Machine Learning application — Census Income Prediction

The economic well-being of a Nation is highly driven by the income of the residents.

Countless decisions in private and public sectors are based on Census data. Census data is the backbone of the democratic system of government, highly affecting the economic sectors. Census-related figures are used to distribute the federal funding by the government into different states and localities.

Not only the above, the census data is also used for post census population estimates and projections, economic and social science research, and many other such applications. Hence, the importance of this data and its correct predictions is very clear to us.

Data has always been the backbone of many important decisions. When an assumption is backed up by facts and numbers, the chances of incorrectness and bad decisions decrease.

Problem Statement

The above introduction had an aim to increase the awareness about how the income factor actually has an impact not only on the personal lives of people, but also an impact on the nation and its betterment. We will today have a look on the data extracted from the 1994 Census bureau database, and try to find insights about how different features have an impact on the income of an individual. Though the data is quite old, and the insights drawn cannot be directly used for derivation in the modern world, but it would surely help us to analyze what role different features play in predicting the income of an individual.

The Dataset

The dataset provided to us contains 32560 rows, and 14 different independent features. We aim to predict if a person earns more than 50k$ per year or not. Since the data predicts 2 values (>50K or <=50K), this clearly is a classification problem, and we will train the classification models to predict the desired outputs.

Mentioned below are the details of the features provided to us, which we will be feeding to our classification model to train it.

1. Age — The age of an individual, this ranges from 17 to 90.

2. Workclass — The class of work to which an individual belongs.

3. Fnlwgt — The weight assigned to the combination of features (an estimate of how many people belong to this set of combination)

4. Education — Highest level of education

5. Education_num — Number of years for which education was taken

6. Marital_Status — Represents the category assigned on the basis of marriage status of a person

7. Occupation — Profession of a person

8. Relationship — Relation of the person in his family

9. Race — Origin background of a person

10. Sex — Gender of a person

11. Capital_gain — Capital gained by a person

12. Capital_loss — Loss of capital for a person

13. Hours_per_week — Number of hours for which an individual works per week

14. Native_Country — Country to which a person belongs

Output:

1. Income — The target variable, which predicts if the income is higher or lower than 50K$.

Contents of the article

The following information/steps will be covered further in the article –

1. Exploratory data analysis

2. Data modeling

3. Outlier detection and skewness treatment

4. Encoding the data — Label Encoder

5. Scaling the data — Standard scaler

6. Fitting the machine learning models

7. Cross-validation of the selected model

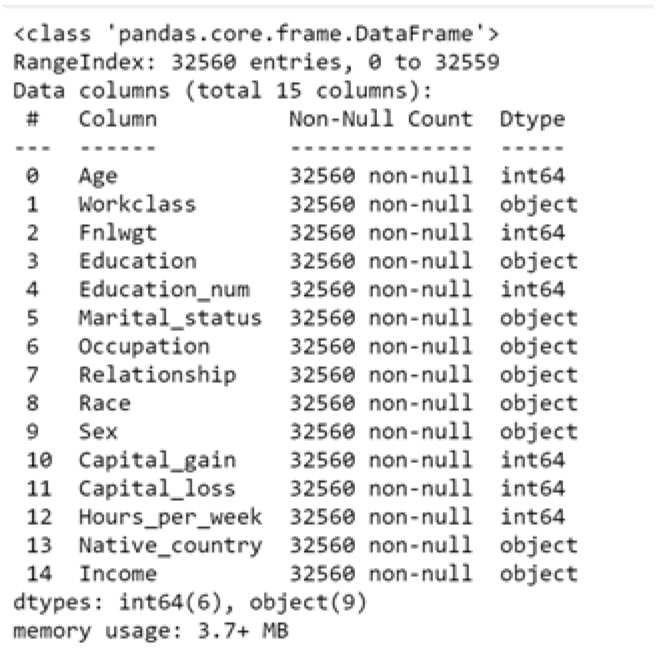
8. Model hypertuning

9. AUC-ROC curve

10. Saving the final model and prediction using saved model

Exploratory Data Analysis

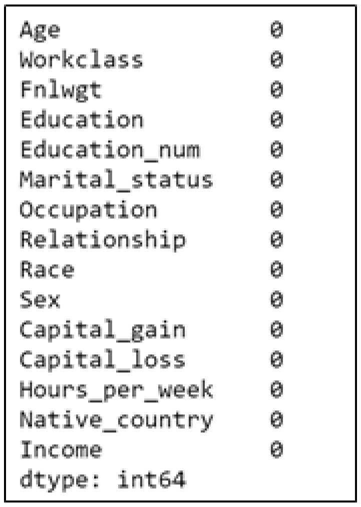
The first step that we do is to check the information about our data. We see the results shown in the image below:



```
<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
```

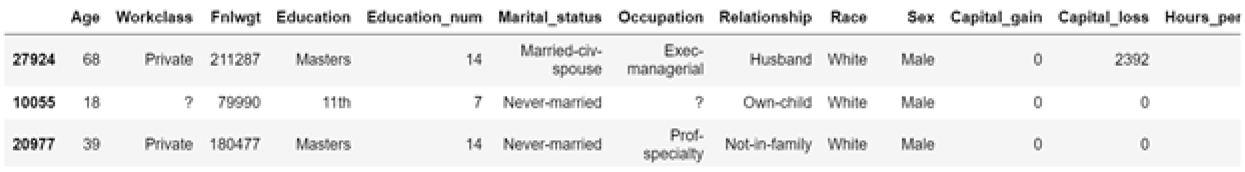
We see that we have a mixture of categorical and numeric columns. We have 6 integer columns and 9 object type columns. We observe that the count of entries is 32560 for all columns, hence no NaN values are present in our dataset.

We confirm this assumption using data.isnull().sum() command –



```
Age               0
Workclass         0
Fnlwgt            0
Education         0
Education_num     0
Marital_status    0
Occupation        0
Relationship      0
Race              0
Sex               0
Capital_gain      0
Capital_loss      0
Hours_per_week    0
Native_country    0
Income            0
dtype: int64
```

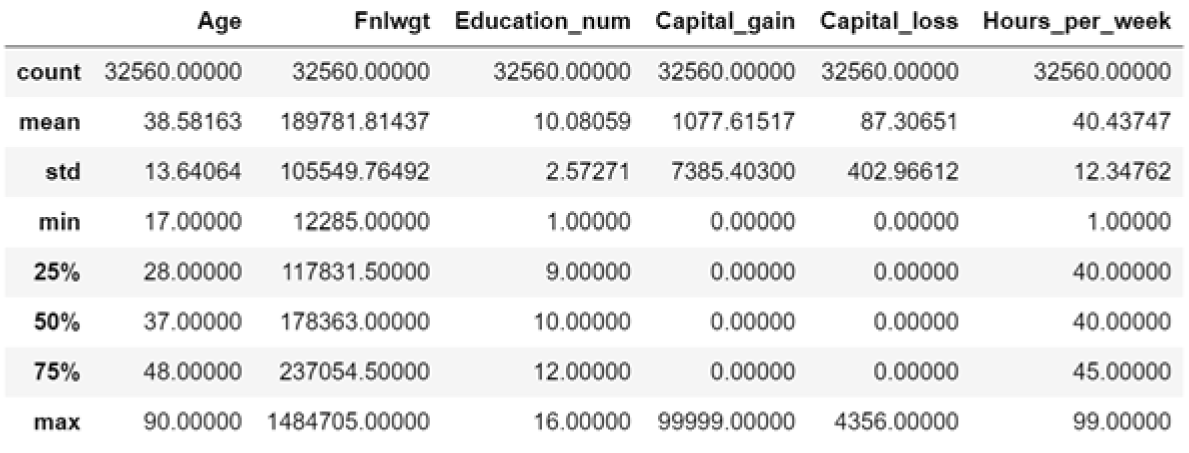
But, while having a close look at our dataset, we observe some of the values as '?', which represent missing values.



	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	Hours_per
27924	68	Private	211287	Masters	14	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	2392	
10055	18	?	79990	11th	7	Never-married	?	Own-child	White	Male	0	0	
20977	39	Private	180477	Masters	14	Never-married	Prof-specialty	Not-in-family	White	Male	0	0	

Hence, we deduce that there are some values in our data set which need to be treated.

We also check the numerical statistics of our data using data.describe() command –



	Age	Fnlwgt	Education_num	Capital_gain	Capital_loss	Hours_per_week
count	32560.00000	32560.00000	32560.00000	32560.00000	32560.00000	32560.00000
mean	38.58163	189781.81437	10.08059	1077.61517	87.30651	40.43747
std	13.64064	105549.76492	2.57271	7385.40300	402.96612	12.34762
min	17.00000	12285.00000	1.00000	0.00000	0.00000	1.00000
25%	28.00000	117831.50000	9.00000	0.00000	0.00000	40.00000
50%	37.00000	178363.00000	10.00000	0.00000	0.00000	40.00000
75%	48.00000	237054.50000	12.00000	0.00000	0.00000	45.00000
max	90.00000	1484705.00000	16.00000	99999.00000	4356.00000	99.00000

Following observations are made in this step –

- The age column has a range of 17 to 90.

- The fnlwgt column has a minimum value of 12285 and maximum value of 1484705

- The education number has a range of 1 to 16

- The capital gain starts from 0 and ends at 99999

- The capital loss starts at 0 and ends at 4356

- Hours per week range between 1–99.

- There are outliers expected in Capital gain column as the values till 75% are 0. Same is the case with capital loss as well.

- The fnlwgt column also has a huge difference between 75% values and the max value. There is a chance of getting outliers here.

Further, we have a look at our dataset and explore the data in various columns, one by one –

· Income column

The income column is our target variable with 2 values — '<=50K' and '>50K'. The count of these values is 24719 and 7841 respectively, suggesting that people with income higher than 50K are significantly less, and our data set is imbalanced considering the target variable.

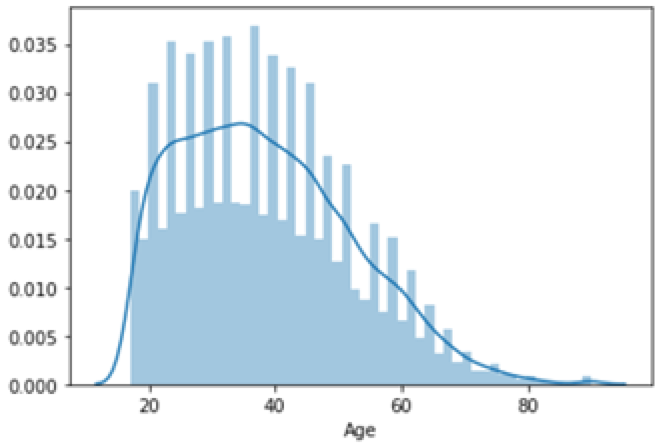
· Age column

The data in age column has a minimum value 17 and max value 90.

We create a distribution plot for the age column –

```
sns.distplot(data.Age)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d41a2775b0>
```



We observe that our data is has right skewness, with majority of the ages falling in the 20–50. The count keeps on decreasing as the age increases.

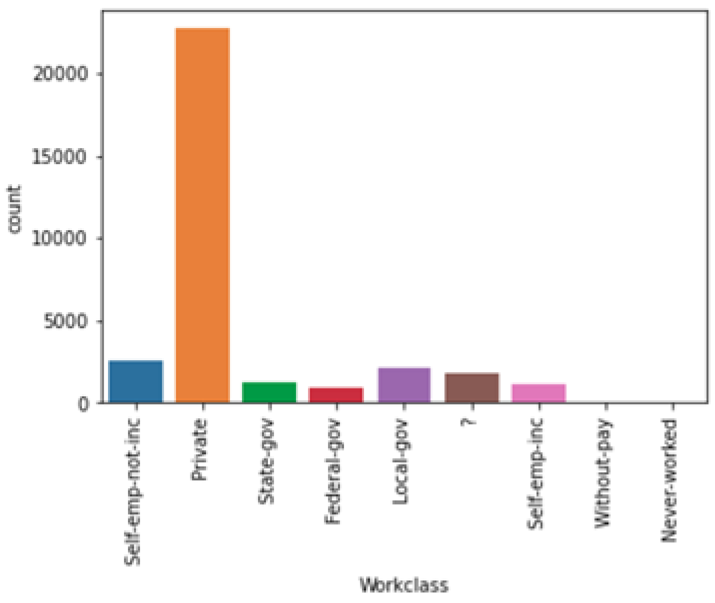
We also observe that we do not have any null values in the age column.

· Workclass column

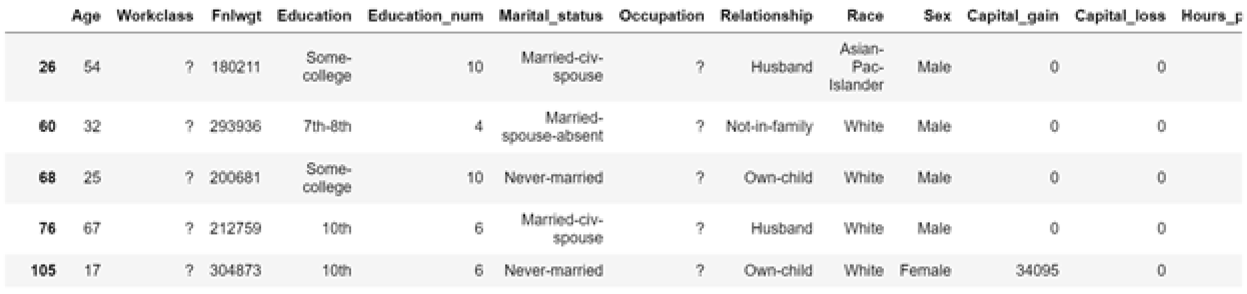
While checking the unique values for workclass, we see that we have 7 different types of values, along with some missing values represented by '?'. The count of null values is 1836, which is around 5% of the data.

```
data.Workclass.value_counts()
```

```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: Workclass, dtype: int64
```



We observe that majority of the people belong to ‘Private’ sector workclass.

