BSc (Hons) Artificial Intelligence and Data Science

Module: CM2604 Machine Learning

Coursework Report

Module Leader: Mr. Sahan Priyanayana

RGU Student ID : 2313081

IIT Student ID : 20221470

Student Name : Withanage Lakshan Anjana Cooray

Contents

Introduction

This report discusses a binary classification approach taken using Random Forest Classifier and Naïve Bayes algorithms (Bernoulli Naïve Bayes) to predict whether the income of a person exceeds fifty thousand dollars or not based on the U.S. adult census data set available at the UCI machine learning repository. This report discusses how the data cleaning process is done, what feature engineering methods are used, visualization of the data with respect to the income rates, outlier detection and handling outliers of the data set, how correlation-based matrices are used to select features to train models, how duplicated values are handled, how imbalanced learning techniques are used to handle imbalanced target variable(income), what hyperparameter optimization techniques are used to train models, and the classification algorithms-based evaluation matrices used to evaluate the models on how they predict adult income rates.

Dataset Description

This income analysis and model training is done by using adult income dataset collected by US census available at UCI machine learning repository to find the people who are having an income greater than 50k and less than 50k based on 14 features. This dataset initially consists with 48842 records. It represents the data of people who are older than 16 years. These features consist with both categorical and numerical data. The age column defines the age of the people who are older than 16 years. The work class column defines whether that person is a self-employee, government employee or an employee working in the private sector. However, as mentioned in the UCI repository this column consists of missing values. The “fnlwgt” column represents the final weight of the person which is a measurement taken based on the census data. The education column defines the educational level of each person including details of the final stage they have done their studies (Bachelors, school level, high school graduate, masters, doctorate etc). The “education-num” column represents the educational level of each person while “marital-status” column has defined whether that person is married, not married, widowed, or divorced. The occupation column defines the area that person is specialized. Hower, the UCI repository has mentioned this occupation column consists of missing values. The “relationship” column consists of categorical values to explain the relationship of the person. The “race” column consists with categorical data that explains whether that person is White, Asian-Pac-Islander, Amer-Indian-Eskimo or black. The “sex” column defines whether

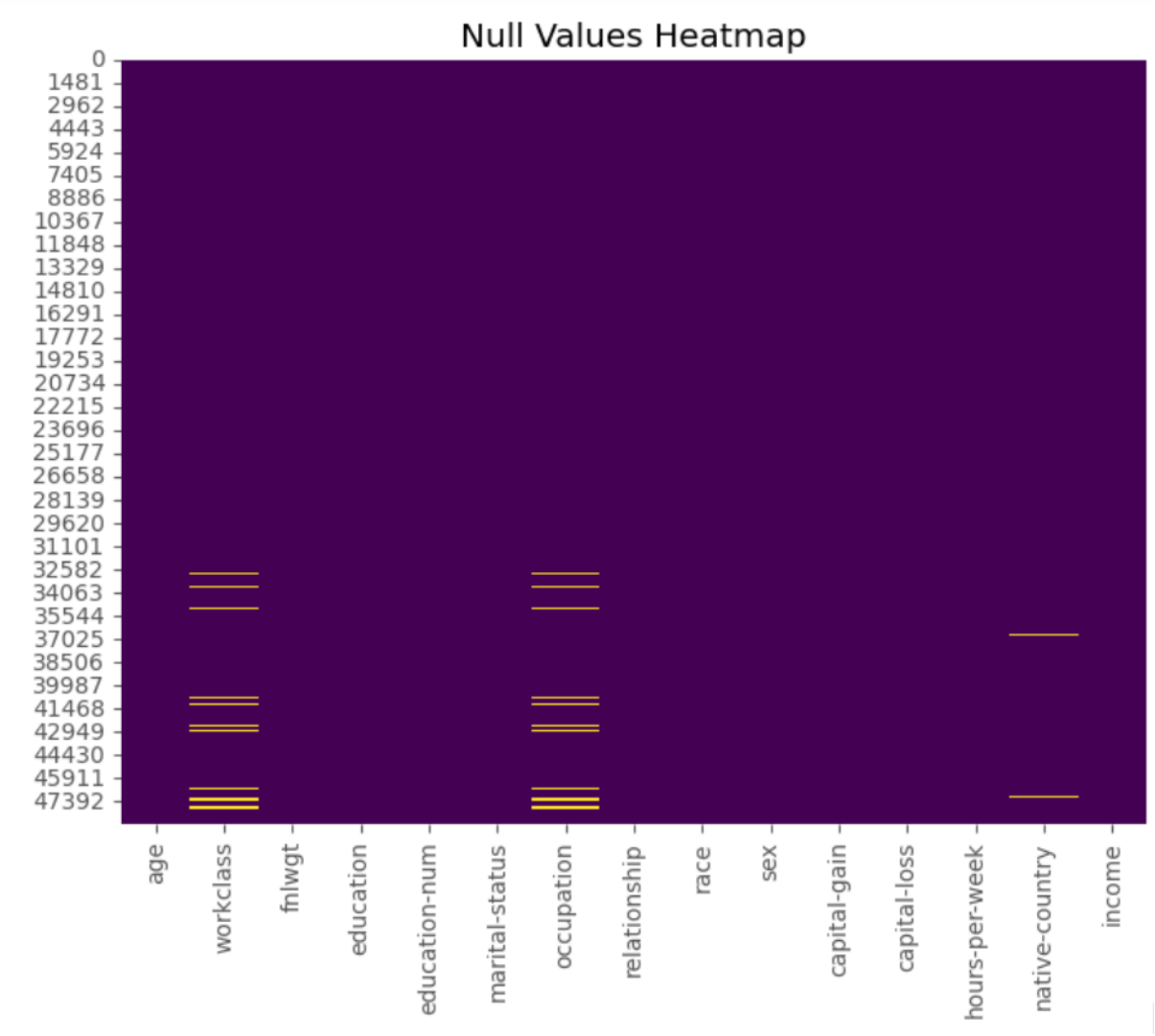
that person is a female or male and it represents binary data. Capital-gain and capital-loss columns defines the profits and loss gain by selling assets. These two columns consist with integer-based data. The hours-per-week column defines the working hours that person works, and it consist with numerical data. The “native country” column defines the country that person is coming from, and it is a column that consists of categorical data. The income column defines whether that person is getting a salary greater than 50K or not. It is the column that represents the target variable of the dataset. It consists of categorical binary data.

Varaible Name	**Data Type**	**Missing Values**
Age	Integer	No
workclass	Categorical	Yes
fnlwgt	Integer	No
education	Categorical	No
education-num	Integer	No
marital status	Categorical	No
occupation	Categorical	Yes
relationship	Categorical	No
race	Categorical	No
sex	Binary	No
Capital-gain	Integer	No
Capital-loss	Integer	No
Hours-per-week	Integer	No
Native-country	Categorical	Yes
Income	Binary	No

Corpus Preparation

Handling missing values in categorical columns

As the first step of handling missing values, first each column is plotted as a heatmap to check what are the columns that consist of null values. That shows, “workclass”, “occupation” and “native country” columns have records that consist of null values.

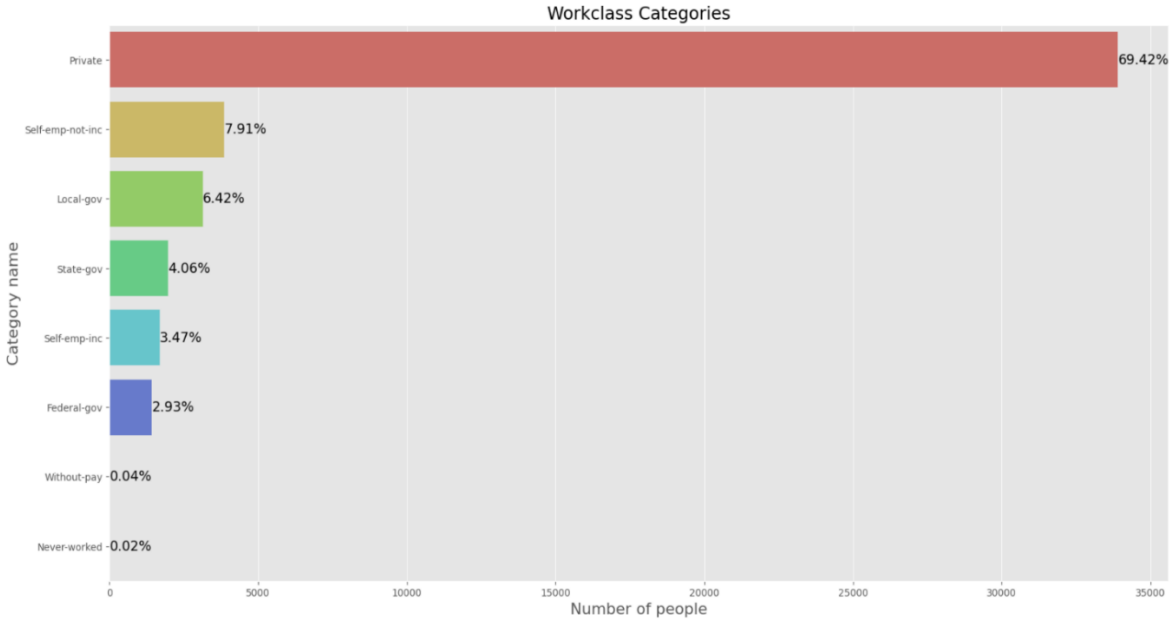


Then checked how many null values are there in each column that consists null values. It showed that work class column contains 963 null values, occupation column contains 966 null values and native-country column contains 274 null values.

```
age                0
workclass        963
fnlwgt             0
education          0
education-num      0
marital-status     0
occupation       966
relationship       0
race               0
sex                0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country   274
income             0
dtype: int64
```

Workclass column

Then each column is checked separately and checked whether there are any other missing values in different text formats. The workclass column contained "?" values that does not convey any meaning. Therefore, it has been identified as a missing value in the dataset. This column contains 1836 "?" values and 963 null values. To handle the null values, the categories are plotted to check how it should be handled. Before that the "?" values are converted to null values to easily handle it. The below plot shows how the other categorical variables distributed in the "workclass" column.

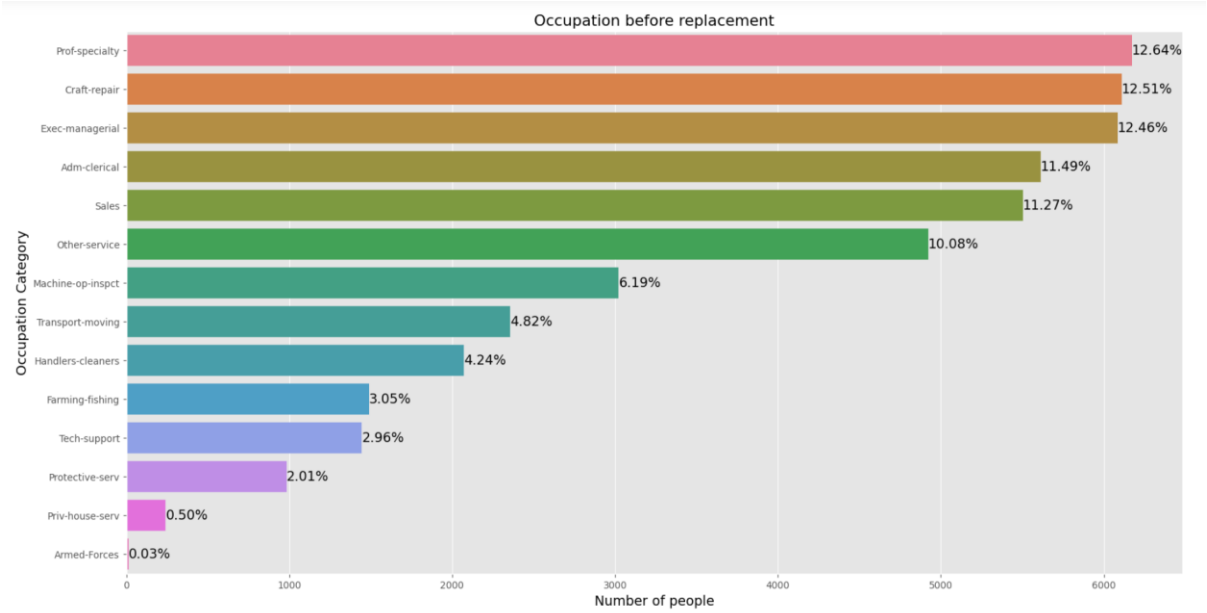


This shows that most of the workclass values contains with people who are working in the private sector. Therefore, the null values are handled by replacing them with the mode category (Private Class). After replacing it shows that it has not impacted a lot on the way that categories are initially distributed. The below picture shows how the categories are distributed after replacing the values.

