# US cencus income : Ensembles, Bagging and Shap Values

# Problem Framework

Our task is to determine the income level for the person represented by the record. Incomes have been binned at the $50K level to present a **binary classification problem**.

The dataset used in this analysis was extracted from the census bureau database found at. The data was split into train/test in approximately 2/3, 1/3 proportions.

The following mappings of the data is as follow :

Datasets can be found in this link : <https://github.com/younesszaim/myportfolio/tree/master/_notebooks/Datasets>

import pandas as pd  
from pandas.api.types import is\_string\_dtype, is\_numeric\_dtype, is\_categorical\_dtype  
import numpy as np

PATH = '/Users/rmbp/Desktop/Dataiku Data Scientist Technical Assessment'

df\_labels = pd.read\_csv(f'{PATH}/census\_income\_metadata\_column.csv', sep=';')  
df\_labels.head(5)

column\_name dtype  
0 age continuous  
1 class\_of\_worker nominal  
2 detailed\_industry\_recode nominal  
3 detailed\_occupation\_recode nominal  
4 education nominal

In any sort of data science work, it's important to look at our data directly to make sure we understand the format, how it's stored, what types of values it holds, etc. Even if we've read a description of the data, the actual data may not be what we expect. We'll start by reading the training set into a Pandas DataFrame :

# Loading the train data   
df = pd.read\_csv(f'{PATH}/census\_income\_learn.csv', names = df\_labels['column\_name'])  
df.shape

(199523, 42)

Let's have a look at the columns, their types defined by Pandas and compared it to their actual mapping types :

# Chekcing the mapping of the data   
d1 = df.dtypes.apply(lambda x: x.name).to\_dict()  
d2 = {c: d for c,d in zip(df\_labels['column\_name'],df\_labels['dtype'])}  
mapping = [d1, d2]  
d = {}  
for k in d1.keys():  
 d[k] = tuple(d[k] for d in mapping)  
d

{'age': ('int64', 'continuous'),  
 'class\_of\_worker': ('object', 'nominal'),  
 'detailed\_industry\_recode': ('int64', 'nominal'),  
 'detailed\_occupation\_recode': ('int64', 'nominal'),  
 'education': ('object', 'nominal'),  
 'wage\_per\_hour': ('int64', 'continuous'),  
 'enroll\_in\_edu\_inst\_last\_wk': ('object', 'nominal'),  
 'marital\_stat': ('object', 'nominal'),  
 'major\_industry\_code': ('object', 'nominal'),  
 'major\_occupation\_code': ('object', 'nominal'),  
 'race': ('object', 'nominal'),  
 'hispanic\_origin': ('object', 'nominal'),  
 'sex': ('object', 'nominal'),  
 'member\_of\_a\_labor\_union': ('object', 'nominal'),  
 'reason\_for\_unemployment': ('object', 'nominal'),  
 'full\_or\_part\_time\_employment\_stat': ('object', 'nominal'),  
 'capital\_gains': ('int64', 'continuous'),  
 'capital\_losses': ('int64', 'continuous'),  
 'dividends\_from\_stocks': ('int64', 'continuous'),  
 'tax\_filer\_stat': ('object', 'nominal'),  
 'region\_of\_previous\_residence': ('object', 'nominal'),  
 'state\_of\_previous\_residence': ('object', 'nominal'),  
 'detailed\_household\_and\_family\_stat': ('object', 'nominal'),  
 'detailed\_household\_summary\_in\_household': ('object', 'nominal'),  
 'ignore': ('float64', 'continuous'),  
 'migration\_code-change\_in\_msa': ('object', 'nominal'),  
 'migration\_code-change\_in\_reg': ('object', 'nominal'),  
 'migration\_code-move\_within\_reg': ('object', 'nominal'),  
 'live\_in\_this\_house\_1\_year\_ago': ('object', 'nominal'),  
 'migration\_prev\_res\_in\_sunbelt': ('object', 'nominal'),  
 'num\_persons\_worked\_for\_employer': ('int64', 'continuous'),  
 'family\_members\_under\_18': ('object', 'nominal'),  
 'country\_of\_birth\_father': ('object', 'nominal'),  
 'country\_of\_birth\_mother': ('object', 'nominal'),  
 'country\_of\_birth\_self': ('object', 'nominal'),  
 'citizenship': ('object', 'nominal'),  
 'own\_business\_or\_self\_employed': ('int64', 'nominal'),  
 "fill\_inc\_questionnaire\_for\_veteran's\_admin": ('object', 'nominal'),  
 'veterans\_benefits': ('int64', 'nominal'),  
 'weeks\_worked\_in\_year': ('int64', 'continuous'),  
 'year': ('int64', 'nominal'),  
 'income\_level': ('object', 'nominal')}

We can see that **detailed\_industry\_recode**, **detailed\_occupation\_recode**, **own\_business\_or\_self\_employed**, **veterans\_benefits** and **year** is set by default as a continuos category.

Let's redifined their types :

# Correcting data types  
d1['detailed\_industry\_recode']='object'  
d1['detailed\_occupation\_recode']='object'  
d1['own\_business\_or\_self\_employed']='object'  
d1['veterans\_benefits']='object'  
d1['year']='object'

Let's reload the data with its correspind feature's mapping :

# reload data with coorexted types  
df = pd.read\_csv(f'{PATH}/census\_income\_learn.csv', names =df\_labels['column\_name'],  
 dtype= d1)

The **info()** method is useful to get a quick description of the data, in particular the total number of rows, each attribute’s type, and the number of nonnull values :

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 199523 entries, 0 to 199522  
Data columns (total 42 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 199523 non-null int64   
 1 class\_of\_worker 199523 non-null object   
 2 detailed\_industry\_recode 199523 non-null object   
 3 detailed\_occupation\_recode 199523 non-null object   
 4 education 199523 non-null object   
 5 wage\_per\_hour 199523 non-null int64   
 6 enroll\_in\_edu\_inst\_last\_wk 199523 non-null object   
 7 marital\_stat 199523 non-null object   
 8 major\_industry\_code 199523 non-null object   
 9 major\_occupation\_code 199523 non-null object   
 10 race 199523 non-null object   
 11 hispanic\_origin 199523 non-null object   
 12 sex 199523 non-null object   
 13 member\_of\_a\_labor\_union 199523 non-null object   
 14 reason\_for\_unemployment 199523 non-null object   
 15 full\_or\_part\_time\_employment\_stat 199523 non-null object   
 16 capital\_gains 199523 non-null int64   
 17 capital\_losses 199523 non-null int64   
 18 dividends\_from\_stocks 199523 non-null int64   
 19 tax\_filer\_stat 199523 non-null object   
 20 region\_of\_previous\_residence 199523 non-null object   
 21 state\_of\_previous\_residence 199523 non-null object   
 22 detailed\_household\_and\_family\_stat 199523 non-null object   
 23 detailed\_household\_summary\_in\_household 199523 non-null object   
 24 ignore 199523 non-null float64  
 25 migration\_code-change\_in\_msa 199523 non-null object   
 26 migration\_code-change\_in\_reg 199523 non-null object   
 27 migration\_code-move\_within\_reg 199523 non-null object   
 28 live\_in\_this\_house\_1\_year\_ago 199523 non-null object   
 29 migration\_prev\_res\_in\_sunbelt 199523 non-null object   
 30 num\_persons\_worked\_for\_employer 199523 non-null int64   
 31 family\_members\_under\_18 199523 non-null object   
 32 country\_of\_birth\_father 199523 non-null object   
 33 country\_of\_birth\_mother 199523 non-null object   
 34 country\_of\_birth\_self 199523 non-null object   
 35 citizenship 199523 non-null object   
 36 own\_business\_or\_self\_employed 199523 non-null object   
 37 fill\_inc\_questionnaire\_for\_veteran's\_admin 199523 non-null object   
 38 veterans\_benefits 199523 non-null object   
 39 weeks\_worked\_in\_year 199523 non-null int64   
 40 year 199523 non-null object   
 41 income\_level 199523 non-null object   
dtypes: float64(1), int64(7), object(34)  
memory usage: 63.9+ MB

# dispplay first rows   
with pd.option\_context('display.max\_rows', None, 'display.max\_columns', None):   
 display(df.head(3))

age class\_of\_worker detailed\_industry\_recode \  
0 73 Not in universe 0   
1 58 Self-employed-not incorporated 4   
2 18 Not in universe 0   
  
 detailed\_occupation\_recode education wage\_per\_hour \  
0 0 High school graduate 0   
1 34 Some college but no degree 0   
2 0 10th grade 0   
  
 enroll\_in\_edu\_inst\_last\_wk marital\_stat major\_industry\_code \  
0 Not in universe Widowed Not in universe or children   
1 Not in universe Divorced Construction   
2 High school Never married Not in universe or children   
  
 major\_occupation\_code race \  
0 Not in universe White   
1 Precision production craft & repair White   
2 Not in universe Asian or Pacific Islander   
  
 hispanic\_origin sex member\_of\_a\_labor\_union reason\_for\_unemployment \  
0 All other Female Not in universe Not in universe   
1 All other Male Not in universe Not in universe   
2 All other Female Not in universe Not in universe   
  
 full\_or\_part\_time\_employment\_stat capital\_gains capital\_losses \  
0 Not in labor force 0 0   
1 Children or Armed Forces 0 0   
2 Not in labor force 0 0   
  
 dividends\_from\_stocks tax\_filer\_stat region\_of\_previous\_residence \  
0 0 Nonfiler Not in universe   
1 0 Head of household South   
2 0 Nonfiler Not in universe   
  
 state\_of\_previous\_residence detailed\_household\_and\_family\_stat \  
0 Not in universe Other Rel 18+ ever marr not in subfamily   
1 Arkansas Householder   
2 Not in universe Child 18+ never marr Not in a subfamily   
  
 detailed\_household\_summary\_in\_household ignore \  
0 Other relative of householder 1700.09   
1 Householder 1053.55   
2 Child 18 or older 991.95   
  
 migration\_code-change\_in\_msa migration\_code-change\_in\_reg \  
0 ? ?   
1 MSA to MSA Same county   
2 ? ?   
  
 migration\_code-move\_within\_reg live\_in\_this\_house\_1\_year\_ago \  
0 ? Not in universe under 1 year old   
1 Same county No   
2 ? Not in universe under 1 year old   
  
 migration\_prev\_res\_in\_sunbelt num\_persons\_worked\_for\_employer \  
0 ? 0   
1 Yes 1   
2 ? 0   
  
 family\_members\_under\_18 country\_of\_birth\_father country\_of\_birth\_mother \  
0 Not in universe United-States United-States   
1 Not in universe United-States United-States   
2 Not in universe Vietnam Vietnam   
  
 country\_of\_birth\_self citizenship \  
0 United-States Native- Born in the United States   
1 United-States Native- Born in the United States   
2 Vietnam Foreign born- Not a citizen of U S   
  
 own\_business\_or\_self\_employed fill\_inc\_questionnaire\_for\_veteran's\_admin \  
0 0 Not in universe   
1 0 Not in universe   
2 0 Not in universe   
  
 veterans\_benefits weeks\_worked\_in\_year year income\_level   
0 2 0 95 - 50000.   
1 2 52 94 - 50000.   
2 2 0 95 - 50000.

# drop 'ignore' column  
df.drop('ignore', axis=1,inplace=True)

# list columns  
df.columns

Index(['age', 'class\_of\_worker', 'detailed\_industry\_recode',  
 'detailed\_occupation\_recode', 'education', 'wage\_per\_hour',  
 'enroll\_in\_edu\_inst\_last\_wk', 'marital\_stat', 'major\_industry\_code',  
 'major\_occupation\_code', 'race', 'hispanic\_origin', 'sex',  
 'member\_of\_a\_labor\_union', 'reason\_for\_unemployment',  
 'full\_or\_part\_time\_employment\_stat', 'capital\_gains', 'capital\_losses',  
 'dividends\_from\_stocks', 'tax\_filer\_stat',  
 'region\_of\_previous\_residence', 'state\_of\_previous\_residence',  
 'detailed\_household\_and\_family\_stat',  
 'detailed\_household\_summary\_in\_household',  
 'migration\_code-change\_in\_msa', 'migration\_code-change\_in\_reg',  
 'migration\_code-move\_within\_reg', 'live\_in\_this\_house\_1\_year\_ago',  
 'migration\_prev\_res\_in\_sunbelt', 'num\_persons\_worked\_for\_employer',  
 'family\_members\_under\_18', 'country\_of\_birth\_father',  
 'country\_of\_birth\_mother', 'country\_of\_birth\_self', 'citizenship',  
 'own\_business\_or\_self\_employed',  
 'fill\_inc\_questionnaire\_for\_veteran's\_admin', 'veterans\_benefits',  
 'weeks\_worked\_in\_year', 'year', 'income\_level'],  
 dtype='object')

We load the test set with the same training data types :

# loading the test set   
test = pd.read\_csv(f'{PATH}/census\_income\_test.csv', names =df\_labels['column\_name'], dtype= d1 )

test.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 99762 entries, 0 to 99761  
Data columns (total 42 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 99762 non-null int64   
 1 class\_of\_worker 99762 non-null object   
 2 detailed\_industry\_recode 99762 non-null object   
 3 detailed\_occupation\_recode 99762 non-null object   
 4 education 99762 non-null object   
 5 wage\_per\_hour 99762 non-null int64   
 6 enroll\_in\_edu\_inst\_last\_wk 99762 non-null object   
 7 marital\_stat 99762 non-null object   
 8 major\_industry\_code 99762 non-null object   
 9 major\_occupation\_code 99762 non-null object   
 10 race 99762 non-null object   
 11 hispanic\_origin 99762 non-null object   
 12 sex 99762 non-null object   
 13 member\_of\_a\_labor\_union 99762 non-null object   
 14 reason\_for\_unemployment 99762 non-null object   
 15 full\_or\_part\_time\_employment\_stat 99762 non-null object   
 16 capital\_gains 99762 non-null int64   
 17 capital\_losses 99762 non-null int64   
 18 dividends\_from\_stocks 99762 non-null int64   
 19 tax\_filer\_stat 99762 non-null object   
 20 region\_of\_previous\_residence 99762 non-null object   
 21 state\_of\_previous\_residence 99762 non-null object   
 22 detailed\_household\_and\_family\_stat 99762 non-null object   
 23 detailed\_household\_summary\_in\_household 99762 non-null object   
 24 ignore 99762 non-null float64  
 25 migration\_code-change\_in\_msa 99762 non-null object   
 26 migration\_code-change\_in\_reg 99762 non-null object   
 27 migration\_code-move\_within\_reg 99762 non-null object   
 28 live\_in\_this\_house\_1\_year\_ago 99762 non-null object   
 29 migration\_prev\_res\_in\_sunbelt 99762 non-null object   
 30 num\_persons\_worked\_for\_employer 99762 non-null int64   
 31 family\_members\_under\_18 99762 non-null object   
 32 country\_of\_birth\_father 99762 non-null object   
 33 country\_of\_birth\_mother 99762 non-null object   
 34 country\_of\_birth\_self 99762 non-null object   
 35 citizenship 99762 non-null object   
 36 own\_business\_or\_self\_employed 99762 non-null object   
 37 fill\_inc\_questionnaire\_for\_veteran's\_admin 99762 non-null object   
 38 veterans\_benefits 99762 non-null object   
 39 weeks\_worked\_in\_year 99762 non-null int64   
 40 year 99762 non-null object   
 41 income\_level 99762 non-null object   
dtypes: float64(1), int64(7), object(34)  
memory usage: 32.0+ MB

