**Census Income Project**

**Problem Statement:**

This data was extracted from the [1994 Census bureau database](http://www.census.gov/en.html). The data set contains different information required to predict the Income of the individuals. The prediction task is to determine whether a person makes over $50K a year or not.

Attribute information:

The dataset contains the following columns->

1. age: Age of the person.
2. workclass: Working class of the person.
3. fnlwgt: Final weight (this is the number of people the census believes the entry represents)
4. education: Highest education done by the person
5. education-num: Education number done by the person.
6. marital-status: Marital status of the person
7. occupation: occupation of the person
8. relationship: Relationship status of the person
9. race: Race of the person
10. sex: Sex of the person
11. capital-gain: Capital-gain of the person
12. capital-loss: Capital-loss of the person
13. hours-per-week: Hours per week worked by the person
14. native-country: Native country of the person
15. Income: Income status of the person

**Data Analysis:**

First, we will import few basic libraries that will be required for this project:



Now we will import the dataset using pandas -



Here df will be our data frame in which we have imported the dataset.

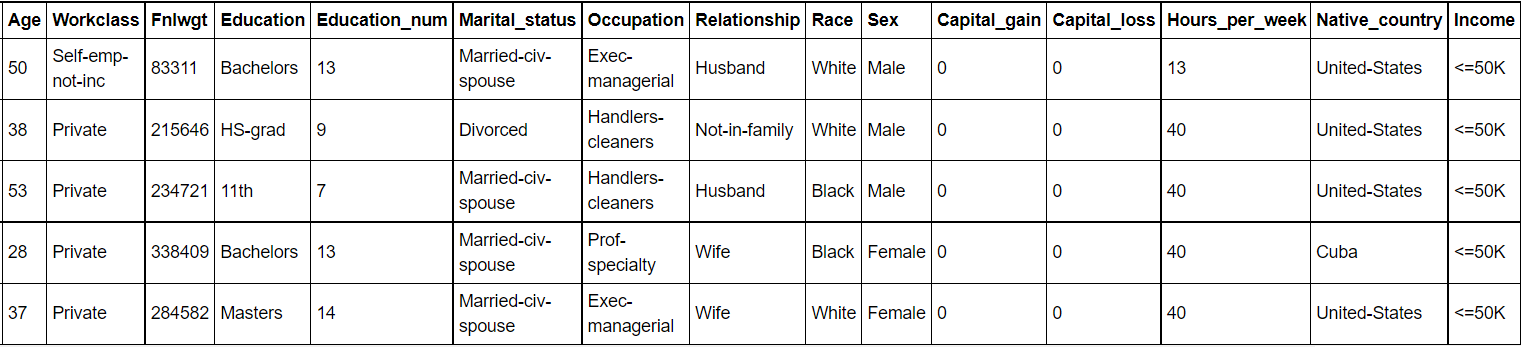
Let’s start exploring our dataset for the Census Income problem.

First step for analysing the data is checking whether we have the right type of data in the columns or not. We can test this by checking the head of the dataset. To check the head of the dataset we can use ‘head()’ function available in pandas.

Code:



Output:

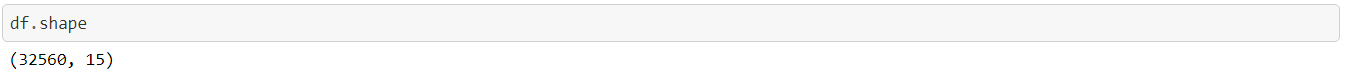


We got the head of the dataset. But the question is what does it means to have right type of data in the columns?

We can explore one example from the above head : For ‘Hours\_per\_week’ column we can infer that numeric i.e. Integer values should be present in this object. But further if we check the data type of the ‘Hours\_per\_week’ column using dtypes() function of pandas and if it turns out to be object type. This means there is a miss match between the object type that is present and object type that is required. In that case, we will be required to convert the type and correct it. But for now, let’s get back to our current dataset and its scenarios.

Further we will check what is the shape of the dataset. We can do this by using shape attribute in pandas.

Code/Output:



So, we have 32560 rows and 15 columns in our dataset.

Now, we will find whether the dataset contains incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data.

Q) What can we do to check all of the above in our dataset?

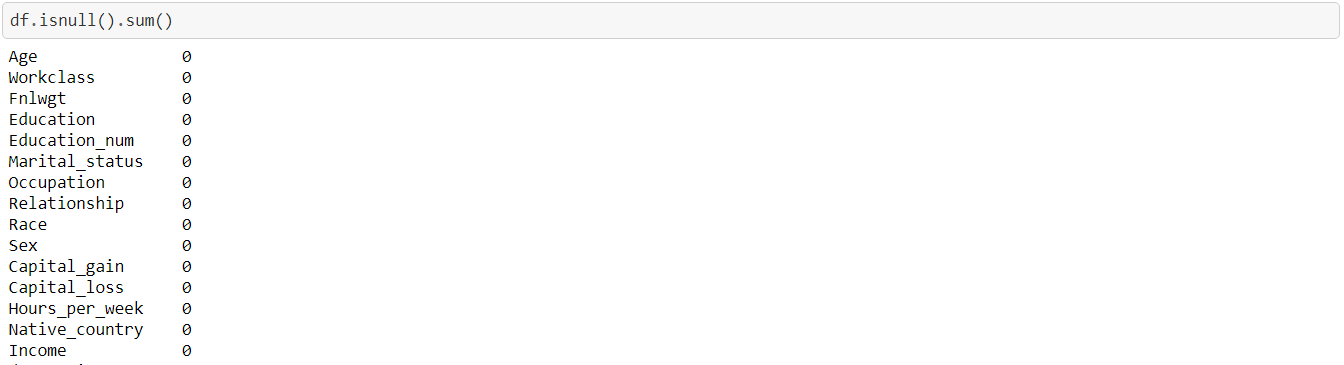
A) There are various techniques to explore our dataset for the anomalies, few of them are checking null values, duplicate values, outliers, skewness and checking whether the data is normally distributed or not.

**Let’s discuss them one by one:**

1. **Checking null values:**

We can use the pandas inbuilt function to check the null values in the dataset:

Code/Output:



We can see that there are no null values in the dataset.

We have both integer and string type values in our dataset, keeping that in mind, is this the correct result? Do we need to cross validate this? What should we do?

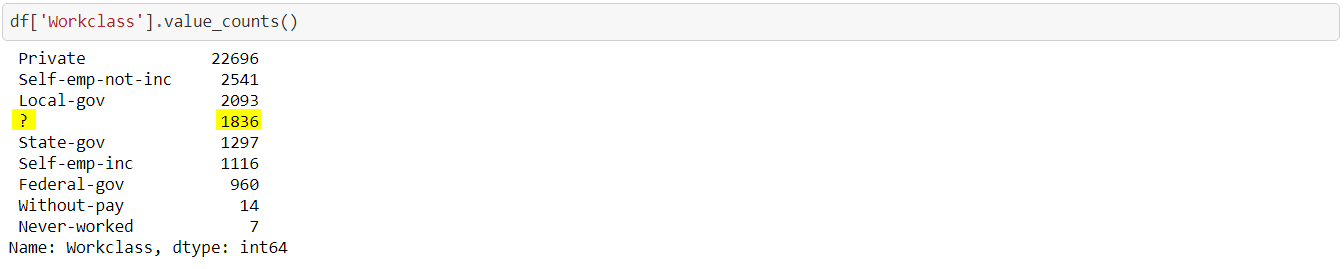
.

.

Yes, we need to cross validate this as in string type columns we might have null values present in the form of ‘?’, ‘- ‘or any other special character. How can we check the same in the string type column? The dataset is huge and checking values manually will cost us time and efforts. But we need not worry as we have huge number of inbuilt functions in python libraries at our hands.

We can use ‘value\_count’ function in pandas to check the unique values and its count present in the columns:

Code/Output:



As highlighted above, there is ‘?’ value present in the ‘Workclass’ column. Hence, null values are present in this column but according to the isnull() function there were no null values. Therefore, for string datatypes it is always better to cross validate the result. The more we explore our dataset the better result it will provide during model building.

Similarly, we can find the null values for other columns.

1. **Duplicate Rows:**

To check the duplicate rows in the dataset we will use ‘duplicated’ function available in pandas



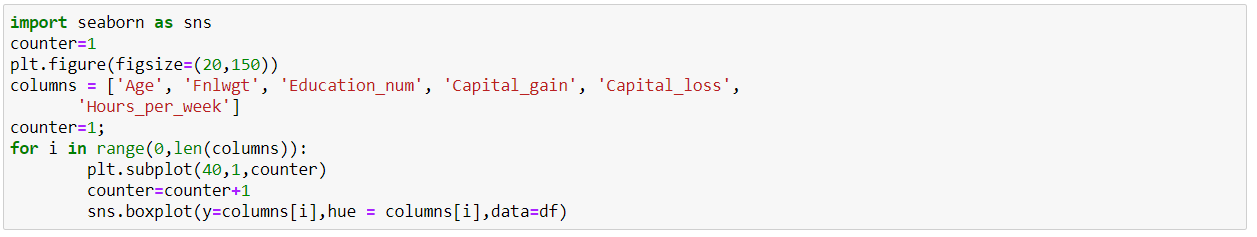
We have 24 duplicated rows in the dataset.