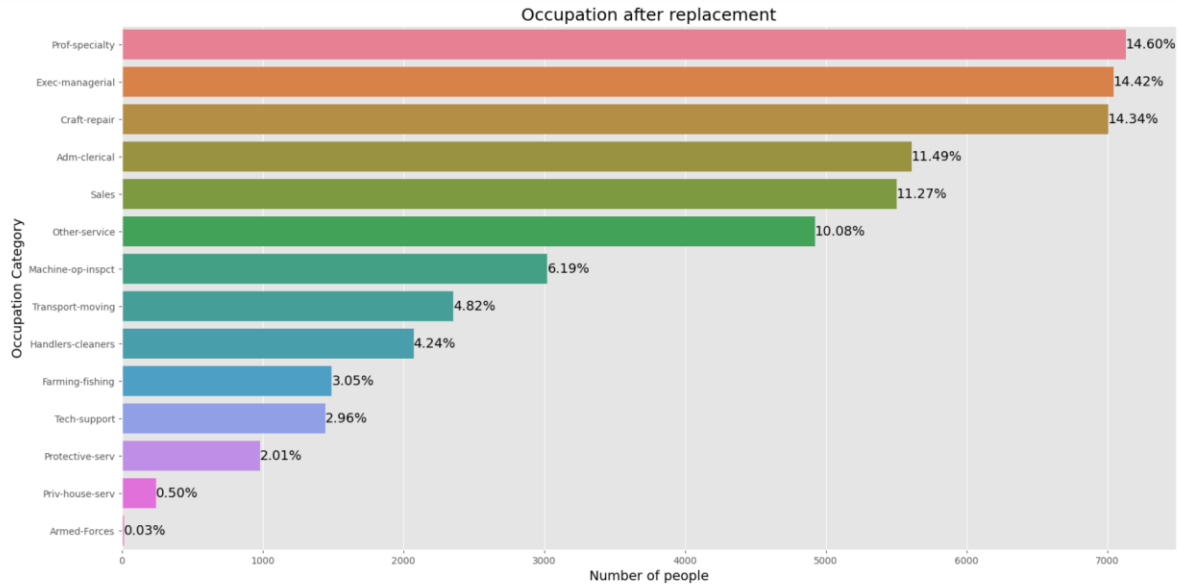
```
workclass
Private                36705
Self-emp-not-inc        3862
Local-gov               3136
State-gov               1981
Self-emp-inc            1695
Federal-gov             1432
Without-pay               21
Never-worked              10
```

Occupation Column

The occupation column consisted with 1843 “?” values and 966 null values. Under handling the null values in the column Occupation column, “?” values are converted to null values and then checked the initial distribution of the occupation categories to handle the missing null values. The below bar chart shows the initial distribution of workclass column without considering null values.



This bar chart shows that the “Prof-speciality” is the mode category, but “Craft repair” and “Exec Managerial” categories are close to the value count of the mode category, therefore the missing values are replaced with “Prof-speciality”, “Craft repair” and “Exec Managerial” categories randomly. The below bar chart shows how the occupation categories distributed after handling the null values.



It shows that it has not impacted a lot on the initial distribution of the Occupation column by randomly replacing the categories that consist with the highest value counts.

Native-Country Column

The native-country column consisted with both null values and "?" mark values as in the occupation and workclass columns. Before handling the null values, the "?" values are converted to null. Then the initial distribution of the native-country categories is taken to check the best fitting method to handle the null values of the native-country column.

```
native-country
United-States                 43832
Mexico                          951
Philippines                     295
Germany                         206
Puerto-Rico                     184
Canada                          182
El-Salvador                     155
India                           151
Cuba                            138
England                         127
China                           122
South                           115
Jamaica                         106
Italy                           105
Dominican-Republic              103
Japan                            92
Guatemala                        88
Poland                           87
Vietnam                          86
Columbia                         85
Haiti                            75
Portugal                         67
Taiwan                           65
Iran                             59
Greece                           49
Nicaragua                        49
Peru                             46
Ecuador                          45
France                           38
Ireland                          37
Hong                             30
Thailand                         30
Cambodia                         28
Trinadad&Tobago                  27
Laos                             23
Yugoslavia                       23
Outlying-US(Guam-USVI-etc)       23
Scotland                         21
Honduras                         20
Hungary                          19
Holand-Netherlands                1
Name: count, dtype: int64
```

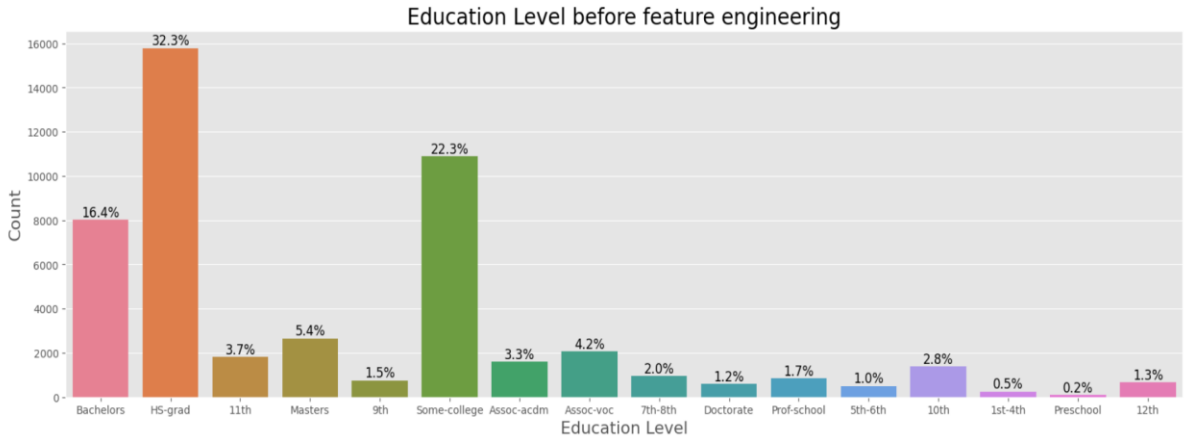
This depicts that there is a huge gap between the United-States (Mode) and other countries for the native country. Therefore, the missing values are replaced with the "United-States".

Feature Engineering for Categorical Values

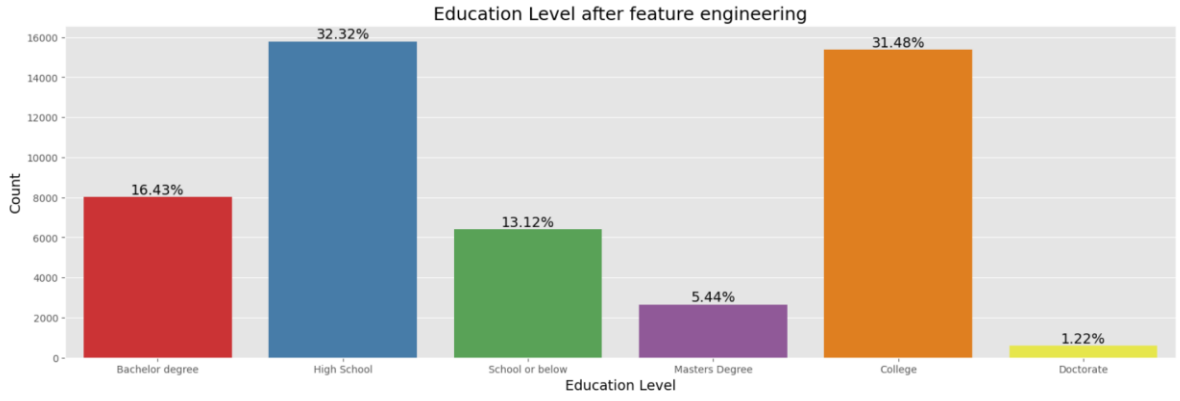
Since most of the categorical columns consist of many categories, the number of categories has been deducted based on the common characteristics of the categories in each categorical columns. If it is not deducted, it will make an impact to the models when it predicts data.

Apply feature engineering in Education Column

Education column consisted with 16 categories. However, there were categories that represented the same educational level of people in different formats. For example, the people who have completed school level studies have represented as 11^{th}, 9^{th}, 7^{th}-8^{th},5^{th}-6^{th} etc. The initial distribution of the education column before doing the feature engineering represents in the below bar chart.

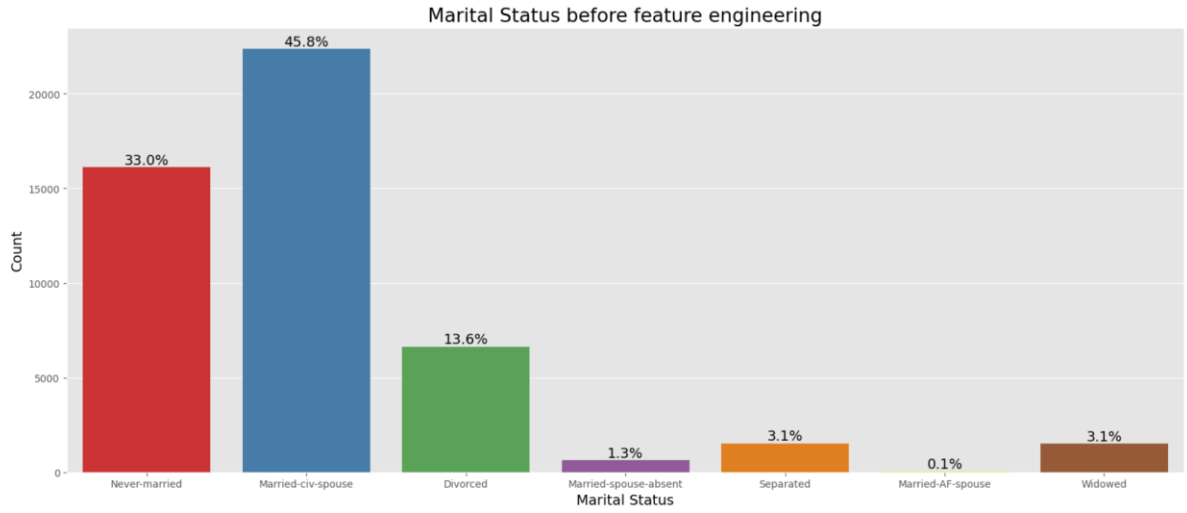


The categories ‘11th’, ‘9th’, ‘7th-8th’, ‘5th-6th’, ‘10th’, ‘1st-4th’, ‘12th’ and ‘Preschool’ are identified as categories that falls within the school or below level. The categories, ‘Assoc-voc’, ‘Assoc-acdm’, ‘Some-college’, ‘Prof-school’ are identified under the category of college level. High school graduate, bachelor's, master’s and doctorate are considered as the other main categories that falls under education level. The below bar chart shows how the education categories distributed after applying feature engineering concepts.

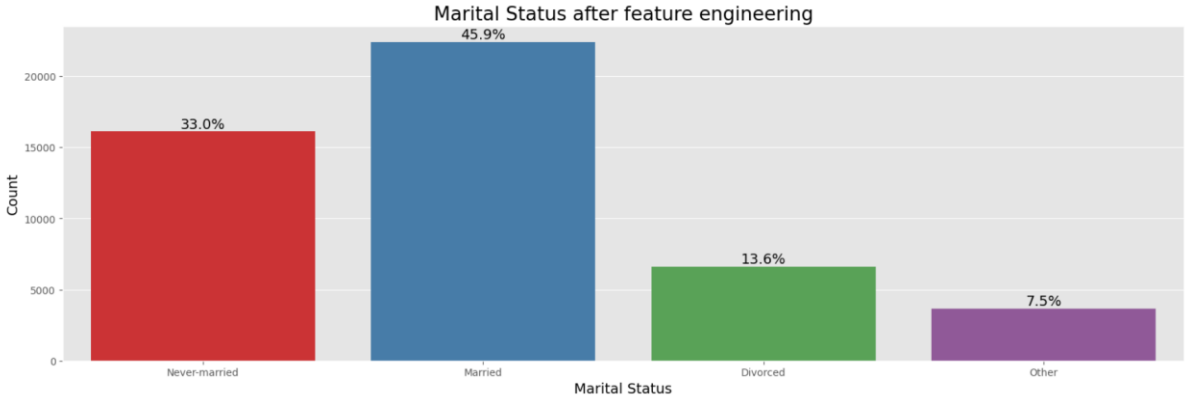


Apply feature engineering in Marital Status Column

The marital status column initially consisted with seven columns. The below bar chart shows the initial categories of Marital status column and its percentages.