We verify if we got the same columns both on the train and the test set :

# checking columns on test set which not in train  
set(test.columns).difference(set(df.columns))

{'ignore'}

# dropping 'ignore' columns  
test.drop('ignore', inplace=True, axis=1)

test.columns

Index(['age', 'class\_of\_worker', 'detailed\_industry\_recode',  
 'detailed\_occupation\_recode', 'education', 'wage\_per\_hour',  
 'enroll\_in\_edu\_inst\_last\_wk', 'marital\_stat', 'major\_industry\_code',  
 'major\_occupation\_code', 'race', 'hispanic\_origin', 'sex',  
 'member\_of\_a\_labor\_union', 'reason\_for\_unemployment',  
 'full\_or\_part\_time\_employment\_stat', 'capital\_gains', 'capital\_losses',  
 'dividends\_from\_stocks', 'tax\_filer\_stat',  
 'region\_of\_previous\_residence', 'state\_of\_previous\_residence',  
 'detailed\_household\_and\_family\_stat',  
 'detailed\_household\_summary\_in\_household',  
 'migration\_code-change\_in\_msa', 'migration\_code-change\_in\_reg',  
 'migration\_code-move\_within\_reg', 'live\_in\_this\_house\_1\_year\_ago',  
 'migration\_prev\_res\_in\_sunbelt', 'num\_persons\_worked\_for\_employer',  
 'family\_members\_under\_18', 'country\_of\_birth\_father',  
 'country\_of\_birth\_mother', 'country\_of\_birth\_self', 'citizenship',  
 'own\_business\_or\_self\_employed',  
 'fill\_inc\_questionnaire\_for\_veteran's\_admin', 'veterans\_benefits',  
 'weeks\_worked\_in\_year', 'year', 'income\_level'],  
 dtype='object')

# display first rows of the test set  
with pd.option\_context('display.max\_rows', None, 'display.max\_columns', None):  
 display(test.head(3))

age class\_of\_worker detailed\_industry\_recode \  
0 38 Private 6   
1 44 Self-employed-not incorporated 37   
2 2 Not in universe 0   
  
 detailed\_occupation\_recode education \  
0 36 1st 2nd 3rd or 4th grade   
1 12 Associates degree-occup /vocational   
2 0 Children   
  
 wage\_per\_hour enroll\_in\_edu\_inst\_last\_wk marital\_stat \  
0 0 Not in universe Married-civilian spouse present   
1 0 Not in universe Married-civilian spouse present   
2 0 Not in universe Never married   
  
 major\_industry\_code major\_occupation\_code \  
0 Manufacturing-durable goods Machine operators assmblrs & inspctrs   
1 Business and repair services Professional specialty   
2 Not in universe or children Not in universe   
  
 race hispanic\_origin sex member\_of\_a\_labor\_union \  
0 White Mexican (Mexicano) Female Not in universe   
1 White All other Female Not in universe   
2 White Mexican-American Male Not in universe   
  
 reason\_for\_unemployment full\_or\_part\_time\_employment\_stat capital\_gains \  
0 Not in universe Full-time schedules 0   
1 Not in universe PT for econ reasons usually PT 0   
2 Not in universe Children or Armed Forces 0   
  
 capital\_losses dividends\_from\_stocks tax\_filer\_stat \  
0 0 0 Joint one under 65 & one 65+   
1 0 2500 Joint both under 65   
2 0 0 Nonfiler   
  
 region\_of\_previous\_residence state\_of\_previous\_residence \  
0 Not in universe Not in universe   
1 Not in universe Not in universe   
2 Not in universe Not in universe   
  
 detailed\_household\_and\_family\_stat \  
0 Spouse of householder   
1 Spouse of householder   
2 Child <18 never marr not in subfamily   
  
 detailed\_household\_summary\_in\_household migration\_code-change\_in\_msa \  
0 Spouse of householder ?   
1 Spouse of householder ?   
2 Child under 18 never married ?   
  
 migration\_code-change\_in\_reg migration\_code-move\_within\_reg \  
0 ? ?   
1 ? ?   
2 ? ?   
  
 live\_in\_this\_house\_1\_year\_ago migration\_prev\_res\_in\_sunbelt \  
0 Not in universe under 1 year old ?   
1 Not in universe under 1 year old ?   
2 Not in universe under 1 year old ?   
  
 num\_persons\_worked\_for\_employer family\_members\_under\_18 \  
0 4 Not in universe   
1 1 Not in universe   
2 0 Both parents present   
  
 country\_of\_birth\_father country\_of\_birth\_mother country\_of\_birth\_self \  
0 Mexico Mexico Mexico   
1 United-States United-States United-States   
2 United-States United-States United-States   
  
 citizenship own\_business\_or\_self\_employed \  
0 Foreign born- Not a citizen of U S 0   
1 Native- Born in the United States 0   
2 Native- Born in the United States 0   
  
 fill\_inc\_questionnaire\_for\_veteran's\_admin veterans\_benefits \  
0 Not in universe 2   
1 Not in universe 2   
2 Not in universe 0   
  
 weeks\_worked\_in\_year year income\_level   
0 12 95 - 50000.   
1 26 95 - 50000.   
2 0 95 - 50000.

df.shape, test.shape

((199523, 41), (99762, 41))

# Looking at the data

The most important data column is the **dependent variable**—that is, the one we want to predict which is **income\_level** :

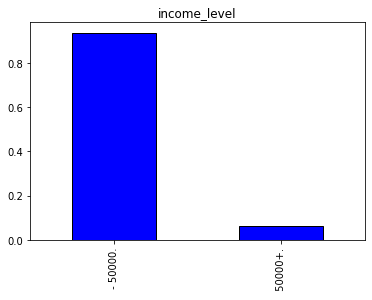
dep\_var = 'income\_level'

Let's see its distribution :

print(df[dep\_var].value\_counts(normalize = True))  
df[dep\_var].value\_counts(normalize = True).plot(kind='bar',  
 edgecolor='black',  
 color='blue',  
 title='income\_level')

- 50000. 0.937942  
 50000+. 0.062058  
Name: income\_level, dtype: float64

<matplotlib.axes.\_subplots.AxesSubplot at 0x124bace90>



We have an **imbalanced dataset** where the income level of -50k is representing more than 93% of the total records.

Next, we automatically handle which columns are **continuous** and which are **categorical** :

# get categorical and numerical variables  
def cont\_cat\_split(df, dep\_var=None):  
 "Helper function that returns column names of cont and cat variables from given `df`."  
 cont\_names, cat\_names = [], []  
 for label in df:  
 if label in [dep\_var]: continue  
 if (pd.api.types.is\_integer\_dtype(df[label].dtype) or  
 pd.api.types.is\_float\_dtype(df[label].dtype)):  
 cont\_names.append(label)  
 else: cat\_names.append(label)  
 return cont\_names, cat\_names  
  
cont, cat = cont\_cat\_split(df, dep\_var= dep\_var)  
cont , cat

(['age',  
 'wage\_per\_hour',  
 'capital\_gains',  
 'capital\_losses',  
 'dividends\_from\_stocks',  
 'num\_persons\_worked\_for\_employer',  
 'weeks\_worked\_in\_year'],  
 ['class\_of\_worker',  
 'detailed\_industry\_recode',  
 'detailed\_occupation\_recode',  
 'education',  
 'enroll\_in\_edu\_inst\_last\_wk',  
 'marital\_stat',  
 'major\_industry\_code',  
 'major\_occupation\_code',  
 'race',  
 'hispanic\_origin',  
 'sex',  
 'member\_of\_a\_labor\_union',  
 'reason\_for\_unemployment',  
 'full\_or\_part\_time\_employment\_stat',  
 'tax\_filer\_stat',  
 'region\_of\_previous\_residence',  
 'state\_of\_previous\_residence',  
 'detailed\_household\_and\_family\_stat',  
 'detailed\_household\_summary\_in\_household',  
 'migration\_code-change\_in\_msa',  
 'migration\_code-change\_in\_reg',  
 'migration\_code-move\_within\_reg',  
 'live\_in\_this\_house\_1\_year\_ago',  
 'migration\_prev\_res\_in\_sunbelt',  
 'family\_members\_under\_18',  
 'country\_of\_birth\_father',  
 'country\_of\_birth\_mother',  
 'country\_of\_birth\_self',  
 'citizenship',  
 'own\_business\_or\_self\_employed',  
 "fill\_inc\_questionnaire\_for\_veteran's\_admin",  
 'veterans\_benefits',  
 'year'])

Let's start by checking the modalties of our categorical variables :

# Check modalities of categorical varaiblies  
for c in cat :  
 print(pd.DataFrame({c : df[c].value\_counts()/len(df)}))

class\_of\_worker  
 Not in universe 0.502423  
 Private 0.361001  
 Self-employed-not incorporated 0.042326  
 Local government 0.039013  
 State government 0.021186  
 Self-employed-incorporated 0.016364  
 Federal government 0.014660  
 Never worked 0.002200  
 Without pay 0.000827  
 detailed\_industry\_recode  
 0 0.504624  
 33 0.085554  
 43 0.041514  
 4 0.029992  
 42 0.023471  
 45 0.022464  
 29 0.021095  
 37 0.020158  
 41 0.019867  
 32 0.018023  
 35 0.016940  
 39 0.014720  
 34 0.013858  
 44 0.012775  
 2 0.011006  
 11 0.008841  
 50 0.008540  
 40 0.008275  
 47 0.008240  
 38 0.008164  
 24 0.007533  
 12 0.006766  
 19 0.006746  
 30 0.005919  
 31 0.005904  
 25 0.005433  
 9 0.004977  
 22 0.004771  
 36 0.004736  
 13 0.004506  
 1 0.004145  
 48 0.003268  
 27 0.003137  
 49 0.003057  
 3 0.002822  
 21 0.002802  
 6 0.002777  
 5 0.002772  
 8 0.002757  
 16 0.002701  
 23 0.002631  
 18 0.002421  
 15 0.002265  
 7 0.002115  
 14 0.001479  
 46 0.000937  
 17 0.000787  
 28 0.000717  
 26 0.000637  
 51 0.000180  
 20 0.000160  
 10 0.000020  
 detailed\_occupation\_recode  
 0 0.504624  
 2 0.043885  
 26 0.039529  
 19 0.027130  
 29 0.025586  
 36 0.020775  
 34 0.020173  
 10 0.018459  
 16 0.017266  
 23 0.017001  
 12 0.016740  
 33 0.016665  
 3 0.016013  
 35 0.015878  
 38 0.015051  
 31 0.013527  
 32 0.012019  
 37 0.011197  
 8 0.010781  
 42 0.009613  
 30 0.009508  
 24 0.009257  
 17 0.008876  
 28 0.008325  
 41 0.007979  
 44 0.007979  
 43 0.006927  
 4 0.006836  
 13 0.006370  
 18 0.005428  
 39 0.005097  
 14 0.004671  
 5 0.004285  
 15 0.004085  
 27 0.003909  
 25 0.003844  
 9 0.003699  
 7 0.003664  
 11 0.003193  
 40 0.003092  
 1 0.002727  
 21 0.002671  
 6 0.002210  
 22 0.002060  
 45 0.000862  
 20 0.000356  
 46 0.000180  
 education  
 High school graduate 0.242614  
 Children 0.237677  
 Some college but no degree 0.139433  
 Bachelors degree(BA AB BS) 0.099562  
 7th and 8th grade 0.040131  
 10th grade 0.037875  
 11th grade 0.034462  
 Masters degree(MA MS MEng MEd MSW MBA) 0.032783  
 9th grade 0.031224  
 Associates degree-occup /vocational 0.026854  
 Associates degree-academic program 0.021867  
 5th or 6th grade 0.016424  
 12th grade no diploma 0.010655  
 1st 2nd 3rd or 4th grade 0.009017  
 Prof school degree (MD DDS DVM LLB JD) 0.008986  
 Doctorate degree(PhD EdD) 0.006330  
 Less than 1st grade 0.004105  
 enroll\_in\_edu\_inst\_last\_wk  
 Not in universe 0.936950  
 High school 0.034542  
 College or university 0.028508  
 marital\_stat  
 Never married 0.433459  
 Married-civilian spouse present 0.422117  
 Divorced 0.063702  
 Widowed 0.052440  
 Separated 0.017341  
 Married-spouse absent 0.007608  
 Married-A F spouse present 0.003333  
 major\_industry\_code  
 Not in universe or children 0.504624  
 Retail trade 0.085554  
 Manufacturing-durable goods 0.045183  
 Education 0.041514  
 Manufacturing-nondurable goods 0.034567  
 Finance insurance and real estate 0.030798  
 Construction 0.029992  
 Business and repair services 0.028323  
 Medical except hospital 0.023471  
 Public administration 0.023105  
 Other professional services 0.022464  
 Transportation 0.021095  
 Hospital services 0.019867  
 Wholesale trade 0.018023  
 Agriculture 0.015151  
 Personal services except private HH 0.014720  
 Social services 0.012775  
 Entertainment 0.008275  
 Communications 0.005919  
 Utilities and sanitary services 0.005904  
 Private household services 0.004736  
 Mining 0.002822  
 Forestry and fisheries 0.000937  
 Armed Forces 0.000180  
 major\_occupation\_code  
 Not in universe 0.504624  
 Adm support including clerical 0.074362  
 Professional specialty 0.069867  
 Executive admin and managerial 0.062624  
 Other service 0.060640  
 Sales 0.059056  
 Precision production craft & repair 0.052716  
 Machine operators assmblrs & inspctrs 0.031971  
 Handlers equip cleaners etc 0.020684  
 Transportation and material moving 0.020148  
 Farming forestry and fishing 0.015768  
 Technicians and related support 0.015126  
 Protective services 0.008325  
 Private household services 0.003909  
 Armed Forces 0.000180  
 race  
 White 0.838826  
 Black 0.102319  
 Asian or Pacific Islander 0.029245  
 Other 0.018329  
 Amer Indian Aleut or Eskimo 0.011282  
 hispanic\_origin  
 All other 0.861590  
 Mexican-American 0.040492  
 Mexican (Mexicano) 0.036256  
 Central or South American 0.019522  
 Puerto Rican 0.016605  
 Other Spanish 0.012455  
 Cuban 0.005643  
 NA 0.004380  
 Do not know 0.001534  
 Chicano 0.001524

