# Predicting Income Levels using Python

**Objective:** The task is to build predictive models to determine the income for people in 7 Susing socio-political data. The income levels are binned below $50,000 and above $50,000.

**1. Introduction**

1. **Data:** Census income dataset (1994).
2. **Source reference:** Refer to the html links in the page to understand more about the data <http://archive.ics.uci.edu/ml/machine-learning-databases/census-income-mld>
3. **Tasks:**
   1. Download the dataset from the above mentioned URL
   2. Clean and manipulate the dataset as required
   3. Check the significance of the predictors in predicting the final outcome, both individually as well as in groups.
   4. Choose the most significant predictors to use in the model. Provide graphs and other visualizations to demonstrate why you believe a certain predictor is more or less significant.
   5. Perform two predictive machine learning techniques on the given dataset (other than linear and logistic regression. Give reasons as to why you would choose these two models in the first place.
   6. Select the best model for this problem
   7. Create comparative AUC graphs to prove the case
4. **Deliverables:**
   1. Jupyter notebook/github repo link with the code. Use comments in your code to improve readability.
   2. A document containing all the graphs, and tables along with your comments. What do you think is the most challenging aspect of the dataset, if any?

**2. Methodolgy**

In this project, US census data is used to build a model in Python using the sklearn library to predict if the income of any individual in the US is greater than or less than USD 50000 based on the information available about that individual in the census data.

The dataset used for the analysis is an extraction from the 1994 census data by Barry Becker and donated to the public site The case study was approached in the following order:

1. Describe the data- Specifically the ondependent variables from the Census data and the dependent variable which is the level of income (either “greater than USD 50000” or “less than USD 50000”).
2. Acquire and Read the data- Download the data from the source and read it.
3. Clean the data- Any data from the real world is always messy and noisy. The data needs to be reshaped in order to aid exploration of the data and modeling to predict the income level.
4. Explore the independent variables of the data- A very crucial step before modeling is the exploration of the independent variables. Exploration provides great insights to on the predicting power of the variable.
5. Build the prediction model with the training data- Since linear regression and logistic regression cannot be used as mentioned in the problem statement, for this project, the non-parametric predicting algorithm of Decision Tree. Census data can have many weak predictors, hence Boosting algorithm is also in consideration . Cross validation can also be used to reduce over fitting while modeling, alongwith with Boosting.
6. Validate the prediction model with the testing data- Here the built model is applied on test data.

**3. Data**

The data was in two csv files titled ‘train’ and ‘test’. Csv is readable by by pandas. The data was read into a pandas DataFrame object. The data had 41 variables and 199523 entries. Four variables had more than 50% entries missing.

The list of variables with data types is :

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 199523 entries, 0 to 199522

Data columns (total 41 columns):

age 199523 non-null int64

class\_of\_worker 199523 non-null object

industry\_code 199523 non-null int64

occupation\_code 199523 non-null int64

education 199523 non-null object

wage\_per\_hour 199523 non-null int64

enrolled\_in\_edu\_inst\_lastwk 199523 non-null object

marital\_status 199523 non-null object

major\_industry\_code 199523 non-null object

major\_occupation\_code 199523 non-null object

race 199523 non-null object

hispanic\_origin 198649 non-null object

sex 199523 non-null object

member\_of\_labor\_union 199523 non-null object

reason\_for\_unemployment 199523 non-null object

full\_parttime\_employment\_stat 199523 non-null object

capital\_gains 199523 non-null int64

capital\_losses 199523 non-null int64

dividend\_from\_Stocks 199523 non-null int64

tax\_filer\_status 199523 non-null object

region\_of\_previous\_residence 199523 non-null object

state\_of\_previous\_residence 198815 non-null object

d\_household\_family\_stat 199523 non-null object

d\_household\_summary 199523 non-null object

migration\_msa 99827 non-null object

migration\_reg 99827 non-null object

migration\_within\_reg 99827 non-null object

live\_1\_year\_ago 199523 non-null object

migration\_sunbelt 99827 non-null object

num\_person\_Worked\_employer 199523 non-null int64

family\_members\_under\_18 199523 non-null object

country\_father 192810 non-null object

country\_mother 193404 non-null object

country\_self 196130 non-null object

citizenship 199523 non-null object

business\_or\_self\_employed 199523 non-null int64

fill\_questionnaire\_veteran\_admin 199523 non-null object

veterans\_benefits 199523 non-null int64

weeks\_worked\_in\_year 199523 non-null int64

year 199523 non-null int64

income\_level 199523 non-null int64

dtypes: int64(13), object(28)

memory usage: 62.4+ MB

Out of the 41 variables, 13 had dtype int64 and rest 28 had np.object.