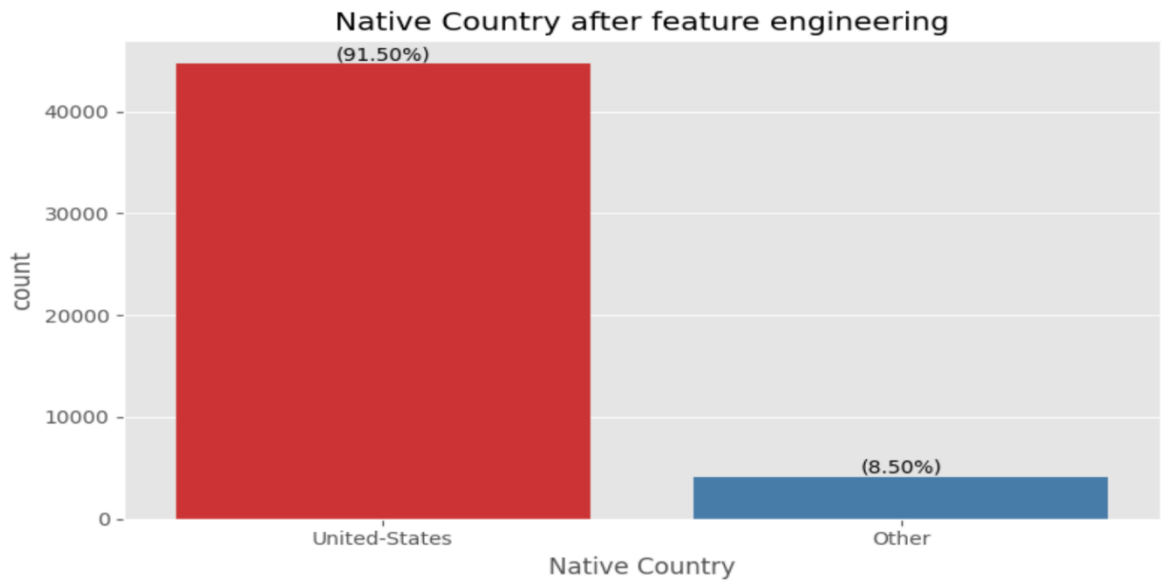


This bar chart shows that 'married-spouce-absent', 'Married-AF-spouce' and 'Widowed' categories consisted with less percentages compared to other categories. Therefore, 'Married-civ-spouse', 'Married-AF-spouse' categories considered under married category. Never Married and Divorced categories considered as two separated categories and 'Married-spouse-absent', 'Widowed' and 'Separated' status is considered under other categories of the marital status. The below chart shows how the categories are distributed after applying the feature engineering concepts to Marital Status column.



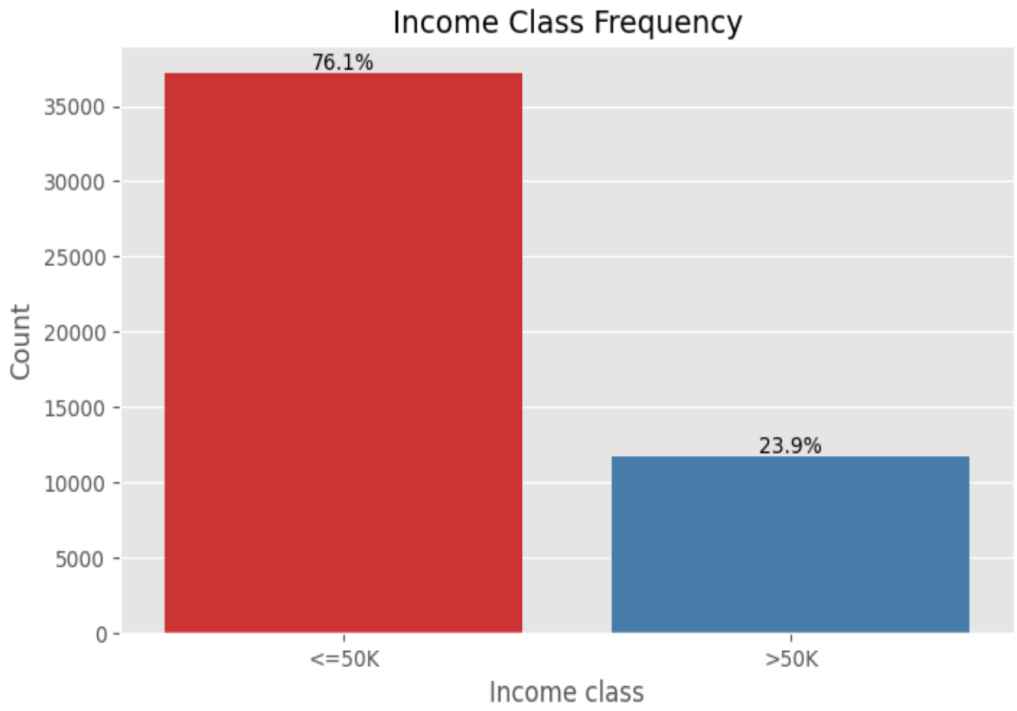
Apply feature engineering in Native country column

The Native Country column consists of ‘United States’ with the highest value count with 44689 records. It represents 91.50% of the records. Therefore, other countries are replaced as “other” since the majority of the columns represent the United States. The below bar chart shows that how native country column looks like after doing feature engineering.



Apply feature engineering for Income column

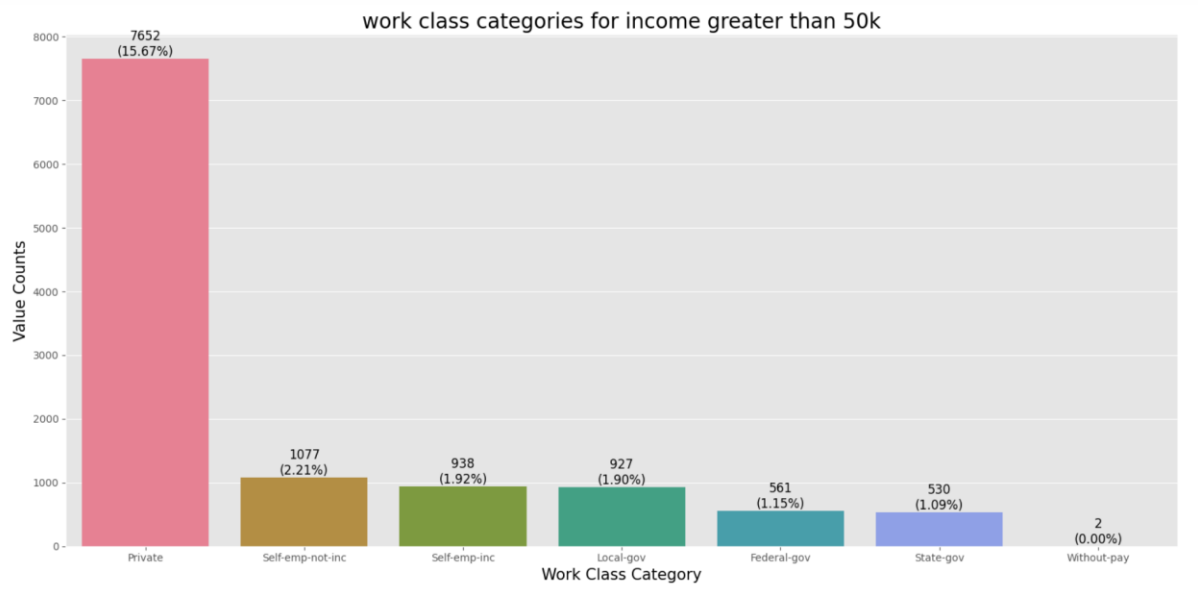
In the given dataset, the income column consisted of four categories, namely, ‘<=50K’, ‘<=50K.’, '>50K’, ‘>50K.’ However, the target column should exist in only two categories since this is a binary classification problem. Therefore, values with ‘<=50K.’ are converted to '<=50K’, and values with ‘>50K.’ are replaced with '>50K’. The below chart shows how the income column is distributed after replacing the values. In addition, it shows that the dataset is imbalanced.



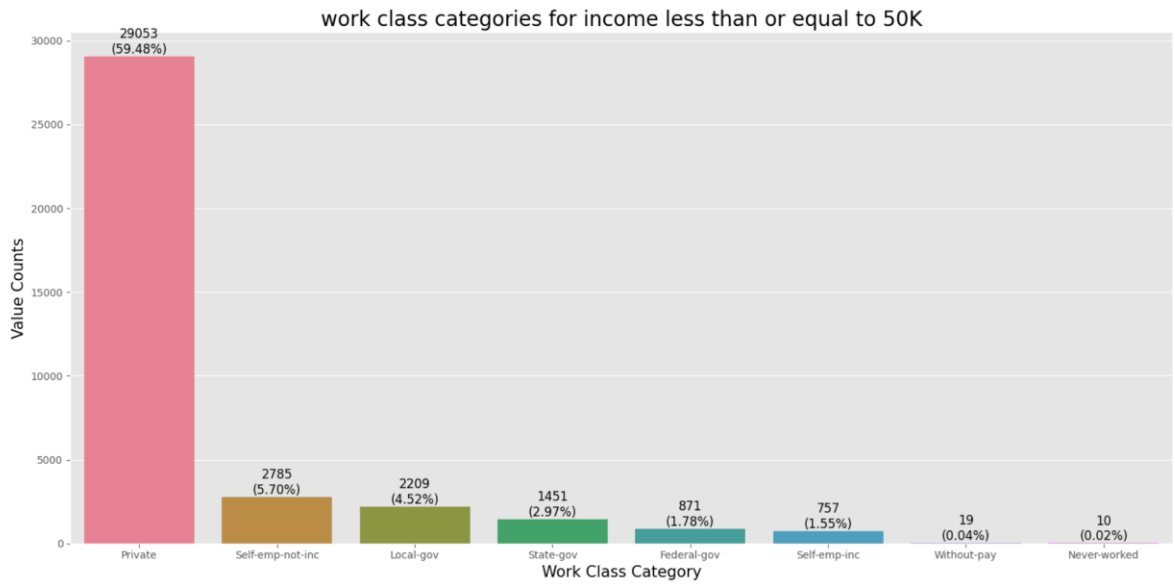
Apply feature engineering for workclass column

Before applying feature engineering concepts for workclass column, checked whether there will be an impact by replacing and deducting the number of categories in workclass column. Therefore, got the percentage of each workclass category with respect to the income to see whether if replacing values will affect for the final predictions.

Workclass when income is greater than 50K



Workclass when income is less than 50K



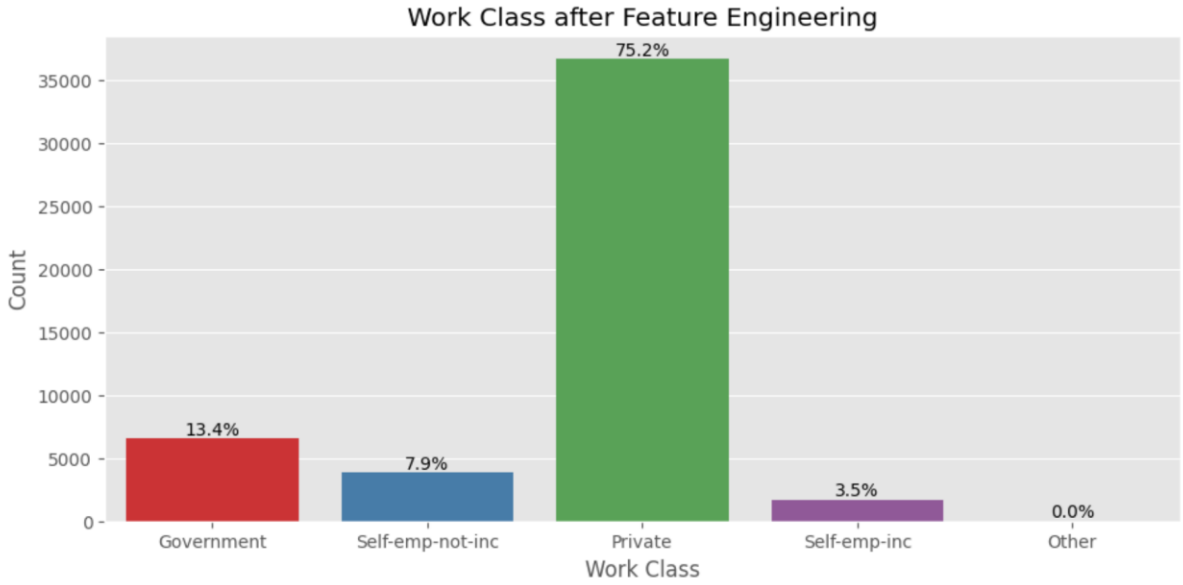
These bar charts show that people who earn more than 50k and earn less than 50k can be seen in all categories except the Never worked category. When never worked category is checked against the working hours it shows there are some people who have working hours. That is an abnormality found in the dataset. Therefore, these values are dropped by filtering the dataset.

Check whether never-worked people have working hours

```
#### This is an abnormality, this cannot be happen
data[data["workclass"] == "Never-worked"]["hours-per-week"]
```

```
5361     40
10845    35
14772    30
20337    10
23232    40
32304    40
32314     4
41346    20
44168    35
46459    35
Name: hours-per-week, dtype: int64
```

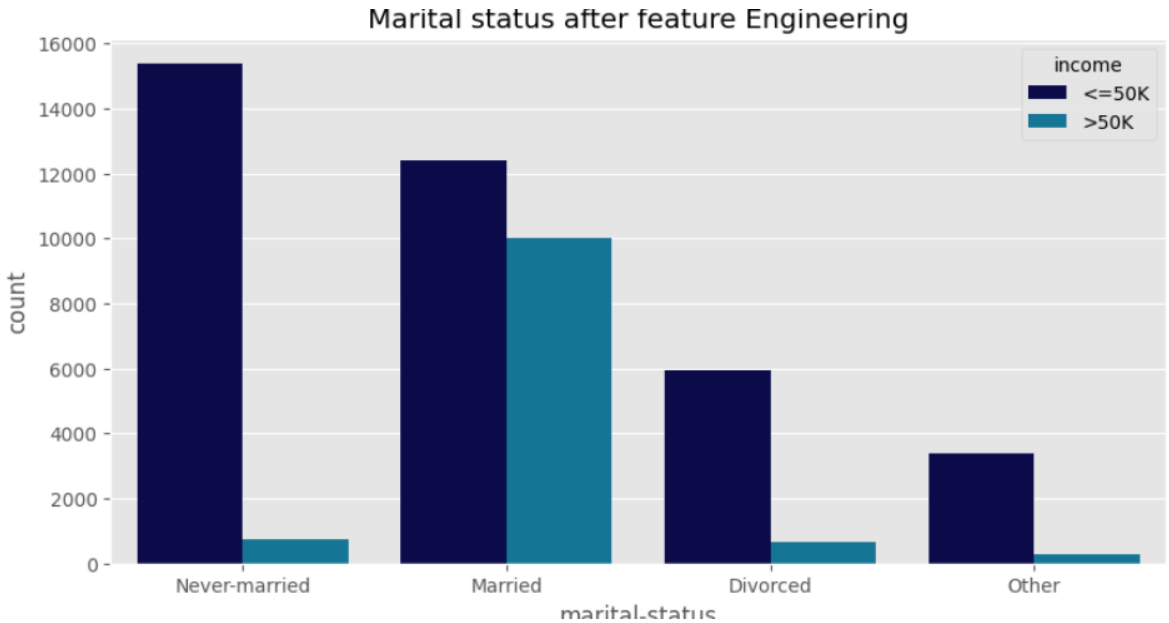
After doing that process, the workclass categories "State-gov", "Local-gov" and "Federal-gov" is renamed as government, and “Private", "Self-emp-inc", "Self-emp-not-inc" categories kept with the same category name and other categories are renamed as “other”. After doing that conversion the below bar chart shows how workclass values are distributed.



Check how each categorical data is distributed with income

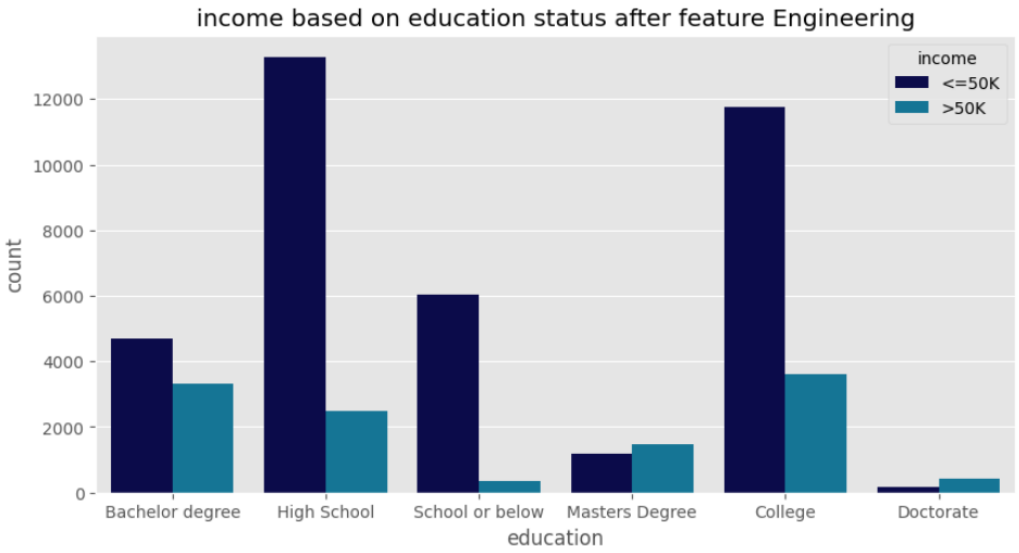
Marital status vs income

When marital status is plotted with respective to the income, Majority of people in all the marital status categories are having an income less than 50k.The below count plot shows how marital status is distributed with respective to the income.



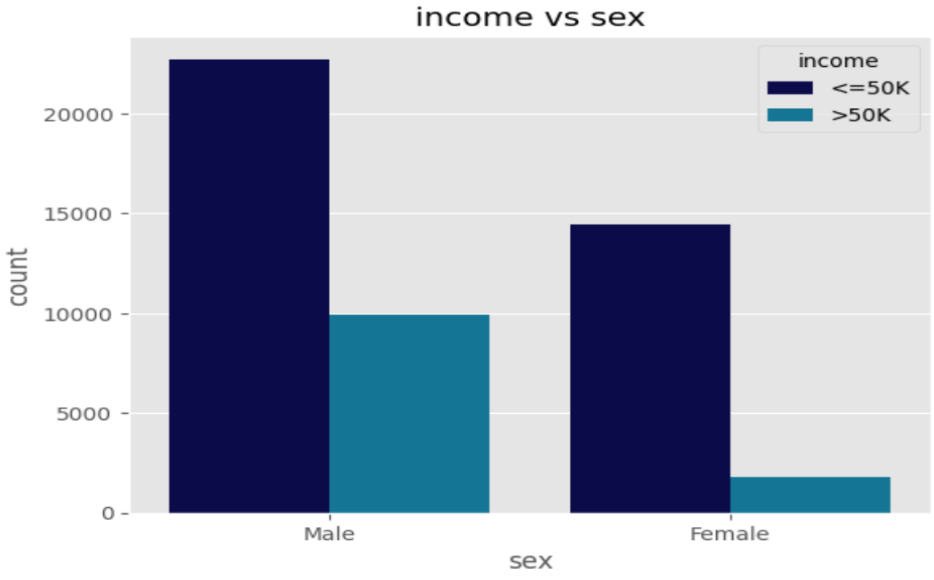
Education Status vs Income

When education status is plotted against the income, majority of people who have completed master's degree and doctorate degree have earned more than 50K, while majority of people who are from bachelor's degree, high school and school or below levels have earned a salary less than 50K. It shows that education status has impacted for their income. The below count plot shows how education status is distributed with respective to the income.



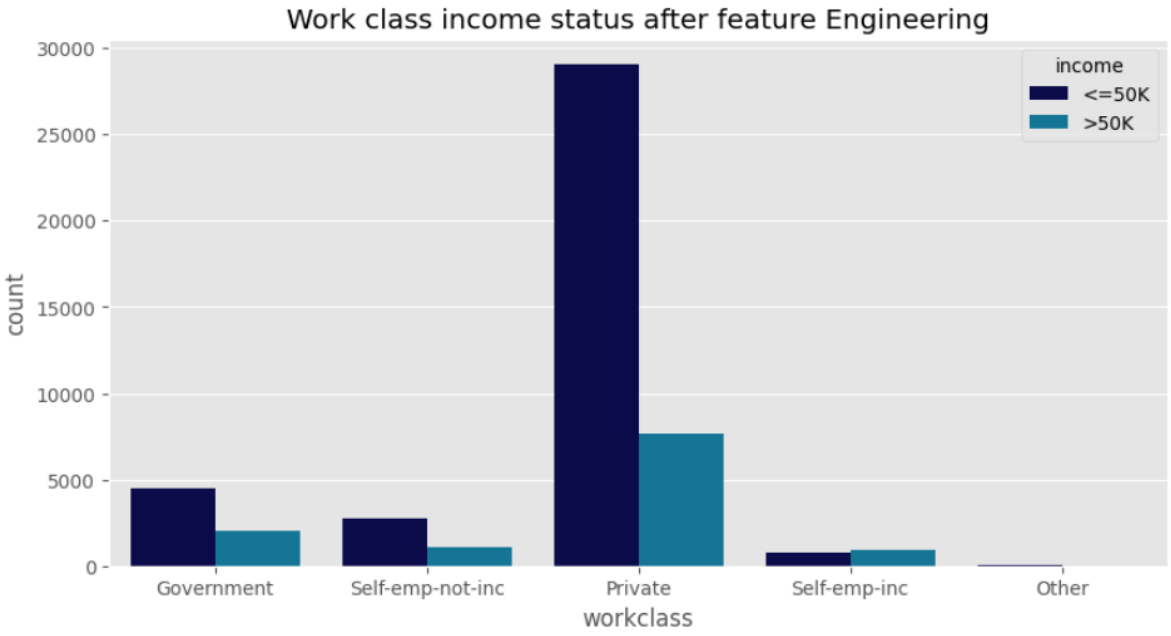
Sex status vs Income

It shows that Majority of both male and female categories have an income less than 50k compared to the people who are having an income greater than 50k. In addition, it shows that male percentage is greater the female percentage in this dataset.



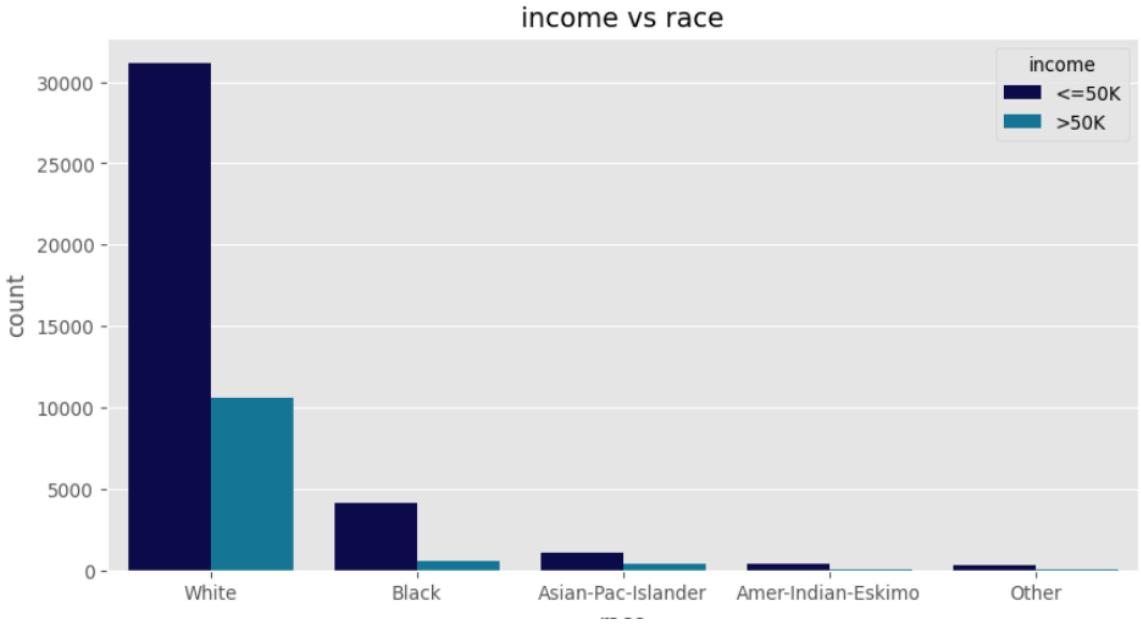
Work class Status vs Income

The below count plot shows that majority of people in all categories have an income less than 50K except the people who are doing a self-employment. It shows how work class categories are distributed respect to the income class.