sex  
 Female 0.521163  
 Male 0.478837  
 member\_of\_a\_labor\_union  
 Not in universe 0.904452  
 No 0.080362  
 Yes 0.015186  
 reason\_for\_unemployment  
 Not in universe 0.969577  
 Other job loser 0.010214  
 Re-entrant 0.010119  
 Job loser - on layoff 0.004892  
 Job leaver 0.002997  
 New entrant 0.002200  
 full\_or\_part\_time\_employment\_stat  
 Children or Armed Forces 0.620324  
 Full-time schedules 0.204167  
 Not in labor force 0.134360  
 PT for non-econ reasons usually FT 0.016650  
 Unemployed full-time 0.011583  
 PT for econ reasons usually PT 0.006059  
 Unemployed part- time 0.004225  
 PT for econ reasons usually FT 0.002631  
 tax\_filer\_stat  
 Nonfiler 0.376368  
 Joint both under 65 0.337720  
 Single 0.187552  
 Joint both 65+ 0.041760  
 Head of household 0.037219  
 Joint one under 65 & one 65+ 0.019381  
 region\_of\_previous\_residence  
 Not in universe 0.920946  
 South 0.024503  
 West 0.020419  
 Midwest 0.017918  
 Northeast 0.013557  
 Abroad 0.002656  
 state\_of\_previous\_residence  
 Not in universe 0.920946  
 California 0.008590  
 Utah 0.005328  
 Florida 0.004255  
 North Carolina 0.004070  
 ? 0.003548  
 Abroad 0.003363  
 Oklahoma 0.003137  
 Minnesota 0.002887  
 Indiana 0.002671  
 North Dakota 0.002501  
 New Mexico 0.002321  
 Michigan 0.002210  
 Alaska 0.001453  
 Kentucky 0.001223  
 Arizona 0.001218  
 New Hampshire 0.001213  
 Wyoming 0.001208  
 Colorado 0.001198  
 Oregon 0.001183  
 West Virginia 0.001158  
 Georgia 0.001138  
 Montana 0.001133  
 Alabama 0.001083  
 Ohio 0.001058  
 Texas 0.001047  
 Arkansas 0.001027  
 Mississippi 0.001022  
 Tennessee 0.001012  
 Pennsylvania 0.000997  
 New York 0.000977  
 Louisiana 0.000962  
 Vermont 0.000957  
 Iowa 0.000947  
 Illinois 0.000902  
 Nebraska 0.000892  
 Missouri 0.000877  
 Nevada 0.000872  
 Maine 0.000837  
 Massachusetts 0.000757  
 Kansas 0.000747  
 South Dakota 0.000692  
 Maryland 0.000682  
 Virginia 0.000632  
 Connecticut 0.000586  
 District of Columbia 0.000581  
 Wisconsin 0.000526  
 South Carolina 0.000476  
 New Jersey 0.000376  
 Delaware 0.000366  
 Idaho 0.000155  
 detailed\_household\_and\_family\_stat  
 Householder 0.266877  
 Child <18 never marr not in subfamily 0.252232  
 Spouse of householder 0.208973  
 Nonfamily householder 0.111331  
 Child 18+ never marr Not in a subfamily 0.060294  
 Secondary individual 0.030683  
 Other Rel 18+ ever marr not in subfamily 0.009803  
 Grandchild <18 never marr child of subfamily RP 0.009362  
 Other Rel 18+ never marr not in subfamily 0.008661  
 Grandchild <18 never marr not in subfamily 0.005343  
 Child 18+ ever marr Not in a subfamily 0.005077  
 Child under 18 of RP of unrel subfamily 0.003669  
 RP of unrelated subfamily 0.003433  
 Child 18+ ever marr RP of subfamily 0.003363  
 Other Rel <18 never marr child of subfamily RP 0.003288  
 Other Rel 18+ ever marr RP of subfamily 0.003288  
 Other Rel 18+ spouse of subfamily RP 0.003198  
 Child 18+ never marr RP of subfamily 0.002952  
 Other Rel <18 never marr not in subfamily 0.002927  
 Grandchild 18+ never marr not in subfamily 0.001879  
 In group quarters 0.000982  
 Child 18+ spouse of subfamily RP 0.000632  
 Other Rel 18+ never marr RP of subfamily 0.000471  
 Child <18 never marr RP of subfamily 0.000401  
 Spouse of RP of unrelated subfamily 0.000261  
 Child <18 ever marr not in subfamily 0.000180  
 Grandchild 18+ ever marr not in subfamily 0.000170  
 Grandchild 18+ spouse of subfamily RP 0.000050  
 Child <18 ever marr RP of subfamily 0.000045  
 Grandchild 18+ ever marr RP of subfamily 0.000045  
 Grandchild 18+ never marr RP of subfamily 0.000030  
 Other Rel <18 ever marr RP of subfamily 0.000030  
 Other Rel <18 never married RP of subfamily 0.000020  
 Other Rel <18 spouse of subfamily RP 0.000015  
 Child <18 spouse of subfamily RP 0.000010  
 Grandchild <18 never marr RP of subfamily 0.000010  
 Grandchild <18 ever marr not in subfamily 0.000010  
 Other Rel <18 ever marr not in subfamily 0.000005  
 detailed\_household\_summary\_in\_household  
 Householder 0.378277  
 Child under 18 never married 0.252733  
 Spouse of householder 0.209044  
 Child 18 or older 0.072322  
 Other relative of householder 0.048631  
 Nonrelative of householder 0.038096  
 Group Quarters- Secondary individual 0.000662  
 Child under 18 ever married 0.000236  
 migration\_code-change\_in\_msa  
 ? 0.499672  
 Nonmover 0.413677  
 MSA to MSA 0.053132  
 NonMSA to nonMSA 0.014089  
 Not in universe 0.007598  
 MSA to nonMSA 0.003959  
 NonMSA to MSA 0.003082  
 Abroad to MSA 0.002270  
 Not identifiable 0.002155  
 Abroad to nonMSA 0.000366

migration\_code-change\_in\_reg  
 ? 0.499672  
 Nonmover 0.413677  
 Same county 0.049177  
 Different county same state 0.014018  
 Not in universe 0.007598  
 Different region 0.005904  
 Different state same division 0.004967  
 Abroad 0.002656  
 Different division same region 0.002331  
 migration\_code-move\_within\_reg  
 ? 0.499672  
 Nonmover 0.413677  
 Same county 0.049177  
 Different county same state 0.014018  
 Not in universe 0.007598  
 Different state in South 0.004877  
 Different state in West 0.003403  
 Different state in Midwest 0.002762  
 Abroad 0.002656  
 Different state in Northeast 0.002160  
 live\_in\_this\_house\_1\_year\_ago  
 Not in universe under 1 year old 0.507270  
 Yes 0.413677  
 No 0.079054  
 migration\_prev\_res\_in\_sunbelt  
 ? 0.499672  
 Not in universe 0.421275  
 No 0.050054  
 Yes 0.028999  
 family\_members\_under\_18  
 Not in universe 0.722884  
 Both parents present 0.195381  
 Mother only present 0.064013  
 Father only present 0.009438  
 Neither parent present 0.008285  
 country\_of\_birth\_father  
 United-States 0.797718  
 Mexico 0.050160  
 ? 0.033645  
 Puerto-Rico 0.013432  
 Italy 0.011086  
 Canada 0.006916  
 Germany 0.006796  
 Dominican-Republic 0.006465  
 Poland 0.006074  
 Philippines 0.005784  
 Cuba 0.005638  
 El-Salvador 0.004922  
 China 0.004290  
 England 0.003974  
 Columbia 0.003077  
 India 0.002907  
 South Korea 0.002656  
 Ireland 0.002546  
 Jamaica 0.002321  
 Vietnam 0.002290  
 Guatemala 0.002230  
 Japan 0.001965  
 Portugal 0.001945  
 Ecuador 0.001900  
 Haiti 0.001759  
 Greece 0.001724  
 Peru 0.001679  
 Nicaragua 0.001579  
 Hungary 0.001534  
 Scotland 0.001238  
 Iran 0.001168  
 Yugoslavia 0.001088  
 Taiwan 0.000997  
 Cambodia 0.000982  
 Honduras 0.000972  
 France 0.000957  
 Outlying-U S (Guam USVI etc) 0.000797  
 Laos 0.000772  
 Trinadad&Tobago 0.000566  
 Thailand 0.000536  
 Hong Kong 0.000531  
 Holand-Netherlands 0.000256  
 Panama 0.000125  
 country\_of\_birth\_mother  
 United-States 0.804313  
 Mexico 0.049022  
 ? 0.030668  
 Puerto-Rico 0.012395  
 Italy 0.009242  
 Canada 0.007272  
 Germany 0.006927  
 Philippines 0.006170  
 Poland 0.005563  
 El-Salvador 0.005553  
 Cuba 0.005553  
 Dominican-Republic 0.005528  
 England 0.004526  
 China 0.003809  
 Columbia 0.003067  
 South Korea 0.003052  
 Ireland 0.003002  
 India 0.002912  
 Vietnam 0.002371  
 Japan 0.002351  
 Jamaica 0.002270  
 Guatemala 0.002225  
 Ecuador 0.001879  
 Peru 0.001779  
 Haiti 0.001769  
 Portugal 0.001714  
 Nicaragua 0.001509  
 Hungary 0.001489  
 Greece 0.001308  
 Scotland 0.001208  
 Taiwan 0.001113  
 Honduras 0.001093  
 France 0.001063  
 Iran 0.000992  
 Yugoslavia 0.000887  
 Cambodia 0.000787  
 Outlying-U S (Guam USVI etc) 0.000787  
 Laos 0.000777  
 Thailand 0.000616  
 Hong Kong 0.000536  
 Trinadad&Tobago 0.000496  
 Holand-Netherlands 0.000246  
 Panama 0.000160  
 country\_of\_birth\_self  
 United-States 0.887061  
 Mexico 0.028904  
 ? 0.017006  
 Puerto-Rico 0.007017  
 Germany 0.004265  
 Philippines 0.004235  
 Cuba 0.004195  
 Canada 0.003508  
 Dominican-Republic 0.003458  
 El-Salvador 0.003453  
 China 0.002396  
 South Korea 0.002361  
 England 0.002290  
 Columbia 0.002175  
 Italy 0.002100  
 India 0.002045  
 Vietnam 0.001960  
 Poland 0.001910  
 Guatemala 0.001724  
 Japan 0.001699  
 Jamaica 0.001604  
 Peru 0.001343  
 Ecuador 0.001293  
 Haiti 0.001143  
 Nicaragua 0.001093  
 Taiwan 0.001007  
 Portugal 0.000872  
 Iran 0.000787  
 Greece 0.000737  
 Honduras 0.000722  
 Ireland 0.000677  
 France 0.000606  
 Outlying-U S (Guam USVI etc) 0.000596  
 Thailand 0.000566  
 Laos 0.000526  
 Hong Kong 0.000501  
 Cambodia 0.000476  
 Hungary 0.000396  
 Scotland 0.000376  
 Trinadad&Tobago 0.000331  
 Yugoslavia 0.000331  
 Panama 0.000140  
 Holand-Netherlands 0.000115  
 citizenship  
 Native- Born in the United States 0.887076  
 Foreign born- Not a citizen of U S 0.067165  
 Foreign born- U S citizen by naturalization 0.029345  
 Native- Born abroad of American Parent(s) 0.008801  
 Native- Born in Puerto Rico or U S Outlying 0.007613  
 own\_business\_or\_self\_employed  
 0 0.905520  
 2 0.080958  
 1 0.013522  
 fill\_inc\_questionnaire\_for\_veteran's\_admin  
 Not in universe 0.990056  
 No 0.007984  
 Yes 0.001960  
 veterans\_benefits  
 2 0.752445  
 0 0.237612  
 1 0.009944

year  
 94 0.500328  
 95 0.499672

Some categorical features are **purely nominal**-having multiple modalities (with modality **?** for nan values) and others are **ordinal columns** like **education** and **year**:

# Ediucation modalities  
df['education'].unique(), df['education'].nunique()

(array([' High school graduate', ' Some college but no degree',  
 ' 10th grade', ' Children', ' Bachelors degree(BA AB BS)',  
 ' Masters degree(MA MS MEng MEd MSW MBA)', ' Less than 1st grade',  
 ' Associates degree-academic program', ' 7th and 8th grade',  
 ' 12th grade no diploma', ' Associates degree-occup /vocational',  
 ' Prof school degree (MD DDS DVM LLB JD)', ' 5th or 6th grade',  
 ' 11th grade', ' Doctorate degree(PhD EdD)', ' 9th grade',  
 ' 1st 2nd 3rd or 4th grade'], dtype=object),  
 17)

# Year modalities  
df['year'].unique(), df['year'].nunique()

(array([' 95', ' 94'], dtype=object), 2)

We can tell Pandas about a suitable ordering of these levels like so:

# Setting the order of education variable  
education = ' Children',' Less than 1st grade',' 1st 2nd 3rd or 4th grade',' 5th or 6th grade',\  
' 7th and 8th grade',' 9th grade',' 10th grade',' 11th grade', ' 12th grade no diploma',\  
' High school graduate', ' Associates degree-academic program',' Associates degree-occup /vocational',\  
' Prof school degree (MD DDS DVM LLB JD)',' Some college but no degree',' Bachelors degree(BA AB BS)',\  
' Masters degree(MA MS MEng MEd MSW MBA)',' Doctorate degree(PhD EdD)'  
len(education)

17

# Setting the order of year variaable  
year = '94', '95'

# apply the defined ordering fot our data :  
df['education'] = df['education'].astype('category')  
df['education'].cat.set\_categories(education, ordered=True, inplace=True)  
  
df['year'] = df['year'].astype('category')  
df['year'].cat.set\_categories(year, ordered=True, inplace=True)  
  
#Same for test set :  
test['education'] = test['education'].astype('category')  
test['education'].cat.set\_categories(education, ordered=True, inplace=True)  
  
test['year'] = test['year'].astype('category')  
test['year'].cat.set\_categories(year, ordered=True, inplace=True)

/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/pandas/core/arrays/categorical.py:2631: FutureWarning: The `inplace` parameter in pandas.Categorical.set\_categories is deprecated and will be removed in a future version. Removing unused categories will always return a new Categorical object.  
 res = method(\*args, \*\*kwargs)

Lets check our continous features: The **describe()** method shows a summary of the numerical attributes