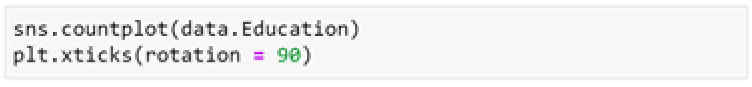


	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	Hours_p
26	54	?	180211	Some-college	10	Married-civ-spouse	?	Husband	Asian-Pac-Islander	Male	0	0	
60	32	?	293936	7th-8th	4	Married-spouse-absent	?	Not-in-family	White	Male	0	0	
68	25	?	200681	Some-college	10	Never-married	?	Own-child	White	Male	0	0	
76	67	?	212759	10th	6	Married-civ-spouse	?	Husband	White	Male	0	0	
105	17	?	304873	10th	6	Never-married	?	Own-child	White	Female	34095	0	

We also make an interesting discovery here — The values where ‘Workclass’ is missing, also has ‘Occupation’ missing!

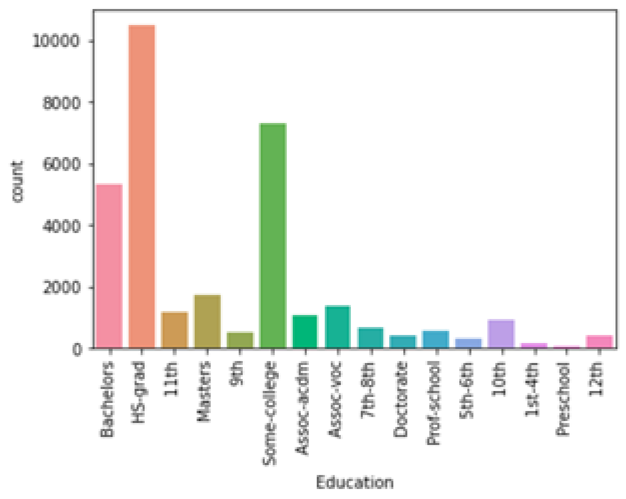
· Education column

The ‘Education’ column has 16 different categories available. Majority of these categories belong to ‘School’ type (different classes are divided into multiple categories)



```
sns.countplot(data.Education)
plt.xticks(rotation = 90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
13, 14, 15]),
 <a list of 16 Text major ticklabel objects>)
```



We observe no missing values in this column, and also find out that majority of the people have education level as ‘HS-grad’, followed by ‘Some-college’ and ‘Bachelors’.

· Education-num column

The education number is the number of years for which a person received education. This is an ordinal column, which contains 16 different values. When we check the division of ‘Education_num’ column, we observe that the count of ‘Education’ column and ‘Education_num’ is exactly same! Which means, the ‘Education_num’ column is providing same information as ‘Education’ column, but in a numeric manner!

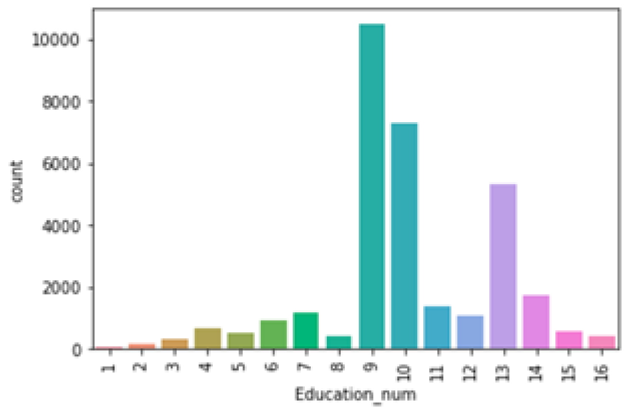
We can map the ‘Education’ level and ‘Education_num’ columns by keeping the information side-by-side as is displayed below:

```
9     10501                                    HS-grad         10501
10     7291                                    Some-college     7291
13     5354                                    Bachelors        5354
14     1723                                    Masters          1723
11     1382                                    Assoc-voc        1382
7      1175                                    11th             1175
12     1067                                    Assoc-acdm       1067
6       933                                    10th              933
4       646                                    7th-8th           646
15      576                                    Prof-school       576
5       514                                    9th               514
8       433                                    12th              433
16      413                                    Doctorate         413
3       333                                    5th-6th           333
2       168                                    1st-4th           168
1        51                                    Preschool          51
Name: Education_num, dtype: int64             Name: Education, dtype: int64
```

The count chart for 'Education_num' is as follows –

```
sns.countplot(data.Education_num)
plt.xticks(rotation = 90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
13, 14, 15]),
 <a list of 16 Text major ticklabel objects>)
```

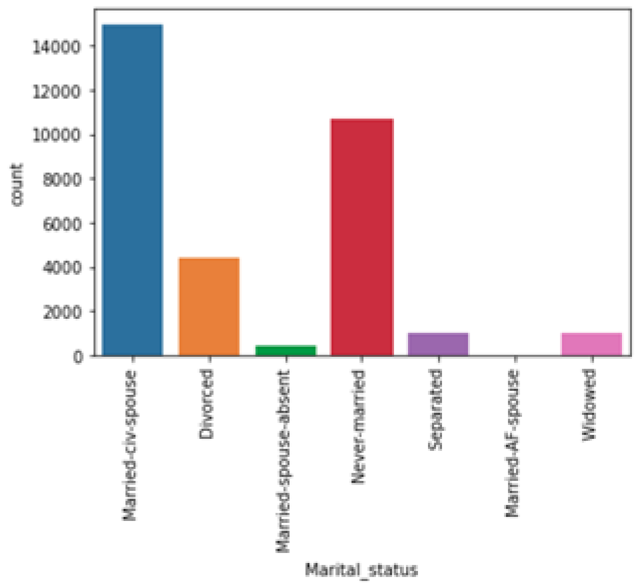


· Marital_Status column

The 'Marital_Status' column has 7 different categories available, and has no missing values.

```
sns.countplot(data.Marital_status)
plt.xticks(rotation = 90)
```

```
(array([0, 1, 2, 3, 4, 5, 6]), <a list of 7 Text major tickl
abel objects>)
```

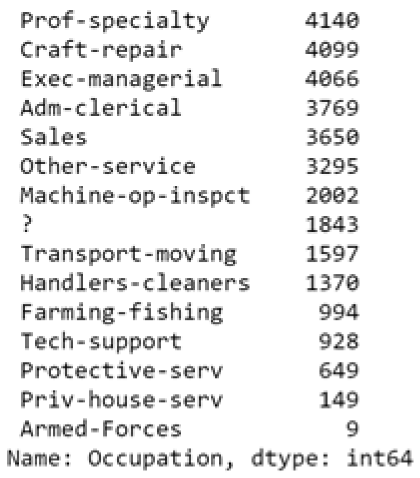


Majority of the people have ‘Marital_Status’ as ‘Married-civ-spouse’, and least have ‘Married-AF-spouse’.

Count of ‘Never-married’ is also quite high.

· Occupation column

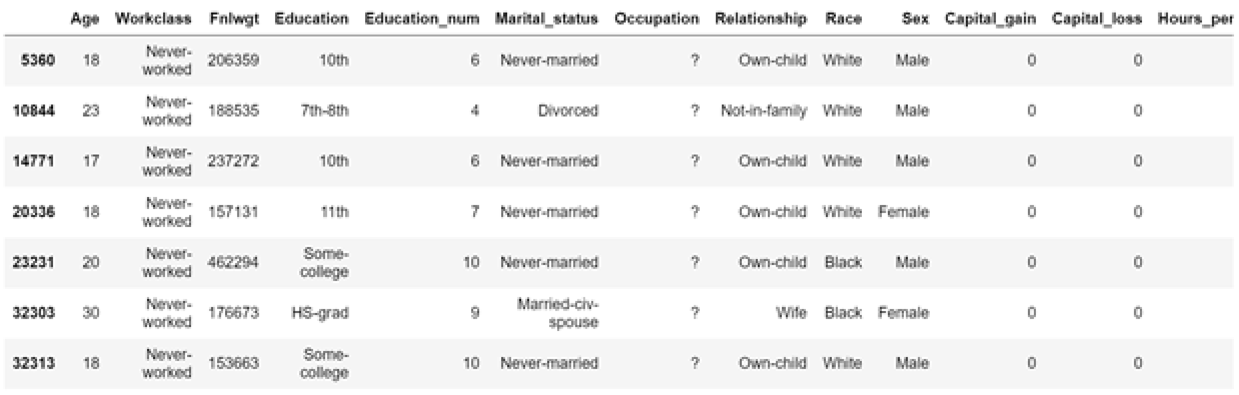
The occupation column contains 14 different categories, and have missing values represented by ‘?’ (which we have already observed, and combined with ‘Workclass’ column).



```
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: Occupation, dtype: int64
```

But the count of missing values is slightly higher than ‘Workclass’ column — 1843.

We try to find out the extra rows where the ‘Occupation’ is missing –

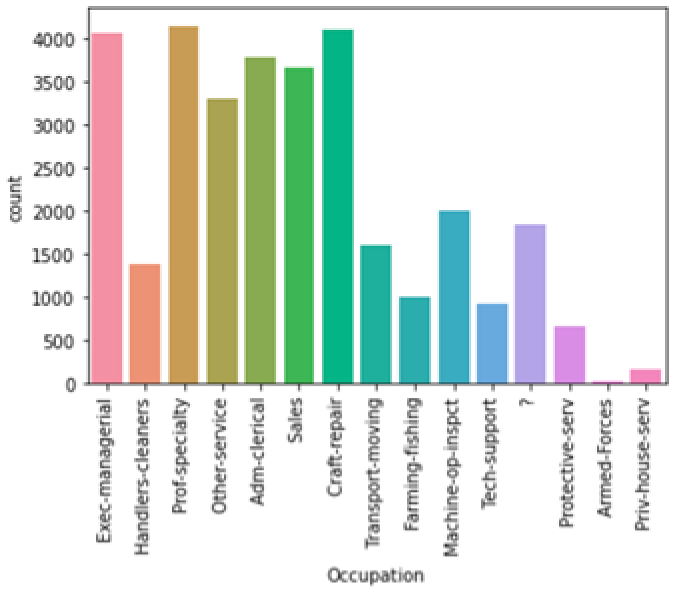


	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	Hours_per
5360	18	Never-worked	206359	10th	6	Never-married	?	Own-child	White	Male	0	0	
10844	23	Never-worked	188535	7th-8th	4	Divorced	?	Not-in-family	White	Male	0	0	
14771	17	Never-worked	237272	10th	6	Never-married	?	Own-child	White	Male	0	0	
20336	18	Never-worked	157131	11th	7	Never-married	?	Own-child	White	Female	0	0	
23231	20	Never-worked	462294	Some-college	10	Never-married	?	Own-child	Black	Male	0	0	
32303	30	Never-worked	176673	HS-grad	9	Married-civ-spouse	?	Wife	Black	Female	0	0	
32313	18	Never-worked	153663	Some-college	10	Never-married	?	Own-child	White	Male	0	0	

We observe that in case the ‘Workclass’ is ‘Never-worked’, then also the ‘Occupation’ is missing.

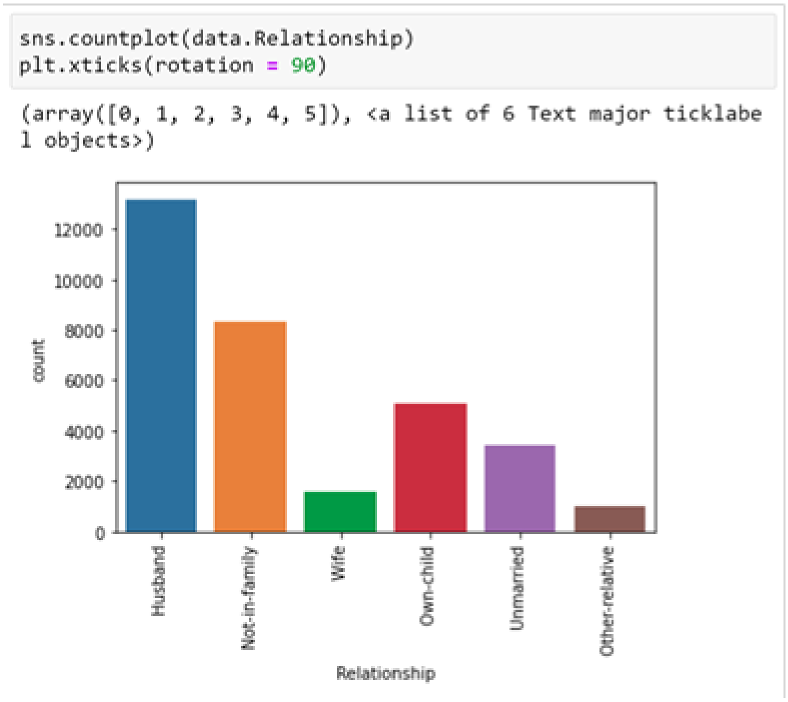
```
sns.countplot(data.Occupation)
plt.xticks(rotation = 90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
13, 14]),
 <a list of 15 Text major ticklabel objects>)
```