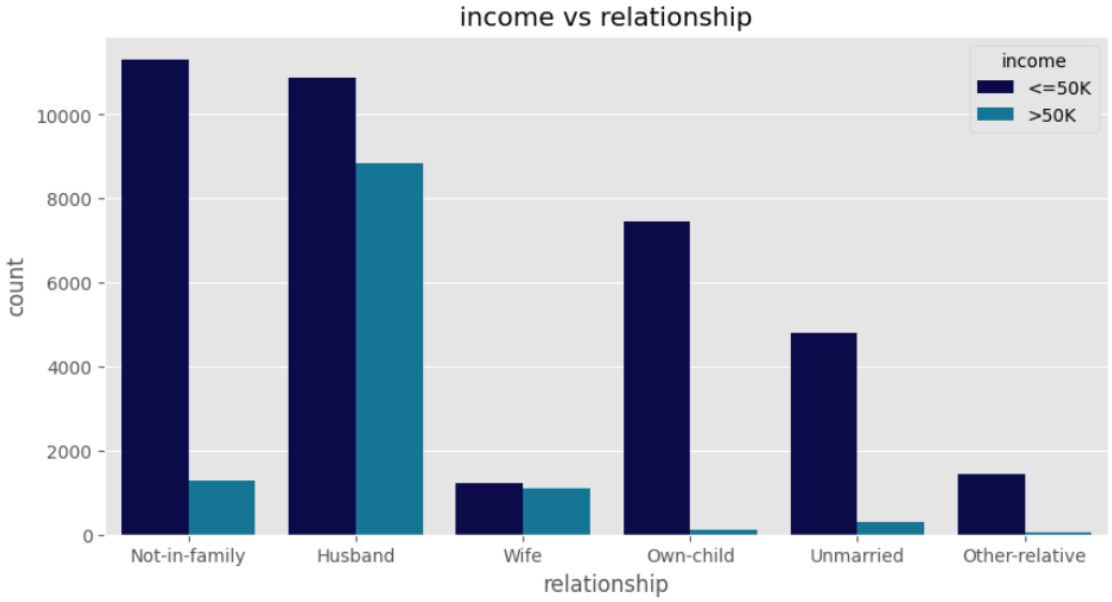
Race vs Income

Most of the people in all the races given in the dataset is having an income less than 50K and the below count plot shows that majority of them are White.



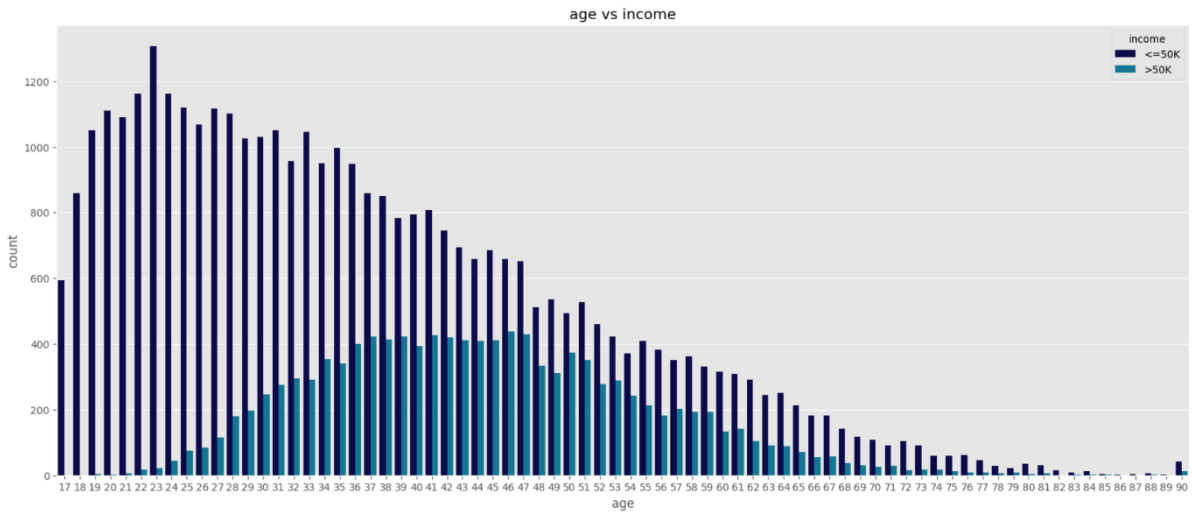
Income vs Relationship

As in the above-mentioned columns, majority of the people in all the relationship categories are having an income less than 50K. The below count plot shows how relationship vs income data is distributed.



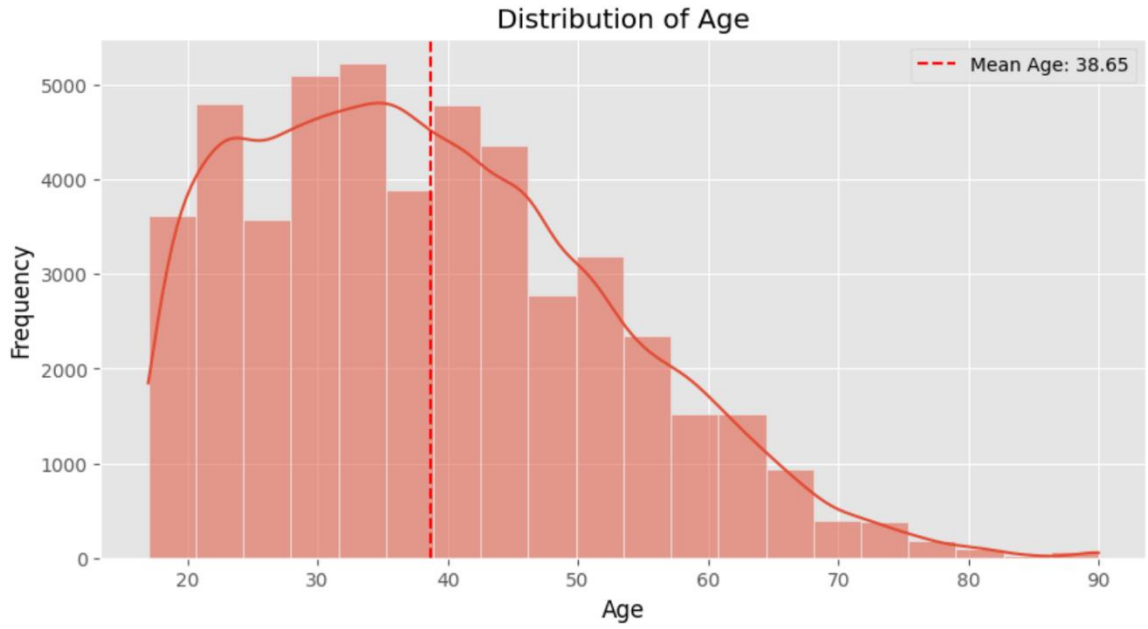
Age vs Income (Numerical data)

The below count plot (bar chart) shows that people who earn less than 50K is greater than the value count of people who earn greater than 50k.The below count plot shows that people who earn greater than 50k in all age categories is approximately distributed in a normal distribution.



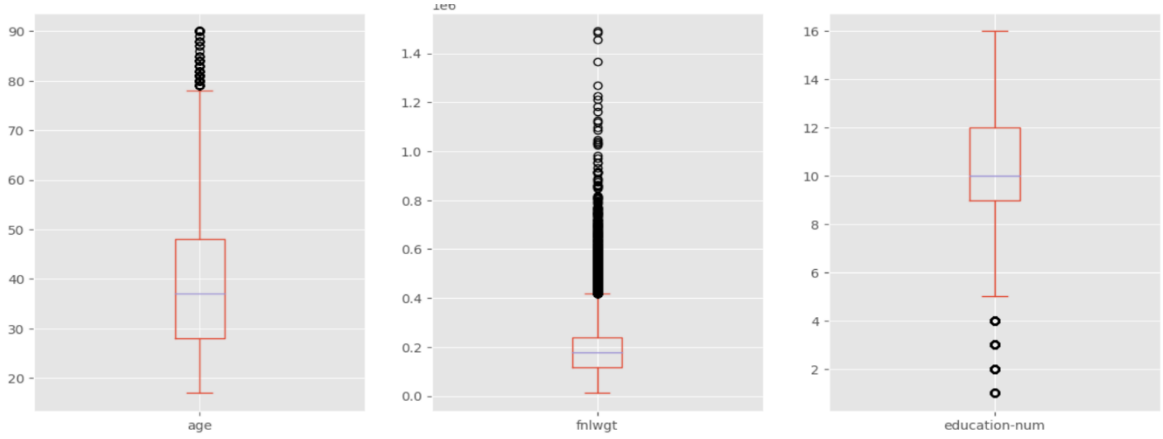
Distribution of age in the dataset

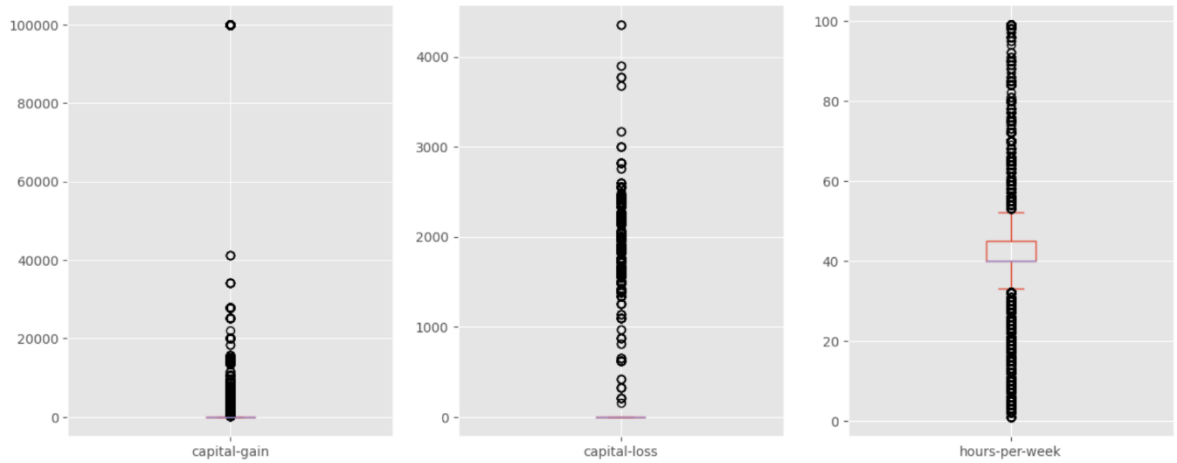
The below histogram shows that, how age of people is distributed in the dataset. It shows that mean age of people is 38.65 and majority of people are from less than 50 years old in their age.



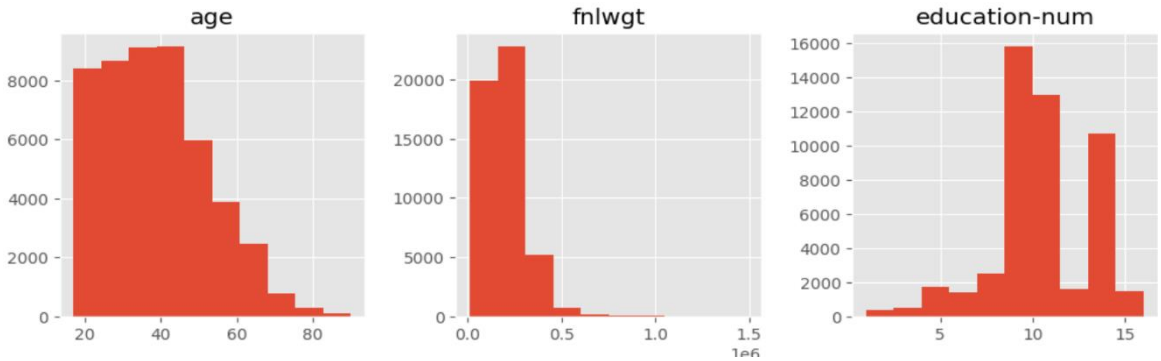
Outlier detection and check correlations in numerical values

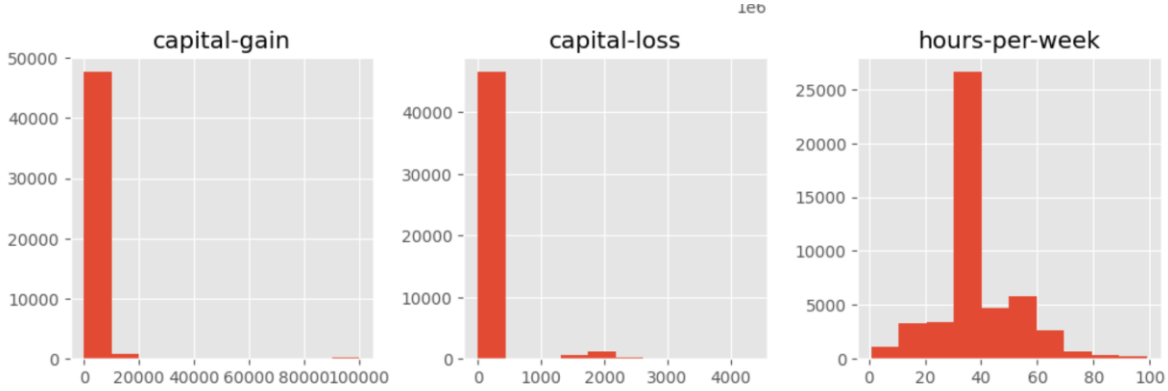
To detect the outliers of the numerical value columns in the dataset, mainly boxplots are used. However, since there is lack of information available on the metadata of the dataset, outliers cannot be handled directly by considering threshold values by looking at its quantiles. The below boxplots show how the outliers of each numerical columns are distributed.





