depth=4, n_estimators=100)
clf_GBC.fit(X_train, y_train)
```

Out[25]:

```
GradientBoostingClassifier(max_depth=4)
```

In [26]:

```
from sklearn.metrics import classification_report, confusion_matrix
#Import metrics class from sklearn
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix

predicted = clf_GBC.predict(X_test)

metrics.accuracy_score(predicted, y_test)
# Confusion matrix in test
plot_confusion_matrix(clf_GBC, X_test, y_test)

plt.title('Confusion matrix sobre test')
plt.xlabel('true label')
plt.ylabel('predicted label');
```

```
# Confusion matrix in train
plot_confusion_matrix(clf_GBC, X_train, y_train)

plt.title('Confusion matrix sobre train')
plt.xlabel('true label')
plt.ylabel('predicted label');

# Precision in model train
score = clf_GBC.score(X_train,y_train)*100

print("Accuracy in training model" in training model {0:.2f} %".format(score))

# Precision in model test
preds = clf_GBC.predict(X_test)

accuracy = np.true_divide(np.sum(preds == y_test), preds.shape[0])*100
cnf_matrix = confusion_matrix(y_test, preds)

print("Accuracy in testing model {0:.2f} %".format(accuracy))

# We will prevent overfitting
print("Classification Report")
print(classification_report(y_test, preds))
```

```
  File "<ipython-input-26-a92a54612606>", line 26
    print("Accuracy in training model" in training model {0:.2f} %".format(score))
                                                       ^
SyntaxError: invalid syntax
```

As Gradient Boosting is a black box (difficult to interpret it), we can not plot it, as there are a multiple combination of decision trees. Hence, we will check the most important variables in a permutation importantes in testing set and in a feature importance test (MDI)

> **Case:** We will check each variable in the fitted model.

In [ ]:

```
from sklearn.inspection import permutation_importance
feature_importance = clf_GBC.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, np.array(columns_names_list)[sorted_idx])
plt.title('Feature Importance (MDI)')

result = permutation_importance(clf_GBC, X_test, y_test, n_repeats=10,
                                random_state=24, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.boxplot(result.importances[sorted_idx].T,
            vert=False, labels=np.array(columns_names_list)[sorted_idx])
plt.title("Permutation Importance (test set)")
fig.tight_layout()
plt.show()
```

> **Analysis:** What are the most important variables? Has overfitting been caused?

Solution

As it has been previously mentioned, the most important variables are income and amount to finance. According to MDI test, the main feature is the cost to finance. However, in permutation importance the most important variable is income. Be that as it may, there is a hefty variability in the boxplot. Furthermore, the following important variable is the civil state (in the case of having an employer couple).

In this case, we can observe an excellent precision (testing set 85,3% and training set 100%). Arbeit, a certain overfitting could be caused (precision training set > testing set). Let me say that we could generate some setback in new predictions. It is plausible that our model is fitting data in excess. However, we have optimised this model using Grid Search. This model minimize error. As classification analysis has reasonable values, our f1-score is 85%. In this case, gradient boosting is greater than decision tree (85,3%>80,49%).

2.4. Prediction of new cases

We will use new data in some cases

Case: It is suposed we work in a bank and we want to conceed a mortage. We will check with new cases what could be the solution per each case. - Case 1: Family with an income of 2000€ and a expenditure 500€, an employer couple without children. They need to finance 200000€. - Case 2: Family with an income of 6000€ and a expenditure 3400€, an employee/freelance couple wit two children. They need to finance 320000€. - Case 3: A single women with an income of 9000€ and an expenditure of 2250€,freelance with a children. She needs to finance 39000€.

In [ ]:

```
# Prediction case 1
# Defining new instance
Xnew = [[2000, 0, 500, 200000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
# New prediction in gradient boosting
ynew_GBC = clf_GBC.predict(Xnew)
# New prediction in decision tree
ynew_DTC = clf_DTC.predict(Xnew)
print("Prediccion according gradient")
print("Prediction is %s in case 1 X=%s, " % (ynew_GBC[0], Xnew[0]))
print("Prediccion according decision tree")
print("La prediccion is %s in case 1 X=%s, " % (ynew_DTC[0], Xnew[0]))
```

In [ ]:

```
# Prediction case 2
# Defining new instance
Xnew = [[6000, 2, 3400, 320000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]]
# New prediction in gradient boosting
ynew_GBC = clf_GBC.predict(Xnew)
# New prediction in decision tree
ynew_DTC = clf_DTC.predict(Xnew)
print("Prediccion according gradient")
print("Prediction is %s in case 2 X=%s, " % (ynew_GBC[0], Xnew[0]))
print("Prediccion according decision tree")
print("La prediccion is %s in case 2 X=%s, " % (ynew_DTC[0], Xnew[0]))
```

In [ ]:

```
# Prediction case 3
# Defining new instance
Xnew = [[9000, 1, 22500, 39000, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
# New prediction in gradient boosting
ynew_GBC = clf_GBC.predict(Xnew)
# New prediction in decision tree
ynew_DTC = clf_DTC.predict(Xnew)
print("Prediccion according gradient")
print("Prediction is %s in case 3 X=%s, " % (ynew_GBC[0], Xnew[0]))
print("Prediccion according decision tree")
print("La prediccion is %s in case 3 X=%s, " % (ynew_DTC[0], Xnew[0]))
```

3. Closure

As it has been previously mentioned in the exploratory analysis, in the first case we would not concede this mortgage as their income is minor than 4500€ (2000 € < 4500 €). In the following cases, as their income is greater and the following variables are fitted in the model, it is more plausible to obtain this mortgage. In both models, the result is the same.

References

[1] Gobierno de España. "Datos abiertos". Extracted from https://datos.gob.es/

[2] INE. "Indicadores demográficos básicos" (publicat en decembre de 2020 amb dades definitives de 2019. Próxima actualizació: juny de 2021)

[3] Yadah, D. "Categorical encoding using Label-Encoding and One-Hot-Encoder". Extracted on 20th April 2021 from https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd