An article/blog on

Census Income Project

In this blog, we will analyze the Census income from the Machine Learning Repository.

We will see how to build a practical machine learning project. In general, any machine learning project requires the following steps:

* Defining the problem statement
* Exploratory Data Analysis
* Training the model
* Fine tuning the model
* Save the model

So let’s get started.

1. Problem definition

The weights on the Current Population Survey (CPS) files are controlled to independent estimates of the civilian non-institutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are:

1. A single cell estimate of the population 16+ for each state.
2. Controls for Hispanic Origin by age and sex.
3. Controls by Race, age and sex.

There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

First, we will import the required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import RandomOverSampler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import f1\_score

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.metrics import classification\_report

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

import warnings

**Downloading the data**

Instead of manually downloading the dataset, we will write a small script which downloads the content for a list of URLs and saves them in a folder.

Next, we load the data into a pandas dataframe using the read\_csv function.

*# Importing dataset*

dataset **=** pd**.**read\_csv('C:/Users/Jai Mata Di/Downloads/census\_income.csv')

**Descriptive analysis**

Describe the dataset and all data shows in the box by using of syntax and shows the preview.

*# Preview dataset*

dataset**.**head()

**shape of data**

its shows the size of data its means that no of columns and rows

*# Shape of dataset*

print('Rows: {} Columns: {}'**.**format(dataset**.**shape[0], dataset**.**shape[1]))

**Exploratory Data Analysis**

Let’s get more information about the training data using train\_data.info()

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

RangeIndex: 32560 entries, 0 to 32559

Data columns (total 15 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Age 32560 non-null int64

1 Workclass 32560 non-null object

2 Fnlwgt 32560 non-null int64

3 Education 32560 non-null object

4 Education\_num 32560 non-null int64

5 Marital\_status 32560 non-null object

6 Occupation 32560 non-null object

7 Relationship 32560 non-null object

8 Race 32560 non-null object

9 Sex 32560 non-null object

10 Capital\_gain 32560 non-null int64

11 Capital\_loss 32560 non-null int64

12 Hours\_per\_week 32560 non-null int64

13 Native\_country 32560 non-null object

14 Income 32560 non-null object

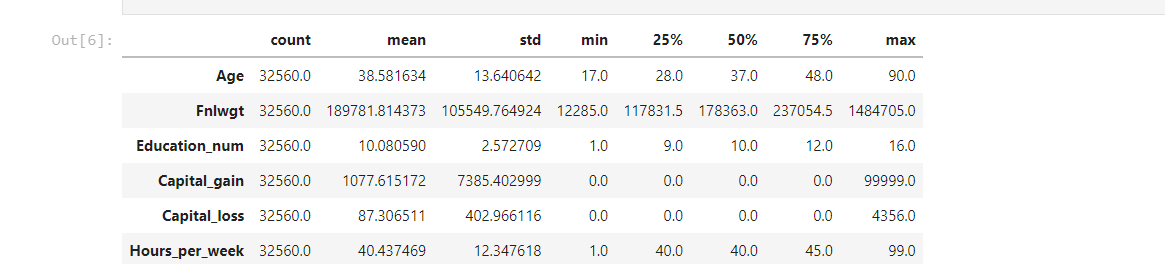
dtypes: int64(6), object(9)

memory usage: 3.7+ MB

**Observations**

* There are **32560** samples in the training dataset
* The columns **workClass**, **occupation**, **native-country** have missing values

**Data Visualizations**

More information about the data can be gathered by using train\_data.describe()

**Observations**

* None of the numerical attributes have missing values
* The values are on different scales. Many machine learning models require the values to be on the same scale. We will use standard scaler from the sklearn library to scale the features.

**Data Visualization**

Univariate Analysis

Its analyze the Univariate analysis explores each variable in a data set, separately. It **looks at the range of values**, as well as the central tendency of the values.