· Relationship column

The relationship column contains 6 different types of values, with highest number set for ‘Husband’ and lowest for ‘Other-relative’. The column does not have any missing value.



```
sns.countplot(data.Relationship)
plt.xticks(rotation = 90)
```

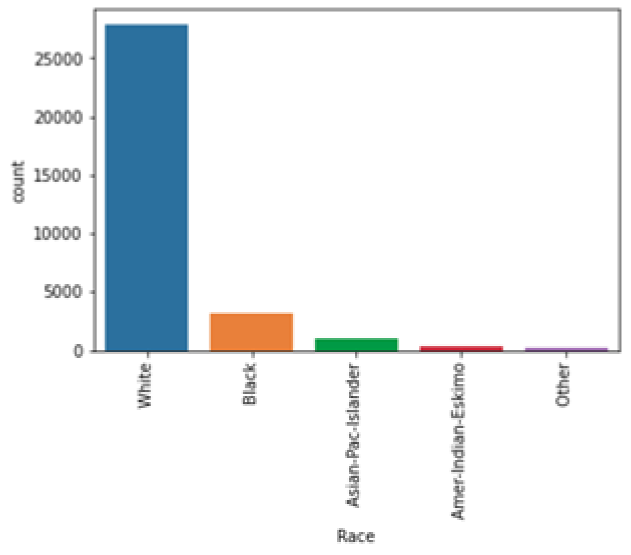
```
(array([0, 1, 2, 3, 4, 5]), <a list of 6 Text major ticklabe
l objects>)
```

· Race column

The Race column has 5 different categories, and no missing data. Highest number of people have race as ‘White’ (significantly high numbers).

```
sns.countplot(data.Race)
plt.xticks(rotation = 90)
```

```
(array([0, 1, 2, 3, 4]), <a list of 5 Text major ticklabel o
bjects>)
```

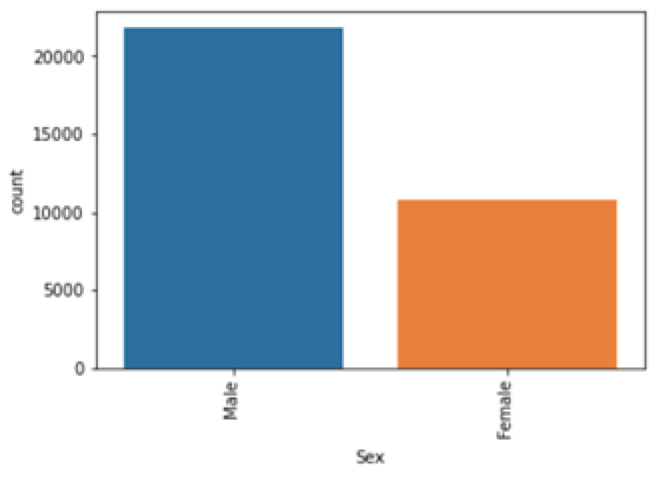


· Sex column

The ‘Sex’ column has 2 categories — Male and Female, where number of males are almost double to number of females. Missing values are not found in this column.

```
sns.countplot(data.Sex)
plt.xticks(rotation = 90)
```

```
(array([0, 1]), <a list of 2 Text major ticklabel objects>)
```

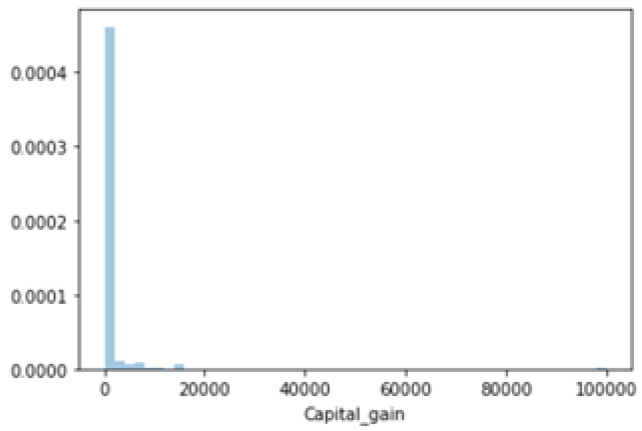


· Capital gain column

‘Capital_gain’ column is a numeric column, with majority of the values set as 0. The distribution plot for ‘Capital_gain’ column is highly right skewed.

```
sns.distplot(data.Capital_gain)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d41b24b790>
```

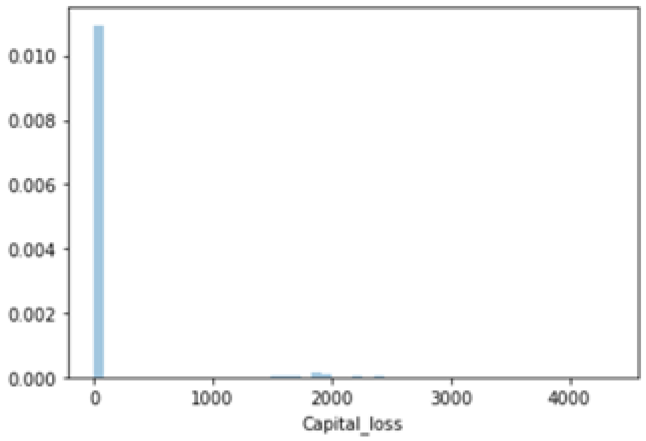


· Capital Loss column

The ‘Capital_loss’ column also has majority of the values set as 0, similar to ‘Capital_gains’. The data is highly right skewed in this case as well.

```
sns.distplot(data.Capital_loss)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d41b3301c0>
```

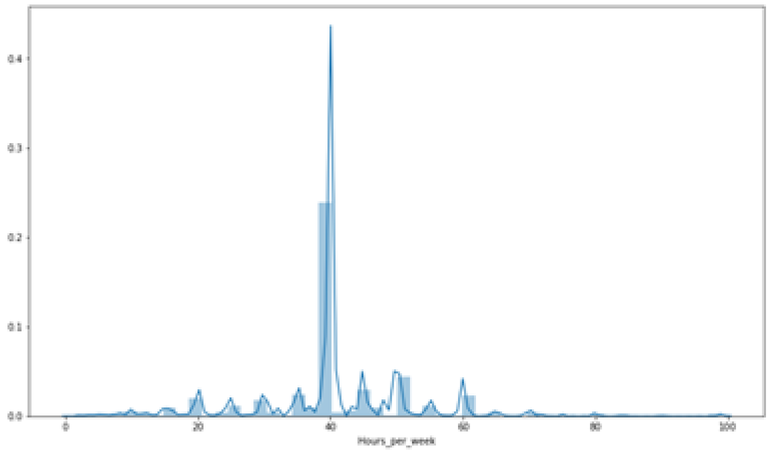


· Hours per week column

The hours per week column has values scattered over a range of 1–99. The column does not have any missing values. Majority of the values have data near 40 hours and hence a high peak can be observed in the below distribution plot –

```
plt.figure(figsize=(14,8))
sns.distplot(data.Hours_per_week)
```