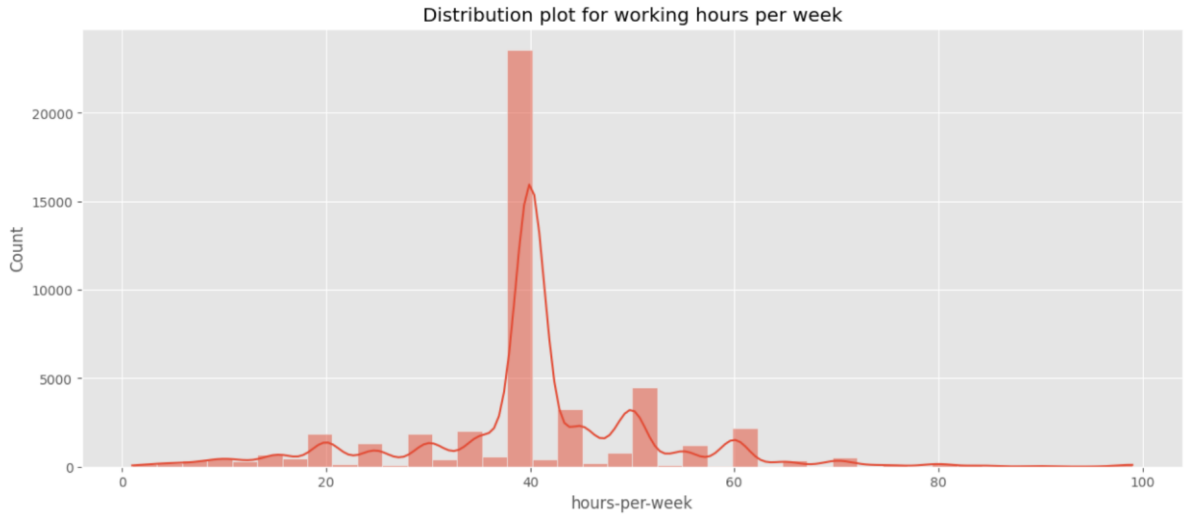
For example, this shows that people who are greater than 80 years in age, as outliers. Hower, in the general situations in well developed countries like U.S.A, there are people who work in 80's as well. (More Americans Are Working Into Their 80s. What's Keeping Them in the Workforce., 2023). Therefore, these values cannot be dropped. When the correlation of education-num is taken with the income, it shows there is a high correlation compared with other columns. In addition, most of the people in the dataset their capital gain and capital loss is 0. This is a usual situation in the general world due to, capital gain and loss is calculated when someone sells their properties. Therefore, it is difficult to drop the data in capital gain and capital loss columns without having much knowledge in metadata. The hours-per week column is distributed in the range of 0 to 99. However, in government and private sectors, it is very rare to find people who are having working hours less than 10 hours. These values have been dropped from the initial dataset. The below histograms show that how each numerical data columns are distributed.





Methods used to drop the abnormal values in the dataset

The below histogram shows how working hours for each people is distributed in the dataset. It depicts that majority of people are working around forty hours per week.



However, there were some people who have over worked with their age limits. When adults over the age of 65 who have worked more than 70 hours per week were checked, there were 26 records.

Check how many people who have over worked after 60

```
# people who have over worked after age 60
data[(data["age"] > 65) & (data["hours-per-week"] > 70)].shape
```

```
(26, 15)
```

As mentioned above, when people who have worked less than 10 hours per week in government and private sector is checked, there were 604 records. Then the dataset is filtered to clear the abnormal values detected in hours per week column to clear outliers.

people who work less than 10 hours a week in private and government

```
data[(data["hours-per-week"]  < 10) & ((data["workclass"].isin(["Government", "Private"])))].shape
```

```
(604, 15)
```

remove data working hours per week less than 10 and work class in (Government or private) workclasses

```
data = data[~((data["hours-per-week"]  < 10) & ((data["workclass"].isin(["Government", "Private"]))))]
```

Then, checked whether there are people who have married less than 18 years. There were records on that, but those records were not dropped due to all the filtered people are from United States and it is legal in some states for people who are less than 18 years and over 16 years to have a married life under some legal conditions.

Check people who are married less than 18 years old in age

```
# Can't remove this, due to USA allows marriage in 17 with certain laws
data[(data["age"] < 18) & (data["marital-status"] != "Never-married")]
```

age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	in
17	Private	221129	School or below	5	Married	Other-service	Husband	White	Male	0	0	40	United-States	
17	Private	186890	School or below	6	Married	Sales	Own-child	White	Female	0	0	30	United-States	
17	Private	27251	School or below	7	Other	Prof-specialty	Own-child	White	Male	0	0	40	United-States	
17	Private	364952	School or below	6	Other	Other-service	Other-relative	White	Male	0	0	40	United-States	

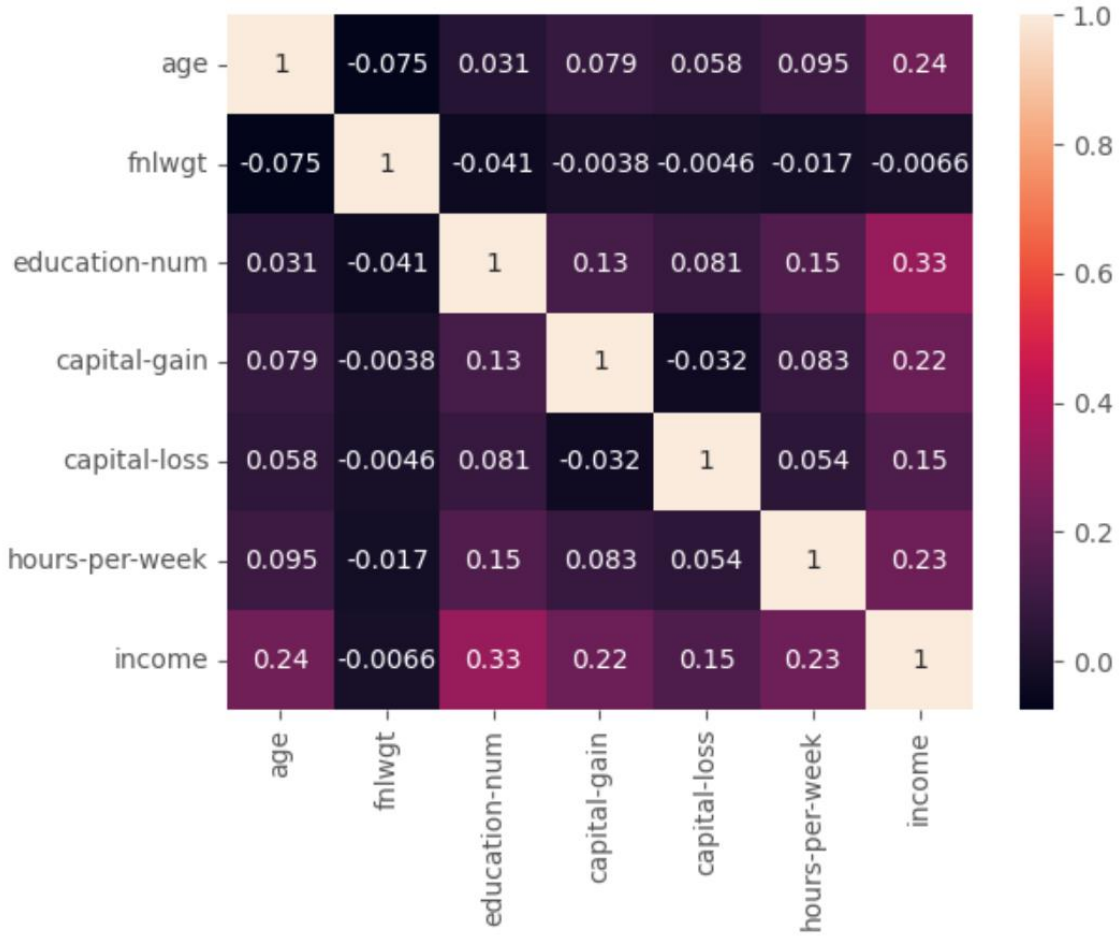
The below value counts show the unique value counts under each column after handling the outliers.

```
age                74
workclass           5
fnlwgt          28153
education           6
education-num      16
marital-status      4
occupation         14
relationship        6
race                5
sex                 2
capital-gain      123
capital-loss       99
hours-per-week     96
native-country      2
income              2
dtype: int64
```

Seaborn pair plot that used to check whether there is a correlation of numerical value-based columns

Then income value is converted to numerical value format to check the correlation values of numerical columns with income column. The below table and heatmap shows how each numerical column is correlated with each other according to Pearson correlation.

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	income
age	1.000000	-0.075415	0.030768	0.079239	0.058053	0.095045	0.236409
fnlwgt	-0.075415	1.000000	-0.040636	-0.003764	-0.004644	-0.017052	-0.006589
education-num	0.030768	-0.040636	1.000000	0.125399	0.080510	0.147669	0.332881
capital-gain	0.079239	-0.003764	0.125399	1.000000	-0.031609	0.082933	0.222800
capital-loss	0.058053	-0.004644	0.080510	-0.031609	1.000000	0.054073	0.147058
hours-per-week	0.095045	-0.017052	0.147669	0.082933	0.054073	1.000000	0.228620
income	0.236409	-0.006589	0.332881	0.222800	0.147058	0.228620	1.000000

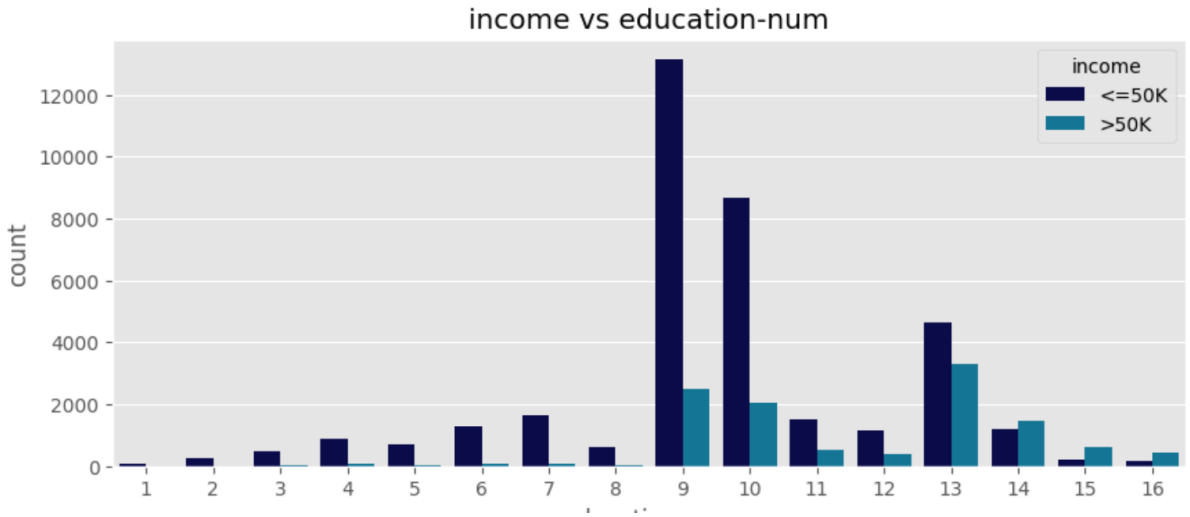


The above heatmap shows there are slight correlations between income and other columns except the fnlwgt column. The fnlwgt column has a very low correlation with the income column (-0.0066). Therefore, it can be dropped before splitting the data into training and testing columns. The highest correlation value with income column is from education-num. Therefore, it shows that education has impacted a lot to the income status.

Analysing numerical Data with income

Education-num vas Income

When education-num plotted with respect to the income, majority of people who have an education-num less than or equals to 14 have got an income less than or equal to 50k. However, people who have education-num greater than or equal to 14, majority of people have got salaries above 50k. Here, it shows that education-num has impacted for income rates. The below seaborn count plot shows how education-num is distributed with respect to the income.