*# Creating a barplot for 'Income'*

income **=** dataset['Income']**.**value\_counts()

plt**.**style**.**use('seaborn-whitegrid')

plt**.**figure(figsize**=**(7, 5))

sns**.**barplot(income**.**index, income**.**values, palette**=**'bright')

plt**.**title('Distribution of Income', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

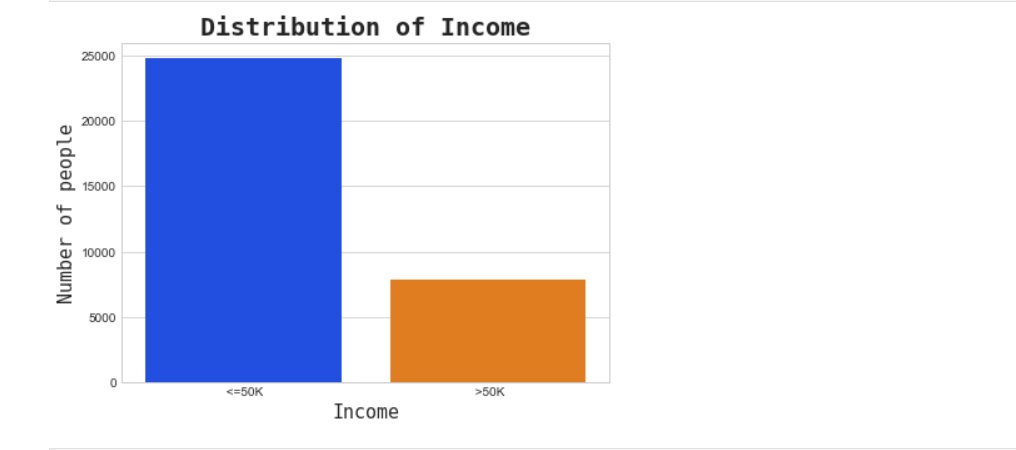
plt**.**xlabel('Income', fontdict**=**{'fontname': 'Monospace', 'fontsize': 15})

plt**.**ylabel('Number of people', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 15})

plt**.**tick\_params(labelsize**=**10)

plt**.**show()



*# Creating a distribution plot for 'Age'*

age **=** dataset['Age']**.**value\_counts()

plt**.**figure(figsize**=**(10, 5))

plt**.**style**.**use('fivethirtyeight')

sns**.**distplot(dataset['Age'], bins**=**20)

plt**.**title('Distribution of Age', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

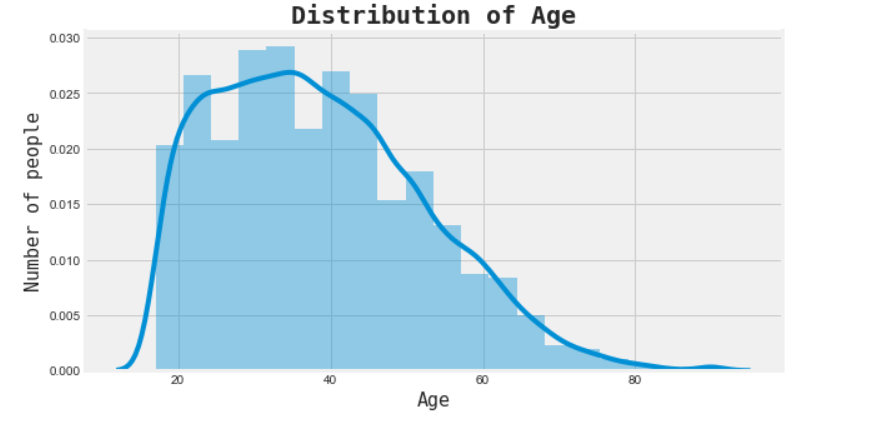
plt**.**xlabel('Age', fontdict**=**{'fontname': 'Monospace', 'fontsize': 15})

plt**.**ylabel('Number of people', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 15})

plt**.**tick\_params(labelsize**=**10)

plt**.**show()



*# Creating a pie chart for 'Marital status'*

marital **=** dataset['Marital\_status']**.**value\_counts()

plt**.**style**.**use('default')

plt**.**figure(figsize**=**(10, 7))

plt**.**pie(marital**.**values, labels**=**marital**.**index, startangle**=**10, explode**=**(