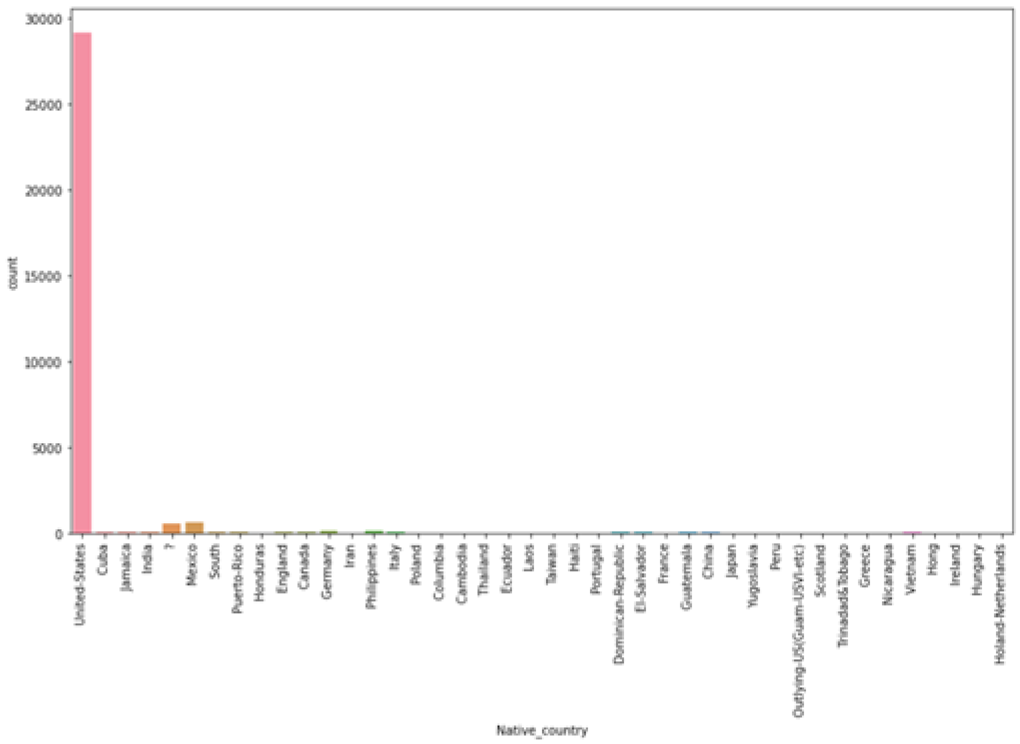
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d41b315fa0>
```



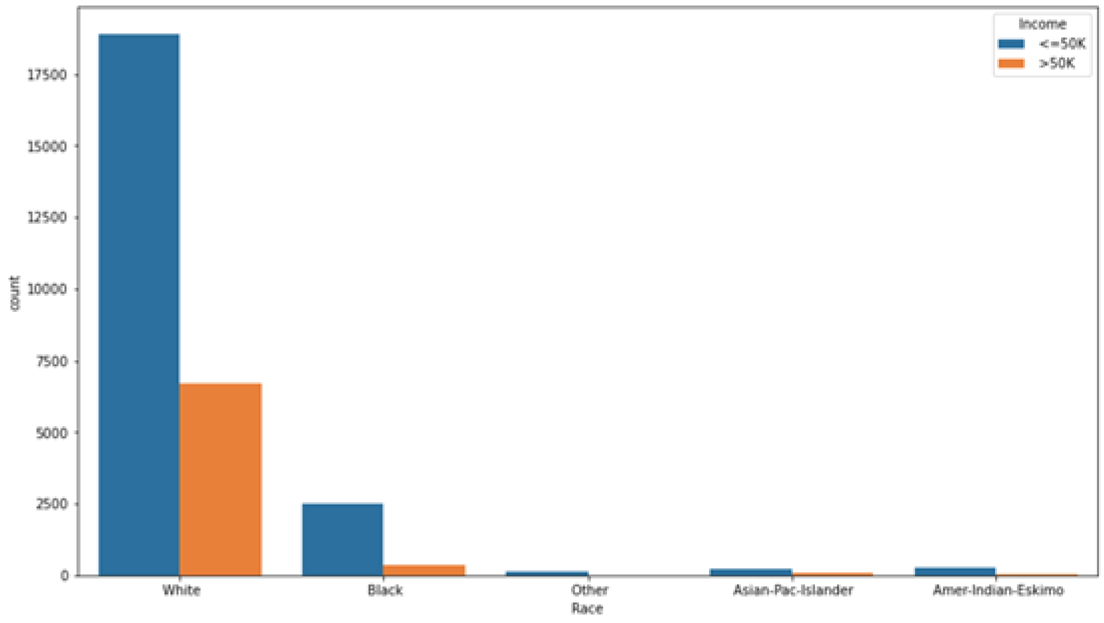
· Native_country column

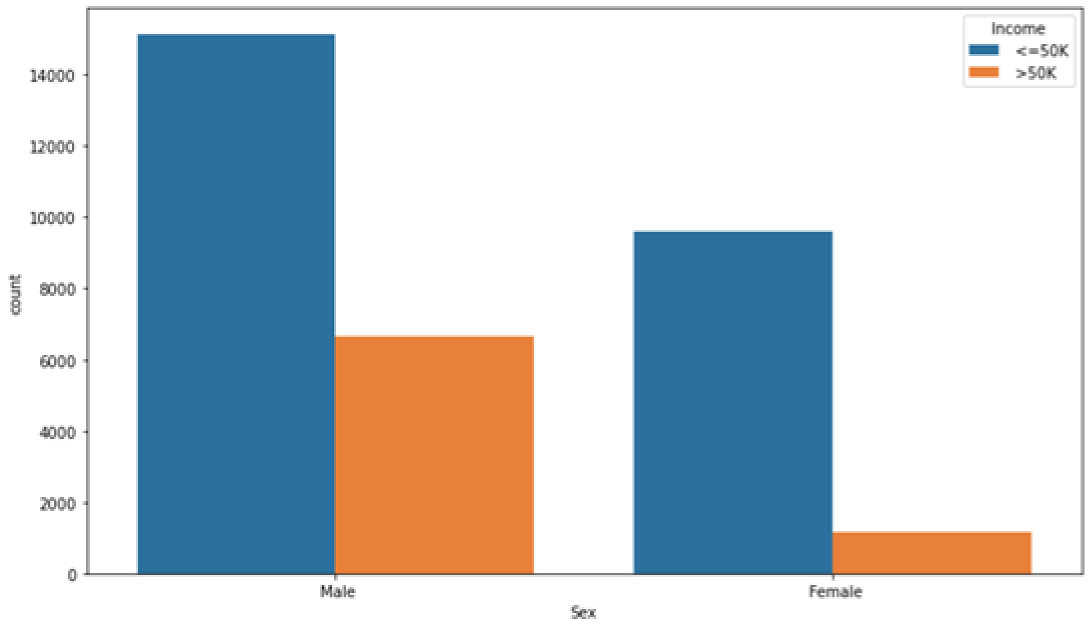
The Native_country column contains the highest count set to ‘United-States’, and rest of the rows contain quite few numbers (highest count after US is 643).

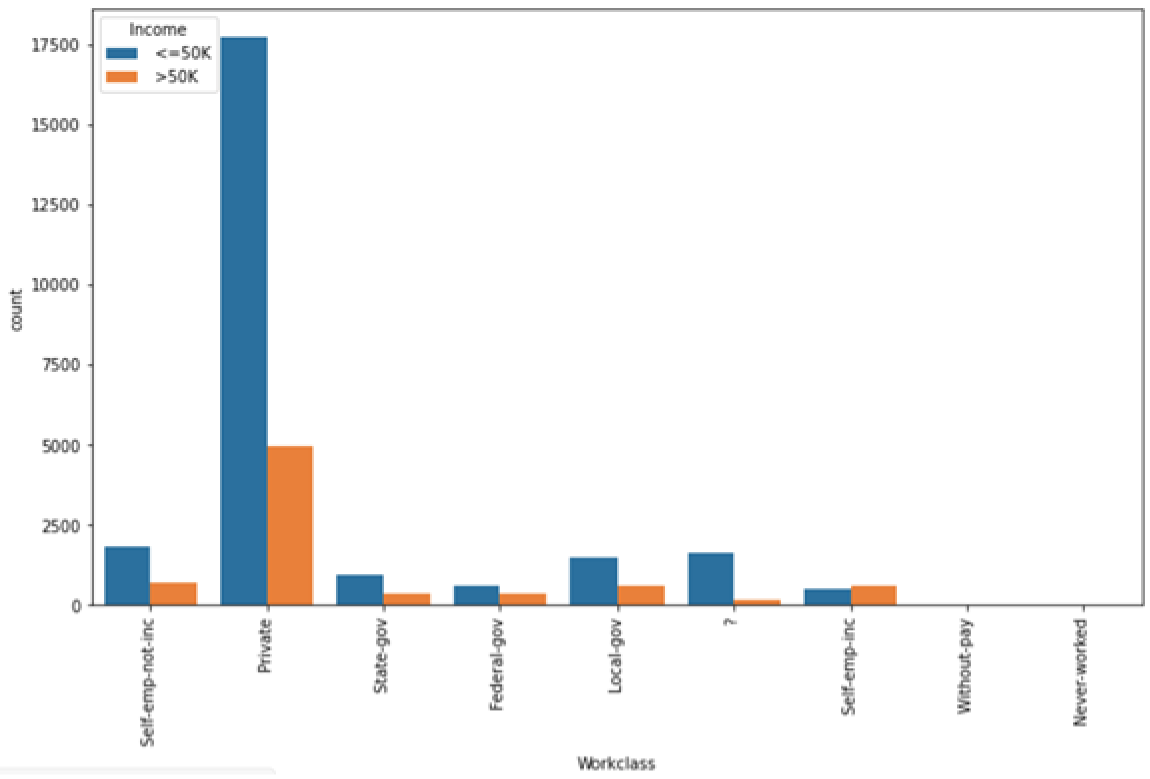
We also have 583 missing values in this column, which we need to treat.

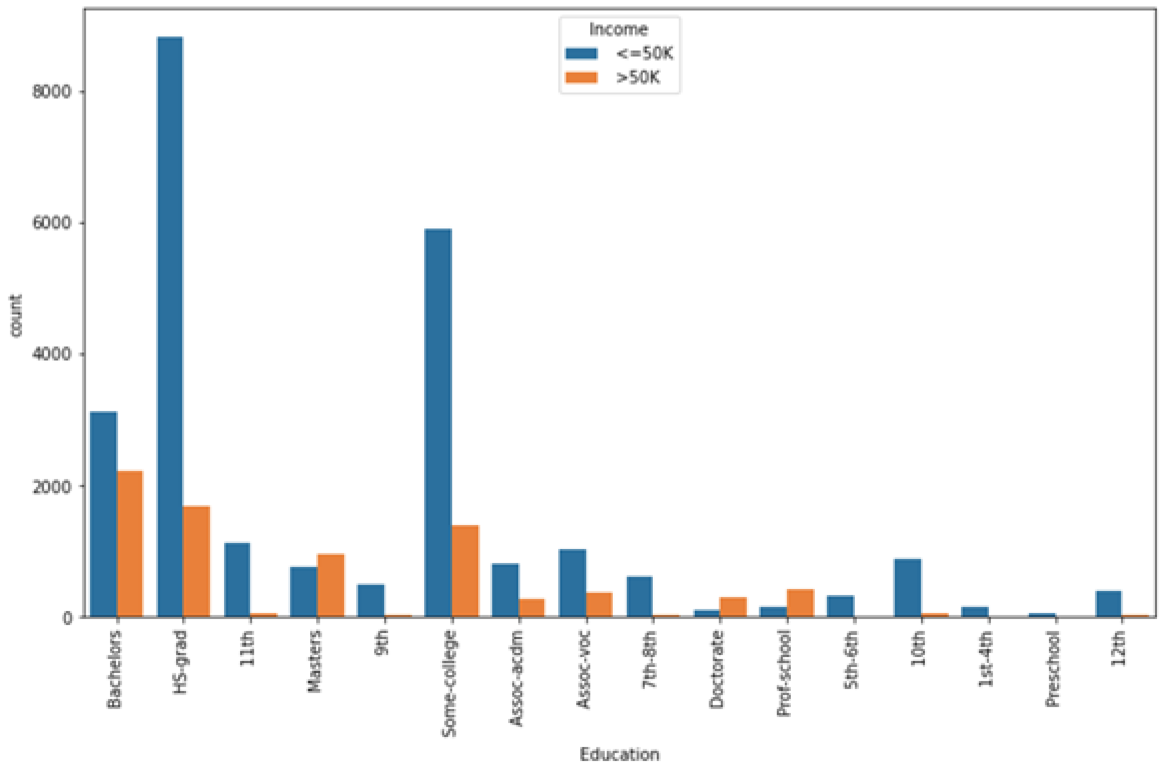


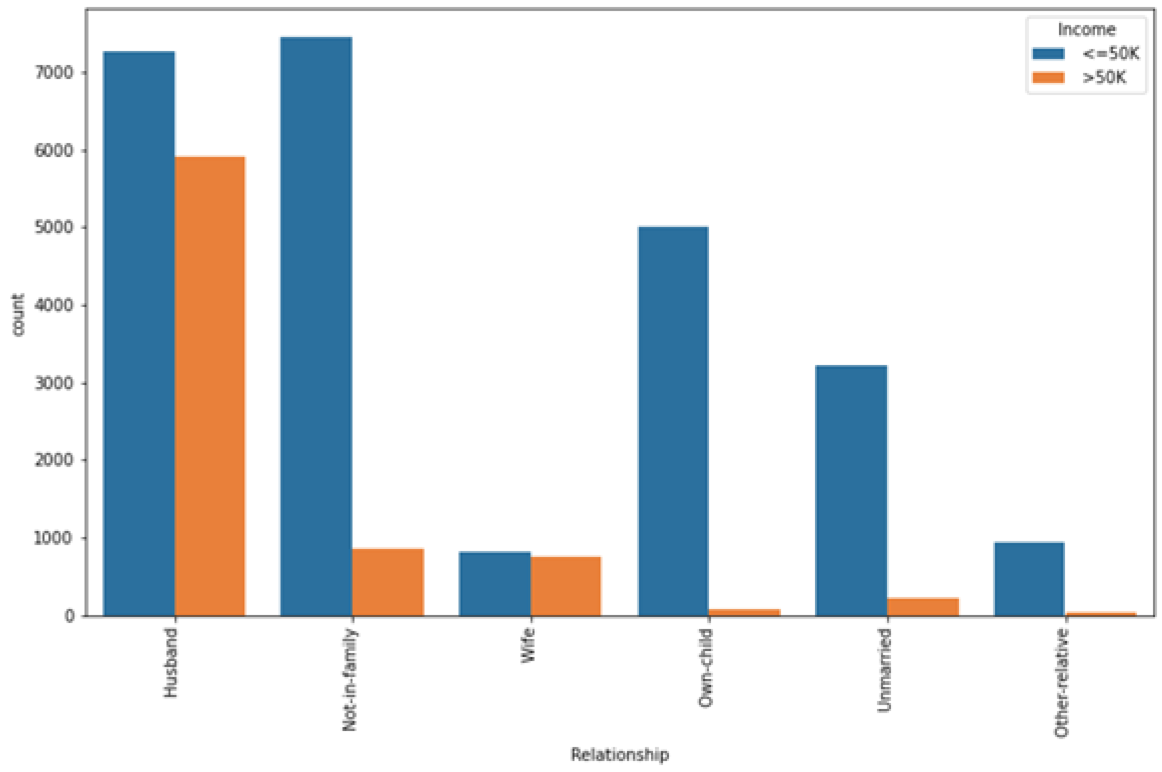
We further check how the income gets impacted due to the features which we just explored.

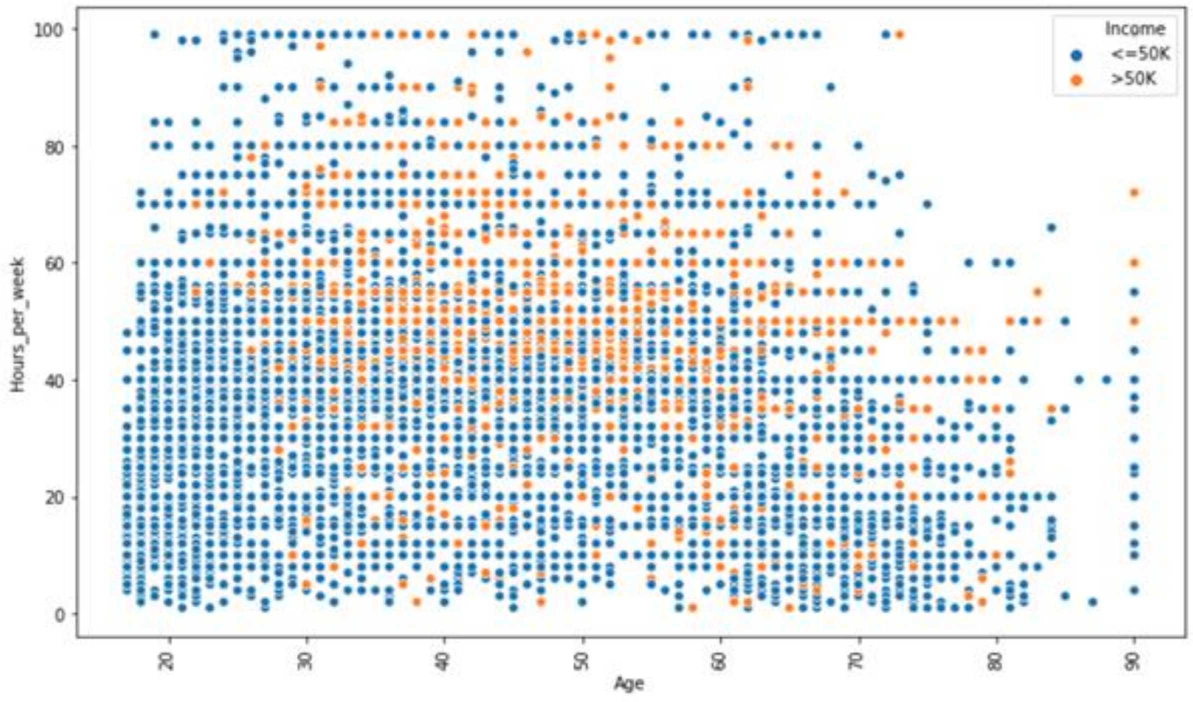

Income
<=50K
>50K
count
17500
15000
12500
10000
7500
5000
2500
0
White
Black
Other
Asian-Pac-Islander
Amer-Indian-Eskimo
Race


Income
<=50K
>50K
count
14000
12000
10000
8000
6000
4000
2000
0
Male
Female
Sex


Income
<=50K
>50K
count
17500
15000
12500
10000
7500
5000
2500
0
Self-emp-not-inc
Private
State-gov
Federal-gov
Local-gov
?
Self-emp-inc
Without-pay
Never-worked
Workclass


Income
<=50K
>50K
count
8000
6000
4000
2000
0
Bachelors
HS-grad
11th
Masters
9th
Some-college
Assoc-acdm
Assoc-voc
7th-8th
Doctorate
Prof-school
5th-6th
10th
1st-4th
Preschool
12th
Education





Looking at the graphs above, we make the following conclusions –

1. A person has high chance of earning >50K in case his/her race is ‘White’/’Asian-pac-islander’.

2. Males have a higher chance of earning more than 50K, than females.

3. Ratio of people earning more than 50K is higher in case Workclass is ‘Self-emp-inc’.

4. People with education level as ‘Masters/Doctorate/Prof-school’ have higher ratios of >50K earning, than <=50K. Bachelors degree also has around 10:7 ratio of <=50K : >50K.

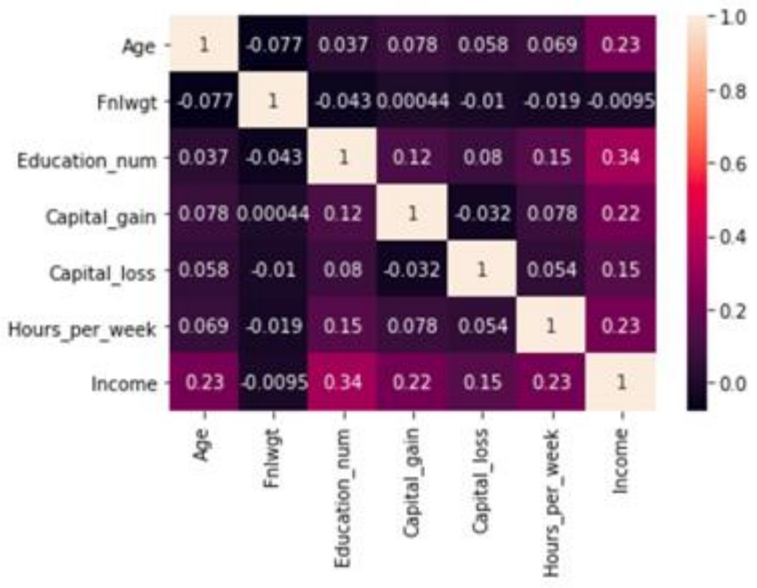
5. If the relationship in family is either ‘Husband/Wife’, the chances of earning more than 50K is high.

6. From the scatterplot between age, hours_per_week and income, we observe that a person needs to be >30 to be earning more than 50K, else needs to work at least 60 hours_per_week to earn >50K.

If we check the correlation between the numeric columns, we observe that –