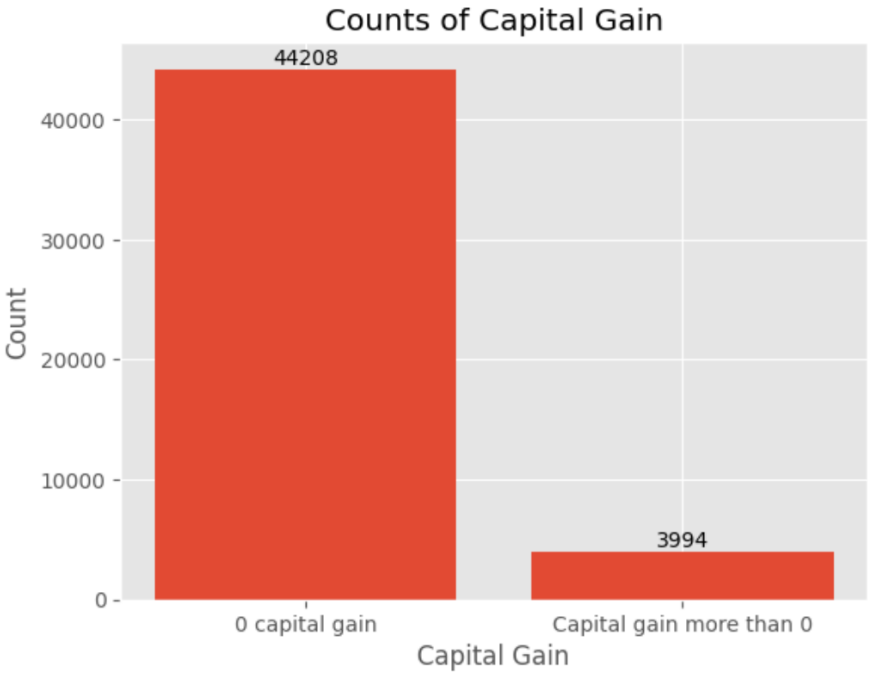


The below table shows how the value counts of each education-num is distributed respect to the income. The zero, represents people who have a salary less than or equals 50k and 1 represents people who have a salary greater than 50k.

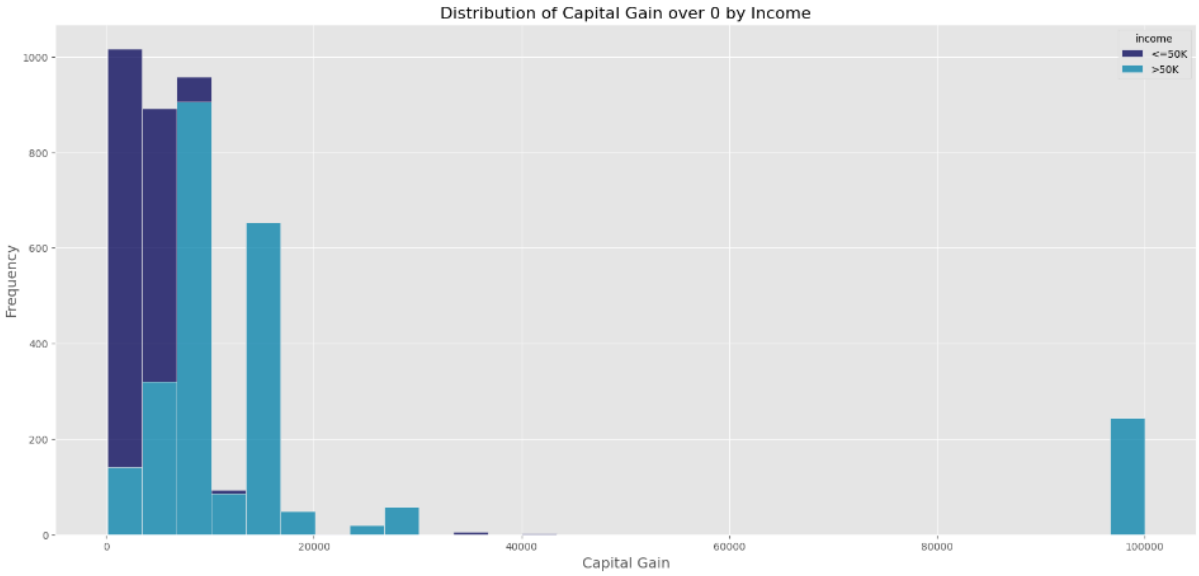
education-num	income	
1	0	82
	1	1
2	0	237
	1	8
3	0	475
	1	27
4	0	867
	1	61
5	0	702
	1	41
6	0	1269
	1	87
7	0	1655
	1	88
8	0	597
	1	48
9	0	13129
	1	2494
10	0	8692
	1	2056
11	0	1521
	1	521
12	0	1163
	1	411
13	0	4636
	1	3297
14	1	1446
	0	1179
15	1	614
	0	212
16	1	425
	0	161

Capital-gain vs Income

Capital-gain is a calculation that is done when people sell their properties. If not, capital gain will remain 0. If a person can sell a property than the value that person is expected, then a capital gain is added to that person. The below bar chart shows that more than 44200 records consist with a capital gain as zero. That usually happens in the real world, due to there are lack of people who are selling their properties.



The below histogram shows how capital gain greater than 0 is distributed with income. In addition, when people who have a capital gain greater than 80000 and an income less than 50K is checked, there were no any records found. Therefore, people who have a capital gain greater than 80000K cannot be considered as outliers to drop them. The below histogram shows, how capital gain greater than 0 is distributed with income.



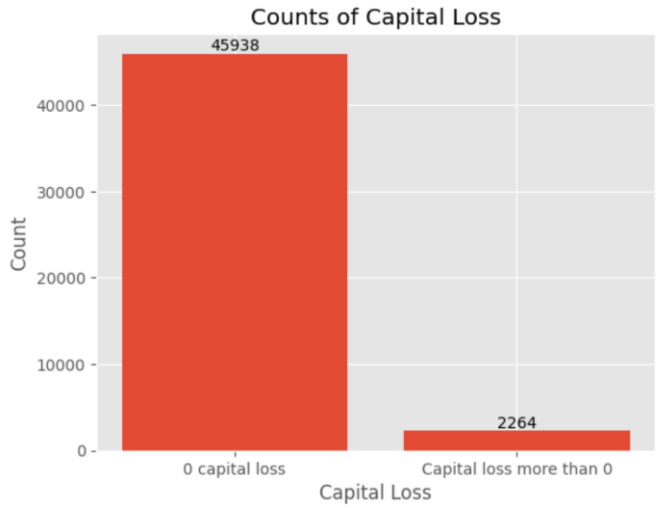
This shows statistical based calculations done for capital gain

```
data["capital-gain"].describe()
```

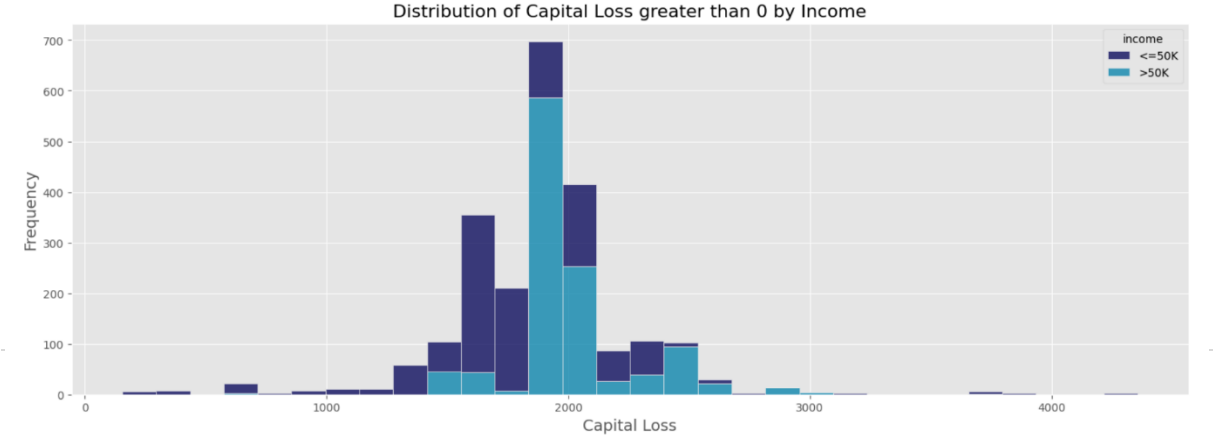
```
count    48202.000000
mean      1088.140886
std       7496.261617
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      99999.000000
Name: capital-gain, dtype: float64
```

Capital-loss vs Income

Capital-loss is also a calculation that is done when a person sells a property and if that person is not able to gain the expected amount by selling their properties. The majority of people with approximately 46000 records have a capital loss with 0. That cannot be considered as an outlier due to very few people are selling their properties. The below bar chart shows how people with zero capital loss and more than zero capital loss is distributed.



The below histogram shows how Capital Loss greater than 0 is distributed with income rates. However, it shows that people who have income rates both grater and less than 50k, are having a capital loss.



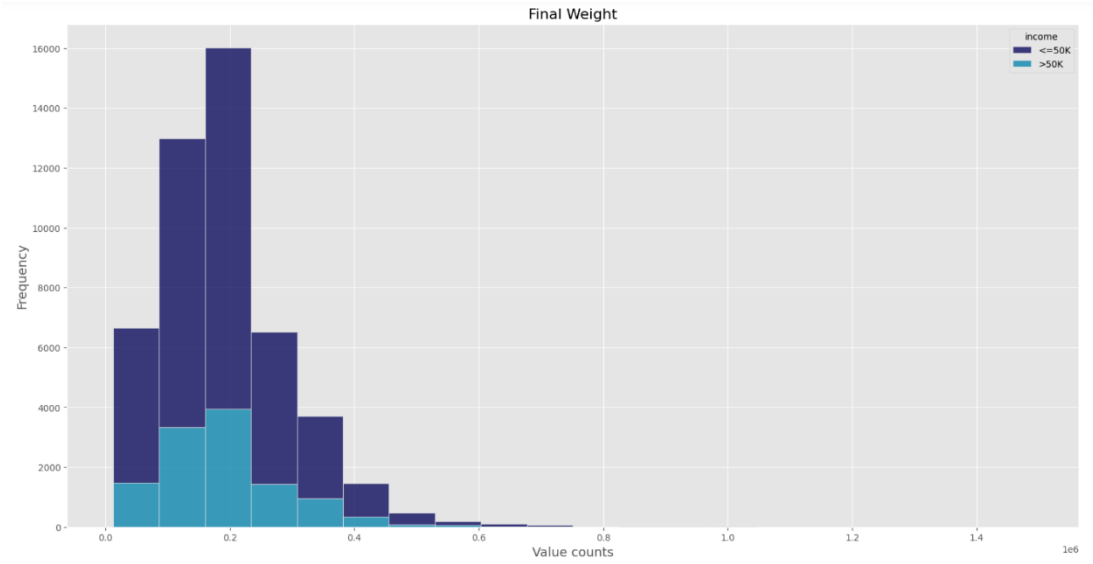
This shows the statistical calculations done for capital loss.

```
: data["capital-loss"].describe()
```

```
: count    48202.000000
  mean        87.977159
  std        404.019342
  min          0.000000
  25%          0.000000
  50%          0.000000
  75%          0.000000
  max       4356.000000
```

fnlwgt vs Income

As mentioned in the introduction, final weight is a calculation that is done based on census data. It is difficult to do an analysis for final weight without having proper metadata. The below histogram shows how final weight data is distributed.



The below table shows some statistical calculations done for final weight

```
: data["fnlwgt"].describe().astype("int32")
```

```
: count      48202
  mean      189814
  std       105532
  min        12285
  25%       117606
  50%       178353
  75%       237868
  max      1490400
```

Check duplicate values of the dataset

Before training the models, it is an important step to drop the duplicated values in the dataset. When duplicated values checked after doing the outlier detection and feature engineering process, there were 54 duplicated rows in the data set. These duplicated rows were dropped.

Check for the duplicated values in the dataset

```
duplicates = data[data.duplicated(keep='first')]
```

check how many rows duplicated

```
duplicates.shape
```

```
(54, 15)
```

Drop duplicates from the dataset

```
data.drop_duplicates(inplace=True, keep='first')
```

Get the shape of the dataset after removing data

```
data.shape
```

```
(48148, 15)
```

Label Encoding Process

For the label encoding process, all the columns that consist of categorical data are encoded. Therefore, columns “workclass”, “education”, “marital-status”, “occupation”, “relationship”, “race”, “sex” and “native-country” are encoded in the label encoding process.