0, 0.20, 0, 0, 0, 0, 0), shadow**=True**, autopct**=**'%1.1f%%')

plt**.**title('Marital distribution', fontdict**=**{

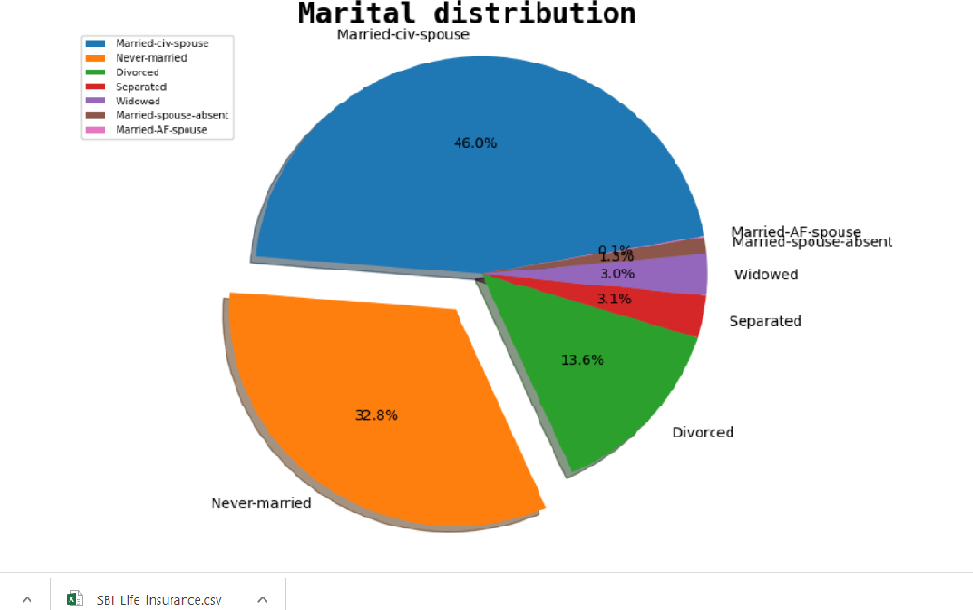
'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

plt**.**legend()

plt**.**legend(prop**=**{'size': 7})

plt**.**axis('equal')

plt**.**show()



Bivariate Analysis

Bivariate analysis means the analysis of bivariate data. It is one of the simplest forms of statistical analysis, **used to find out if there is a relationship between two sets of values**

*# Creating a countplot of income across age*

plt**.**style**.**use('default')

plt**.**figure(figsize**=**(20, 7))

sns**.**countplot(dataset['Age'], hue**=**dataset['Income'])

plt**.**title('Distribution of Income across Age', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

plt**.**xlabel('Age', fontdict**=**{'fontname': 'Monospace', 'fontsize': 15})

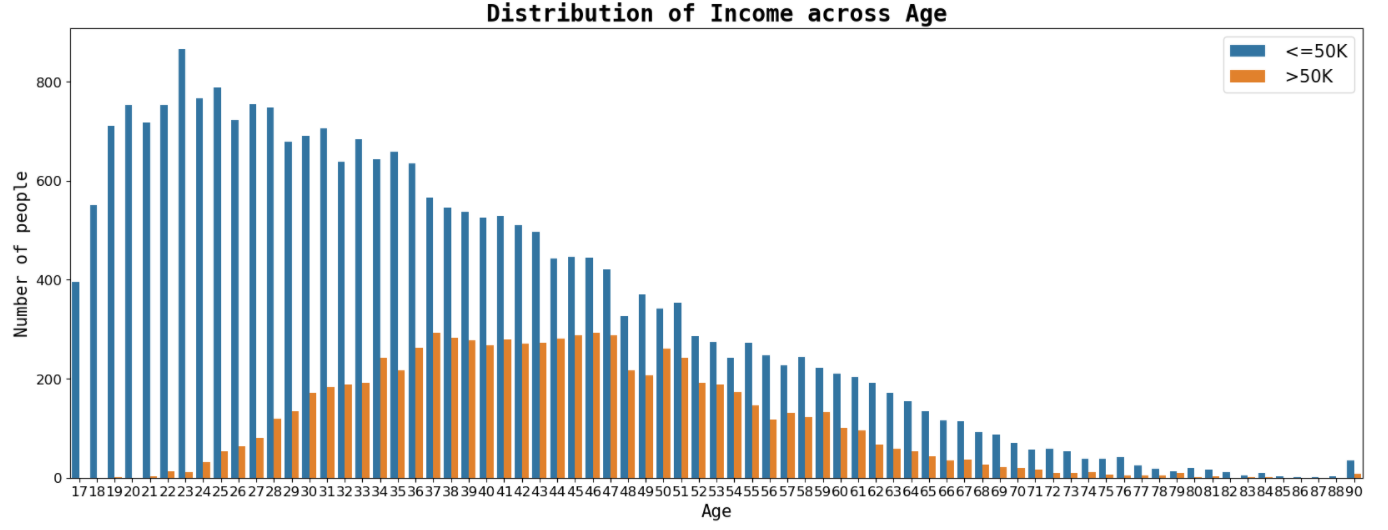
plt**.**ylabel('Number of people', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 15})

plt**.**tick\_params(labelsize**=**12)

plt**.**legend(loc**=**1, prop**=**{'size': 15})

plt**.**show()



*# Creating a countplot of income across education*

plt**.**style**.**use('seaborn')

plt**.**figure(figsize**=**(20, 7))

sns**.**countplot(dataset['Education'],

hue**=**dataset['Income'], palette**=**'colorblind')

plt**.**title('Distribution of Income across Education', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})

plt**.**xlabel('Education', fontdict**=**{'fontname': 'Monospace', 'fontsize': 15})

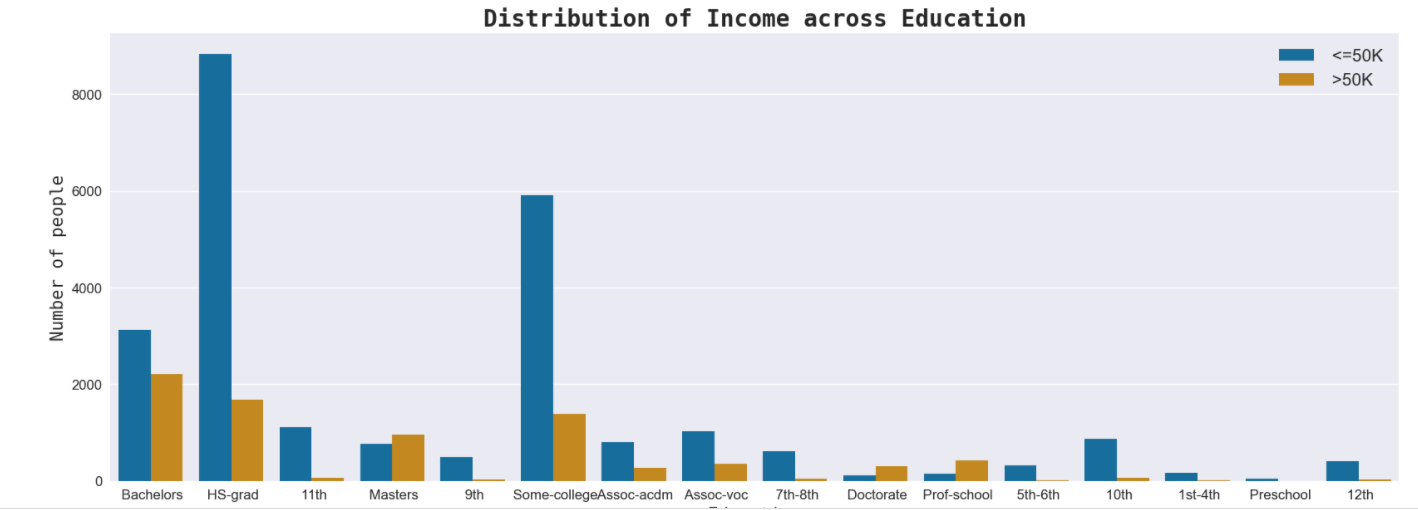
plt**.**ylabel('Number of people', fontdict**=**{

'fontname': 'Monospace', 'fontsize': 15})

plt**.**tick\_params(labelsize**=**12)

plt**.**legend(loc**=**1, prop**=**{'size': 15})

plt**.**show()