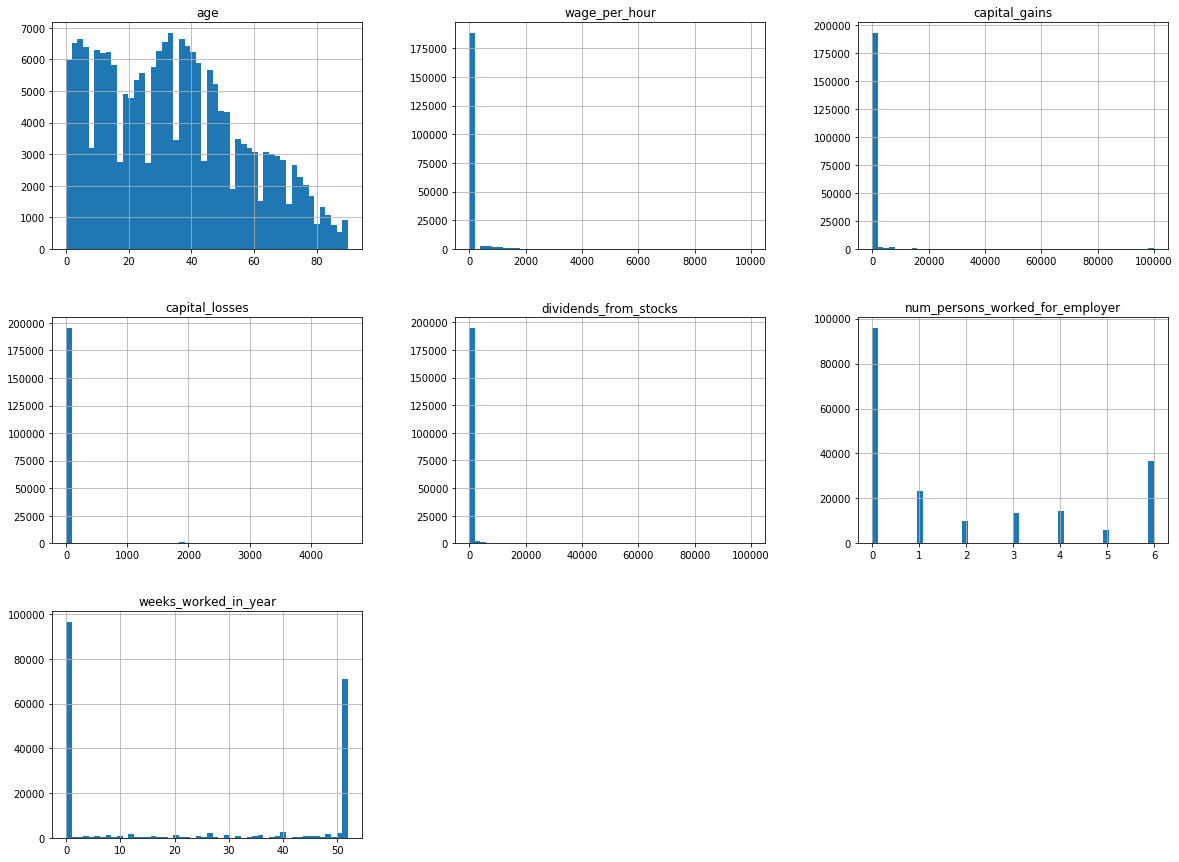
df[cont].describe()

age wage\_per\_hour capital\_gains capital\_losses \  
count 199523.000000 199523.000000 199523.00000 199523.000000   
mean 34.494199 55.426908 434.71899 37.313788   
std 22.310895 274.896454 4697.53128 271.896428   
min 0.000000 0.000000 0.00000 0.000000   
25% 15.000000 0.000000 0.00000 0.000000   
50% 33.000000 0.000000 0.00000 0.000000   
75% 50.000000 0.000000 0.00000 0.000000   
max 90.000000 9999.000000 99999.00000 4608.000000   
  
 dividends\_from\_stocks num\_persons\_worked\_for\_employer \  
count 199523.000000 199523.000000   
mean 197.529533 1.956180   
std 1984.163658 2.365126   
min 0.000000 0.000000   
25% 0.000000 0.000000   
50% 0.000000 1.000000   
75% 0.000000 4.000000   
max 99999.000000 6.000000   
  
 weeks\_worked\_in\_year   
count 199523.000000   
mean 23.174897   
std 24.411488   
min 0.000000   
25% 0.000000   
50% 8.000000   
75% 52.000000   
max 52.000000

The **count**, **mean**, **min**, and **max** rows are self-explanatory.The **std** row shows the standard deviation, which measures how dispersed the values are. The 25%, 50%, and 75% rows show the corresponding percentiles.

We plot a histogram for each numerical attribute :

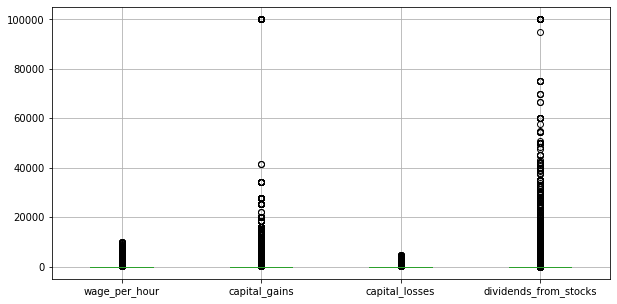
%matplotlib inline   
import matplotlib.pyplot as plt  
df[cont].hist(bins=50, figsize=(20,15))  
plt.show()



* We can see that these attributes have very **different scales**.
* Some numerical varaibles are countinous like **age** and others are discrete and finite like **weeks\_worked\_in\_year** or infinete **num\_persons\_worked\_for\_employer**.
* Some features as **wage\_per\_hour**,**capital\_gains**,**capital\_losses**,**dividends\_from\_stocks** are tail-heavy: they extend much farther to the median right with high coefficient of variation :

df[cont].boxplot(column=['wage\_per\_hour','capital\_gains','capital\_losses','dividends\_from\_stocks'],  
 figsize=(10,5))

<matplotlib.axes.\_subplots.AxesSubplot at 0x125ab4190>



We can see the presence of **extreme values** for those features.

Using the **skewness value**, which explains the extent to which the data is normally distributed, in order to confirm that. Ideally, the skewness value should be between -1 and +1, and any major deviation from this range indicates the presence of extreme values.

We can calculate the skwenss value :

# skewness value  
df[['wage\_per\_hour','capital\_gains','capital\_losses','dividends\_from\_stocks']].skew()

wage\_per\_hour 8.935097  
capital\_gains 18.990822  
capital\_losses 7.632565  
dividends\_from\_stocks 27.786502  
dtype: float64

Using the **IQR score**, let's see the number of obseravtions that are not in the (Q1 - 1.5 IQR) and (Q3 + 1.5 IQR) range :

# IQR score  
Q1 = df[['wage\_per\_hour','capital\_gains','capital\_losses','dividends\_from\_stocks']].quantile(0.25)  
Q3 = df[['wage\_per\_hour','capital\_gains','capital\_losses','dividends\_from\_stocks']].quantile(0.75)  
IQR = Q3 - Q1  
print(IQR)

wage\_per\_hour 0.0  
capital\_gains 0.0  
capital\_losses 0.0  
dividends\_from\_stocks 0.0  
dtype: float64

# number of observation out of the definied range  
out = df[['wage\_per\_hour','capital\_gains','capital\_losses','dividends\_from\_stocks']]  
df\_out = out[((out < (Q1 - 1.5 \* IQR)) |(out > (Q3 + 1.5 \* IQR))).any(axis=1)]  
  
out.shape, df\_out.shape, df\_out.shape[0]/out.shape[0]

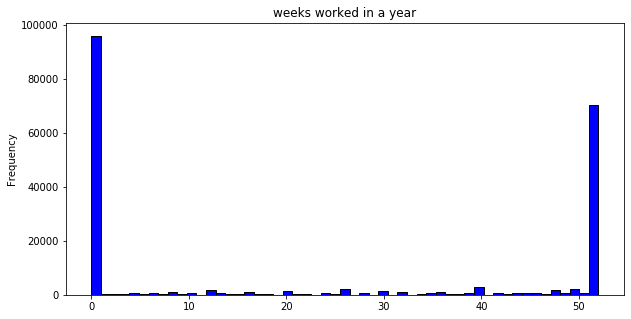
((199523, 4), (38859, 4), 0.1947595014108649)

From 199.523 observation of the selcted features, 38.859 records (19%) represent extrem values.

For **weeks\_worked\_in\_year** :

df['weeks\_worked\_in\_year'].plot( kind='hist',  
 bins=53,  
 edgecolor='black',  
 color='blue',  
 title='weeks worked in a year',  
 figsize=(10,5))

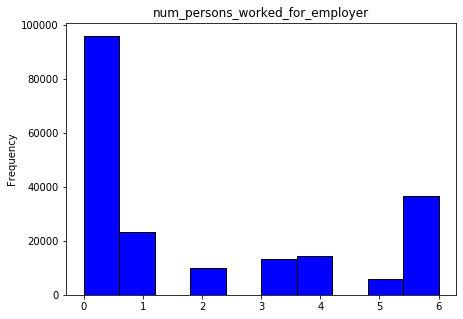
<matplotlib.axes.\_subplots.AxesSubplot at 0x12311dd50>



For **num\_persons\_worked\_for\_employer** :

df['num\_persons\_worked\_for\_employer'].plot( kind='hist',  
   
 edgecolor='black',  
 color='blue',  
 title='num\_persons\_worked\_for\_employer',  
 figsize=(7,5))

<matplotlib.axes.\_subplots.AxesSubplot at 0x1236b43d0>



We notice an increase in the 7th bins **num\_persons\_worked\_for\_employer=6**. Check if this variable is capped ?

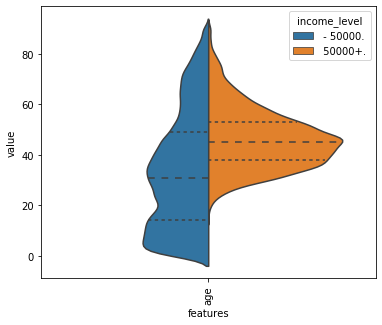
# Exploratory data analysis

* Starting with **numerical variables** :

import seaborn as sns

data\_dia = df[dep\_var]  
data = df[['age']]  
data = pd.concat([data\_dia,data],axis=1)  
data = pd.melt(data,id\_vars="income\_level",  
 var\_name="features",  
 value\_name='value')  
plt.figure(figsize=(6,5))  
sns.violinplot(x="features", y="value", hue="income\_level", data=data,split=True, inner="quartile")  
plt.xticks(rotation=90)

(array([0]), <a list of 1 Text xticklabel objects>)



For the **age** feature, we can see that the medians of the income levels +/- 50k look separated. The income level of +50k with a median of 50 years old has a lower interquntile range (IQR) with value spread of 10 years. Whereas The income level of -50k has a median of 30 years old has and interquantile range (IQR) of 40 years. So, **age** can be good for classification.

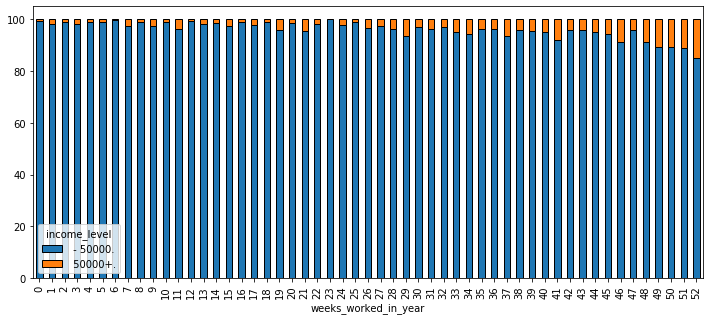
Let's look at the **weeks\_worked\_in\_year** feature :

# get the number of income class in each week  
weeks\_worked\_in\_year = df.groupby(["weeks\_worked\_in\_year", "income\_level"])\  
 .size()\  
 .groupby(level=0).apply(lambda x: 100\*x/x.sum()).unstack()  
  
# print the percentage class for the first and last weeks   
weeks\_worked\_in\_year.iloc[[0,1,2, -3,-2,-1]]

income\_level - 50000. 50000+.  
weeks\_worked\_in\_year   
0 99.379057 0.620943  
1 98.275862 1.724138  
2 98.908297 1.091703  
50 89.279514 10.720486  
51 89.010989 10.989011  
52 85.199249 14.800751

weeks\_worked\_in\_year.plot(kind='bar',   
 stacked=True,  
 edgecolor='black',   
 figsize=(12,5))

<matplotlib.axes.\_subplots.AxesSubplot at 0x125e2ded0>



We can see that the propotion of people making more than 50k a year is increasing with the number of working weeks in a given year where it can reach more than 14% for those working 52 weeks . However, the -50k level of income is representing the higher propotion regardless of the number of working weeks. We notice that among those how don't work at all, 0.6% still make more than 50k a year.

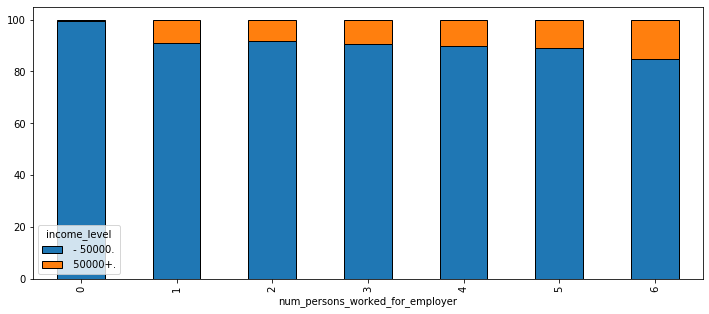
Let's look at **num\_persons\_worked\_for\_employer** :

num\_persons\_worked\_for\_employer = df.groupby(["num\_persons\_worked\_for\_employer", "income\_level"])\  
 .size()\  
 .groupby(level=0).apply(lambda x: 100\*x/x.sum()).unstack()  
  
num\_persons\_worked\_for\_employer

income\_level - 50000. 50000+.  
num\_persons\_worked\_for\_employer   
0 99.379057 0.620943  
1 90.942923 9.057077  
2 91.687333 8.312667  
3 90.763501 9.236499  
4 89.776758 10.223242  
5 88.980944 11.019056  
6 84.990825 15.009175

num\_persons\_worked\_for\_employer.plot(kind='bar',   
 stacked=True,  
 edgecolor='black',   
 figsize=(12,5))

<matplotlib.axes.\_subplots.AxesSubplot at 0x125fbe4d0>

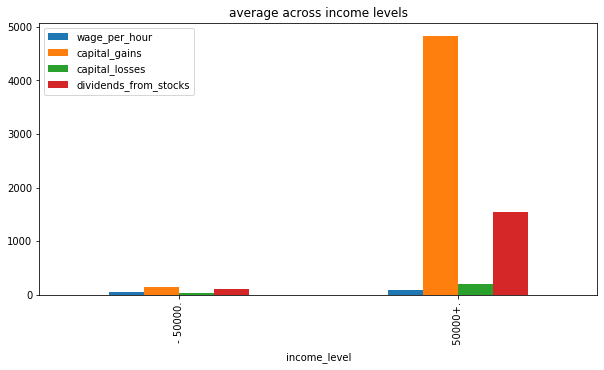


The proportion of +50k income level increases with the number of the num\_preson\_worked\_for\_employer where it reaches **16% for num\_preson\_worked\_for\_employer= 6**.

Let's see the average of **wage\_per\_hour**,**capital\_gains**,**capital\_losses**,**dividends\_from\_stocks** across the income levels :

avg = df[['wage\_per\_hour','capital\_gains','capital\_losses','dividends\_from\_stocks','income\_level']]\  
.groupby('income\_level')\  
.mean()  
  
avg.plot(kind='bar', title = 'average across income levels', figsize=(10,5))  
  
avg

wage\_per\_hour capital\_gains capital\_losses \  
income\_level   
 - 50000. 53.692526 143.848013 27.003730   
 50000+. 81.640284 4830.930060 193.139557   
  
 dividends\_from\_stocks   
income\_level   
 - 50000. 107.816518   
 50000+. 1553.448070



We can see that people making more than 50k a year, have on average, **higher wage per hour**,**higher return on capital asset** and **dividends from stock options**.