```
sns.heatmap(data.corr(), annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d41a68a280>
```



	Age	Fnlwgt	Education_num	Capital_gain	Capital_loss	Hours_per_week	Income
Age	1	-0.077	0.037	0.078	0.058	0.069	0.23
Fnlwgt	-0.077	1	-0.043	0.00044	-0.01	-0.019	-0.0095
Education_num	0.037	-0.043	1	0.12	0.08	0.15	0.34
Capital_gain	0.078	0.00044	0.12	1	-0.032	0.078	0.22
Capital_loss	0.058	-0.01	0.08	-0.032	1	0.054	0.15
Hours_per_week	0.069	-0.019	0.15	0.078	0.054	1	0.23
Income	0.23	-0.0095	0.34	0.22	0.15	0.23	1

Income has 34% correlation with ‘Education_num’, 23% correlation with ‘hours_per_week’ and ‘age’, and 22% correlation with ‘Capital_gain’. The correlations are moderate.

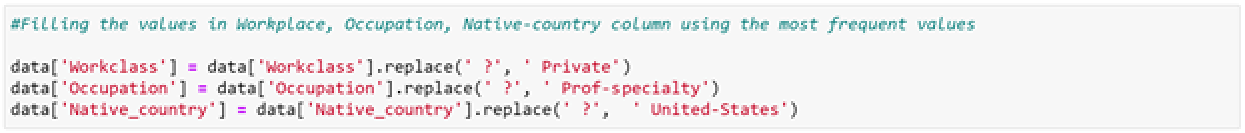
Data modeling

We now proceed to an important part of our process — data modeling. Based on our analysis above, we will fill the missing values in our data, and group certain categories logically, to allow our model to learn better.

Replacing missing values –

We choose to replace the missing values with the mode of the data, i.e. the most frequently occurring values.

Hence, we replace ‘?’ is ‘Workclass’ column by ‘Private’, ‘Occupation’ column by ‘Prof-speciality’ and ‘Native_country’ by ‘United_States’.

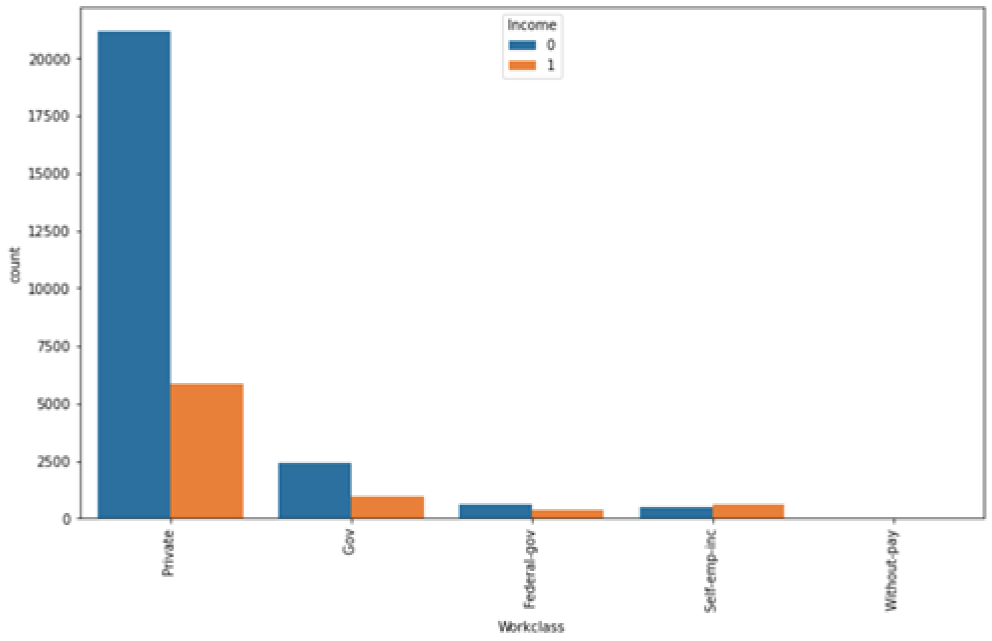


```
#Filling the values in Workplace, Occupation, Native-country column using the most frequent values

data['Workclass'] = data['Workclass'].replace(' ?', ' Private')
data['Occupation'] = data['Occupation'].replace(' ?', ' Prof-specialty')
data['Native_country'] = data['Native_country'].replace(' ?',  ' United-States')
```

Combining the data logically to reduce categories -

Further, we combine the data in various columns –

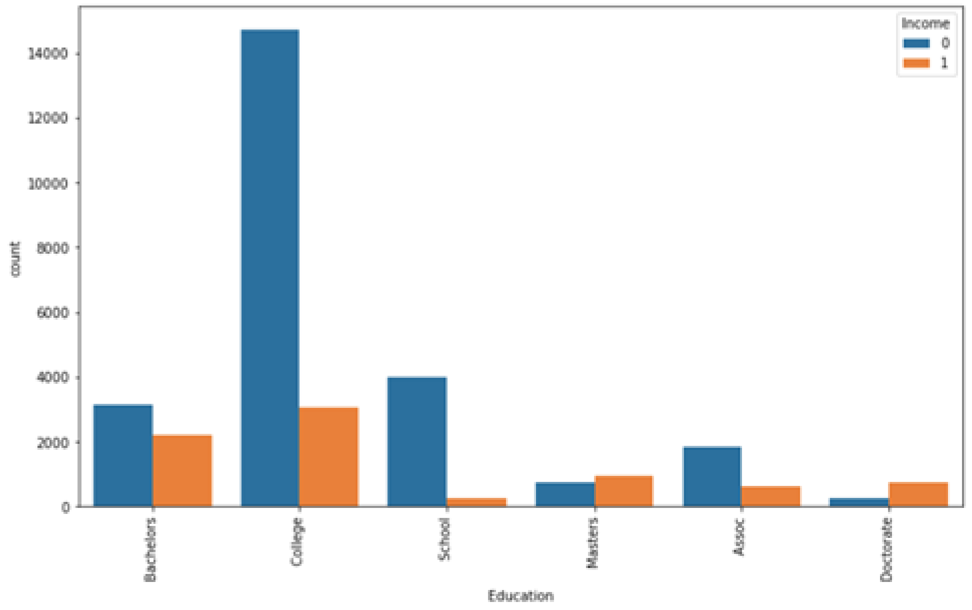
· Workclass –



We put ‘Never-worked’ and ‘Without-pay’ in one category, we classify ‘State-gov’ and ‘Local-gov’ as ‘Gov’, and we add ‘Self-emp-not-inc’ in ‘Private’ category, since the distributions are similar.

We obtain 5 different categories in Workclass column

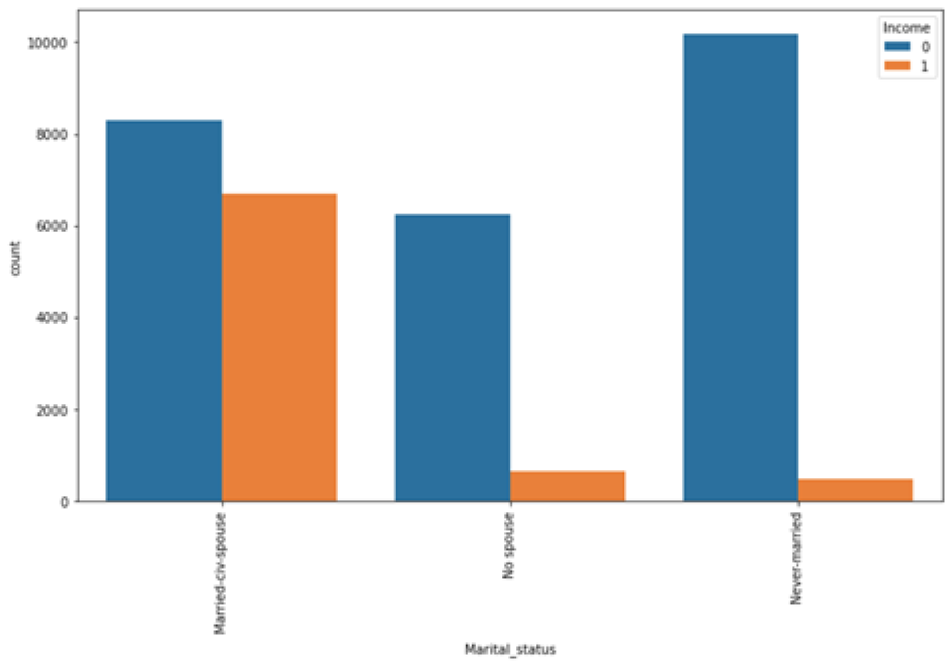
· Education –



We combine all the columns relevant to schools in ‘School’ category, put ‘Doctorate’ and ‘Prof school’ in a single category ‘Doctorate’, ‘Assoc-acdm’ and ‘Assoc-voc’ in one category ‘Assoc’, and ‘HS-Grad’ and ‘Some-college’ in one category ‘College’.

We now obtain 6 categories of education which we feed to our machine learning model.

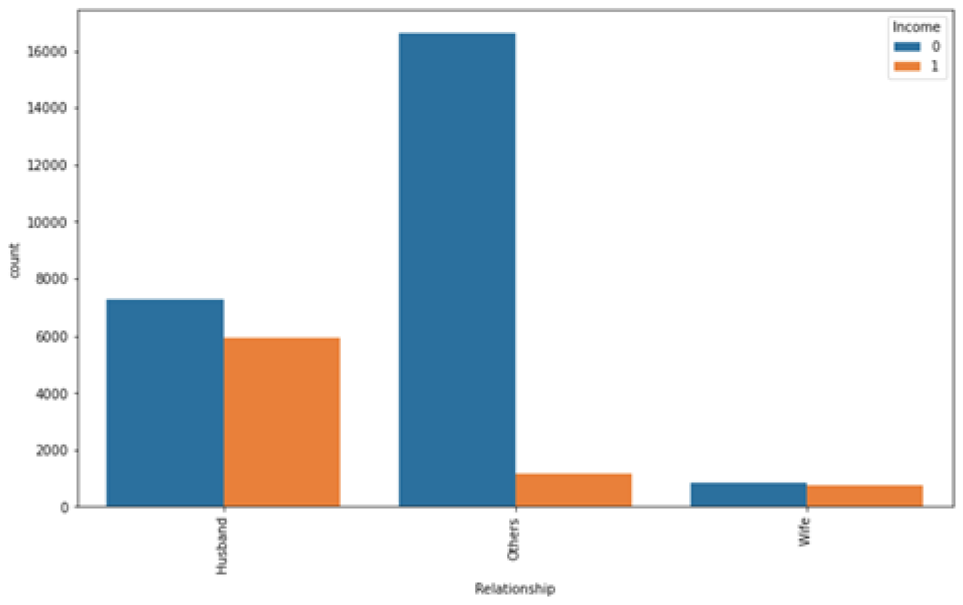
· Marital Status -



We combine ‘Divorced’, ‘Married-spouse-absent’, ‘Separated’, ‘Widowed’ and ‘Married-AF-Spouse’ to one category and name it as ‘No spouse’.

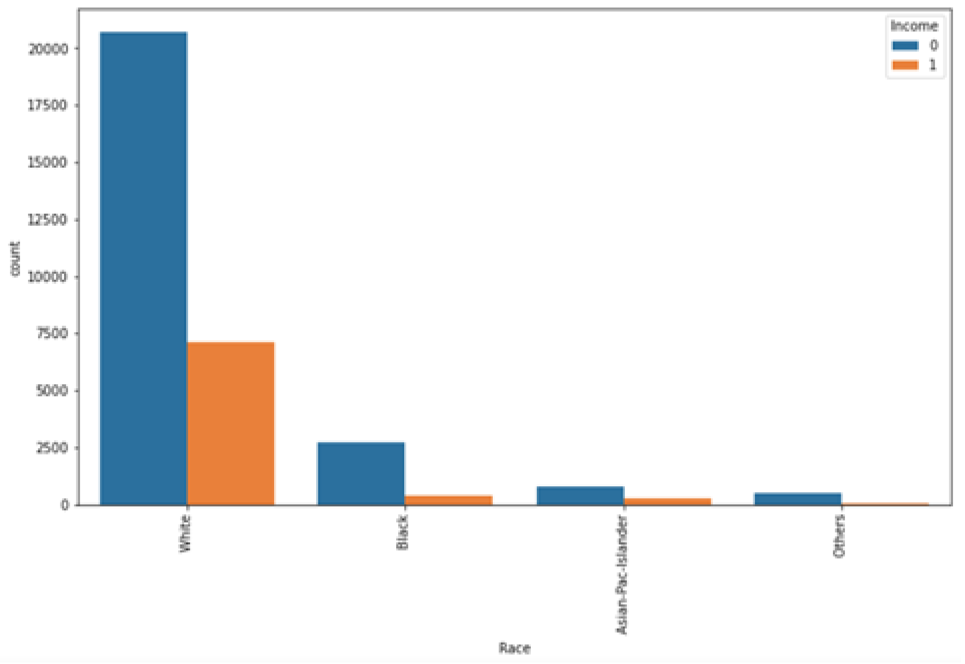
We now obtain 3 categories.

· Relationship column -



We combine ‘Not-in-family’, ‘Own-child’, ‘Unmarried’ and ‘Other-relative’ columns to a single category looking at the distributions, and name is as ‘Other’.

· Race column -

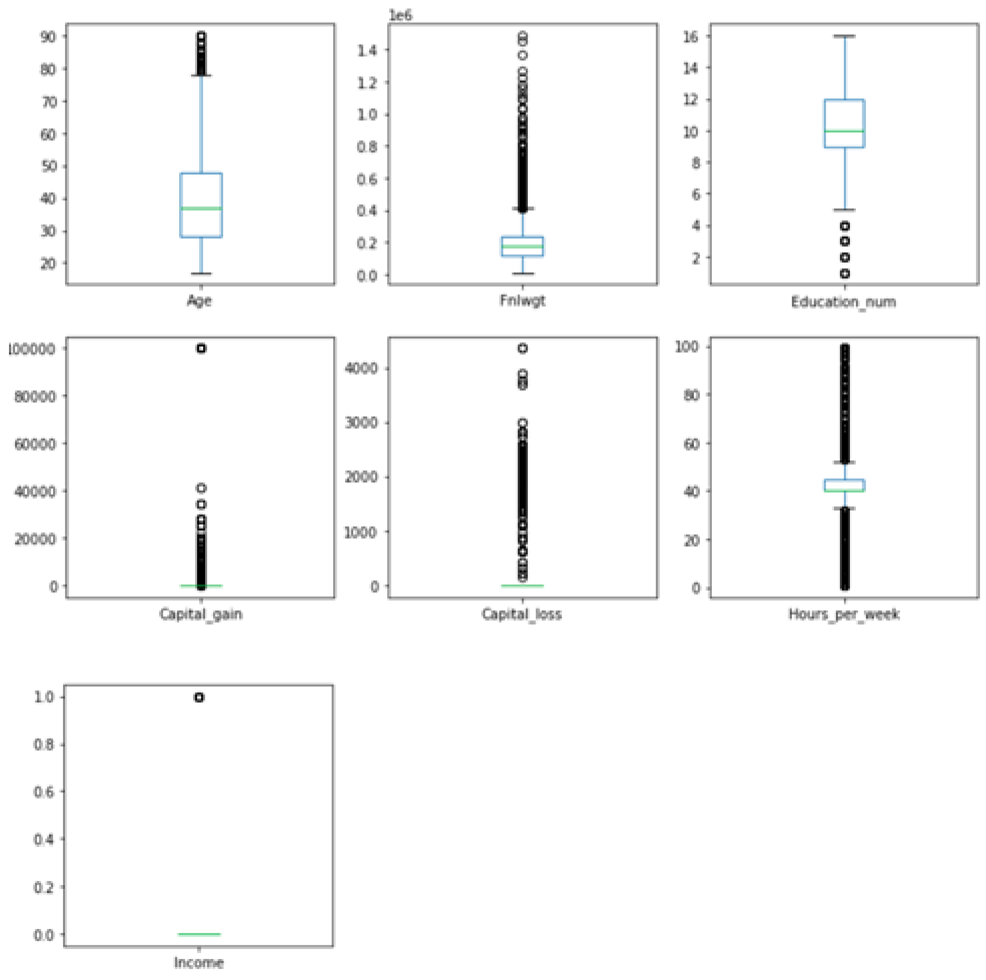


We combine the categories ‘Amer-Indian-Eskimo’ and ‘Other’ to ‘Others’ category, since they have similar distributions.

Our data categorization is now done. The next step is to identify if our data has any outliers, and deal with them.

Outlier Detection

We further proceed to detect outliers in our data and decide how to deal with them. The best way to interpret outliers using visualizations is boxplot, hence we plot boxplots for our numerical columns, which result in the below visuatization –



We observe outliers in all the numeric columns. We use Zscore values to confirm how much data, out of the complete dataset, falls in an outlier range.

‘***Z-score (also called a*** *standard score*) ***gives an idea of how far from the mean a data point is.*** *But more technically it’s a measure of how many standard deviations below or above the population mean a raw score is.’*

After calculating the number of rows containing outliers, we find that we would lose 2733 rows taking a threshold value of zscore as 3. This data has 1546 rows with census income less than 50K$ and 1187 rows with higher income than 50K$. Since our dataset is already imbalanced, losing this number of rows with further increase the imbalance, and would be a significant loss if we consider the rows with income higher than 50K$.

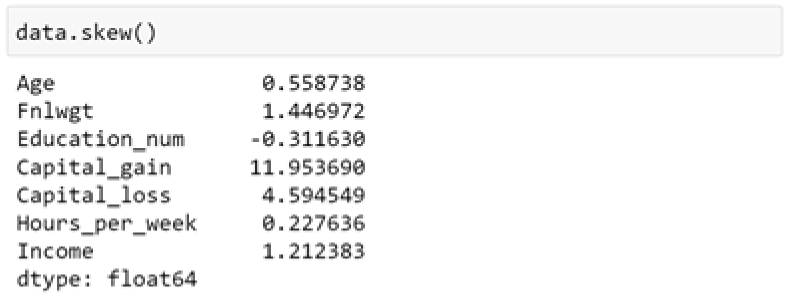
Hence, we keep the rows and proceed with the next steps.

Skewness treatment

We now proceed with treating skewness in our data, which allows us to fit

our data in a symmetric distribution, which further allows our model to learn better.

The skewness of the data without any transformation is –

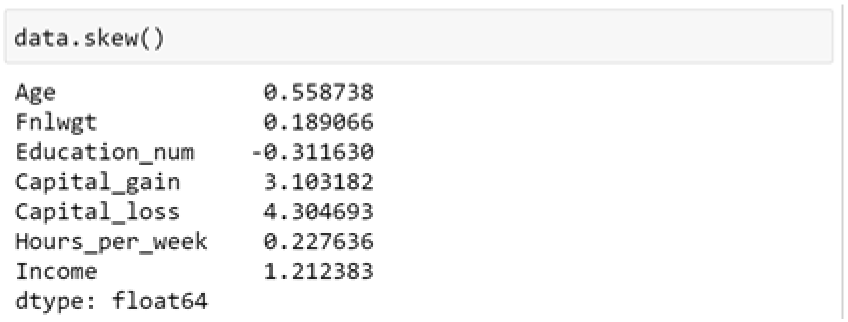