 Multivariate Analysis

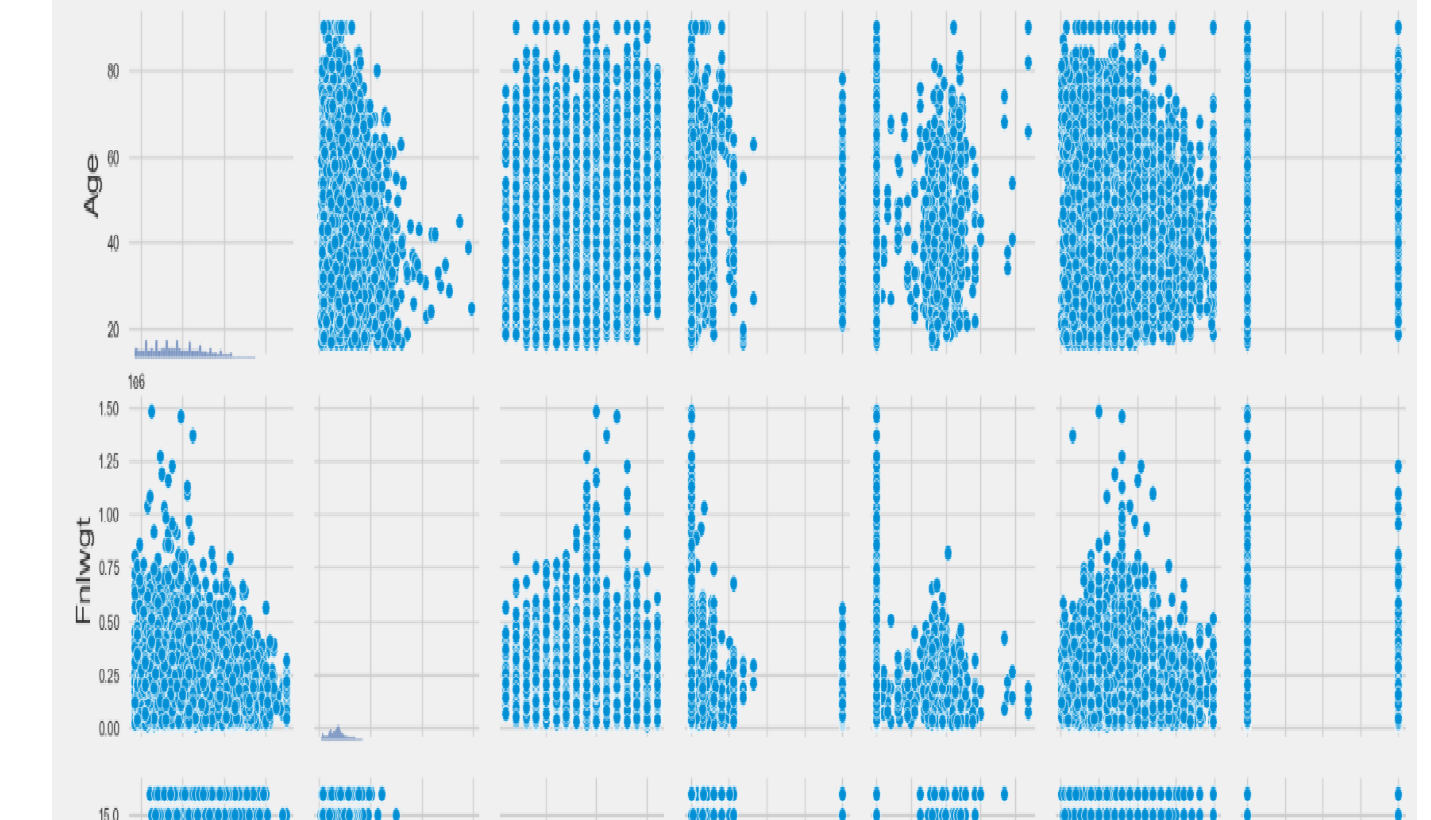
Multivariate analysis (MVA) is **a Statistical procedure for analysis of data involving more than one type of measurement or observation**.

*# Creating a pairplot of dataset*

sns**.**pairplot(dataset)

plt**.**savefig('multi1.png')

plt**.**show()



Correlation

Correlation (to be exact Correlation in Statistic) is **a measure of a mutual relationship between two variables whether they are causal or not**.

corr **=** dataset**.**corr()

mask **=** np**.**zeros\_like(corr)