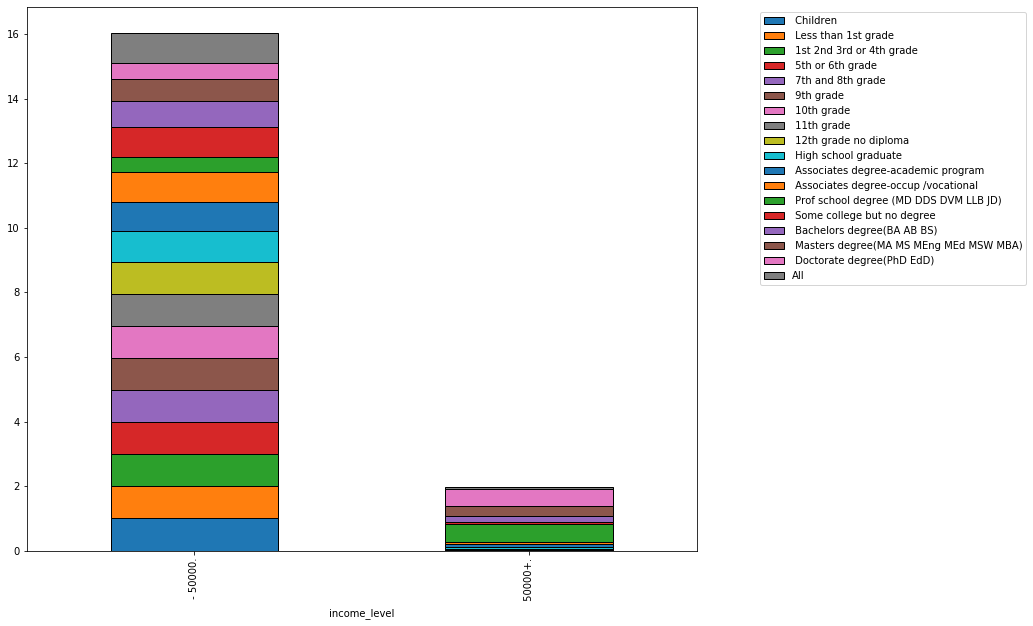
* Next, let's analyse some **categorical variables** :

# Education variable  
pd.crosstab(df['income\_level'],   
 df['education'],  
 margins = True,  
 normalize = 'columns').style.format('{:.2%}')

<pandas.io.formats.style.Styler at 0x125f7b790>

pd.crosstab(df['income\_level'],   
 df['education'],  
 margins = True,  
 normalize = 'columns').plot(kind='bar',stacked=True, edgecolor='black',   
 figsize=(12,10))  
  
plt.legend(bbox\_to\_anchor=(1.5, 1.0))

<matplotlib.legend.Legend at 0x125b9d110>



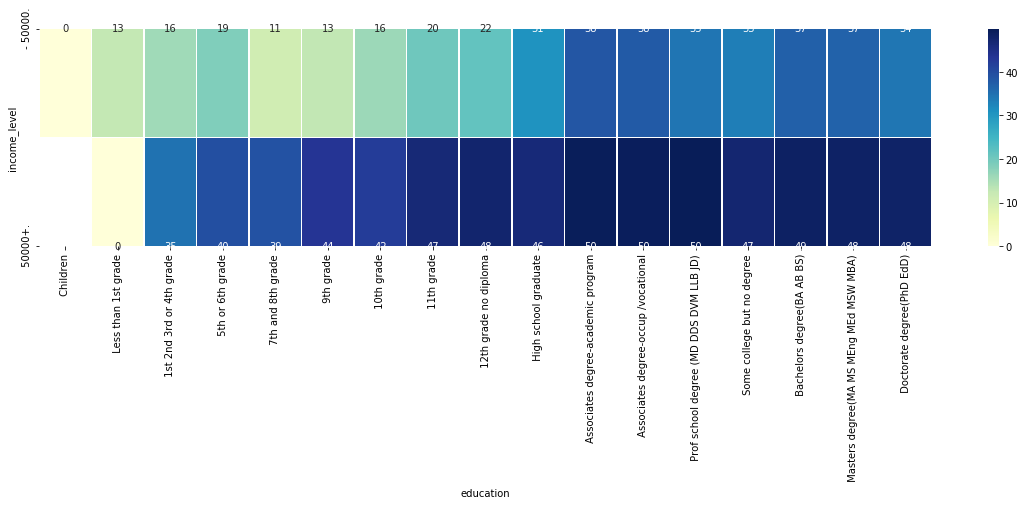
We can see the effect of education on income level where more than 50% of **Prof school degree** and **Doctorate degree** earn more than 50k a year. On the other hand, the majority of people (more than 90%) with **no degree** earn less than 50k a year.

Let's further this analysis and see the effect of **education** and **the number of working weeks** :

pd.crosstab(df['income\_level'],   
 df['education'],  
 values = df['weeks\_worked\_in\_year'],  
 aggfunc = 'mean').round(2)

education Children Less than 1st grade 1st 2nd 3rd or 4th grade \  
income\_level   
 - 50000. 0.0 12.69 15.93   
 50000+. NaN 0.00 35.00   
  
education 5th or 6th grade 7th and 8th grade 9th grade 10th grade \  
income\_level   
 - 50000. 18.56 11.12 13.01 16.23   
 50000+. 39.64 39.00 43.47 42.29   
  
education 11th grade 12th grade no diploma High school graduate \  
income\_level   
 - 50000. 20.42 21.59 30.88   
 50000+. 46.59 47.71 46.53   
  
education Associates degree-academic program \  
income\_level   
 - 50000. 38.48   
 50000+. 49.63   
  
education Associates degree-occup /vocational \  
income\_level   
 - 50000. 37.99   
 50000+. 49.46   
  
education Prof school degree (MD DDS DVM LLB JD) \  
income\_level   
 - 50000. 34.55   
 50000+. 49.88   
  
education Some college but no degree Bachelors degree(BA AB BS) \  
income\_level   
 - 50000. 33.35 37.24   
 50000+. 47.17 48.66   
  
education Masters degree(MA MS MEng MEd MSW MBA) \  
income\_level   
 - 50000. 36.94   
 50000+. 48.47   
  
education Doctorate degree(PhD EdD)   
income\_level   
 - 50000. 34.51   
 50000+. 48.23

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
plt.figure(figsize=(20, 4))  
sns.heatmap(  
 pd.crosstab(df['income\_level'],   
 df['education'],  
 values = df['weeks\_worked\_in\_year'],  
 aggfunc = 'mean').round(1)   
 ,annot = True  
 ,linewidths=.5  
 ,cmap="YlGnBu"  
   
)  
plt.show()



We can see that earning more than 50k a year demands high level of education but also lot of hard work !

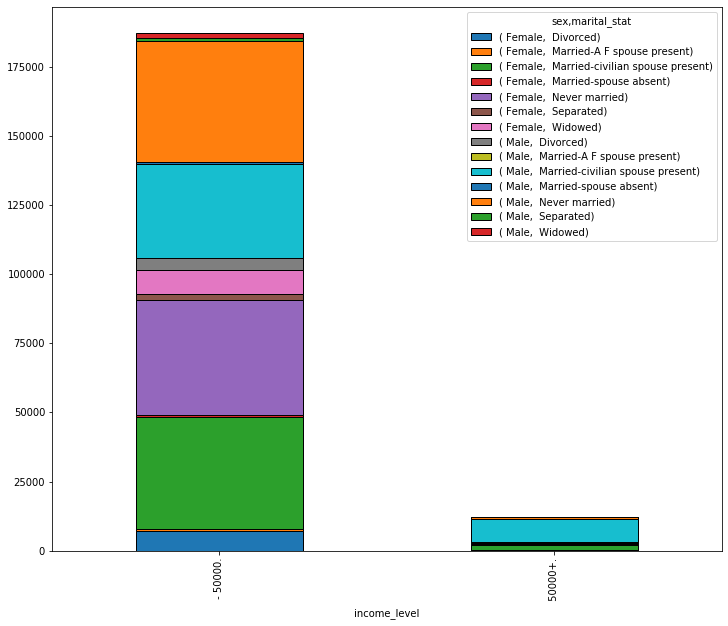
Let's analyse the effect of **sex** and **marital\_stat** on income level :

pd.crosstab(df['income\_level'],   
 [df['sex'],df['marital\_stat']],  
 margins = True,  
 normalize = 'columns').style.format('{:.2%}')

<pandas.io.formats.style.Styler at 0x123262f50>

pd.crosstab(df['income\_level'],   
 [df['sex'],df['marital\_stat']]).plot(kind='bar',stacked=True, edgecolor='black',  
   
 figsize=(12,10))

<matplotlib.axes.\_subplots.AxesSubplot at 0x12932b290>



We can see that the highest proportion of people earning less than 50k a year are mostly **female Married-A F spouse present** or **never married** and **seperated male**. On the other hand, **male Married-civilian spouse present** represent the highest propotion on the +50k income level.

We can further the analysis more as we have got many interesting features with several modalities but for now let's see how machine learning models can help us understanding more our data.

# Data preparation

We set **the feature vector** and **the target variable** :

# setting feature vector and target variable for the train set  
  
X = df.drop(['income\_level'], axis = 1)  
y = df['income\_level']  
  
# setting feature vector and target variable for the test set  
test\_x = test.drop(['income\_level'], axis = 1)  
test\_y = test['income\_level']

# Cheching the result  
df.shape, X.shape, y.shape

((199523, 41), (199523, 40), (199523,))

We will keep the provided **test set** hidden and will use it as a realtime dataset when we make our model on production in order to avoid the risk of **data snooping**.

For that, we will be using a validation set derived from our training set (30%). Scikit-Learn provides a few functions to split datasets into multiple subsets in various ways. The simplest function is **train\_test\_split()**.

Since we have an imbalanced dataset, we can't considered purely random sampling methods. For that, we do stratified sampling based on the **income level**.

from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_val, y\_train, y\_val = train\_test\_split(X,   
 y,   
 test\_size=0.3,   
 random\_state=12,  
 stratify=y)

# Checking the train and validation set  
X\_train.shape, X\_val.shape

((139666, 40), (59857, 40))

# Checking the income level proportion  
y\_train.value\_counts(normalize = True), y\_val.value\_counts(normalize = True)

( - 50000. 0.937945  
 50000+. 0.062055  
 Name: income\_level, dtype: float64,  
 - 50000. 0.937935  
 50000+. 0.062065  
 Name: income\_level, dtype: float64)

What if we didn't stratify with respect to income level ?

We can compare the income level proportions in the **overall dataset**, in the **test set** generated with stratified sampling, and in **a test set** generated using purely random sampling.

def income\_cat\_proportions(data):  
 return data["income\_level"].value\_counts() / len(data)  
  
train\_set, test\_set = train\_test\_split(df, test\_size=0.3, random\_state=12)  
  
train\_set, test\_set\_strat = train\_test\_split(df, test\_size=0.3, random\_state=12,stratify=df['income\_level'])  
  
compare\_props = pd.DataFrame({  
 "Overall": income\_cat\_proportions(df),  
 "Stratified": income\_cat\_proportions(test\_set\_strat),  
 "Random": income\_cat\_proportions(test\_set),  
}).sort\_index()  
compare\_props["Rand. %error"] = 100 \* compare\_props["Random"] / compare\_props["Overall"] - 100  
compare\_props["Strat. %error"] = 100 \* compare\_props["Stratified"] / compare\_props["Overall"] - 100

As we can see, the test set generated using stratified sampling has income level proportions almost identical to those in the full dataset, whereas the test set generated using purely random sampling is skewed.

compare\_props

Overall Stratified Random Rand. %error Strat. %error  
 - 50000. 0.937942 0.937935 0.939305 0.145356 -0.000701  
 50000+. 0.062058 0.062065 0.060695 -2.196901 0.010601

Now that we defined our training set, It’s time to prepare the data for our machine Learning algorithms.

* Data cleaning :

We have seen previously that we don't have any **missing values**. For some cataegorical features, we assumed that the **?** modality is encoded for NaN values.

* Handling Text and Categorical Attributes :

Strating with the target variable **income\_level**, we use **LabelEncoder()** to encode target labels with value between 0 and n\_classes-1 = 1.

We have seen also that we have some ordinal variable as **education** and **year**, so we use **OrdinalEncoder** to encode the categorical features as an integer array. The results in a single column of integers (0 to n\_categories - 1) per feature.

Since the remaining categorical features have several modalities per feature, we use also **OrdinalEncoder** instead of **OneHotEncoder**.

Working with **OneHotEncoder** leads, in our case, to high memory consumption. We can combine **OneHotEncoder** and **PCA** : The benefit in PCA is that combination of N attributes is better than any individual attribute. And the disadvantage is in harder explanation what exactly that PCA component means. Therefore, for this work, we will sacrifice a bit of predictive power to get more understandable model.

# categorical variables encoding  
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder  
  
# For the traget varaible  
le = LabelEncoder()  
y\_train = le.fit\_transform(y\_train) #fit on training set  
y\_val = le.transform(y\_val)   
test\_y = le.transform(test\_y)   
  
  
# For categorical features :  
Or = OrdinalEncoder(handle\_unknown='use\_encoded\_value', unknown\_value = -1)  
for c in cat :   
 X\_train[c] = Or.fit\_transform(np.array(X\_train[c]).reshape(-1,1).astype(str)) #fit on training set  
 X\_val[c] = Or.transform(np.array(X\_val[c]).reshape(-1,1).astype(str))  
 test\_x[c] = Or.transform(np.array(test\_x[c]).reshape(-1,1).astype(str))

# Cheking categorical features encoding  
X\_train[cat].head(3)

class\_of\_worker detailed\_industry\_recode detailed\_occupation\_recode \  
88634 4.0 31.0 14.0   
148296 8.0 37.0 12.0   
163953 3.0 0.0 0.0   
  
 education enroll\_in\_edu\_inst\_last\_wk marital\_stat \  
88634 9.0 2.0 2.0   
148296 12.0 2.0 2.0   
163953 10.0 2.0 4.0   
  
 major\_industry\_code major\_occupation\_code race hispanic\_origin \  
88634 2.0 0.0 4.0 0.0   
148296 12.0 2.0 4.0 0.0   
163953 14.0 6.0 4.0 0.0   
  
 ... migration\_prev\_res\_in\_sunbelt family\_members\_under\_18 \  
88634 ... 0.0 4.0   
148296 ... 2.0 4.0   
163953 ... 0.0 0.0   
  
 country\_of\_birth\_father country\_of\_birth\_mother \  
88634 40.0 40.0   
148296 40.0 40.0   
163953 40.0 40.0   
  
 country\_of\_birth\_self citizenship own\_business\_or\_self\_employed \  
88634 40.0 4.0 2.0   
148296 40.0 4.0 0.0   
163953 40.0 4.0 0.0   
  
 fill\_inc\_questionnaire\_for\_veteran's\_admin veterans\_benefits year   
88634 1.0 2.0 0.0   
148296 1.0 2.0 0.0   
163953 1.0 0.0 0.0   
  
[3 rows x 33 columns]

# Cheking target feature encoding  
set(y\_train)

{0, 1}

* Feature Scaling :

We saw previously that out numerical inputs have different scales like the **weeks\_worked\_in\_year** and **capital\_gains**. We will be using **StandardScaler** since standardization is much less affected by outliers.

from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
  
for c in cont:  
 X\_train[c] = scaler.fit\_transform(np.array(X\_train[c]).reshape(-1,1)) # fir on the train set  
 X\_val[c] = scaler.transform(np.array(X\_val[c]).reshape(-1,1))  
 test\_x[c] = scaler.transform(np.array(test\_x[c]).reshape(-1,1))