```
data.skew()
```

```
Age                0.558738
Fnlwgt             1.446972
Education_num     -0.311630
Capital_gain      11.953690
Capital_loss       4.594549
Hours_per_week     0.227636
Income             1.212383
dtype: float64
```

We treat 'Fnlwgt', 'Capital_gain' and 'Capital_loss' column for skewness, and use square-root transform and cube-root transform methods (since we cannot apply log and boxcox transform to columns where 0 values are present)

The final skewness that we receive after multiple transformations (sqrt transform for Fnlwgt column, and cbrt transform for Capital_gain and Capital_loss columns twice) –



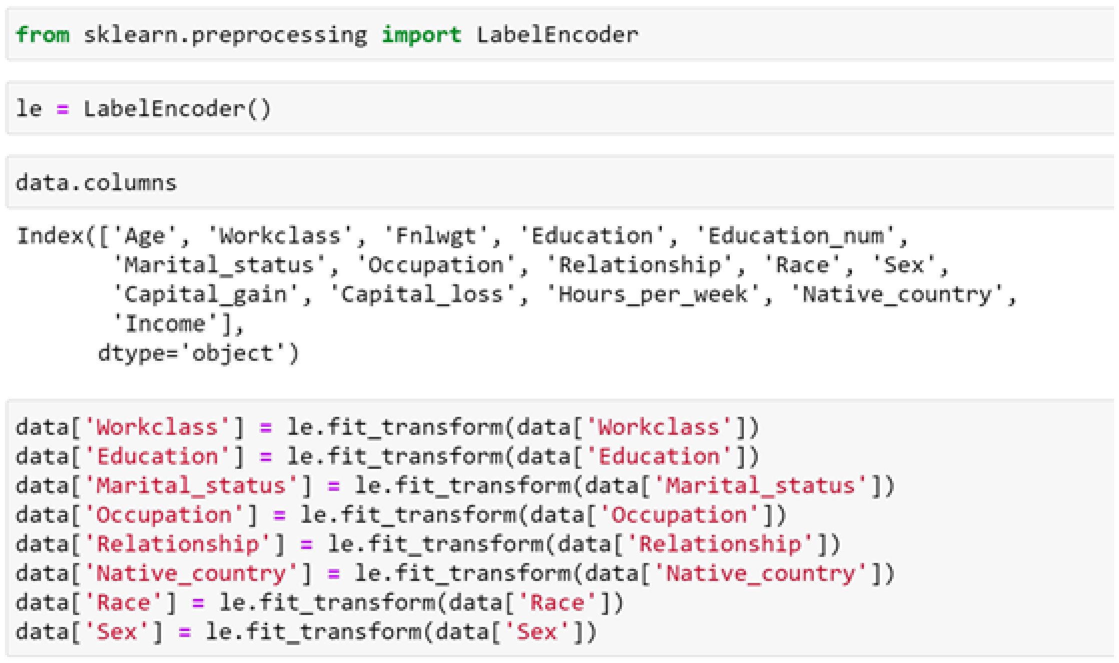
```
data.skew()
```

```
Age                0.558738
Fnlwgt             0.189066
Education_num     -0.311630
Capital_gain       3.103182
Capital_loss       4.304693
Hours_per_week     0.227636
Income             1.212383
dtype: float64
```

We further proceed to next steps since the skewness does not get further decreased.

Encoding the data

Since majority of the classification models need input as ‘int/float’, and do not work on ‘string’ data, we encode our categorical columns using ‘Label Encoder’



```
from sklearn.preprocessing import LabelEncoder
```

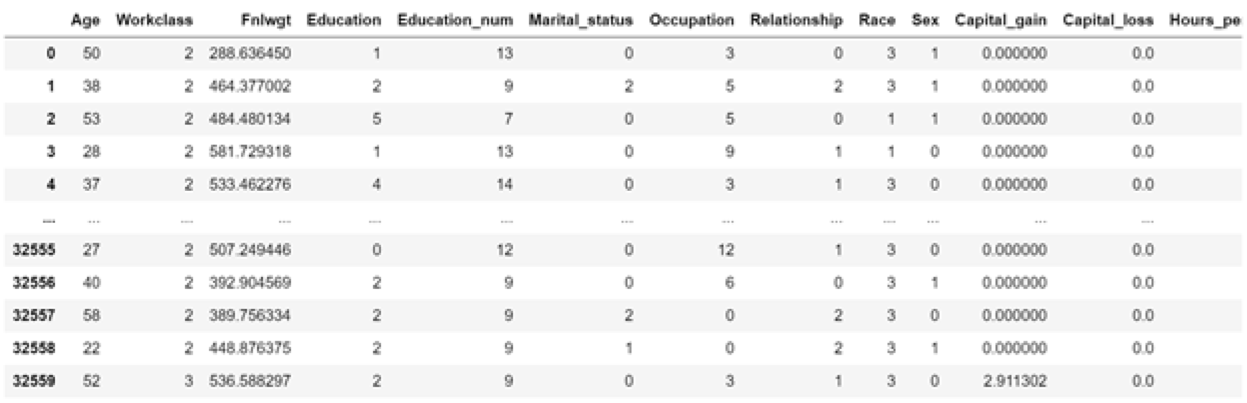
```
le = LabelEncoder()
```

```
data.columns
```

```
Index(['Age', 'Workclass', 'Fnlwgt', 'Education', 'Education_num',
       'Marital_status', 'Occupation', 'Relationship', 'Race', 'Sex',
       'Capital_gain', 'Capital_loss', 'Hours_per_week', 'Native_country',
       'Income'],
      dtype='object')
```