1. **Outliers:**

We will check the outliers in the Integer/Float type columns. Various methods are available to check outliers in the dataset and we will use one of the methods i.e., by plotting Box-Plot to check the outliers in the dataset.

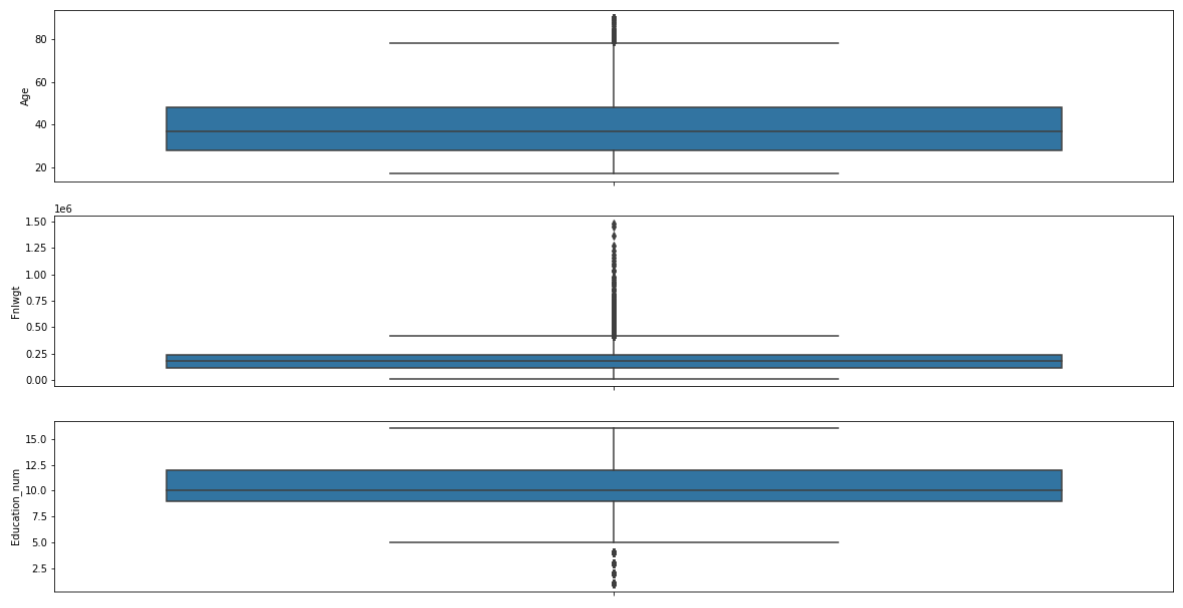
We will import the seaborn library to plot the boxplot.

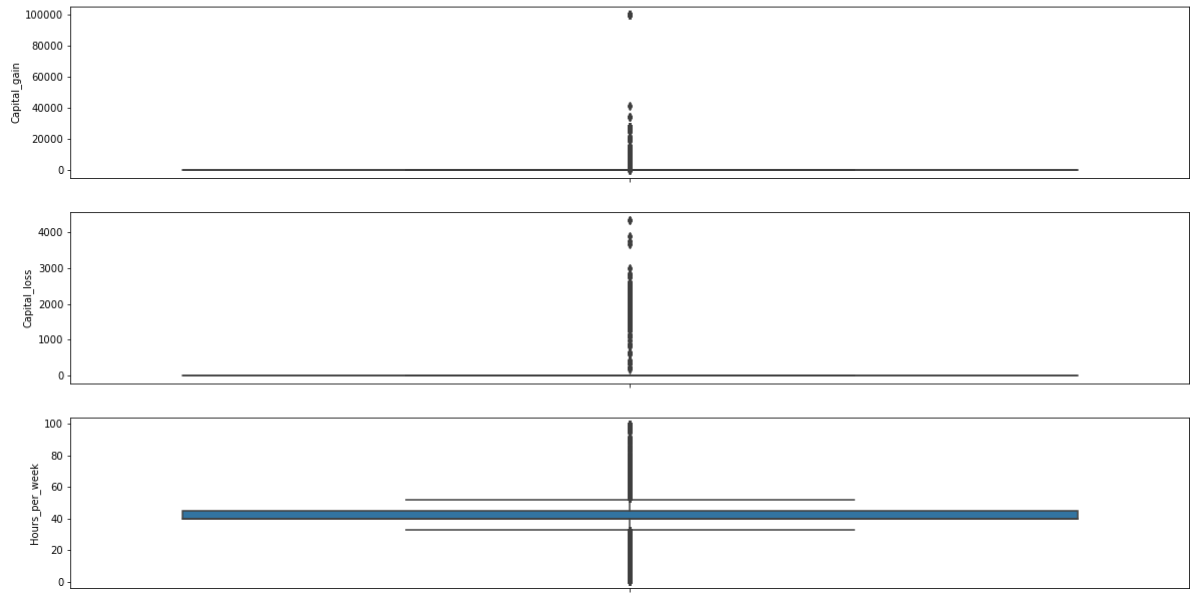
The code for checking the outliers for the numeric columns of our dataset:



In the above code, we have stored the Numeric/Integer/Float columns name in a list to plot their box-plot one by one.

Output:





We can observe that outliers are present in all the numeric columns. How did we got to know from the above plots that outliers are present? Well, if any value is present in the plot outside of the upper limit and lower limit of the box plot that are considered as outliers.

If we need to calculate the upper/lower limit we can do it using below formula:

IQR = df[i].quantile(0.75)-df[i].quantile(0.25)

Upper\_limit = df[i].quantile(0.75) + 1.5\*IQR

Lowerl\_imit = df[i].quantile(0.25) - 1.5\*IQR

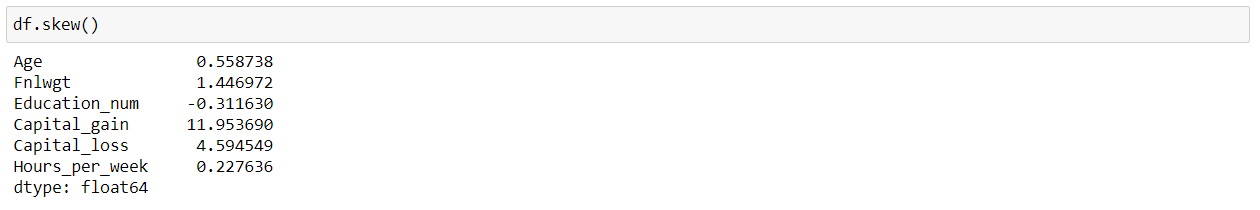
(Here ‘i’ denotes the column name)

Any values below the lower limit and any values above the upper limit will be identified as outliers.

1. **Skewness:**

Let’s check the skewness of the dataset using pandas function skew()

Code/ Output:



As we know that the skewness outside the range of -0.5 to +0.5 is not good for model building and in our dataset skewness for few columns is very high.

We can see that ‘Capital\_gain’ and ‘Capital\_loss’ columns ‘Fnlwgt’ are highly right skewed.