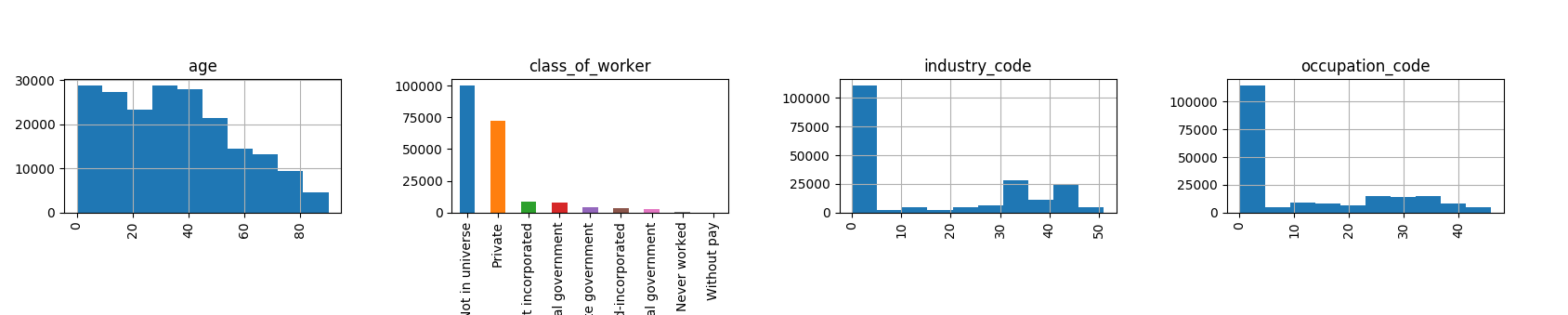
**4. Visualising Data**

The value\_counts were plotted for each variable, separating out variables with dtype. It revealed that data was heavily unbalanced with only one value dominating. Upon checking the value\_counts of the dependent variable, it was conformed that the ‘above 50000’ bin had only 6% support.

-50000 187141

50000 12382

Name: income\_level, dtype: int64



Further visualisation was not done as most characteristics were suppessed due to skewness.

**5. Cleaning Data**

The training data set was cleaned for missing data. Apart from the ‘NaN’ values, 6 columns had more than 90% ‘Not in universe’ values.

class\_of\_worker -- 100245

enrolled\_in\_edu\_inst\_lastwk -- 186943

major\_occupation\_code -- 100684

member\_of\_labor\_union -- 180459

reason\_for\_unemployment -- 193453

region\_of\_previous\_residence -- 183750

state\_of\_previous\_residence -- 183750

migration\_msa -- 1516

migration\_reg -- 1516

migration\_within\_reg -- 1516

migration\_sunbelt -- 84054

family\_members\_under\_18 -- 144232

fill\_questionnaire\_veteran\_admin – 197539

Droping variables with missing data may cause loss of information, and every information is valuable for the given unbalanced data, so missing values were replaced with ‘Not in universe’.

**6. Up-sampling Data**

Training the model with unbalanced data will cause the model to predict all test data to the majority class, hence upsampling was done. The train data was split into majority and minority class, majority being the ‘less than 50000’ bin encoded with (zero) while cleaning. The minority class was upsampled using sklearn.utils.resample and then the upsampled data was concatenated with the majority data.