mask[np**.**triu\_indices\_from(mask)] **=** **True**

**with** sns**.**axes\_style("white"):

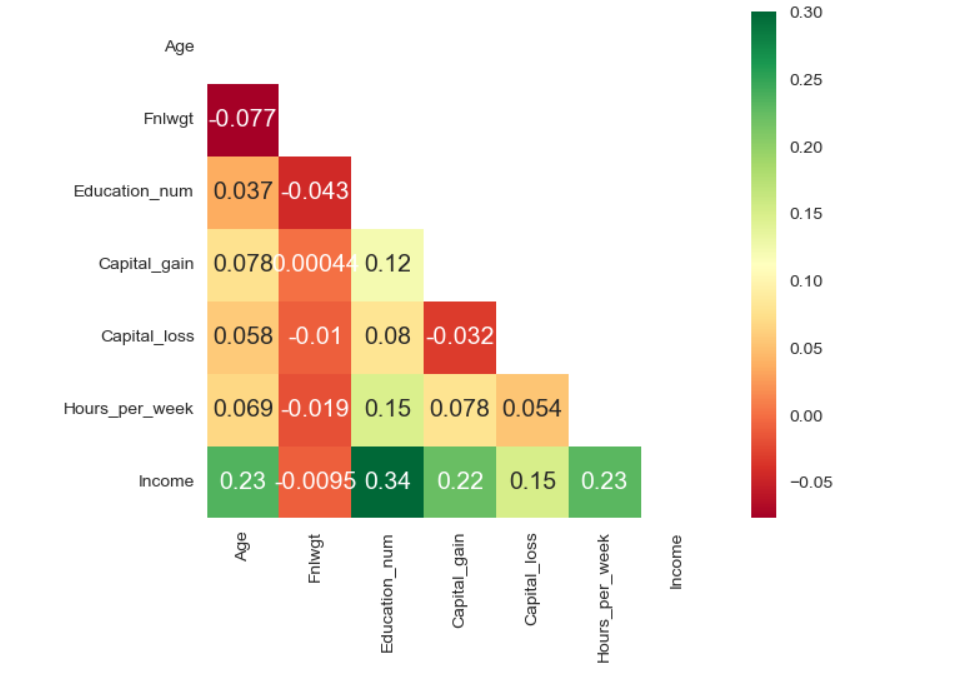
f, ax **=** plt**.**subplots(figsize**=**(7, 5))

ax **=** sns**.**heatmap(corr, mask**=**mask, vmax**=**.3, square**=True**,

annot**=True**, cmap**=**'RdYlGn')

plt**.**savefig('multi2.png')

plt**.**show()



**Data Preprocessing**

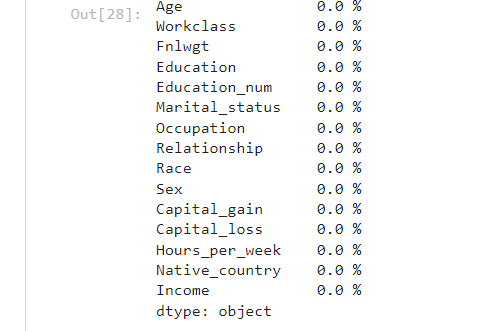
Data Preprocessing includes **the steps we need to follow to transform or encode data so that it may be easily parsed by the machine**. The main agenda for a model to be accurate and precise in predictions is that the algorithm should be able to easily interpret the data's features.

dataset **=** dataset**.**replace('?', np**.**nan)

In [28]:

*# Checking null values*

round((dataset**.**isnull()**.**sum() **/** dataset**.**shape[0]) **\*** 100, 2)**.**astype(str) **+** ' %'

****

columns\_with\_nan **=** ['Workclass', 'Occupation', 'Native\_country']

In [30]:

**for** col **in** columns\_with\_nan:

dataset[col]**.**fillna(dataset[col]**.**mode()[0], inplace**=True**)

Label Encoding

**for** col **in** dataset**.**columns:

**if** dataset[col]**.**dtypes **==** 'object':

encoder **=** LabelEncoder()

dataset[col] **=** encoder**.**fit\_transform(dataset[col])

X **=** dataset**.**drop('Income', axis**=**1)

Y **=** dataset['Income']

**Feature Selection**

Feature Selection is the **process where you automatically or manually select those features which contribute most to your prediction variable or output** in which you are interested in.

selector **=** ExtraTreesClassifier(random\_state**=**42)

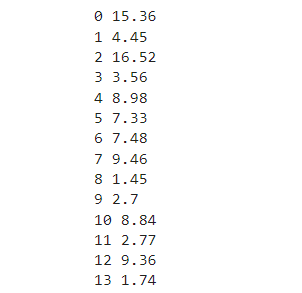
selector**.**fit(X, Y)

feature\_imp **=** selector**.**feature\_importances\_

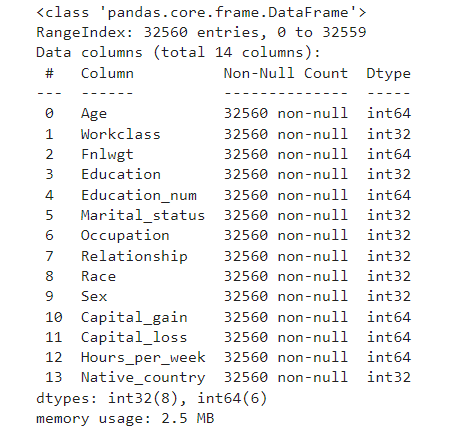
In [37]:

**for** index, val **in** enumerate(feature\_imp):

print(index, round((val **\*** 100), 2))



X**.**info()



**Feature Scaling**

Feature scaling is **a method used to normalize the range of independent variables or features of data**.