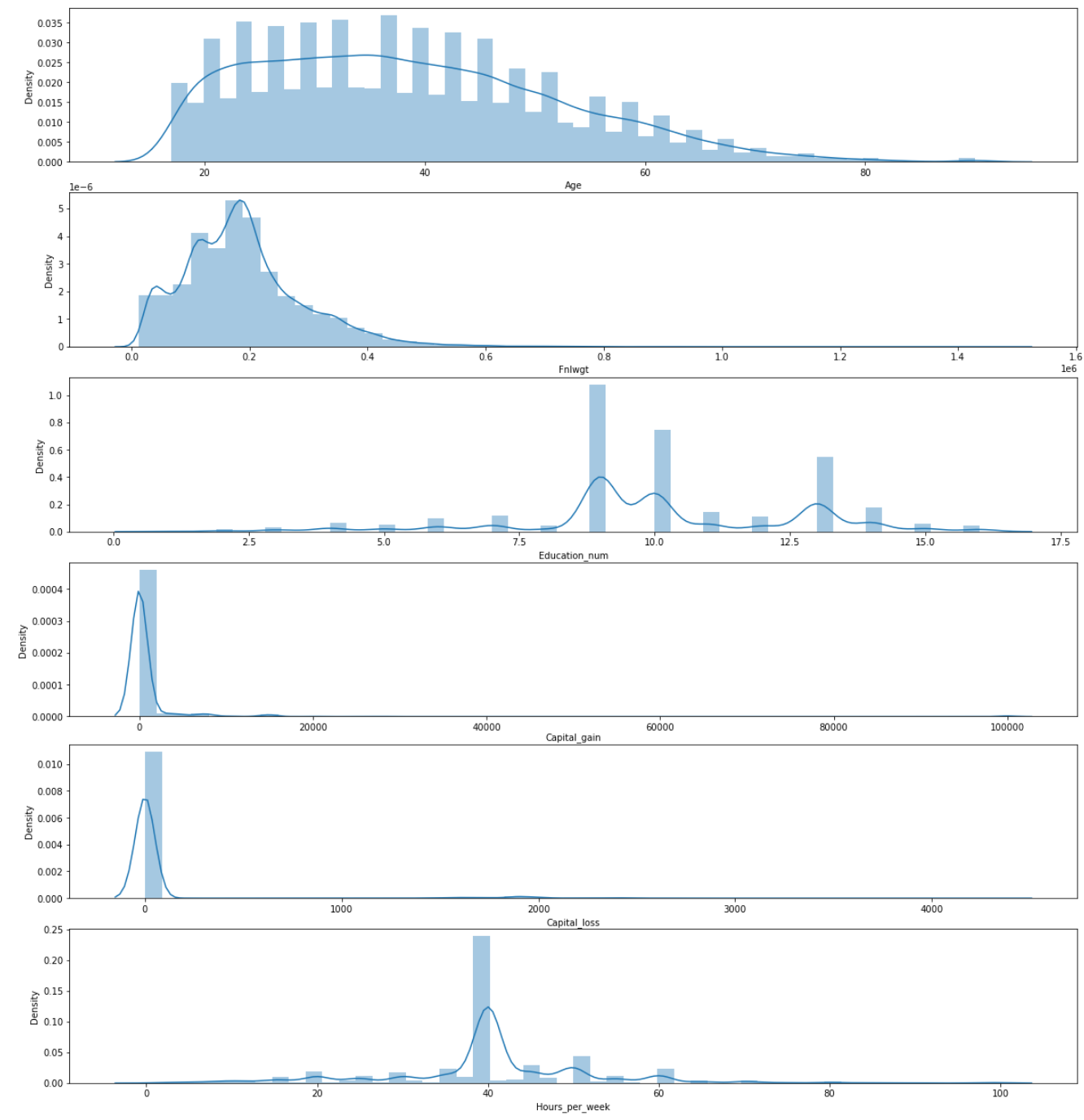
1. **Distribution of the data in columns:**

We can check whether the data is normally distributed or not using distance plot. We will use same sea born library to plot this graph

Code:



Output:



From above plots we can see for ‘Fnlwgt’, ‘Education\_num’ and ‘Hours\_per\_week’ have multiple peeks in the graph which means that the data is not normally distributed.

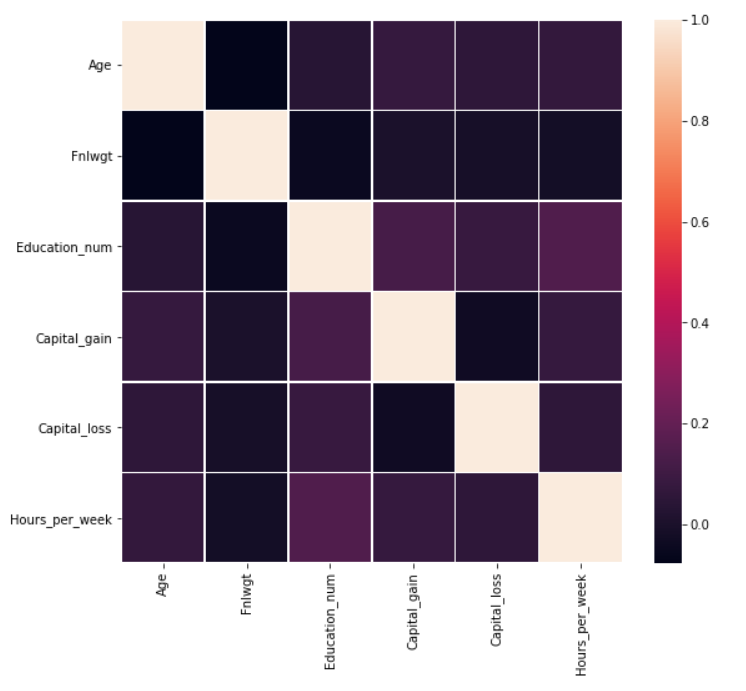
Now, after checking anomalies in our dataset, let’s check whether there is any correlation between the columns.

We will use ‘corr()’ method to find correlation and ‘heatmap’ plot to visualize the same.

Code:



Output:



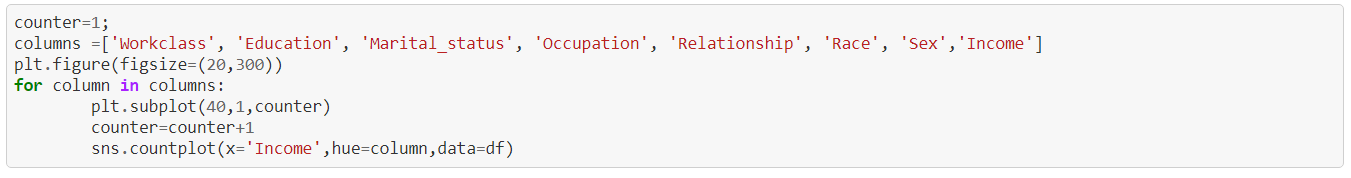
We can see that there is very weak correlation between the Numeric columns.

Now, let’s explore our String type or we can say object type columns in the dataset.

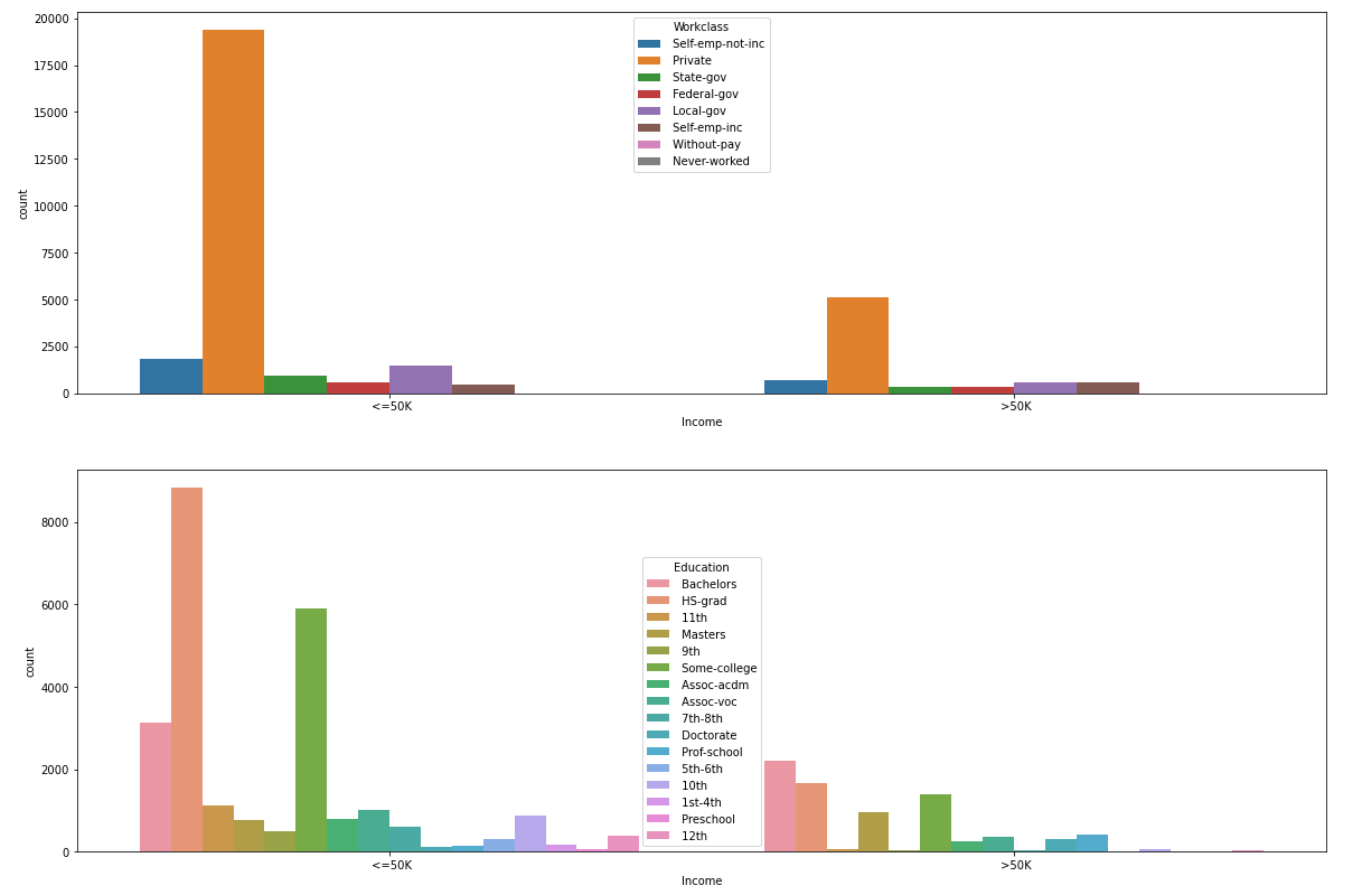
One important thing to note, as the target variable is of string type, the exploration of the string type column might become important for us. We will discuss why it is important to observe the target column data later after visualizing the data.