```
data['Workclass'] = le.fit_transform(data['Workclass'])
data['Education'] = le.fit_transform(data['Education'])
data['Marital_status'] = le.fit_transform(data['Marital_status'])
data['Occupation'] = le.fit_transform(data['Occupation'])
data['Relationship'] = le.fit_transform(data['Relationship'])
data['Native_country'] = le.fit_transform(data['Native_country'])
data['Race'] = le.fit_transform(data['Race'])
data['Sex'] = le.fit_transform(data['Sex'])
```

The final dataset looks something like –



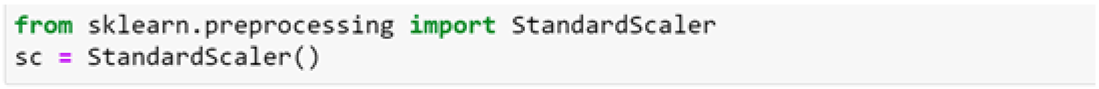
	Age	Workclass	Fnlwgt	Education	Education_num	Marital_status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	Hours_pe
0	50	2	288.636450	1	13	0	3	0	3	1	0.000000	0.0	
1	38	2	464.377002	2	9	2	5	2	3	1	0.000000	0.0	
2	53	2	484.480134	5	7	0	5	0	1	1	0.000000	0.0	
3	28	2	581.729318	1	13	0	9	1	1	0	0.000000	0.0	
4	37	2	533.462276	4	14	0	3	1	3	0	0.000000	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32555	27	2	507.249446	0	12	0	12	1	3	0	0.000000	0.0	
32556	40	2	392.904569	2	9	0	6	0	3	1	0.000000	0.0	
32557	58	2	389.756334	2	9	2	0	2	3	0	0.000000	0.0	
32558	22	2	448.876375	2	9	1	0	2	3	1	0.000000	0.0	
32559	52	3	536.588297	2	9	0	3	1	3	0	2.911302	0.0	

Scaling the data

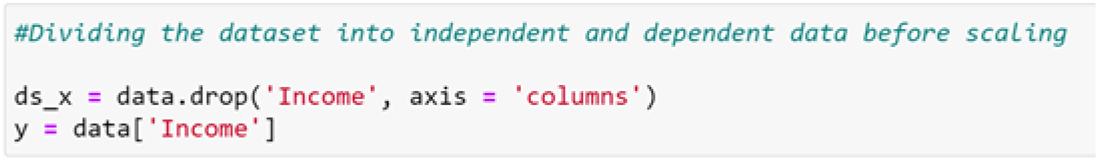
The next step is to bring the data to a common scale, since there are certain columns with very small values and some columns with high values. This process is important as values on a similar scale allow the model to learn better.

We use standard scaler for this process –

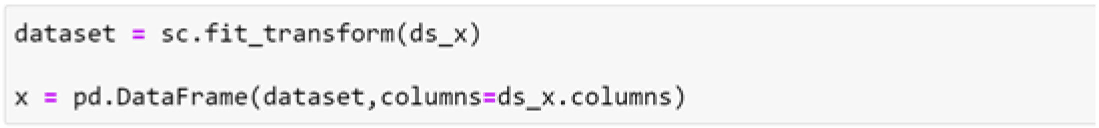
'StandardScaler follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance'



```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```



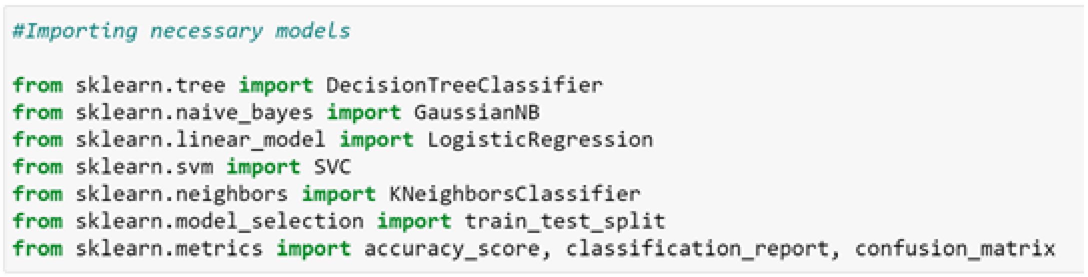
```
#Dividing the dataset into independent and dependent data before scaling

ds_x = data.drop('Income', axis = 'columns')
y = data['Income']
```



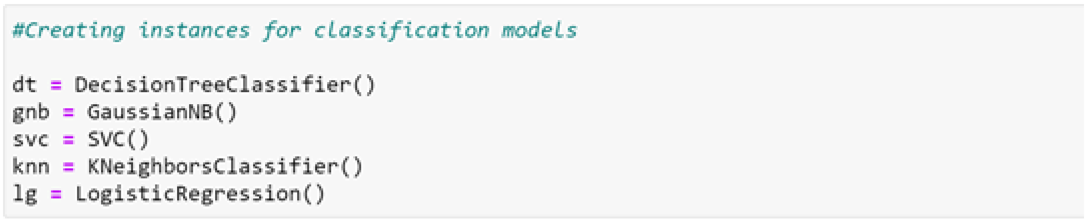
```
dataset = sc.fit_transform(ds_x)

x = pd.DataFrame(dataset,columns=ds_x.columns)
```

Fitting data into classification models

We now proceed to the main step of our machine learning, fitting the model and predicting the outputs. We fit the data into multiple classification models to compare the performance of all models and select the best model –

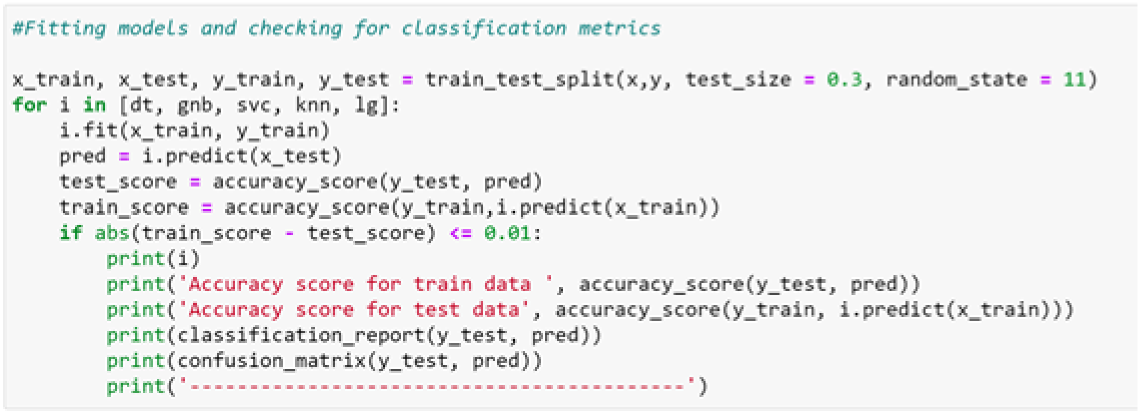


```python
#Importing necessary models

from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```



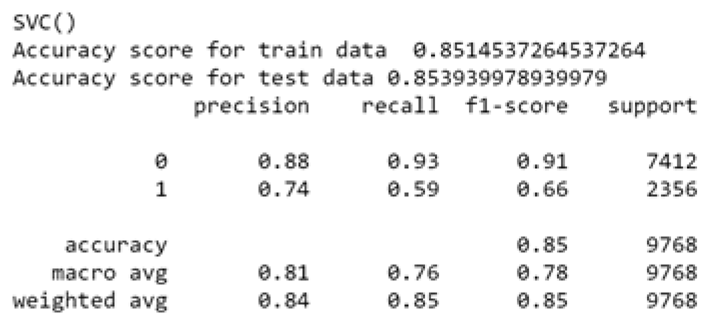
```python
#Creating instances for classification models

dt = DecisionTreeClassifier()
gnb = GaussianNB()
svc = SVC()
knn = KNeighborsClassifier()
lg = LogisticRegression()
```

We use the below mentioned code snipped to fit the data into ML models and predict the output –



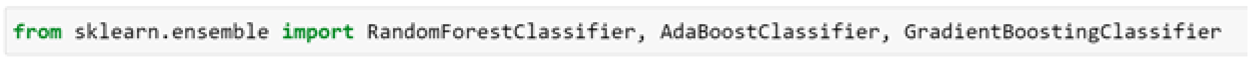
```python
#Fitting models and checking for classification metrics

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3, random_state = 11)
for i in [dt, gnb, svc, knn, lg]:
    i.fit(x_train, y_train)
    pred = i.predict(x_test)
    test_score = accuracy_score(y_test, pred)
    train_score = accuracy_score(y_train,i.predict(x_train))
    if abs(train_score - test_score) <= 0.01:
        print(i)
        print('Accuracy score for train data ', accuracy_score(y_test, pred))
        print('Accuracy score for test data', accuracy_score(y_train, i.predict(x_train)))
        print(classification_report(y_test, pred))
        print(confusion_matrix(y_test, pred))
        print('-----------------------------------------')
```

We achieve the best results using — ‘Support Vector Classifier’, which provides an accuracy of 85% on test data. Below is the classification report for the SVC model –

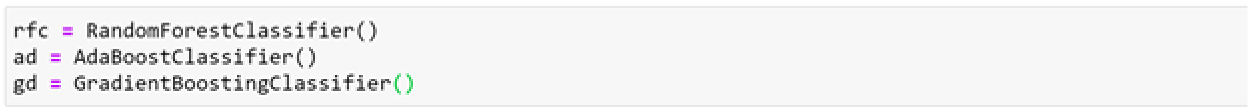


```
SVC()
Accuracy score for train data  0.8514537264537264
Accuracy score for test data 0.853939978939979
              precision    recall  f1-score   support

           0       0.88      0.93      0.91      7412
           1       0.74      0.59      0.66      2356

    accuracy                           0.85      9768
   macro avg       0.81      0.76      0.78      9768
weighted avg       0.84      0.85      0.85      9768
```

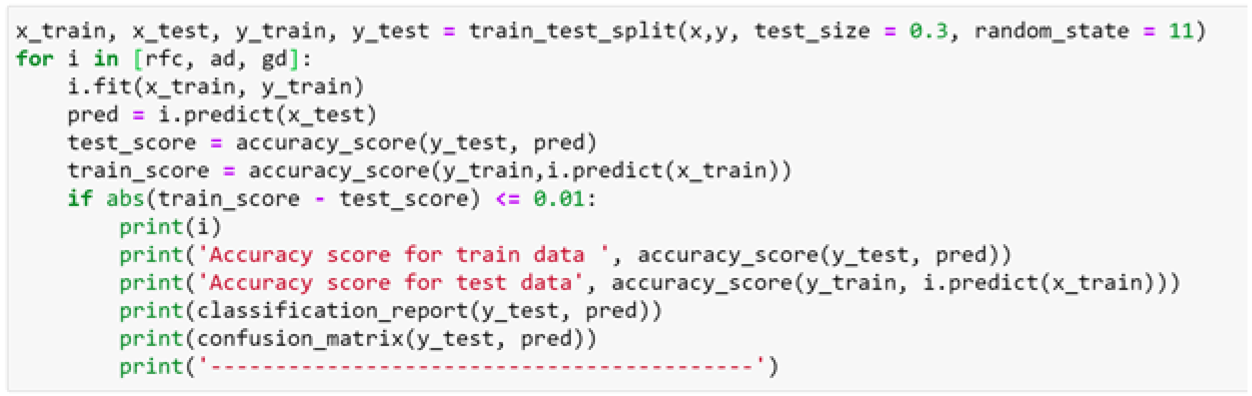
We further try fitting the data to classification models to check how our ensemble models perform on the given dataset.



```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
```

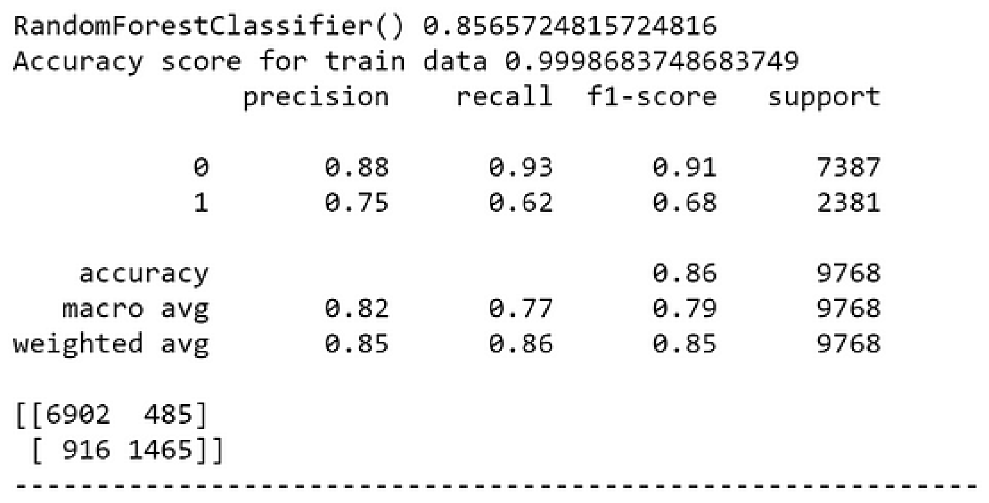


```
rfc = RandomForestClassifier()
ad = AdaBoostClassifier()
gd = GradientBoostingClassifier()
```

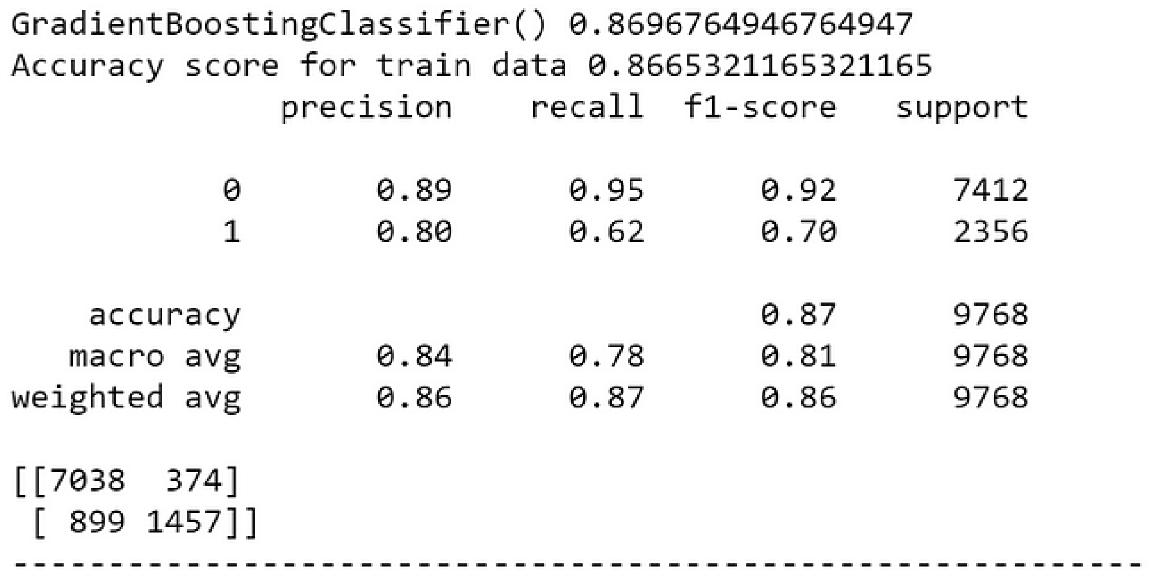


```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3, random_state = 11)
for i in [rfc, ad, gd]:
    i.fit(x_train, y_train)
    pred = i.predict(x_test)
    test_score = accuracy_score(y_test, pred)
    train_score = accuracy_score(y_train,i.predict(x_train))
    if abs(train_score - test_score) <= 0.01:
        print(i)
        print('Accuracy score for train data ', accuracy_score(y_test, pred))
        print('Accuracy score for test data', accuracy_score(y_train, i.predict(x_train)))
        print(classification_report(y_test, pred))
        print(confusion_matrix(y_test, pred))
        print('-----------------------------------------')
```

As a result, we observe that Random Forest Classifier over fits the train set, and gives a 99% accuracy on train data.