#checking the standardization  
X\_train[cont].head(3)

age wage\_per\_hour capital\_gains capital\_losses \  
88634 0.334269 -0.201648 -0.092139 -0.137611   
148296 0.468715 -0.201648 -0.092139 -0.137611   
163953 -1.368713 -0.201648 -0.092139 -0.137611   
  
 dividends\_from\_stocks num\_persons\_worked\_for\_employer \  
88634 -0.069026 1.705234   
148296 -0.098703 -0.405865   
163953 -0.098703 -0.828085   
  
 weeks\_worked\_in\_year   
88634 1.177773   
148296 1.177773   
163953 -0.950488

So far, we have handled the **categorical columns** and the **numerical columns** :

# Checking the training set  
with pd.option\_context('display.max\_rows', None, 'display.max\_columns', None):   
 display(X\_train.head(3))  
  
X\_train.shape, X\_val.shape, test\_x.shape

age class\_of\_worker detailed\_industry\_recode \  
88634 0.334269 4.0 31.0   
148296 0.468715 8.0 37.0   
163953 -1.368713 3.0 0.0   
  
 detailed\_occupation\_recode education wage\_per\_hour \  
88634 14.0 9.0 -0.201648   
148296 12.0 12.0 -0.201648   
163953 0.0 10.0 -0.201648   
  
 enroll\_in\_edu\_inst\_last\_wk marital\_stat major\_industry\_code \  
88634 2.0 2.0 2.0   
148296 2.0 2.0 12.0   
163953 2.0 4.0 14.0   
  
 major\_occupation\_code race hispanic\_origin sex \  
88634 0.0 4.0 0.0 1.0   
148296 2.0 4.0 0.0 0.0   
163953 6.0 4.0 0.0 1.0   
  
 member\_of\_a\_labor\_union reason\_for\_unemployment \  
88634 1.0 3.0   
148296 1.0 3.0   
163953 1.0 3.0   
  
 full\_or\_part\_time\_employment\_stat capital\_gains capital\_losses \  
88634 1.0 -0.092139 -0.137611   
148296 0.0 -0.092139 -0.137611   
163953 0.0 -0.092139 -0.137611   
  
 dividends\_from\_stocks tax\_filer\_stat region\_of\_previous\_residence \  
88634 -0.069026 2.0 3.0   
148296 -0.098703 2.0 3.0   
163953 -0.098703 4.0 3.0   
  
 state\_of\_previous\_residence detailed\_household\_and\_family\_stat \  
88634 36.0 37.0   
148296 36.0 37.0   
163953 36.0 8.0   
  
 detailed\_household\_summary\_in\_household migration\_code-change\_in\_msa \  
88634 7.0 0.0   
148296 7.0 7.0   
163953 2.0 0.0   
  
 migration\_code-change\_in\_reg migration\_code-move\_within\_reg \  
88634 0.0 0.0   
148296 6.0 7.0   
163953 0.0 0.0   
  
 live\_in\_this\_house\_1\_year\_ago migration\_prev\_res\_in\_sunbelt \  
88634 1.0 0.0   
148296 2.0 2.0   
163953 1.0 0.0   
  
 num\_persons\_worked\_for\_employer family\_members\_under\_18 \  
88634 1.705234 4.0   
148296 -0.405865 4.0   
163953 -0.828085 0.0   
  
 country\_of\_birth\_father country\_of\_birth\_mother \  
88634 40.0 40.0   
148296 40.0 40.0   
163953 40.0 40.0   
  
 country\_of\_birth\_self citizenship own\_business\_or\_self\_employed \  
88634 40.0 4.0 2.0   
148296 40.0 4.0 0.0   
163953 40.0 4.0 0.0   
  
 fill\_inc\_questionnaire\_for\_veteran's\_admin veterans\_benefits \  
88634 1.0 2.0   
148296 1.0 2.0   
163953 1.0 0.0   
  
 weeks\_worked\_in\_year year   
88634 1.177773 0.0   
148296 1.177773 0.0   
163953 -0.950488 0.0

((139666, 40), (59857, 40), (99762, 40))

# Data modeling

* Selecting a Performance Measure :

**Accuracy** is the simplest way to measure the effectiveness of a classification task, and it's the percentage of correct predictions over all predictions. In other words, in a binary classification task, you can calculate this by adding the number of True Positives (TPs) and True Negatives (TNs) and dividing them by a tally of all predictions made. As with regression metrics, you can measure accuracy for both train and test to gauge **overfitting**.

But, we can get an accuracy of 94%, which sounds pretty good, but it turns out we are always predicting **-50k**! In other words, even if we get high accuracy, it is meaningless unless we are predicting accurately for the least represented class, **+50k**.

For this reasing, we will be using **F1-score**. The **F1-score** is also called the harmonic average of precision and recall because it's calculated like this: 2TP / 2TP + FP + FN. Since it includes both precision and recall metrics, which pertain to the proportion of true positives, it's a good metric choice to use when the dataset is **imbalanced**, and we don't prefer either precision or recall.

* Base model :

Let's start with **Decision tree ensembles**.

A decision tree asks a series of binary (that is, yes or no) questions about the data. After each question the data at that part of the tree is split between a "yes" and a "no" branch. After one or more questions, either a prediction can be made on the basis of all previous answers or another question is required.

We illustarte a tree classification using **4 leaf nodes**.

from sklearn.tree import DecisionTreeClassifier, plot\_tree  
  
m = DecisionTreeClassifier(max\_leaf\_nodes=4, random\_state=14) # to plot the tree classification  
m.fit(X\_train, y\_train)

DecisionTreeClassifier(max\_leaf\_nodes=4, random\_state=14)

# to get the class output  
m.classes\_

array([0, 1])

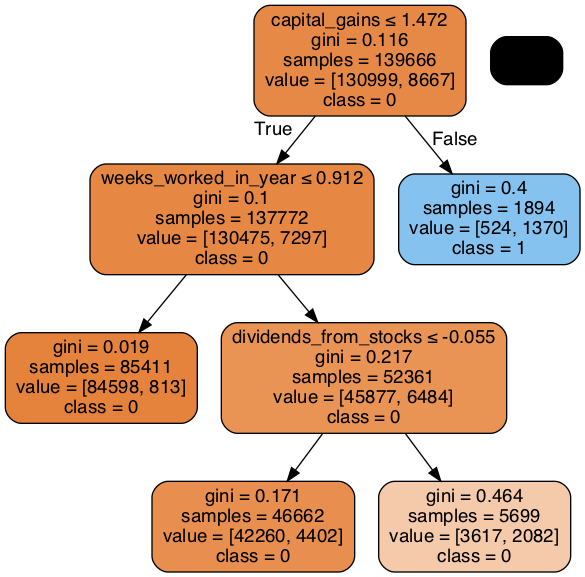
!pip install pydotplus

Requirement already satisfied: pydotplus in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (2.0.2)
  
Requirement already satisfied: pyparsing>=2.0.1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from pydotplus) (3.0.4)

!pip install graphviz

Requirement already satisfied: graphviz in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (0.16)

from sklearn.tree import export\_graphviz  
from io import StringIO  
from IPython.display import Image   
import pydotplus  
feature\_cols = X\_train.columns  
  
  
dot\_data = StringIO()  
export\_graphviz(m, out\_file=dot\_data,   
 filled=True, rounded=True,  
 special\_characters=True,feature\_names = feature\_cols, class\_names=['0','1'])  
graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())   
Image(graph.create\_png())



The top node represents the **initial model** before any splits have been done, when all the data is in the initial income levels. This is the simplest possible model. It is the result of asking zero questions and will always predict the more represented class which is -50k. We use the **Gini method** to create split points. The strategy is to select each pair of adjacent values as a possible split-point and the point with smaller gini index chosen as the splitting point. In our case, the **capital gains** at 1.47 was choosen first.

Moving down and to the left, this node shows us that there were 130,999 records for income level of -50k where **capital gains** was less than 1.47. The class predicted is -50k in this case. Moving down and to the right from the initial model takes us to the records where **capital gains** was greater than 1.47. The class predicted is +50k in this case where 1370 records have an income of +50k and **capital gains** >0.4

The bottom row contains our leaf nodes: the nodes with no answers coming out of them, because there are no more questions to be answered.

Returning back to the top node after the first decision point, we can see that a second binary decision split has been made, based on asking whether **weeks\_worked\_per\_year** is less than or equal to 0.9. For the group where this is true, the class predicted is -50k with a gini of 0.019 and there are 85,411 records. For the records where this decision is false, the class predicted is -50k with a gini of 0.019, and there are 52,361 records. So again, we can see that the decision tree algorithm has successfully split out more records into two more groups which differ in gini value significantly.

Now, let's run our base model :

m = DecisionTreeClassifier(random\_state=14)  
m.fit(X\_train, y\_train)

DecisionTreeClassifier(random\_state=14)

We evaluate the model on our validation set using **accuracy**, **recall** and **f1 score** :

from sklearn.metrics import accuracy\_score, f1\_score, recall\_score  
  
# on the train set   
accuracy\_score(y\_train,m.predict(X\_train)) , \  
recall\_score(y\_train,m.predict(X\_train)), \  
f1\_score(y\_train,m.predict(X\_train), average='binary', pos\_label=1)

(0.9996348431257428, 0.9943463712934117, 0.997049806212761)

# on the valid set   
accuracy\_score(y\_val,m.predict(X\_val)) , \  
recall\_score(y\_val,m.predict(X\_val)), \  
f1\_score(y\_val,m.predict(X\_val), average='binary', pos\_label=1)

(0.9307683311893346, 0.4888290713324361, 0.4670781893004115)

It's seems that we are doing badly on the validation set. Let's see houw many leaf nodes we got :

m.get\_n\_leaves(), len(X\_train)

(8005, 139666)

Sklearn's default settings allow it to continue splitting nodes until there is only one item in each leaf node. Let's change the stopping rule to tell sklearn to ensure every leaf node contains at least 25 records:

m = DecisionTreeClassifier(min\_samples\_leaf=25,random\_state=14)  
m.fit(X\_train, y\_train)  
  
# on the train set   
accuracy\_score(y\_train,m.predict(X\_train)) , \  
recall\_score(y\_train,m.predict(X\_train)), \  
f1\_score(y\_train,m.predict(X\_train), average='binary', pos\_label=1)

(0.9571048071828505, 0.46198223145263645, 0.5720408600614331)

# on the valid set   
accuracy\_score(y\_val,m.predict(X\_val)) , \  
recall\_score(y\_val,m.predict(X\_val)), \  
f1\_score(y\_val,m.predict(X\_val), average='binary', pos\_label=1)

(0.9504986885410228, 0.42449528936742936, 0.515612228216446)

That looks much better. Let's check the number of leaves again:

m.get\_n\_leaves(), len(X\_train)

(1533, 139666)

We got less leaf nodes than before. So, the more we increase the number of leaf nodes, the more is the possibility of **overfitting**.

Building a **decision tree** is a good way to create a model of our data. It is very flexible, since it can clearly handle nonlinear relationships and interactions between variables. But we can see there is a fundamental compromise between how well it generalizes (which we can achieve by creating small trees) and how accurate it is on the training set (which we can achieve by using large trees).

So how do we get the best of both worlds?

* Ensembling :

An an example of an Ensemble method is **Random Forest** : we can train a group of Decision Tree classifiers, each on a different random subset of the training set. The process of subseting the data is called **bagging** done with **max\_samples** hyperparameter ( we set it at 100.00 samples) and the ramdom selection process this called **bootsraping** done by setting **bootstrap = True**.

With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all. The remaining sampled are called **out-of-bag (oob) instances** used as **validation set** in the training process and done by setting **oob\_score=True**.

We train a Random Forest classifier **with 50 trees** (each limited to **minimum 5 samples per leaf**). and instead of searching for the very best feature when splitting a node, we searches for the best feature among a random subset of 50% of our initial features.

from sklearn.ensemble import RandomForestClassifier  
  
rf = RandomForestClassifier(n\_estimators = 50, max\_samples=100\_000, max\_features=0.5, min\_samples\_leaf= 5,   
 bootstrap= True,oob\_score = True,random\_state=14)

%%time  
rf.fit(X\_train,y\_train)

/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"

CPU times: user 29.1 s, sys: 643 ms, total: 29.7 s  
Wall time: 32.8 s

RandomForestClassifier(max\_features=0.5, max\_samples=100000, min\_samples\_leaf=5,  
 n\_estimators=50, oob\_score=True, random\_state=14)

# on the train set   
accuracy\_score(y\_train,rf.predict(X\_train)) , \  
recall\_score(y\_train,rf.predict(X\_train)), \  
f1\_score(y\_train,rf.predict(X\_train), average='binary', pos\_label=1)

(0.9670571219910358, 0.5334025614399446, 0.6677258611973712)

# on the valid set   
accuracy\_score(y\_val,rf.predict(X\_val)) , \  
recall\_score(y\_val,rf.predict(X\_val)), \  
f1\_score(y\_val,rf.predict(X\_val), average='binary', pos\_label=1)

(0.9546418965200394, 0.41668909825033645, 0.5327826535880227)

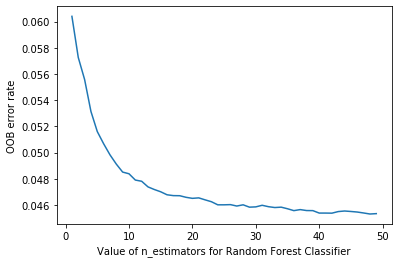
Looking at what happens to the **oob error rate** as we add more and more trees, we you can see that the improvement levels off quite a bit after around 40 trees:

scores =[]  
for k in range(1, 50):  
 rfc = RandomForestClassifier(n\_estimators = k, max\_samples=100\_000, max\_features=0.5, min\_samples\_leaf= 5,   
 bootstrap= True,oob\_score = True,random\_state=14)  
 rfc.fit(X\_train, y\_train)  
 #y\_pred = rfc.predict(X\_val)  
 #scores.append(accuracy\_score(y\_test, y\_pred)) oob\_score\_  
 oob\_error = 1 - rfc.oob\_score\_  
 scores.append(oob\_error)  
  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
# plot the relationship between K and testing accuracy  
# plt.plot(x\_axis, y\_axis)  
plt.plot(range(1, 50), scores)  
plt.xlabel('Value of n\_estimators for Random Forest Classifier')  
#plt.ylabel('Testing Accuracy')  
plt.ylabel('OOB error rate')

/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,

/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/\_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.  
 UserWarning,  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"  
/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"

Text(0, 0.5, 'OOB error rate')



Let's try to improve our model :

We may ask **which columns are the strongest predictors, which can we ignore?**

It's not normally enough just to know that a model can make accurate predictions—we also want to know how it's making predictions. Feature importance gives us insight into this. We can get these directly from sklearn's random forest by looking in the feature\_importances\_ attribute. Here's a simple function we can use to pop them into a DataFrame and sort them:

def rf\_feat\_importance(m, df):  
 return pd.DataFrame({'cols':df.columns, 'imp':m.feature\_importances\_}  
 ).sort\_values('imp', ascending=False)

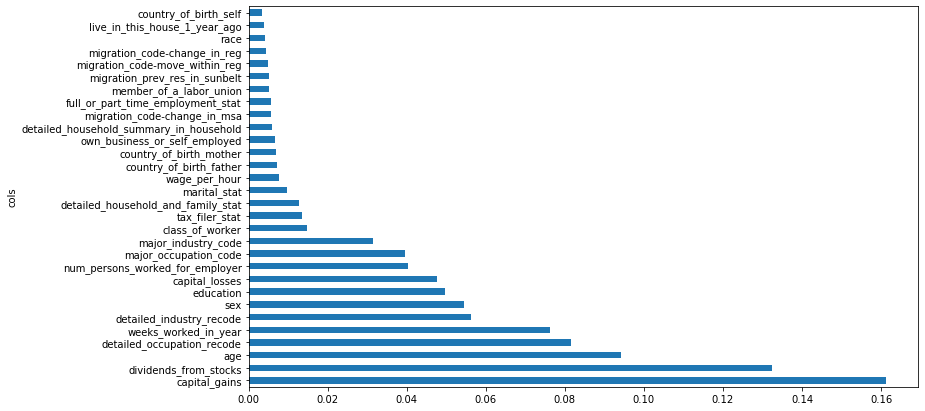
fi = rf\_feat\_importance(rf, X\_train)  
fi[:14]

cols imp  
16 capital\_gains 0.161368  
18 dividends\_from\_stocks 0.132382  
0 age 0.094106  
3 detailed\_occupation\_recode 0.081452  
38 weeks\_worked\_in\_year 0.076202  
2 detailed\_industry\_recode 0.056239  
12 sex 0.054394  
4 education 0.049652  
17 capital\_losses 0.047552  
29 num\_persons\_worked\_for\_employer 0.040222  
9 major\_occupation\_code 0.039596  
8 major\_industry\_code 0.031488  
1 class\_of\_worker 0.014878  
19 tax\_filer\_stat 0.013542

The feature importances for our model show that the first few most important columns have much higher importance scores than the rest, with (not surprisingly) **capital\_gains** and **dividends\_from\_stocks** being at the top of the list.