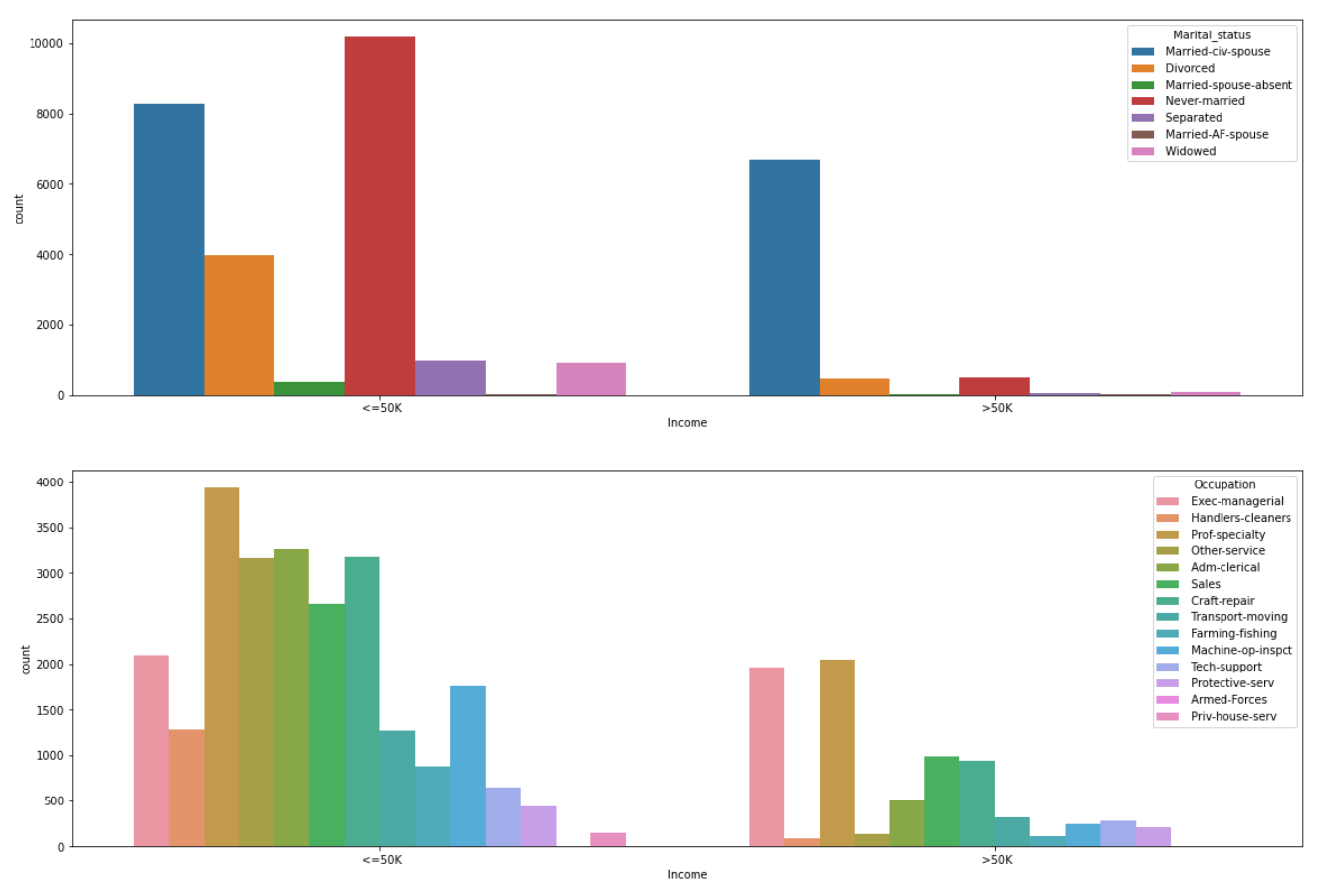
So, let’s visualize the categorical columns against the Target column:

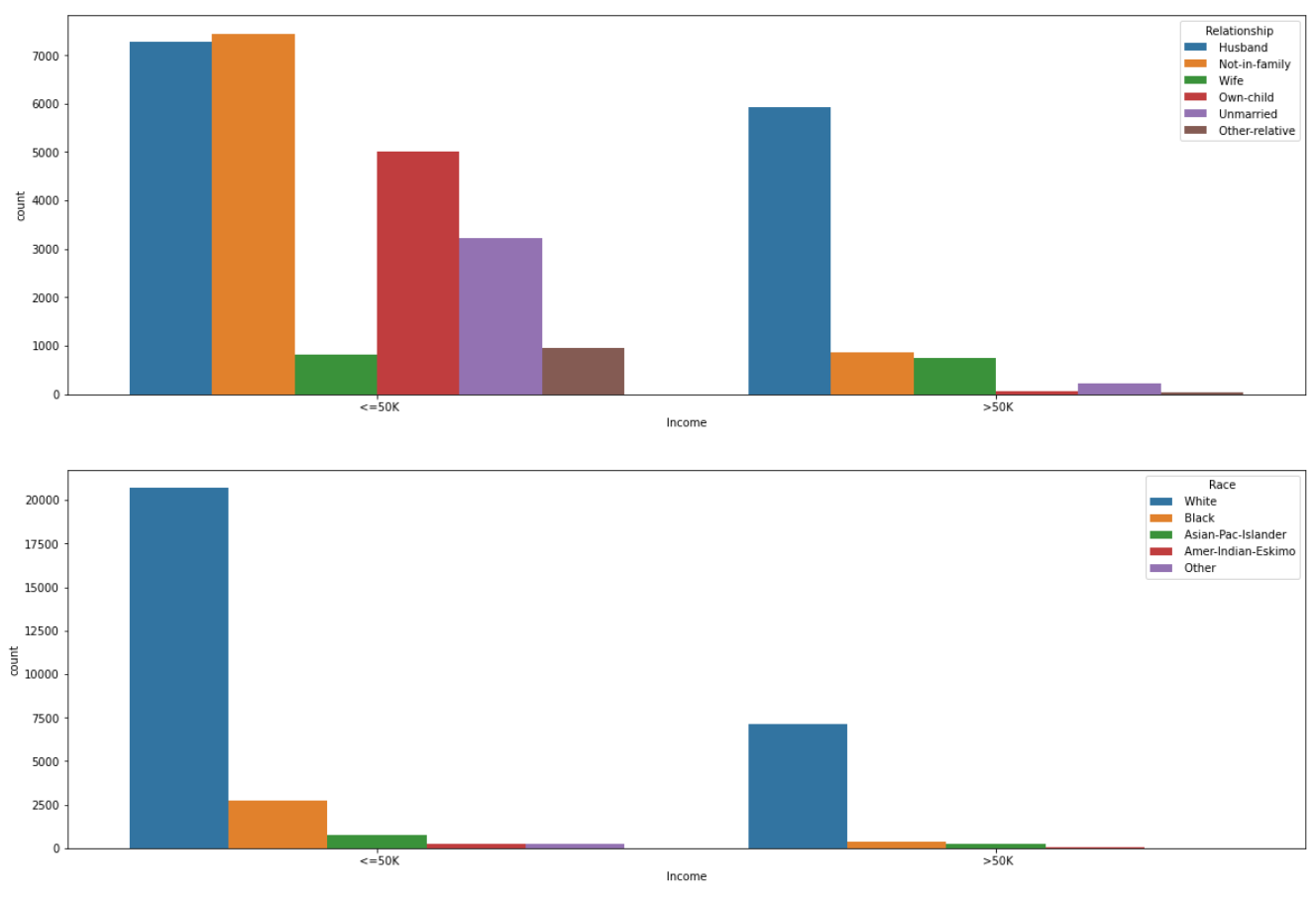
Code:

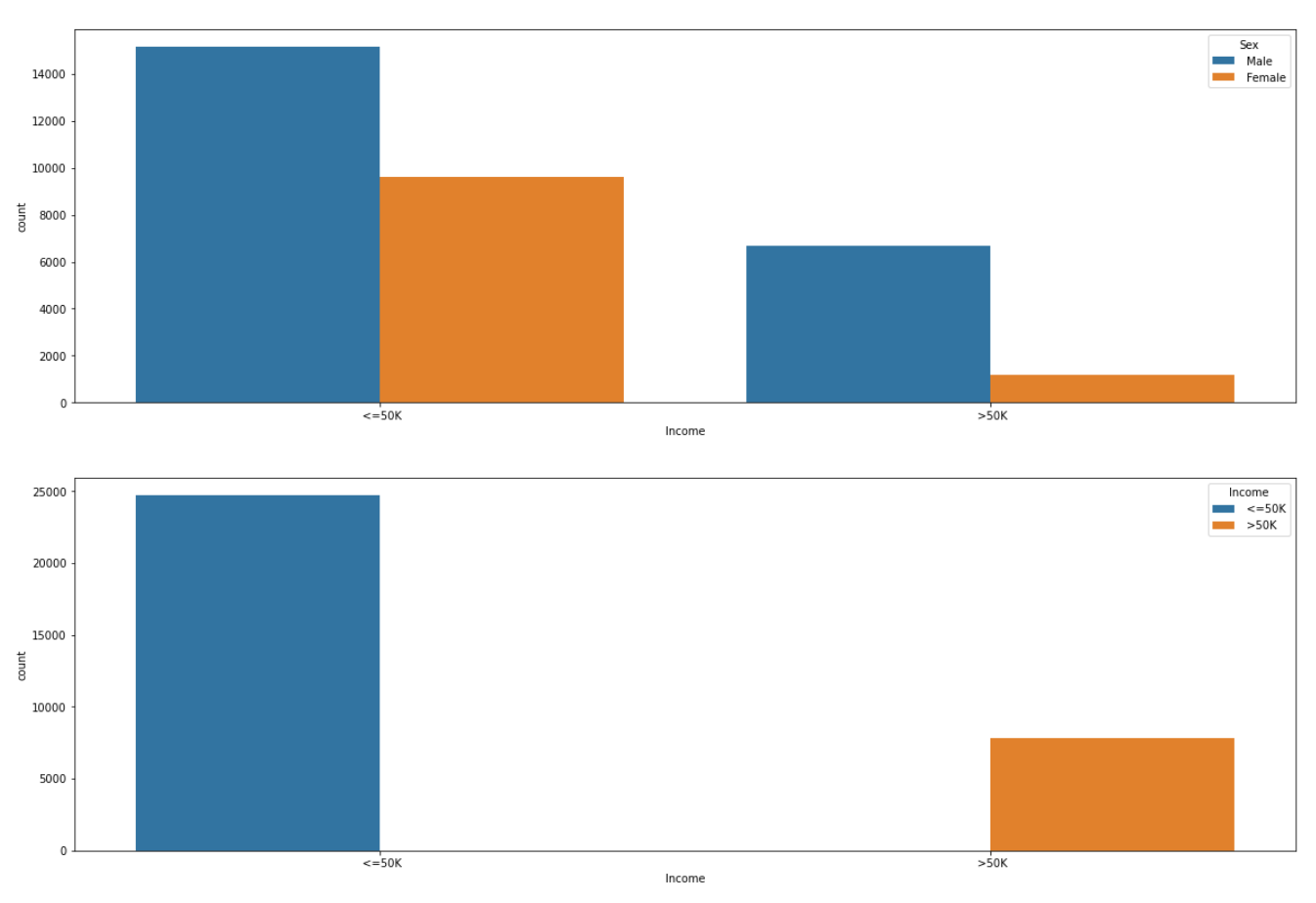


Output:









Let’s discuss our findings from above graphs.

* Private Work class is majorly earning Income <=50K
* Education qualification of HS-grad have income <=50K
* Married-civ-spouse have >50K income as compared to other marital status
* Husbands are the major earning members as compared to other relations
* Majority of the Income earning race is White
* Majority of females have income <=50K
* Data in Income (target variable) is highly unbalanced

Before visualizing above graphs, we said observing target variable is important, let’s discuss it now.

From the above findings, we saw data in ‘Income’ column is highly unbalanced. Does this mean anything for us? Will this affect the model building later on?

Yes, this is the most important observation for us. The data is highly unbalanced because we have more than 25000 records for the income of <=50K and only 8000 records for the income of >=50K. If we train our prediction model for the dataset in this current situation, then it will be highly biased towards income <=50K. We can fix this before building our model using certain python libraries specifically designed for this purpose. So, let’s move forward as we will fix this in the Model Building Section.

**EDA Concluding Remark:**

We found that in our dataset we have null values/missing values, after that we also checked that our dataset contains duplicate rows. For numeric values we have lots of outliers which also contributes in making our dataset skewed. Then we found that few columns are not normally distributed and the target variable is highly unbalanced. We will solve all of these to make our dataset clean and increase the model efficiency in the next section. Please note, the more we analyse/clean the dataset the better the predicting models will perform.

**Pre-Processing Pipeline:**

Let’s start processing our data to move one step closer for model building. Previously, we found that there are many anomalies which are required to be removed or resolved in the dataset. We will process them one by one.

**Removing Null values:**

There are several methods for solving Null values issue. We can remove them or replace them with Mean/Median/Mode depending upon the data present in the column.

We will replace the ‘?’ in the columns with mode of that particular column. We can also choose the remove method. But let’s assume that data is very costly and we cannot afford huge data loss. So, best method will be replacement of null or garbage values.

Code:



Here, we have used replace method to replace the ‘?’ values with mode.

**Removing the duplicate rows:**

As the duplicate rows will only increase the load on the model while training and it does not help us in any way for improving the prediction models, we will drop them

Code:

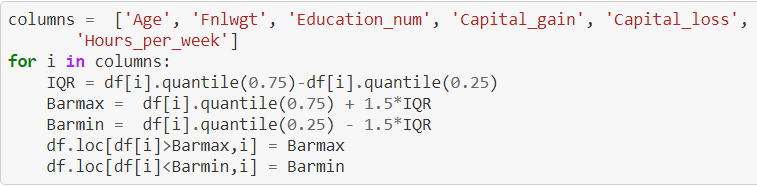


**Outlier treatment:**

We found that in numeric columns there were huge numbers of outliers. Treating them is important for better prediction. There are several ways to treat outliers. Either we can remove them from the dataset or replace them with appropriate values.

But as we have discussed before we have to keep the data loss as low as possible. So, we will use the replacement method.

Code:



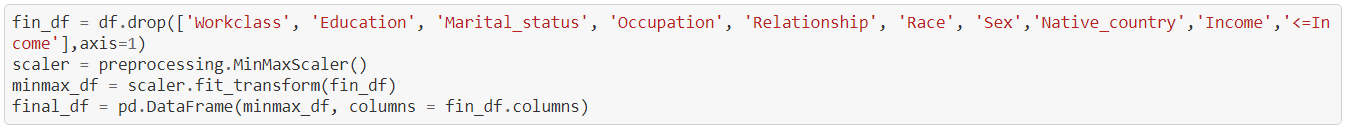
In the above code, we have replaced the outliers with upper limit and lower limit as per the Bar-plot.

**Scaling the data:**

While visualising the dataset, we observed that few columns are not normally distributed. We can fix this issue by scaling the data. There are several ways to scale the data like using Standard scaler, Min-Max scaler etc. We will use Min-Max scaler in this case:

The Min-Max scaler is available in ‘sklearn’ library.

Code:



Did you notice that first we dropped few columns? And here the question arises why did we dropped them? Answer to this question is very simple. If we observe that we dropped only categorical columns or we can say string type columns, scaling of the data is only applied on numeric data. Hence, we dropped the non-required columns before scaling the data. We need not worry as we will join them later on to our dataset.

**Removing Skewness:**

The skewness can be removed using several different methods. We can use log, sqrt, cbrt, binning etc methods for this. But here we will use binning method for our dataset. Here, we will again use python library to use the power\_tranform the dataset.

The ‘power\_transform’ is available in ‘sklearn.preprocessing’

Code:

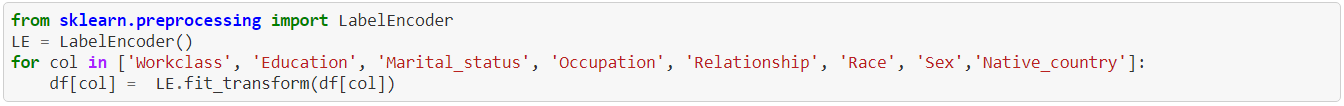


Now, the only thing left is joining the categorical strings back to the transformed or we can anomaly free dataset. But are we sure that this is enough? Do we need to focus on anything else? Can we start building the data set yet?