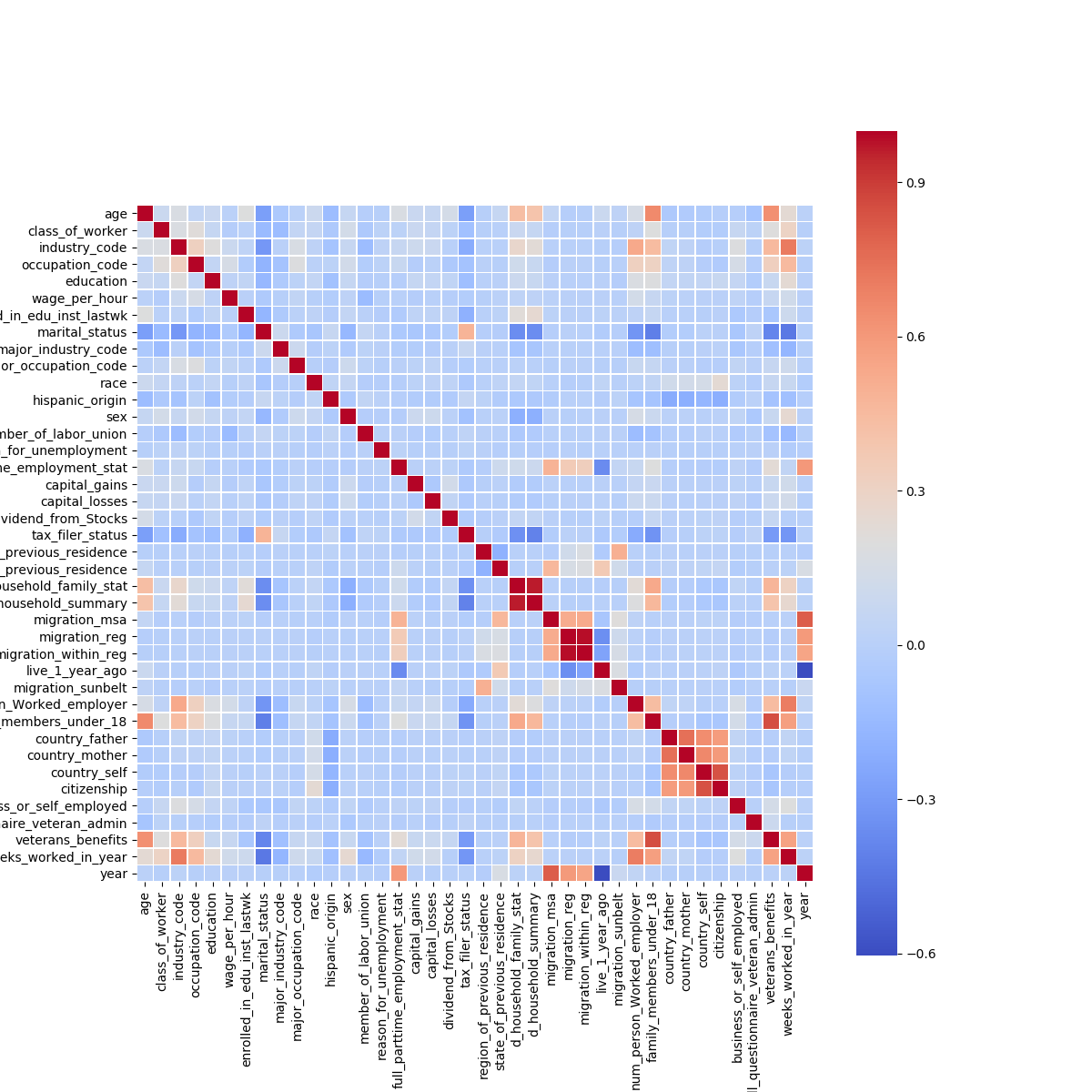
**7. Numerically encoding categorical features**

The sklearn models depend on numerical data to create models. Therefore, it was indespensable to encode categorical features to numerical values. A resuable function was defined to encode train and test data using sklearn.preprocessing.LabelEncoder. After cross checking the encoding was found to be consistent.

The dtype of ‘income\_level’ variable, the dependent variable, was be np.object, due to which the endoder() function encoded this column too which was realised after proceeding further in the analysis. To find the issue, encoders{} was inserted into endoder() function which returned the encoded columns and the issue came to light. It was rectified by inspecting income\_level values in the train data and replacing with binary values, 0 (zero) being the value for ‘less than 50000’ bin. Binarising the income\_level was moved to the cleaning section to ensure income\_level dtype is int64 while using encoder() function.

The train and test data was split into X and y.

**8. Plotting correlation**

****

Correlations were were observed in country, migration and householding data. Correlation in householding data was due to the maximum occurence of ‘household’ in both the columns.