A plot of the feature importances shows the relative importances more clearly:

def plot\_fi(fi):  
 return fi.plot('cols', 'imp', 'barh', figsize=(12,7), legend=False)  
  
plot\_fi(fi[:30]);



The way these importances are calculated is quite simple yet elegant. The feature importance algorithm loops through each tree, and then recursively explores each branch. At each branch, it looks to see what feature was used for that split, and how much the model improves as a result of that split. The improvement (weighted by the number of rows in that group) is added to the importance score for that feature. This is summed across all branches of all trees, and finally the scores are normalized such that they add to 1.

It seems likely that we could use just a subset of the columns by removing the variables of low importance and still get good results. Let's try just keeping those with a feature importance greater than **0.005**:

to\_keep = fi[fi.imp>0.005].cols  
len(to\_keep)

25

We can retrain our model using just this subset of the columns:

X\_train\_imp = X\_train[to\_keep]  
X\_val\_imp = X\_val[to\_keep]

m = RandomForestClassifier(n\_estimators = 50, max\_samples=100\_000, max\_features=0.5, min\_samples\_leaf= 5,   
 bootstrap= True,oob\_score = True,random\_state=14)  
  
m.fit(X\_train\_imp,y\_train)

/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names  
 "X does not have valid feature names, but"

RandomForestClassifier(max\_features=0.5, max\_samples=100000, min\_samples\_leaf=5,  
 n\_estimators=50, oob\_score=True, random\_state=14)

# on the train set   
accuracy\_score(y\_train,m.predict(X\_train\_imp)) , \  
recall\_score(y\_train,m.predict(X\_train\_imp)), \  
f1\_score(y\_train,m.predict(X\_train\_imp), average='binary', pos\_label=1)

(0.9670857617458795, 0.5334025614399446, 0.6679188037275157)

# on the valid set   
accuracy\_score(y\_val,m.predict(X\_val\_imp)) , \  
recall\_score(y\_val,m.predict(X\_val\_imp)), \  
f1\_score(y\_val,m.predict(X\_val\_imp), average='binary', pos\_label=1)

(0.9543077668443123, 0.4142664872139973, 0.5295028384655084)

Our **accuracy is about the same**, but we have **far fewer columns** to study:

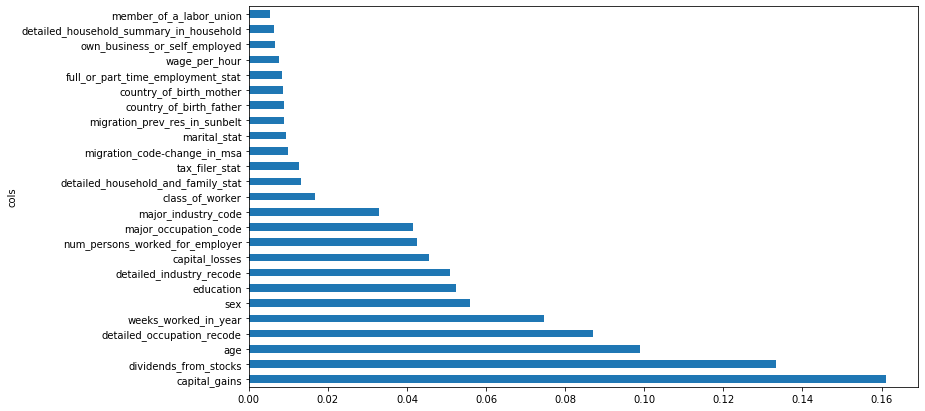
len(X\_train.columns), len(X\_train\_imp.columns)

(40, 25)

We've found that generally the first step to improving a model is **simplifying it**—48 columns was too many for us to study them all in depth! Furthermore, in practice often a simpler, more interpretable model is easier to roll out and maintain.

This also makes our feature importance plot easier to interpret. Let's look at it again:

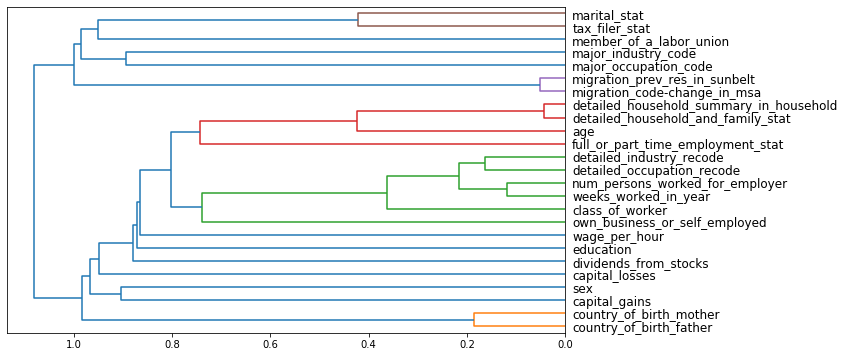
plot\_fi(rf\_feat\_importance(m, X\_train\_imp));



Let's see if we have redundent feature in our model by determining their similarities :

Determining Similarity: The most similar pairs are found by calculating the rank correlation, which means that all the values are replaced with their rank (i.e., first, second, third, etc. within the column), and then the correlation is calculated.

import scipy  
from scipy.cluster import hierarchy as hc  
  
def cluster\_columns(df, figsize=(10,6), font\_size=12):  
 corr = np.round(scipy.stats.spearmanr(df).correlation, 4)  
 corr\_condensed = hc.distance.squareform(1-corr)  
 z = hc.linkage(corr\_condensed, method='average')  
 fig = plt.figure(figsize=figsize)  
 hc.dendrogram(z, labels=df.columns, orientation='left', leaf\_font\_size=font\_size)  
 plt.show()  
  
cluster\_columns(X\_train\_imp)



Looking good! This is really not much worse than the model with all the fields. Let's create DataFrames without these columns, and save them:

X\_train\_final = X\_train\_imp # train  
X\_val\_final = X\_val\_imp # valid   
  
test\_x\_final = test\_x[to\_keep] # test set

X\_train\_final.shape , X\_val\_final.shape, test\_x\_final.shape

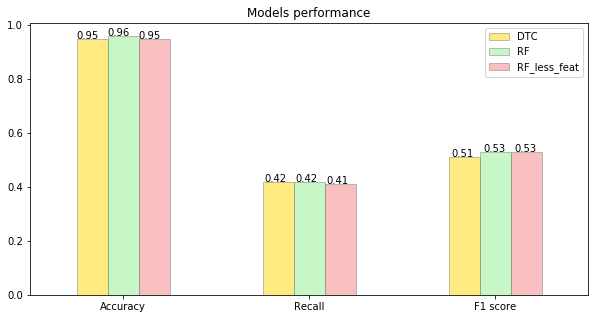
((139666, 25), (59857, 25), (99762, 25))

# Model Assesment :

We have seen the **DecisionTreeClassifier** as our basemodel, then we tried **RandomForestClassifier** and finaly we tried to optimize so we can have less features for better interpretation.

Here is the model metrics on our **validation set** :

models\_metrics = {'DTC': [0.95, 0.42, 0.51],   
 'RF': [0.96, 0.42, 0.53],  
 'RF\_less\_feat': [0.95, 0.41, 0.53]  
   
 }  
df = pd.DataFrame(data = models\_metrics)  
df.rename(index={0:'Accuracy',1:'Recall', 2: 'F1 score'},   
 inplace=True)  
ax = df.plot(kind='bar', figsize = (10,5),   
 color = ['gold', 'lightgreen','lightcoral'],  
 rot = 0, title ='Models performance',  
 edgecolor = 'grey', alpha = 0.5)  
for p in ax.patches:  
 ax.annotate(str(p.get\_height()), (p.get\_x() \* 1.01, p.get\_height() \* 1.0005))  
plt.show()



Based on **F1 score**, we select the **RandomForestClassifier** with 25 features as our best model.

Let's see the **experiment results**\* of this model :

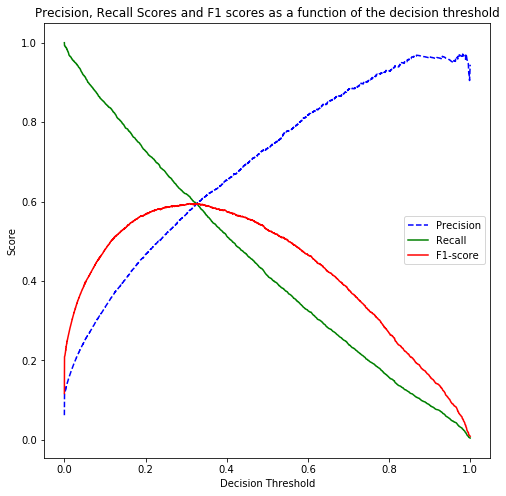
The **precision\_recall\_curve** and **roc\_curve** are useful tools to visualize the **sensitivity-specificty** tradeoff in the classifier. They help inform a data scientist where to set the decision threshold of the model to maximize either sensitivity or specificity. This is called the **operating point** of the model.

from sklearn.metrics import roc\_curve, precision\_recall\_curve, \  
auc, make\_scorer, recall\_score, accuracy\_score, precision\_score, confusion\_matrix

# We create an array of the class probabilites called y\_scores  
y\_scores = m.predict\_proba(X\_val\_imp)[:, 1]  
  
# we enerate the precision-recall curve for the classifier:  
p, r, thresholds = precision\_recall\_curve(y\_val, y\_scores)  
  
# We calculate the F1 scores  
f1\_scores = 2\*r\*p/(r+p)

Let's plot the **decision chart** of our model :

def plot\_precision\_recall\_vs\_threshold(precisions, recalls, thresholds):  
   
 plt.figure(figsize=(8, 8))  
 plt.title("Precision, Recall Scores and F1 scores as a function of the decision threshold")  
 plt.plot(thresholds, precisions[:-1], "b--", label="Precision")  
 plt.plot(thresholds, recalls[:-1], "g-", label="Recall")  
 plt.plot(thresholds, f1\_scores[:-1], "r-", label="F1-score")  
 plt.ylabel("Score")  
 plt.xlabel("Decision Threshold")  
 plt.legend(loc='best')  
   
plot\_precision\_recall\_vs\_threshold(p, r, thresholds)



We can see that the the optimal threshold to achieve the highest F1 score is set at **0.30** with **59% F1-score**.

print('Best threshold: ', thresholds[np.argmax(f1\_scores)])  
print('Best F1-Score: ', np.max(f1\_scores))

Best threshold: 0.30834595959595956  
Best F1-Score: 0.5950888192267503

Let's creat **an animated confusion matrix** where the users get to choose the **threesholds** and we **dislpay the confusion matrix and recall vs precision curve** :

pip install ipywidgets

Requirement already satisfied: ipywidgets in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (7.6.5)  
Requirement already satisfied: ipython-genutils~=0.2.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipywidgets) (0.2.0)  
Requirement already satisfied: ipykernel>=4.5.1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipywidgets) (5.3.4)  
Requirement already satisfied: ipython>=4.0.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipywidgets) (7.22.0)  
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipywidgets) (1.0.0)  
Requirement already satisfied: nbformat>=4.2.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipywidgets) (5.1.3)  
Requirement already satisfied: traitlets>=4.3.1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipywidgets) (5.1.1)  
Requirement already satisfied: widgetsnbextension~=3.5.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipywidgets) (3.5.1)  
Requirement already satisfied: tornado>=4.2 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipykernel>=4.5.1->ipywidgets) (6.1)  
Requirement already satisfied: jupyter-client in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipykernel>=4.5.1->ipywidgets) (7.0.1)  
Requirement already satisfied: appnope in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipykernel>=4.5.1->ipywidgets) (0.1.2)  
Requirement already satisfied: decorator in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (5.1.0)  
Requirement already satisfied: pygments in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (2.10.0)  
Requirement already satisfied: backcall in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (0.2.0)  
Requirement already satisfied: pexpect>4.3 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (4.8.0)  
Requirement already satisfied: setuptools>=18.5 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (58.0.4)  
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (3.0.20)  
Requirement already satisfied: jedi>=0.16 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (0.18.1)  
Requirement already satisfied: pickleshare in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from ipython>=4.0.0->ipywidgets) (0.7.5)  
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jedi>=0.16->ipython>=4.0.0->ipywidgets) (0.8.3)  
Requirement already satisfied: jupyter-core in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbformat>=4.2.0->ipywidgets) (4.9.1)  
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbformat>=4.2.0->ipywidgets) (3.2.0)  
Requirement already satisfied: pyrsistent>=0.14.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets) (0.18.0)  
Requirement already satisfied: importlib-metadata in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets) (4.8.1)  
Requirement already satisfied: attrs>=17.4.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets) (21.2.0)  
Requirement already satisfied: six>=1.11.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets) (1.16.0)  
Requirement already satisfied: ptyprocess>=0.5 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from pexpect>4.3->ipython>=4.0.0->ipywidgets) (0.7.0)  
Requirement already satisfied: wcwidth in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=4.0.0->ipywidgets) (0.2.5)  
Requirement already satisfied: notebook>=4.4.1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from widgetsnbextension~=3.5.0->ipywidgets) (6.4.5)  
Requirement already satisfied: prometheus-client in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.11.0)  
Requirement already satisfied: pyzmq>=17 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (22.2.1)  
Requirement already satisfied: jinja2 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (2.11.3)  
Requirement already satisfied: nbconvert in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (6.1.0)  
Requirement already satisfied: terminado>=0.8.3 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.9.4)  
Requirement already satisfied: argon2-cffi in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (20.1.0)  
Requirement already satisfied: Send2Trash>=1.5.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (1.8.0)  
Requirement already satisfied: python-dateutil>=2.1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jupyter-client->ipykernel>=4.5.1->ipywidgets) (2.8.2)  
Requirement already satisfied: nest-asyncio>=1.5 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jupyter-client->ipykernel>=4.5.1->ipywidgets) (1.5.1)  
Requirement already satisfied: entrypoints in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jupyter-client->ipykernel>=4.5.1->ipywidgets) (0.3)  
Requirement already satisfied: cffi>=1.0.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (1.15.0)  
Requirement already satisfied: pycparser in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from cffi>=1.0.0->argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (2.21)  
Requirement already satisfied: typing-extensions>=3.6.4 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from importlib-metadata->jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets) (3.10.0.2)  
Requirement already satisfied: zipp>=0.5 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from importlib-metadata->jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets) (3.6.0)  
Requirement already satisfied: MarkupSafe>=0.23 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from jinja2->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (2.0.1)  
Requirement already satisfied: jupyterlab-pygments in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.1.2)  
Requirement already satisfied: mistune<2,>=0.8.1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.8.4)  
Requirement already satisfied: testpath in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.5.0)  
Requirement already satisfied: pandocfilters>=1.4.1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (1.4.3)  
Requirement already satisfied: defusedxml in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.7.1)  
Requirement already satisfied: bleach in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (4.0.0)  
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.5.3)