**for** col **in** X**.**columns:

scaler **=** StandardScaler()

X[col] **=** scaler**.**fit\_transform(X[col]**.**values**.**reshape(**-**1, 1))

round(Y**.**value\_counts(normalize**=True**) **\*** 100, 2)**.**astype('str') **+** ' %'

0 75.92 %

1 24.08 %

Name: Income, dtype: object

ros **=** RandomOverSampler(random\_state**=**42)

ros**.**fit(X, Y)

RandomOverSampler(random\_state=42)

X\_resampled, Y\_resampled **=** ros**.**fit\_resample(X, Y)

round(Y\_resampled**.**value\_counts(normalize**=True**) **\*** 100, 2)**.**astype('str') **+** ' %'

**Creating a train test split**

The train-test split is a **technique for evaluating the performance of a machine learning algorithm**. It can be used for classification or regression problems and can be used for any supervised learning algorithm.

X\_train, X\_test, Y\_train, Y\_test **=** train\_test\_split(

X\_resampled, Y\_resampled, test\_size**=**0.2, random\_state**=**42)

print("X\_train shape:", X\_train**.**shape)

print("X\_test shape:", X\_test**.**shape)

print("Y\_train shape:", Y\_train**.**shape)

print("Y\_test shape:", Y\_test**.**shape)

output-:

X\_train shape: (39550, 8)

X\_test shape: (9888, 8)

Y\_train shape: (39550,)

Y\_test shape: (9888,)

**Data Modelling**

Data modeling is **the process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures**.

log\_reg **=** LogisticRegression(random\_state**=**42)

log\_reg**.**fit(X\_train, Y\_train)

LogisticRegression(random\_state=42)

Y\_pred\_log\_reg **=** log\_reg**.**predict(X\_test)

 KNN Classifier

KNN works by **finding the distances between a query and all the examples in the data**, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

knn **=** KNeighborsClassifier()

knn**.**fit(X\_train, Y\_train)

output-:

KNeighborsClassifier()

Y\_pred\_knn **=** knn**.**predict(X\_test)

**Support Vector Classifier**

SVM algorithm **finds the closest point of the lines from both the classes**. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin.

svc **=** SVC(random\_state**=**42)

svc**.**fit(X\_train, Y\_train)

output-:

SVC(random\_state=42)

Y\_pred\_svc **=** svc**.**predict(X\_test)

**Naive Bayes Classifier**

Naive Bayes classifiers are **a collection of classification algorithms based on Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

**from** sklearn.naive\_bayes **import** GaussianNB

nb **=** GaussianNB()

nb**.**fit(X\_train, Y\_train)

ouput-:

GaussianNB()

Y\_pred\_nb **=** nb**.**predict(X\_test)

**Decision Tree Classifier**

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The **goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features**.

dec\_tree **=** DecisionTreeClassifier(random\_state**=**42)

dec\_tree**.**fit(X\_train, Y\_train)

output-:

DecisionTreeClassifier(random\_state=42)

Y\_pred\_dec\_tree **=** dec\_tree**.**predict(X\_test)

**Random Forest Classifier**

The random forest classifier is a versatile classification tool that makes **an aggregated prediction using a group of decision trees trained using the bootstrap method with extra randomness.**

ran\_for **=** RandomForestClassifier(random\_state**=**42)

ran\_for**.**fit(X\_train, Y\_train)

output-:

RandomForestClassifier(random\_state=42)

Y\_pred\_ran\_for **=** ran\_for**.**predict(X\_test)