**9. Building and Evaluating models**

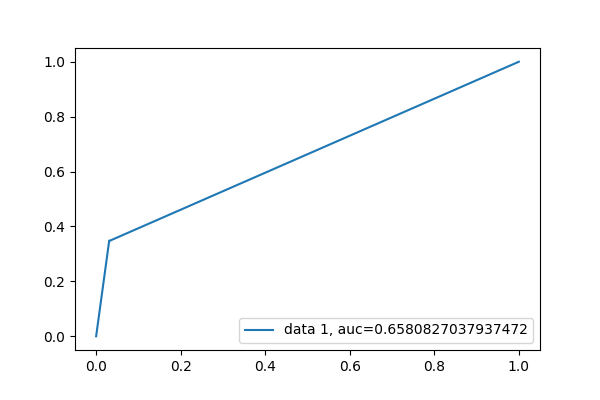
The first model was DecisionTree. All the independent variables were used to train the model. The classification report and roc\_auc\_curve are following:

precision recall f1-score support

0 0.96 0.97 0.96 93576

1 0.42 0.35 0.38 6186

avg / total 0.92 0.93 0.93 99762

Plot of feature importances of the decison tree showed that weeks\_worked\_in\_year had highest importance, which almost 275% more than then next most important feature.

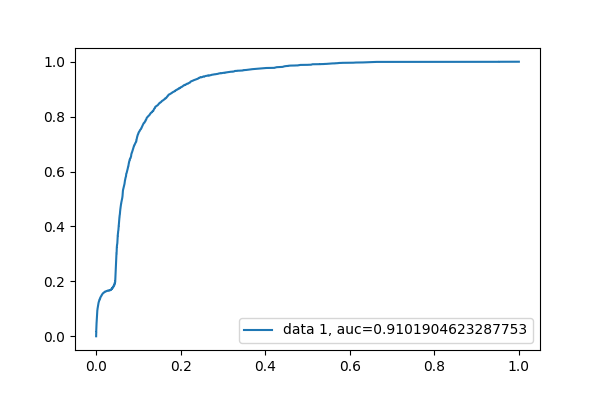
A GradientBoostingClassifier was used for the mext model. The classifcation\_report and roc\_auc\_curve are following:

precision recall f1-score support

0 0.99 0.84 0.91 93576

1 0.26 0.86 0.40 6186

avg / total 0.94 0.84 0.88 99762

Though auc was significantly higher, the model failed to predict with precision. Possibly the numercal encoding caused the model to fail.

**10. Binarizing the data**

To get the best prediction from GBM, it was required to binarize the data. Since we do not intend to reverse encode the binary data, buit-in get\_dummies attribute of the pandas DataFrame object is the most suitable, the code is also very clean.

To ensure uniformity in binarizing, train and test data was concatenated (axis=0), binarized and separated again. X and y data structures were created again. The new X and y dataframes are very heavy on memory due to approx 800 columns, hence DataFrames not in use were deleted to free up memory.

**11. DecisionTree with binarized data**

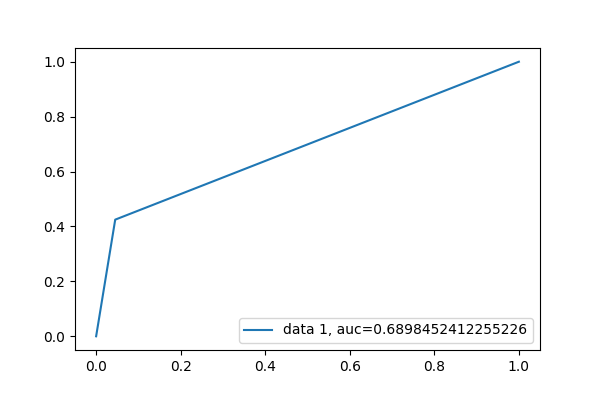
The classifcation\_report and roc\_auc\_curve for DecisionTree model trained with binary data are following:

precision recall f1-score support

0 0.96 0.97 0.96 93576

1 0.42 0.36 0.39 6186

avg / total 0.92 0.93 0.93 99762

There was no significant improvement in the metrics for DecisionTree model after binarizing.

Despite several attempts with altered parameters, GBM failed to train. Hence DecisionTree model with binary training data is recommended for the given data.

## 12. Conclusion

The most challenging aspect of the problem was to select the independent variables. The DecisionTree model trained with binary data predicts the minority clss with 42% precision, overall precision being 92%. There is no benchmark for a heavily skewed data like this. It is safe to assume that computationally intensive GBM will produce better results if trained with binarised data.