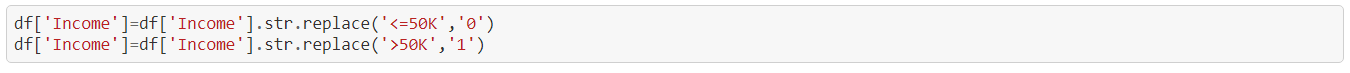
Well, we have not fully prepared our dataset yet for model training. The models only take numeric values as input/output for the training/testing. But we have categorical/string values present in the dataset. Hence, we will be required to convert the categorical values into numeric values. We can do this using Labels.

We will again use the ‘sklearn’ library to perform the same using Label Encoders

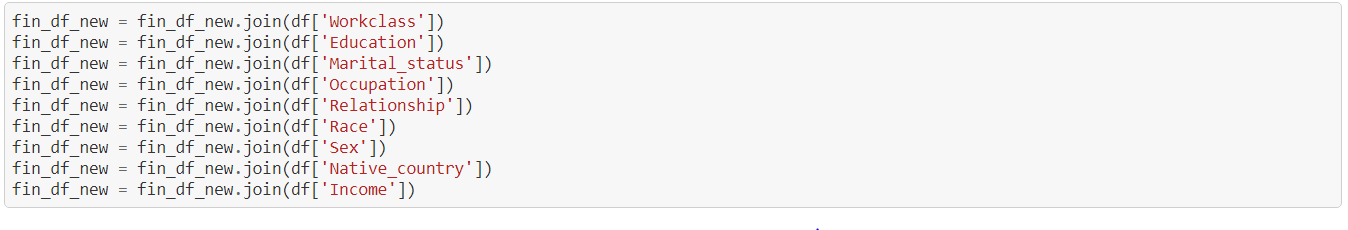
Code:



For target variable:



Let’s join the converted categorical variables back to the final dataset:

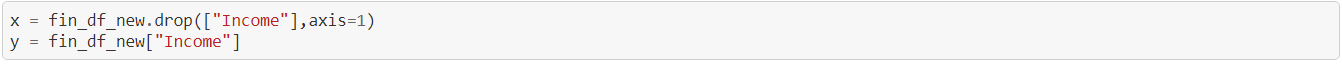


Now we are ready to move onto the next phase i.e., model building

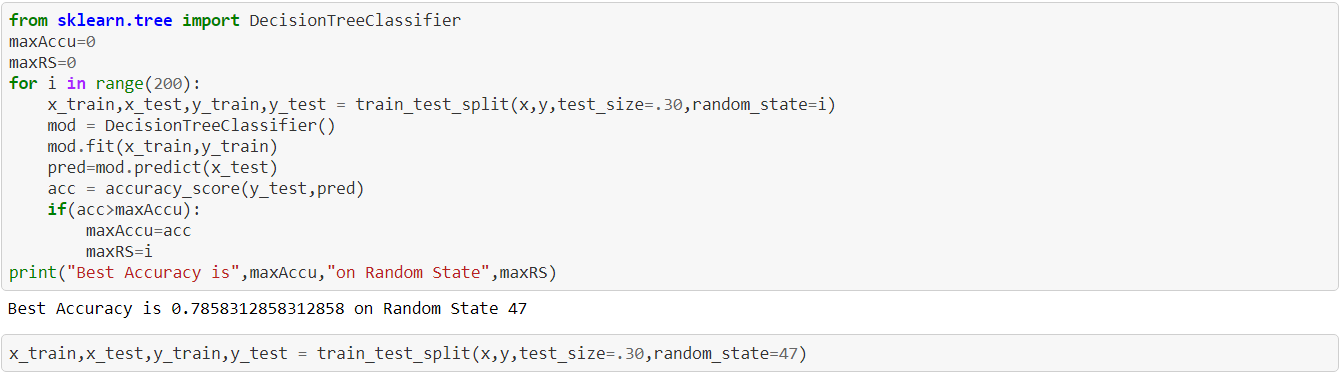
**Building Machine Learning Models:**

We will build Classification models for the prediction of Census Income.

First, we need to split the dataset into training set and test set.



Here, we are using a model to test on which random state the accuracy will be best, so that we can do the train-test slit on that random state. Below we got random state as 47, so we will perform the train-test split at random state 47.



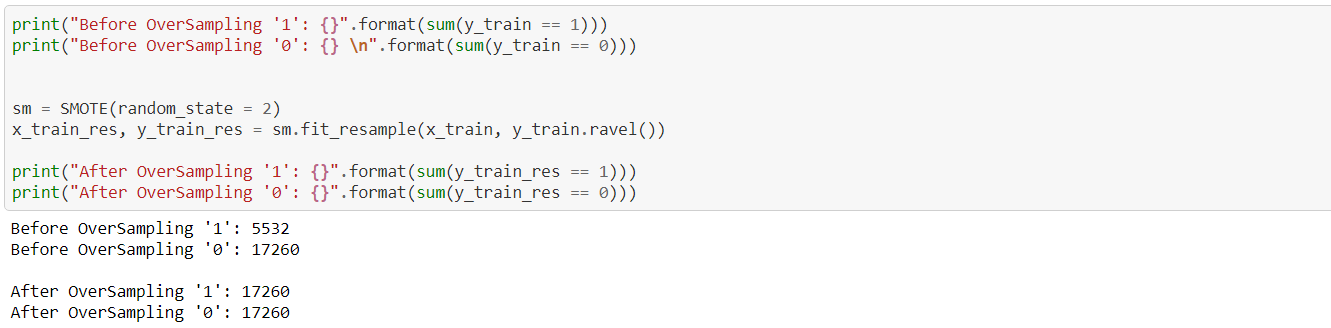
Above, we have kept the 70% of data as training data and we will perform testing on the remaining 30% of data.

So, shall we start building our model? Wait, if we remember correctly, we have observed that our target variable was unbalanced and will affect our model efficiency and make our model biased and till now we have not fixed it. Let’s fix this imbalance now using SMOTE technique:

For this we will need to import ‘SMOTE’ from ‘imblearn’ library as below:

from imblearn.over\_sampling import SMOTE

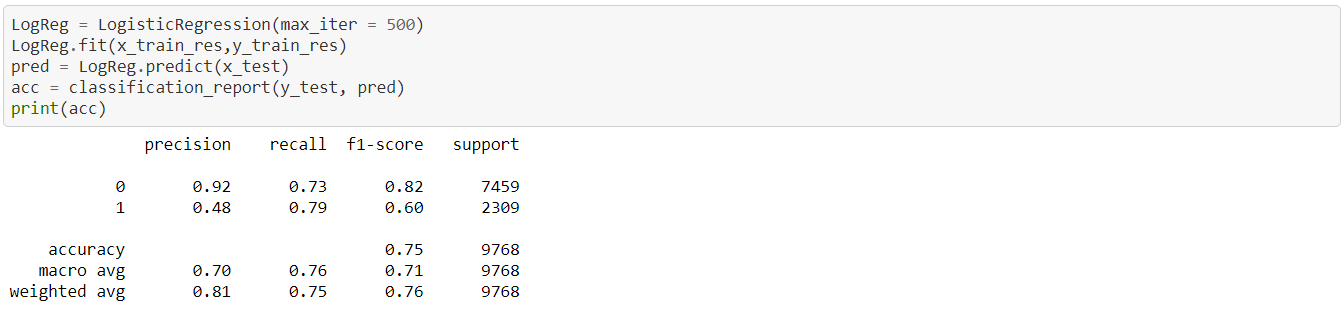
Code:



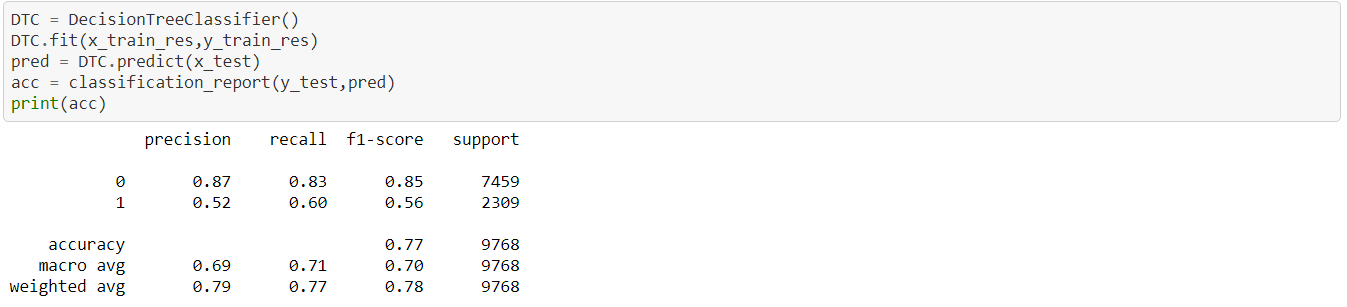
Now that we have overcome the imbalance, let’s finally move into model building phase.