Requirement already satisfied: async-generator in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from nbclient<0.6.0,>=0.5.0->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (1.10)  
Requirement already satisfied: webencodings in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (0.5.1)  
Requirement already satisfied: packaging in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (21.0)  
Requirement already satisfied: pyparsing>=2.0.2 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from packaging->bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets) (3.0.4)  
Note: you may need to restart the kernel to use updated packages.

import ipywidgets as widgets

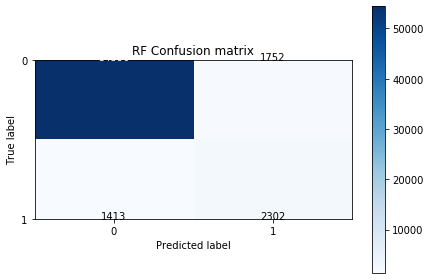
import itertools  
def plot\_confusion\_matrix(cm, classes,  
 normalize = False,  
 title = 'Confusion matrix"',  
 cmap = plt.cm.Blues) :  
 plt.imshow(cm, interpolation = 'nearest', cmap = cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation = 0)  
 plt.yticks(tick\_marks, classes)  
  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])) :  
 plt.text(j, i, cm[i, j],  
 horizontalalignment = 'center',  
 color = 'white' if cm[i, j] > thresh else 'black')  
  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
   
def adjusted\_classes(y\_scores, t):  
 """  
 This function adjusts class predictions based on the prediction threshold (t).  
 Will only work for binary classification problems.  
 """  
 return [1 if y >= t else 0 for y in y\_scores]  
  
def precision\_recall\_threshold(p, r, thresholds, t=0.5):  
 """  
 plots the precision recall curve and shows the current value for each  
 by identifying the classifier's threshold (t).  
 """  
   
 # generate new class predictions based on the adjusted\_classes  
 # function above and view the resulting confusion matrix.  
 y\_pred\_adj = adjusted\_classes(y\_scores, t)  
   
 cm = confusion\_matrix(y\_val, y\_pred\_adj)  
 class\_names = [0,1]  
 plt.figure()  
 plot\_confusion\_matrix(cm,   
 classes=class\_names,   
 title='RF Confusion matrix')  
  
 plt.figure(figsize=(8,8))  
 plt.title("Precision and Recall curve ^ = current threshold")  
 plt.step(r, p, color='b', alpha=0.2,  
 where='post')  
 plt.fill\_between(r, p, step='post', alpha=0.2,  
 color='b')  
   
 plt.xlabel('Recall');  
 plt.ylabel('Precision');  
   
 # plot the current threshold on the line  
 close\_default\_clf = np.argmin(np.abs(thresholds - t))  
 plt.plot(r[close\_default\_clf], p[close\_default\_clf], '^', c='k',  
 markersize=15)

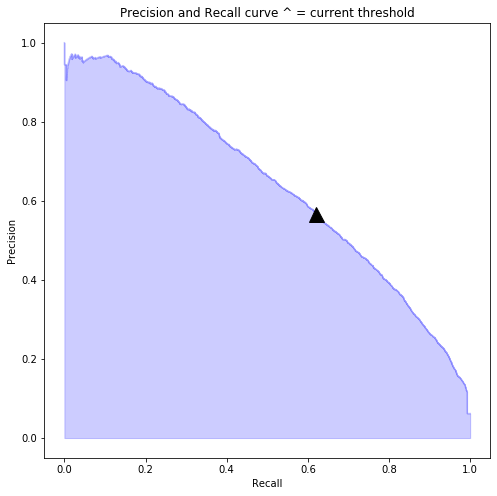
slider = widgets.IntSlider(  
 min=0,  
 max=10,  
 step=1,  
 description='Slider:',  
 value=3 # The best threshhold for our model  
)  
display(slider)

{"model\_id":"64c4bfbbeb2a4242855806666b132d49","version\_major":2,"version\_minor":0}

print(f'For this threshold : {slider.value/10}, the confusion matrix is as follow :')  
precision\_recall\_threshold(p, r, thresholds, slider.value/10)

For this threshold : 0.3, the confusion matrix is as follow :

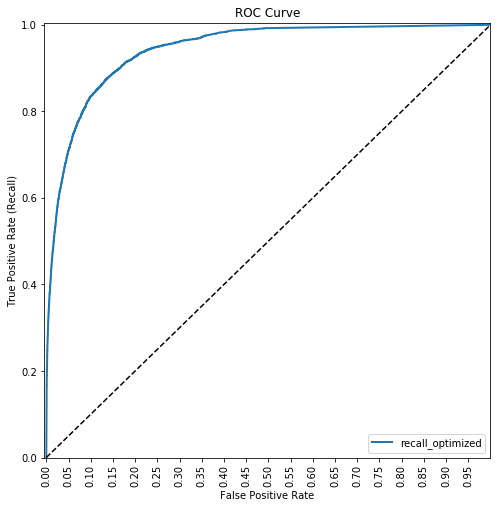




def plot\_roc\_curve(fpr, tpr, label=None):  
 plt.figure(figsize=(8,8))  
 plt.title('ROC Curve')  
 plt.plot(fpr, tpr, linewidth=2, label=label)  
 plt.plot([0, 1], [0, 1], 'k--')  
 plt.axis([-0.005, 1, 0, 1.005])  
 plt.xticks(np.arange(0,1, 0.05), rotation=90)  
 plt.xlabel("False Positive Rate")  
 plt.ylabel("True Positive Rate (Recall)")  
 plt.legend(loc='best')

fpr, tpr, auc\_thresholds = roc\_curve(y\_val, y\_scores)  
print(f'AUC : {auc(fpr, tpr)}') # AUC of ROC  
plot\_roc\_curve(fpr, tpr, 'recall\_optimized')

AUC : 0.9433941778952841



Now, let's test this model on our **test set** :

accuracy\_score(test\_y,m.predict(test\_x\_final)) , \  
recall\_score(test\_y,m.predict(test\_x\_final)) , \  
f1\_score(test\_y,m.predict(test\_x\_final), average='binary', pos\_label=1)

(0.9546420480744171, 0.4183640478499838, 0.5335532419338213)

# Results : Partial dependency and SHAP values

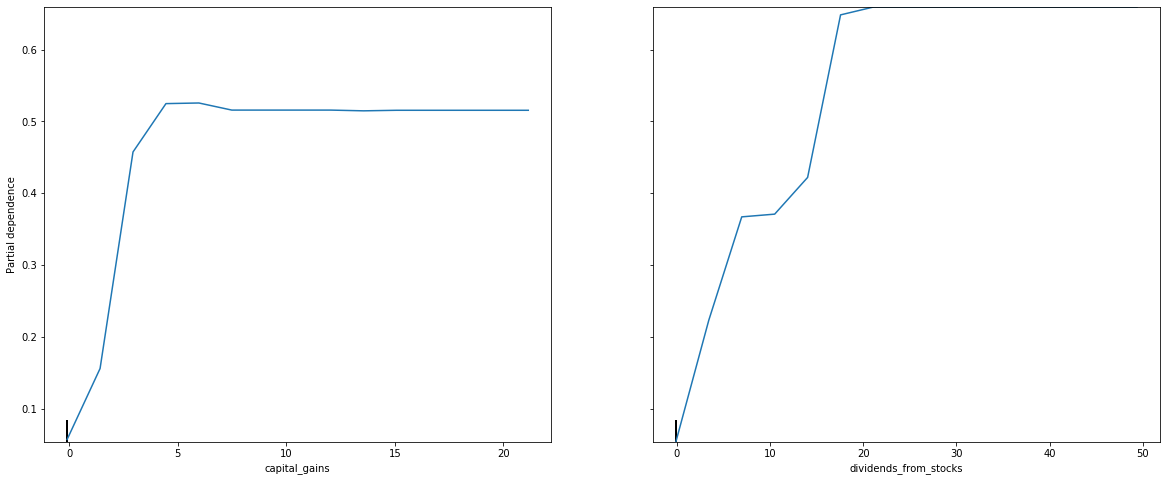
Let's look at **partial dependence plots**.

**Partial dependence** plots try to answer the question: if a row varied on nothing other than the feature in question, how would it impact the dependent variable?

For instance, how does **capital\_gains** and **dividends\_from\_stocks** impact probability of belonging to the +50k income levl, all other things being equal?

from sklearn.inspection import plot\_partial\_dependence  
  
fig,ax = plt.subplots(figsize=(20, 8))  
plot\_partial\_dependence(m, X\_val\_final, ['capital\_gains','dividends\_from\_stocks'], percentiles=(0,1),  
 grid\_resolution=15, ax=ax);

/Users/rmbp/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot\_partial\_dependence is deprecated; Function `plot\_partial\_dependence` is deprecated in 1.0 and will be removed in 1.2. Use PartialDependenceDisplay.from\_estimator instead  
 warnings.warn(msg, category=FutureWarning)



Looking first at the **dividends\_from\_stocks** plot, we can see a nearly linear relationship between capital dividends\_from\_stocks and the probabillity of income level. Same for **capital\_gains** at 5 standad deviation from the mean after reaching a steady state above that.

!pip install shap

Requirement already satisfied: shap in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (0.40.0)  
Requirement already satisfied: packaging>20.9 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (21.0)  
Requirement already satisfied: cloudpickle in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (2.0.0)  
Requirement already satisfied: tqdm>4.25.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (4.62.3)  
Requirement already satisfied: pandas in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (1.3.4)  
Requirement already satisfied: scipy in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (1.7.1)  
Requirement already satisfied: slicer==0.0.7 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (0.0.7)  
Requirement already satisfied: numpy in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (1.21.2)  
Requirement already satisfied: numba in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (0.53.1)  
Requirement already satisfied: scikit-learn in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from shap) (1.0.1)  
Requirement already satisfied: pyparsing>=2.0.2 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from packaging>20.9->shap) (3.0.4)  
Requirement already satisfied: llvmlite<0.37,>=0.36.0rc1 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from numba->shap) (0.36.0)  
Requirement already satisfied: setuptools in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from numba->shap) (58.0.4)  
Requirement already satisfied: python-dateutil>=2.7.3 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from pandas->shap) (2.8.2)  
Requirement already satisfied: pytz>=2017.3 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from pandas->shap) (2021.3)  
Requirement already satisfied: six>=1.5 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas->shap) (1.16.0)  
Requirement already satisfied: joblib>=0.11 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from scikit-learn->shap) (1.1.0)  
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/rmbp/opt/anaconda3/lib/python3.7/site-packages (from scikit-learn->shap) (2.2.0)

import shap

row\_to\_show = 20  
data\_for\_prediction = test\_x\_final.iloc[row\_to\_show] # use 1 row of data here. Could use multiple rows if desired

# Create object that can calculate shap values  
explainer = shap.TreeExplainer(m)  
  
# Calculate Shap values  
shap\_values = explainer.shap\_values(data\_for\_prediction)

shap.initjs()  
shap.force\_plot(explainer.expected\_value[1], shap\_values[1], data\_for\_prediction)

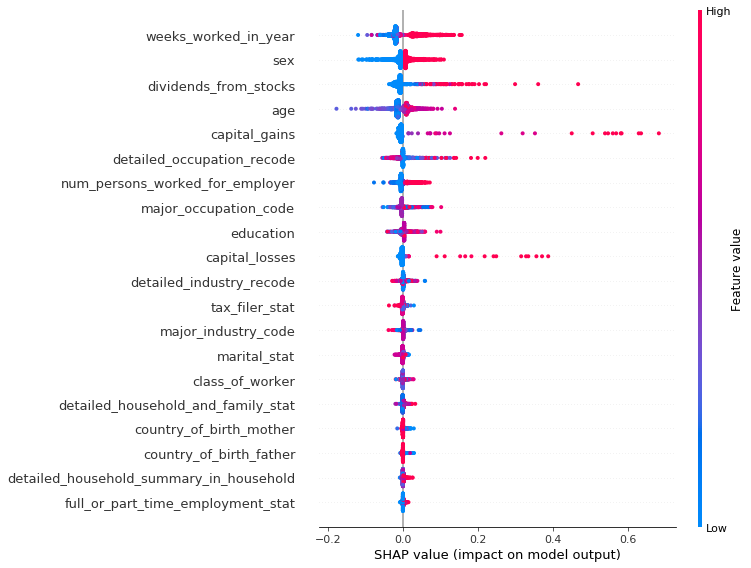
<IPython.core.display.HTML object>

<shap.plots.\_force.AdditiveForceVisualizer at 0x13c39bd50>

The above explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in blue

* The base\_value here is 0.062 while our predicted value is 0.0.
* sex = 1 has the biggest impact on increasing the prediction, while
* Weeks\_worked\_im\_year (below the average) and Age (below the average) feature has the biggest effect in decreasing the prediction.

explainer = shap.TreeExplainer(m)  
  
# calculate shap values. This is what we will plot.  
# Calculate shap\_values for all of val\_X rather than a single row, to have more data for plot.  
shap\_values = explainer.shap\_values(test\_x\_final.iloc[:1000,])  
  
# Make plot. Index of [1] is explained in text below.  
shap.summary\_plot(shap\_values[1],test\_x\_final.iloc[:1000,])



For every dot:

* Vertical location shows what feature it is depicting
* Color shows whether that feature was high or low for that row of the dataset
* Horizontal location shows whether the effect of that value caused a higher or lower prediction.

For the **age** variable, the point in the upper left was depicts a person whose age level is less thereby reducing the prediction of income level +50k class by 0.2.

# Conclusion :

In this work, we presented some techniques for dealing with a machine learning project :

* We used Decision Tree ensembles : Random Forest are the easiest to train, because they are extremely resilient to hyperparameter choices and require very little preprocessing. They are very fast to train, and should not overfit if we have enough trees.
* we used the model for feature selection and partial dependence analysis and Shap values, to get a better understanding of our data.

For futur improvements :

* We can try Gradient Boosting machines as in theory are just as fast to train as random forests, but in practice we will have to try lots of different hyperparameters. They can overfit, but they are often a little more accurate than random forests.
* We can try OneHotEncoder with PCA to deal with the multiple modalities on our categorical variables.
* We can creat new features to challenge the model performance.