**XGB Classifier**

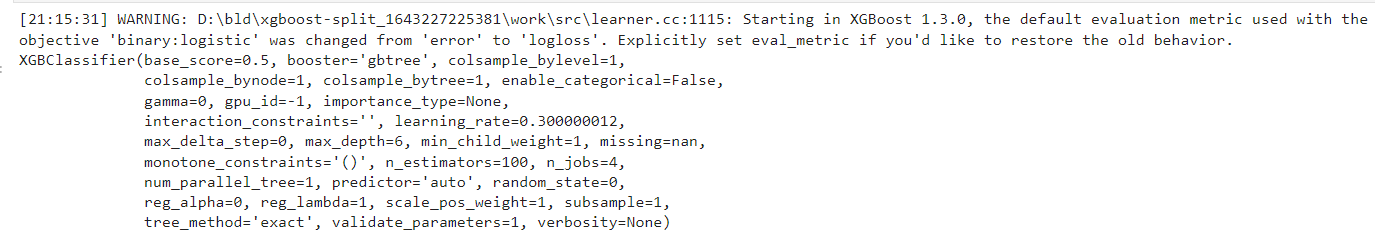
XGBoost provides **a wrapper class to allow models to be treated like classifiers or regressors in the** scikit-learn framework. This means we can use the full scikit-learn library with XGBoost models. The XGBoost model for classification is called XGBClassifier. We can create and and fit it to our training dataset.

**from** xgboost **import** XGBClassifier

xgb **=** XGBClassifier()

In [69]:

xgb**.**fit(X\_train, Y\_train)

****

# Model Evaluation

We will use [accuracy\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html" \t "_blank) from sklearn to find the accuracy of the model

The accuracy is 92.59

Let’s plot the [confusion matrix](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html).

cm **=** confusion\_matrix(Y\_test, Y\_pred\_rf\_best)

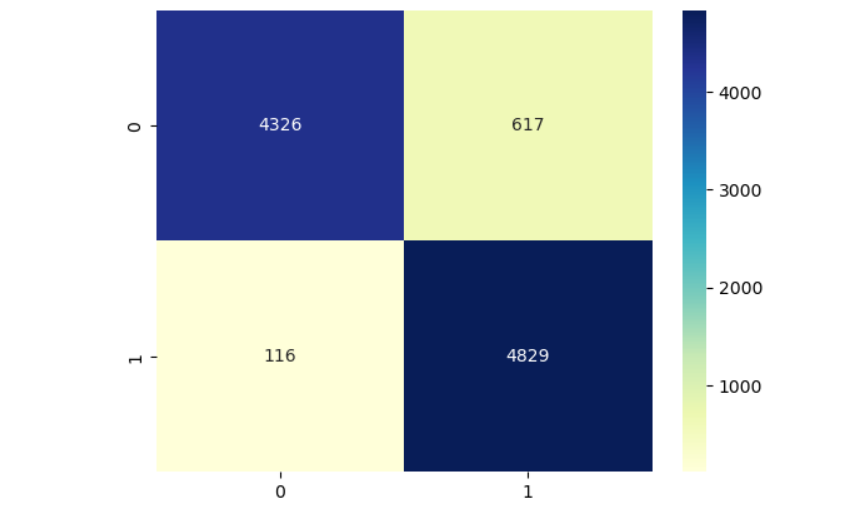
In [92]:

plt**.**style**.**use('default')

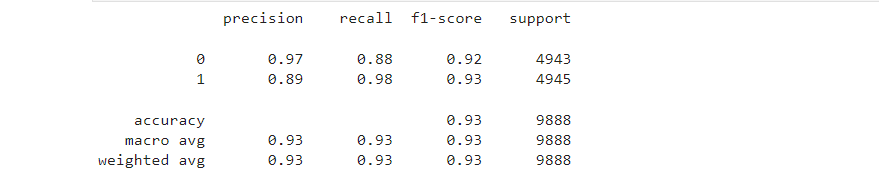
sns**.**heatmap(cm, annot**=True**, fmt**=**'d', cmap**=**'YlGnBu')

plt**.**savefig('heatmap.png')

plt**.**show()



print(classification\_report(Y\_test, Y\_pred\_rf\_best))



The X-axis represents the Predicted classes and the Y-axis represents the Actual classes. How do we interpret the confusion matrix? 1.2e+04 times the model correctly predicted the class 0 when the actual class was 0. Similarly, conclusions can be drawn for the remaining cases.

# **Final Remarks**

We have learned to build a complete machine learning project. In the process, we built custom transformers that can be used with different algoriths and different libraries.

Please let me know if there is any part I could have done better.

Thanks for Reading!!