We will build many different models. We will not discuss how the model internally works but we will compare different parameters to conclude which model is predicting the best and then perform hyper tuning on the selected model.

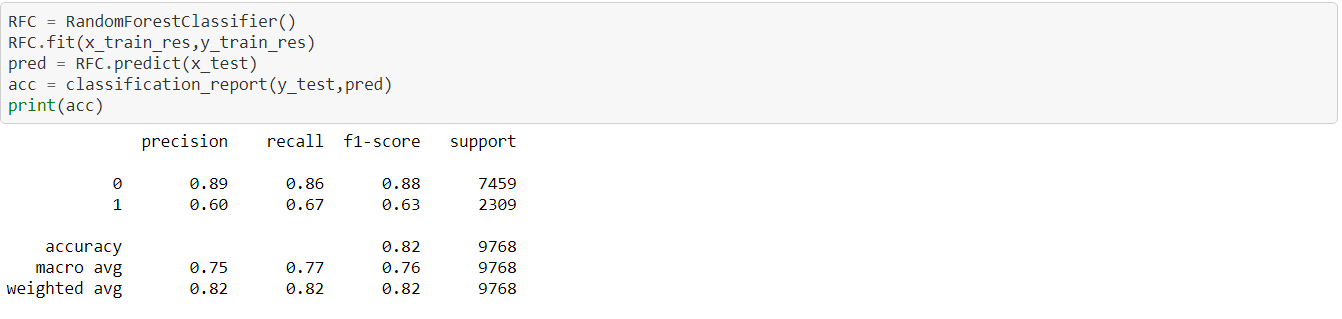
**Logistic regression:**



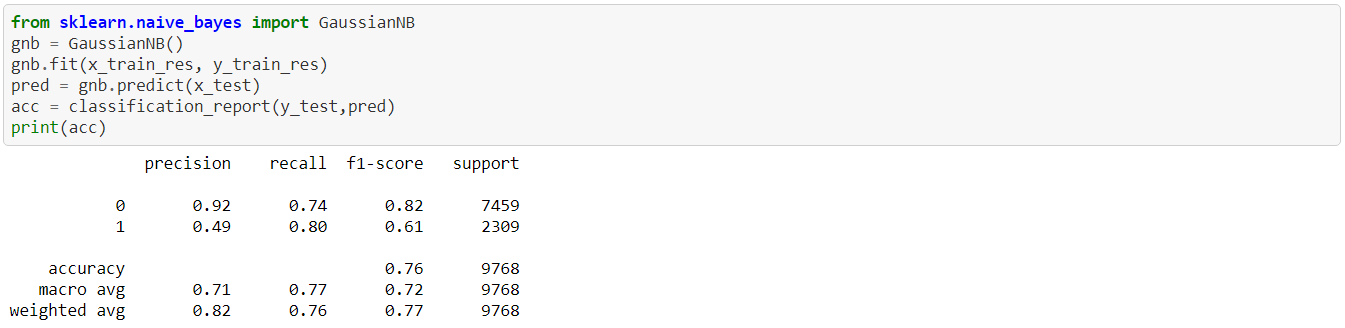
**Decision Tree:**



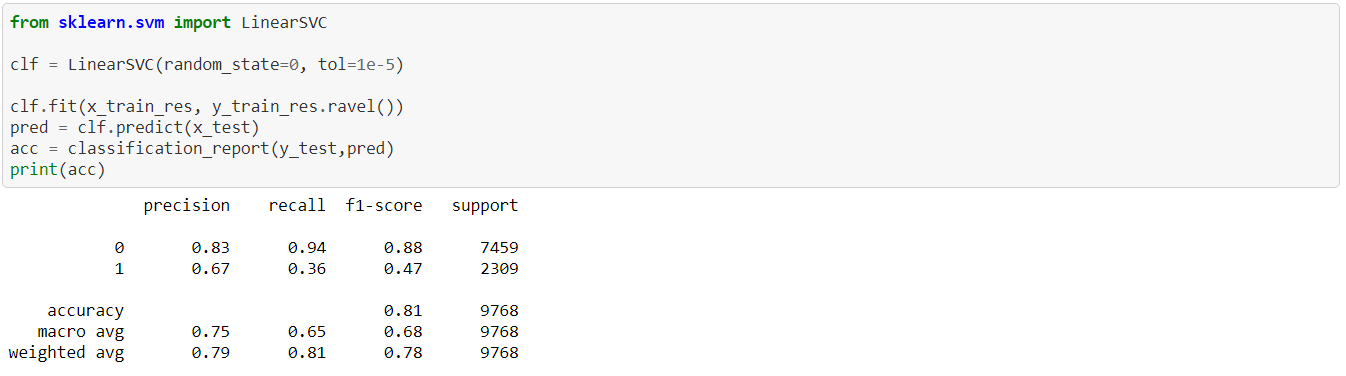
**Random Forest:**



**Naïve Bayes:**



**SVM:**

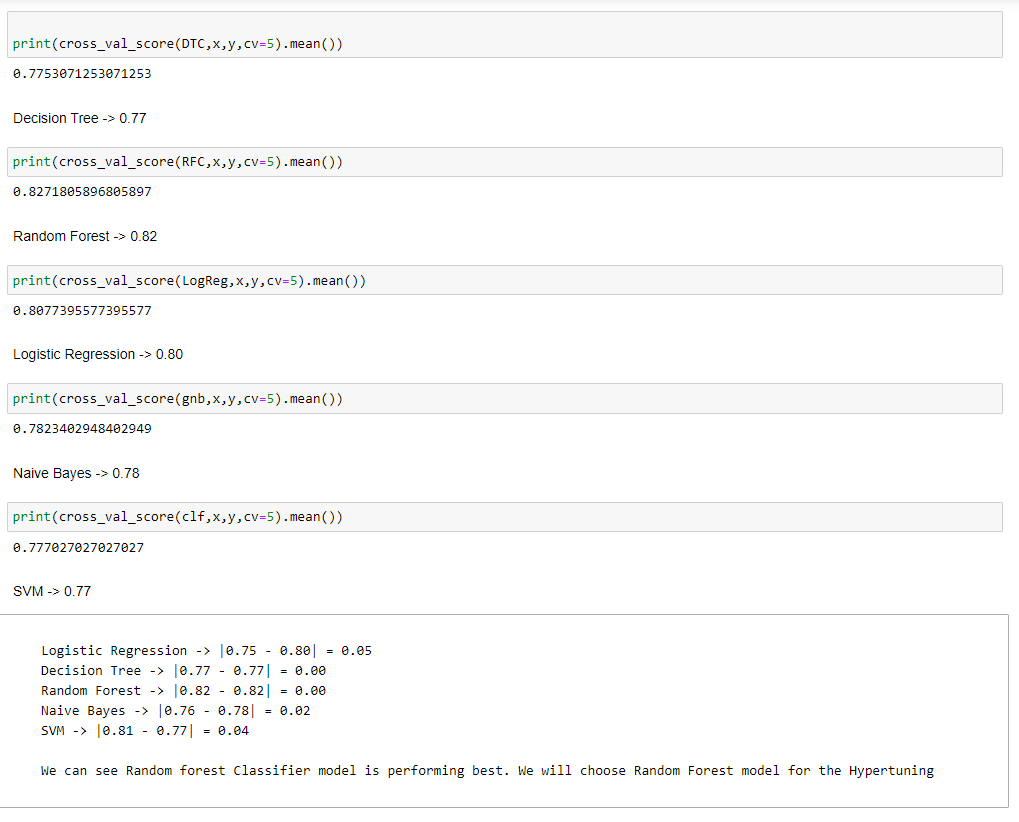


Let’s summarize the data from the above models in a table to get more clear view and compare the different models.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Models** | **Precision (0)** | **Precision (1)** | **Recall (0)** | **Recall (1)** | **F1-score (0)** | **F1-score (1)** | **Accuracy** |
| **Logistic Regression** | 0.92 | 0.48 | 0.73 | 0.79 | 0.82 | 0.60 | 0.75 |
| **Decision Tree** | 0.87 | 0.52 | 0.83 | 0.60 | 0.85 | 0.56 | 0.77 |
| **Random Forest** | 0.89 | 0.60 | 0.86 | 0.67 | 0.88 | 0.63 | 0.82 |
| **Naïve Bayes** | 0.92 | 0.49 | 0.74 | 0.80 | 0.82 | 0.61 | 0.76 |
| **SVM** | 0.83 | 0.67 | 0.94 | 0.36 | 0.88 | 0.47 | 0.81 |

We should keep in mind that we should compare all parameters for both 0 & 1. If we observe the above table carefully, we can see that Random Forest model is performing the best by keeping both 0 & 1 parameters into consideration.