```
RandomForestClassifier() 0.8565724815724816
Accuracy score for train data 0.9998683748683749
              precision    recall  f1-score   support

           0       0.88      0.93      0.91      7387
           1       0.75      0.62      0.68      2381

    accuracy                           0.86      9768
   macro avg       0.82      0.77      0.79      9768
weighted avg       0.85      0.86      0.85      9768

[[6902  485]
 [ 916 1465]]
-----------------------------------------------------------
```



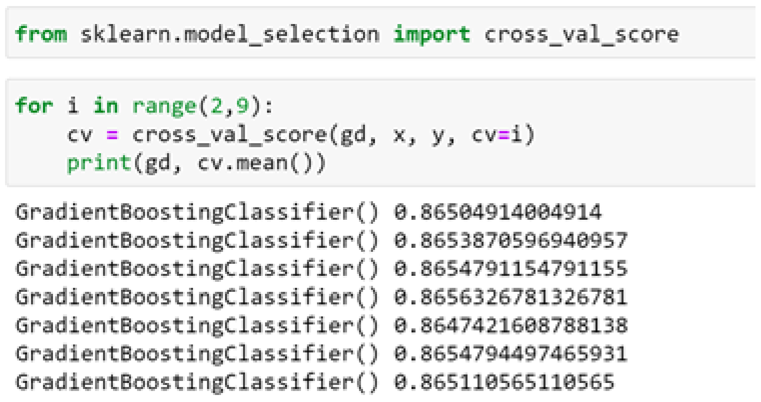
```
GradientBoostingClassifier() 0.8696764946764947
Accuracy score for train data 0.8665321165321165
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      7412
           1       0.80      0.62      0.70      2356

    accuracy                           0.87      9768
   macro avg       0.84      0.78      0.81      9768
weighted avg       0.86      0.87      0.86      9768

[[7038  374]
 [ 899 1457]]
-----------------------------------------------------------
```

We see that the Gradient Boosting Classifier gives us an accuracy of ~87% (higher than SVC), and the f1-score, recall and precision scores also improve. ***Hence we choose 'Gradient boosting classifier' as our final model***, and proceed with hypertuning the model. But before this, we perform k-folds cross validation on our dataset.

Cross validation

The **goal of cross-validation** is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

We obtain the following results using cross-validation –



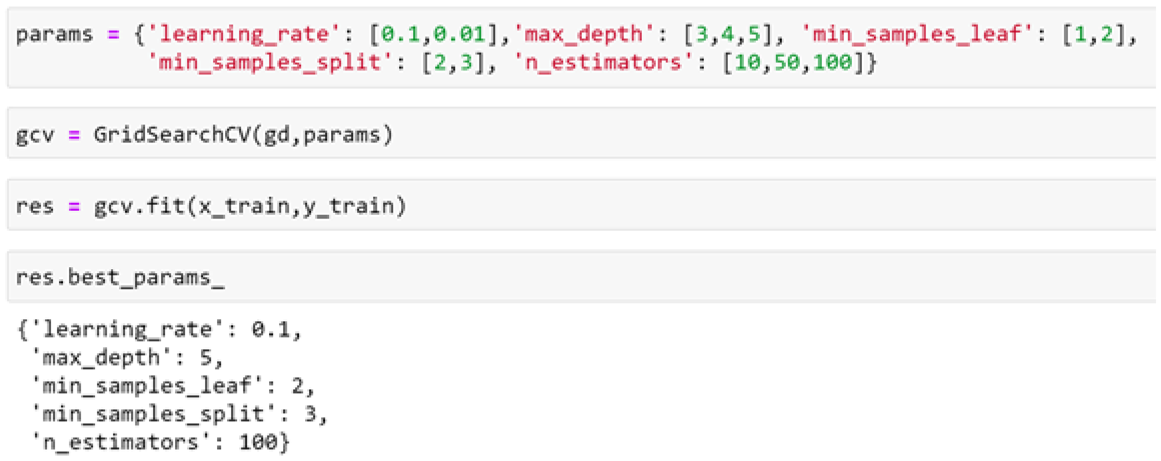
```
from sklearn.model_selection import cross_val_score
```

```
for i in range(2,9):
    cv = cross_val_score(gd, x, y, cv=i)
    print(gd, cv.mean())
```

```
GradientBoostingClassifier() 0.86504914004914
GradientBoostingClassifier() 0.8653870596940957
GradientBoostingClassifier() 0.8654791154791155
GradientBoostingClassifier() 0.8656326781326781
GradientBoostingClassifier() 0.8647421608788138
GradientBoostingClassifier() 0.8654794497465931
GradientBoostingClassifier() 0.865110565110565
```

This helps us interpret that the model is not overfitting and will perform well for new data that we feed to our model. We now proceed with hypertuning the model, using GridSearch CV.

Hypertuning the model



```
params = {'learning_rate': [0.1,0.01],'max_depth': [3,4,5], 'min_samples_leaf': [1,2],
          'min_samples_split': [2,3], 'n_estimators': [10,50,100]}
```

```
gcv = GridSearchCV(gd,params)
```

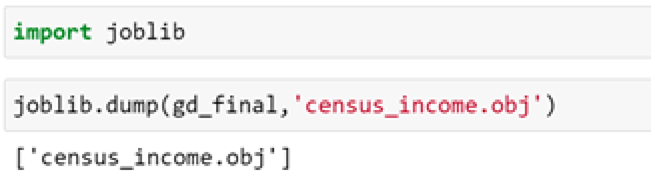
```
res = gcv.fit(x_train,y_train)
```

```
res.best_params_
```

```
{'learning_rate': 0.1,
 'max_depth': 5,
 'min_samples_leaf': 2,
 'min_samples_split': 3,
 'n_estimators': 100}
```

We receive the best params for our model, which result in a best score of 87.7%. We increased our model accuracy by 1% using hypertuning.

We now save the model with best parameters that we identified using GridSearch, and create an object for our model using ‘Joblib’.



```
import joblib
```

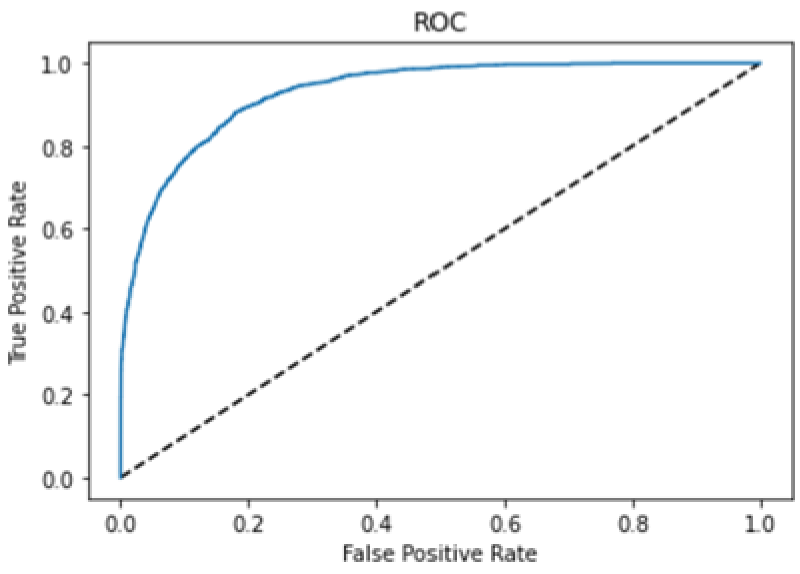
```
joblib.dump(gd_final,'census_income.obj')
```

```
['census_income.obj']
```

AUC ROC curve

AUC — ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0’s as 0’s and 1’s as 1’s.

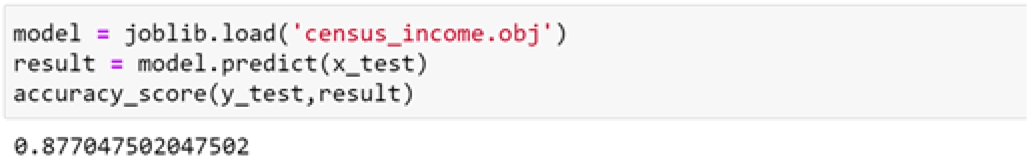
We draw the AUC-ROC curve to obtain the following output –



The area under the ROC curve represents the ability of our model to predict correct values, and the curve that we got is quite a good score.

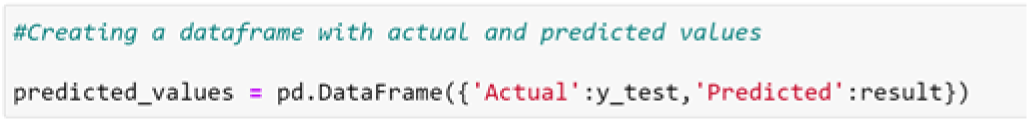
Conclusion

We further proceed to test the object that we saved using joblib, and create a dataframe of predicted values –



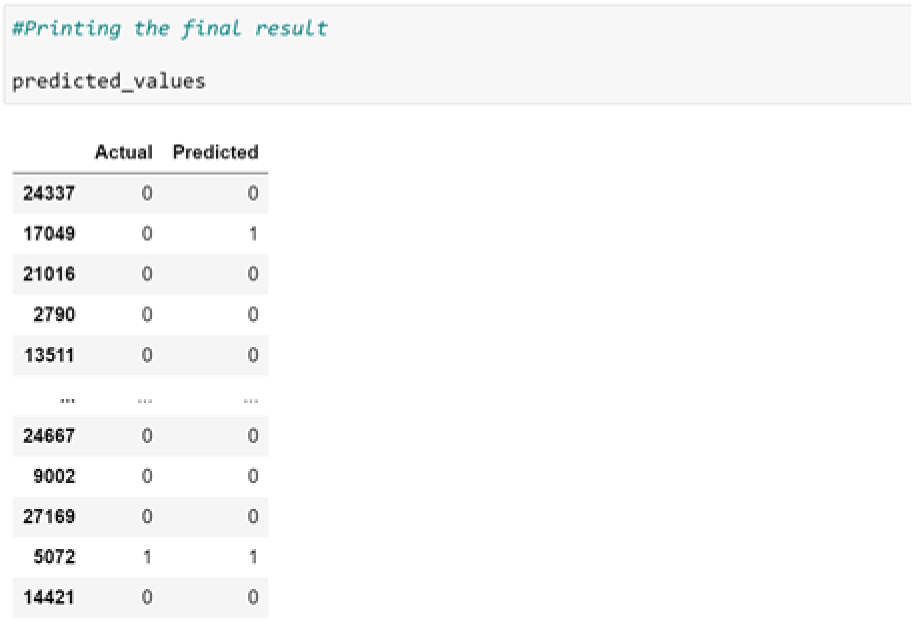
```
model = joblib.load('census_income.obj')
result = model.predict(x_test)
accuracy_score(y_test,result)
```

```
0.877047502047502
```



```
#Creating a dataframe with actual and predicted values

predicted_values = pd.DataFrame({'Actual':y_test,'Predicted':result})
```

Following are the results that we achieve, with an accuracy of 87.7%.



```
#Printing the final result

predicted_values
```

	Actual	Predicted
24337	0	0
17049	0	1
21016	0	0
2790	0	0
13511	0	0
...	...	...
24667	0	0
9002	0	0
27169	0	0
5072	1	1
14421	0	0

This marks the end of our process; we have successfully trained our model to predict the income of a person, with an accuracy of ~88%.

We moved step by step, analyzing, cleaning and modeling the data, and applied various machine learning models to achieve the desired predictions. We also tuned the model to improve the accuracy, and were able to achieve a model with quite a good accuracy.