Let’s Cross Validate the score by performing Cross Validation:

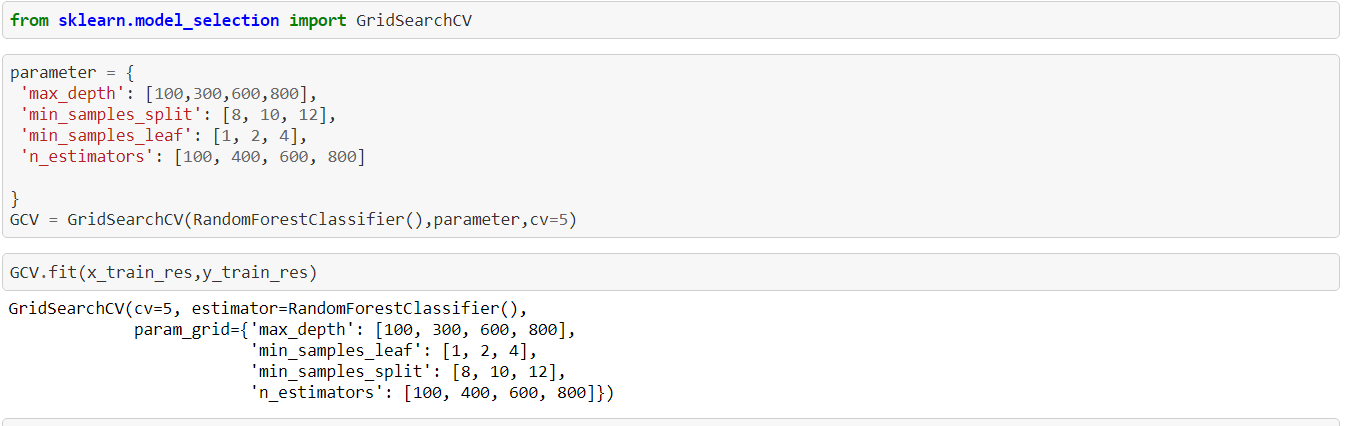


We have calculated the score by subtracting accuracy from the cross-validation score. As we can observe that Random Forest is having 0 difference and have highest accuracy. It is the best performing algorithm for this dataset.

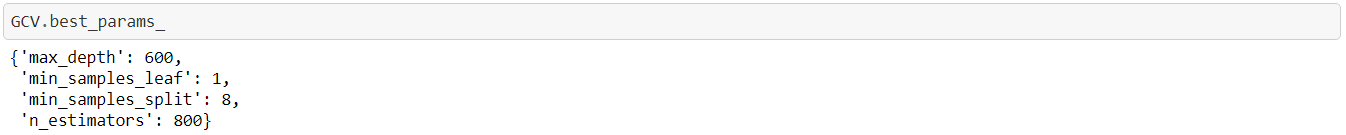
Hence, let’s Hyper tune the Random Forest model further.

We will use ‘GridSearchCV’ from ‘sklearn’ library for the Hyper tuning, It tries different combinations of the parameters that we will pass and check which parameter suits the model best:

Code:

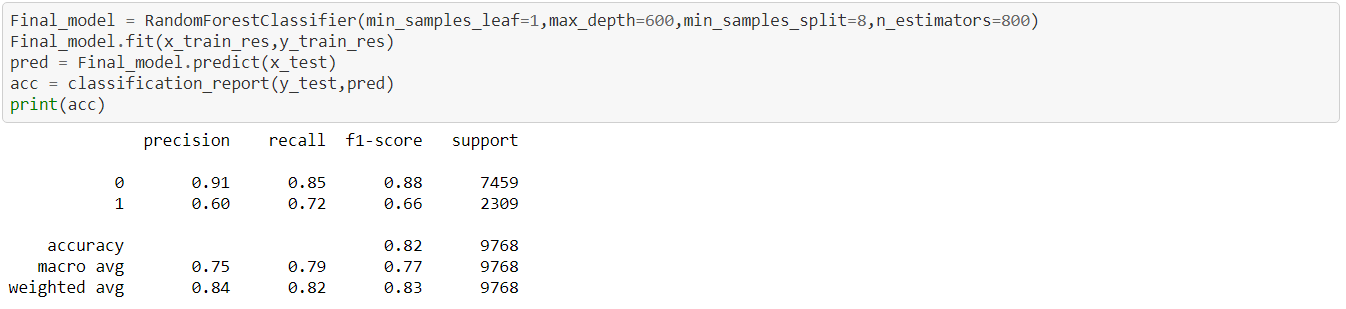


Now let’s check the best parameters for the model:



We have the best parameters now, let’s build our final model i.e., Random Forest model using these parameters:

Code:



Let’s compare the old model values and values of the model after hyper tuning

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Random Forest** | **Precision (0)** | **Precision (1)** | **Recall (0)** | **Recall (1)** | **F1-score (0)** | **F1-score (1)** | **Accuracy** |
| **Before** | 0.89 | 0.60 | 0.86 | 0.67 | 0.88 | 0.63 | 0.82 |
| **After** | 091 | 0.60 | 0.85 | 0.72 | 0.88 | 0.66 | 0.82 |

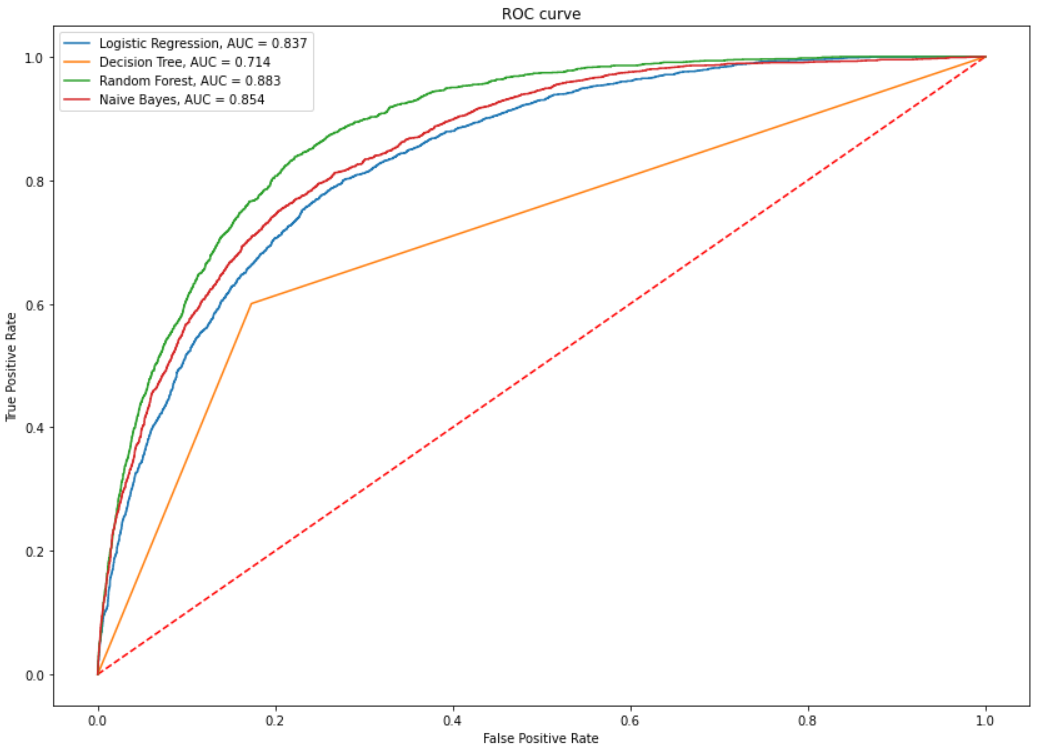
We can observe that precision for 0 has increased as well as, recall for 1 has also increased.

Now, the final check to confirm and conclude that Random Forest model is performing the best is by plotting AUC-ROC curve

Code:



Output:



The AUC-ROC curve takes True-Positive Rate, True-Negative Rate, False-Positive Rate and False-Negative Rate into the consideration

We can see that AUC value for Random Forest is 0.883 which is the largest in our models. Hence, Random Forest model is the best for making predictions on the available data set of Census income.

**Concluding Remarks**

After several iterations of exploring and conditioning on the data, we have built a useful algorithm for predicting the Census Income. The technique applied in this project is an implementation of a simple machine learning model, the Random Forest.

We can conclude that Random Forest algorithm is performing best among all the algorithms.