Encode labels in columns that consist with categorical data

```
from sklearn.preprocessing import LabelEncoder

final_data = data.copy()

encoders = {}

for column in final_data.columns:
    if final_data[column].dtype == 'object':
        encoder = LabelEncoder()
        final_data[column] = encoder.fit_transform(final_data[column])
        encoders[column] = encoder

final_data.head()
```

The below table shows how the values look like after label encoding the process.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	39	0	77516	0	13	2	0	1	4	1	2174	0	40	1
1	50	4	83311	0	13	1	3	0	4	1	0	0	13	1
2	38	2	215646	3	9	0	5	1	4	1	0	0	40	1
3	53	2	234721	5	7	1	5	0	2	1	0	0	40	1
4	28	2	338409	0	13	1	9	5	2	0	0	0	40	0

The below dictionaries show how each column related categories are encoded with its original labels and the encoded values.

```
workclass:
{0: 'Government', 1: 'Other', 2: 'Private', 3: 'Self-emp-inc', 4: 'Self-emp-not-inc'}
education:
{0: 'Bachelor degree', 1: 'College', 2: 'Doctorate', 3: 'High School', 4: 'Masters Degree', 5: 'School or below'}
marital-status:
{0: 'Divorced', 1: 'Married', 2: 'Never-married', 3: 'Other'}
occupation:
{0: 'Adm-clerical', 1: 'Armed-Forces', 2: 'Craft-repair', 3: 'Exec-managerial', 4: 'Farming-fishing', 5: 'Handlers-cleaners',
6: 'Machine-op-inspct', 7: 'Other-service', 8: 'Priv-house-serv', 9: 'Prof-specialty', 10: 'Protective-serv', 11: 'Sales', 12:
'Tech-support', 13: 'Transport-moving'}
relationship:
{0: 'Husband', 1: 'Not-in-family', 2: 'Other-relative', 3: 'Own-child', 4: 'Unmarried', 5: 'Wife'}
race:
{0: 'Amer-Indian-Eskimo', 1: 'Asian-Pac-Islander', 2: 'Black', 3: 'Other', 4: 'White'}
sex:
{0: 'Female', 1: 'Male'}
native-country:
{0: 'Other', 1: 'United-States'}
income:
{0: '<=50K', 1: '>50K'}
```

Apply Standardization for final weight column

Since values in the final weight column ranges from 12285 – 1490400, standardization technique is used to scale the data into a lower range in the final weight column. The below image shows how the standardization process is done and the statistical calculations done to calculate final weights after standardization process.

Put Standard Scaler to fnlwgt since it has values ranging in (12285 - 1490400)

```
from sklearn.preprocessing import StandardScaler
```

Standard Scaler formula: $z = \frac{x - \mu}{\sigma}$

```
scaler = StandardScaler()
final_data["fnlwgt"] = scaler.fit_transform(final_data[["fnlwgt"]])
```

Display final weight after fitting standard scaler

```
final_data["fnlwgt"].describe().astype("float64")
```

```
count    4.814800e+04
mean     4.604331e-17
std      1.000010e+00
min     -1.681979e+00
25%     -6.841415e-01
50%     -1.087229e-01
75%      4.549951e-01
max      1.232205e+01
Name: fnlwgt, dtype: float64
```

Apply Normalization for Capital gain and Capital loss columns

Since capital gain and capital loss values are spread in a wide range, the columns capital-gain and capital-loss have to be normalized or standardized. Capital gain column is spread in a range of 0 to 99999 and Capital loss column is spread in a range of 0 to 48148. However, most of the capital gain and capital loss values are 0. Therefore, Normalization technique is used to scale down data between 0 to 1. The below image shows how the normalization applied for both capital gain and capital loss columns.

Apply Normalization for capital gain

describe data in capital gain before normalizing

```
In [149]: final_data["capital-gain"].describe()
```

```
Out[149]: count    48148.000000
          mean      1089.361282
          std       7500.375585
          min          0.000000
          25%          0.000000
          50%          0.000000
          75%          0.000000
          max      99999.000000
          Name: capital-gain, dtype: float64
```

Apply the normalizer for capital-gain

```
In [150]: capital_normalizer = MinMaxScaler()
          final_data["capital-gain"] = capital_normalizer.fit_transform(final_data[["capital-gain"]])
```

Get the normalized value for example capital gain

```
In [151]: capital_normalizer.data_max_, capital_normalizer.data_min_
```

```
Out[151]: (array([99999.]), array([0.]))
```

Apply Normalization for capital loss

Put Normalizer for capital loss

```
from sklearn.preprocessing import MinMaxScaler
```

describe data in capital loss before normalizing

```
final_data["capital-loss"].describe()
```

```
count    48148.000000
mean        88.075829
std        404.235096
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       4356.000000
Name: capital-loss, dtype: float64
```

Apply the normalizer for capital-loss

```
capital_loss_normalizer = MinMaxScaler()
final_data["capital-loss"] = capital_loss_normalizer.fit_transform(final_data[["capital-loss"]])
```

```
capital_loss_normalizer.data_max_ , capital_loss_normalizer.data_min_
```

```
(array([4356.]), array([0.]))
```

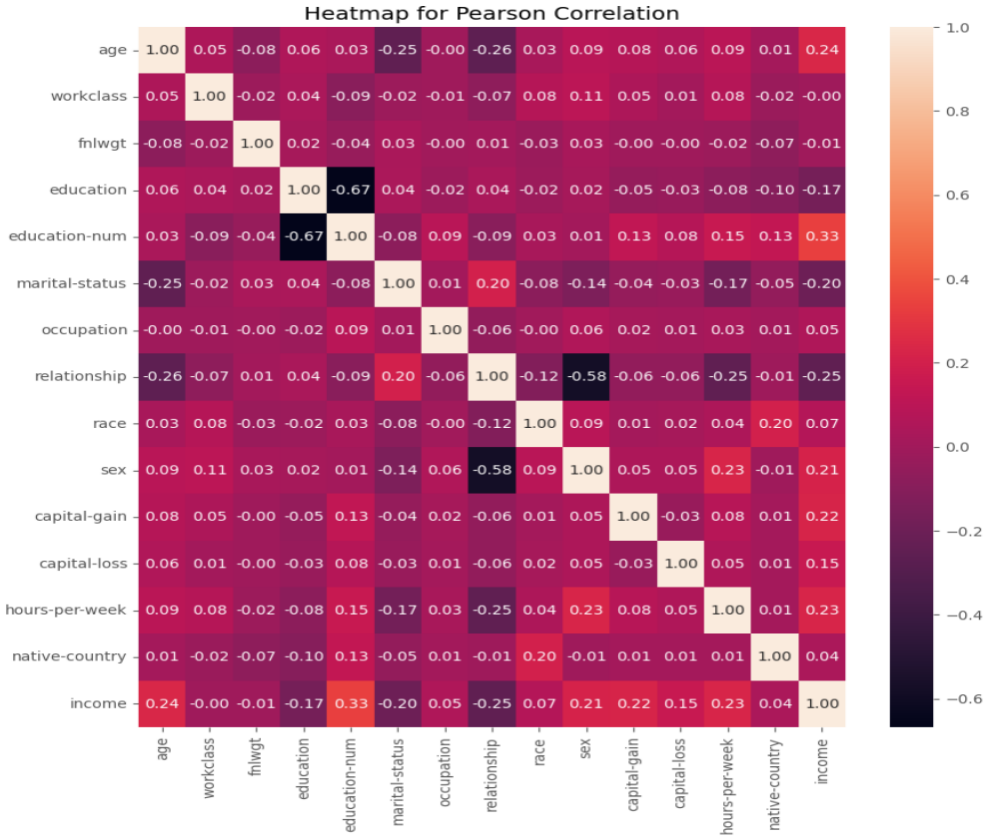
Check correlations of columns for feature selection

The correlation of the columns is checked with the income column to check whether how each column is correlated and to choose the best columns to train the model. The below image shows how each column is correlated with the income column.

```
cor = final_data.corr(numeric_only=True, method="pearson")
cor["income"]
```

```
age               0.236172
workclass        -0.003656
fnlwgt           -0.006542
education        -0.172500
education-num     0.333051
marital-status   -0.201073
occupation        0.053403
relationship     -0.254417
race              0.071065
sex               0.214371
capital-gain      0.222832
capital-loss      0.147040
hours-per-week    0.228572
native-country    0.037483
income            1.000000
Name: income, dtype: float64
```

The below heatmap shows how each column is correlated with rounded two decimal points.



This heatmap shows that work class and final weight columns have a very low correlation with the income compared to other columns. Therefore, work class and final weight columns can be

dropped before training the models. In addition, there is a good correlation between “education” and “education-num” columns. Therefore, one of the columns can be dropped by keeping the highest correlated column with the income. Therefore, “education” column can be dropped. When the correlation with relationship and sex columns are checked, there is a high correlation compared to other columns. The same concept can be applied to the relationship and sex columns, and the sex column can be dropped since it has a low correlation with the income column compared to the relationship column.’

Feature Selection Process

Before, selecting the features, the variable values are assigned to a NumPy array X and the target variable values income has assigned to a NumPy array Y. The below image shows how that process is done.

Section 8 : Feature Selection process

assign the input column values to X and target variable column values to Y

```
X = final_data.drop('income', axis=1)
Y = final_data['income'].map({0: "<=50K", 1: ">50K"})
```

print first five values of both X and Y

```
X.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	39	0	-1.063964	0	13	2	0	1	4	1	0.02174	0.0	40	1
1	50	4	-1.009061	0	13	1	3	0	4	1	0.00000	0.0	13	1
2	38	2	0.244714	3	9	0	5	1	4	1	0.00000	0.0	40	1
3	53	2	0.425435	5	7	1	5	0	2	1	0.00000	0.0	40	1
4	28	2	1.407802	0	13	1	9	5	2	0	0.00000	0.0	40	0

Then dropped the columns, "fnlwgt", "education", "sex", "workclass" from the initial dataset to get the X data frame that consist with the variables that only used for training. The below image shows how that process is done.

Drop columns based on low correlations and similar correlation patterns with other columns

```
X.drop(columns=["fnlwgt", "education", "sex", "workclass"], inplace=True)
```

Display first five column of variable dataset after feature reduction

```
X.head()
```

	age	education-num	marital-status	occupation	relationship	race	capital-gain	capital-loss	hours-per-week	native-country
0	39	13	2	0	1	4	0.02174	0.0	40	1
1	50	13	1	3	0	4	0.00000	0.0	13	1
2	38	9	0	5	1	4	0.00000	0.0	40	1
3	53	7	1	5	0	2	0.00000	0.0	40	1
4	28	13	1	9	5	2	0.00000	0.0	40	0

Train some base models and see how it works before imbalanced learning process

Base models (Random Forest classifier and Naïve Bayes) are trained to check whether how they perform before doing the imbalanced learning process. Therefore, dataset is split into train and test by giving a ratio 0.3 for the testing dataset.

split the dataset

```
from sklearn.model_selection import train_test_split

train_X, test_X, train_Y, test_Y = train_test_split(X, Y, test_size=0.3, random_state=42)
```

```
train_X.shape, train_Y.shape, test_X.shape, test_Y.shape
```

```
((33703, 10), (33703,), (14445, 10), (14445,))
```

Base Model built with Random Forest classifier

The below image shows the model created for random forest classifier and it shows that mode l is overfitting. (Training accuracy - 0.9612497403791948, Testing accuracy - 0.84742125302 87296). Therefore, it is trained again by doing a parameterized optimization with randomized search cv.

Apply random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

model_random_forest = RandomForestClassifier(random_state=42)

model_random_forest.fit(train_X, train_Y)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Get the training accuracy

```
model_random_forest.score(train_X, train_Y)
```

```
0.9612497403791948
```

Get the testing Accuracy

```
from sklearn.metrics import accuracy_score

accuracy_score(test_Y, model_random_forest.predict(test_X))
```

```
0.8474212530287296
```

The below image shows how parameters are tuned in the random forest classifier model with tuned parameters. These parameters changes time to time, when it runs due to these values are generated randomly. Number of estimators, max features, min sample split and min sample leaf are the values that are optimized through the process.

Check whether it works properly by doing a parameterized optimization

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

np.random.seed(20)

# parameter grid
param_grid = {
    'n_estimators': list(range(5, 50)),
    'max_features': ['log2', 'auto', 'sqrt'],
    'min_samples_split': list(range(6, 15)),
    'min_samples_leaf': list(range(4, 15))
}

# Create a RandomForestClassifier instance
rf_classifier = RandomForestClassifier(n_jobs=5)

# Create RandomizedSearchCV instance
random_search = RandomizedSearchCV(estimator=rf_classifier, param_distributions=param_grid, n_iter=50, cv=5, n_jobs=5,
                                   random_state=42)

# Fit the model
random_search.fit(train_X, train_Y)

# Print best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

Training model with the tuned parameters

The below image shows how the random forest classifier model is trained with the tuned param eters. There was a training accuracy of 0.87185 and a testing accuracy of 0.8634. It shows that the overfitting issue of the model is solved when it is trained with changing the default values of its parameters.

get training accuracy

```
model_random_forest.score(train_X, train_Y)
```

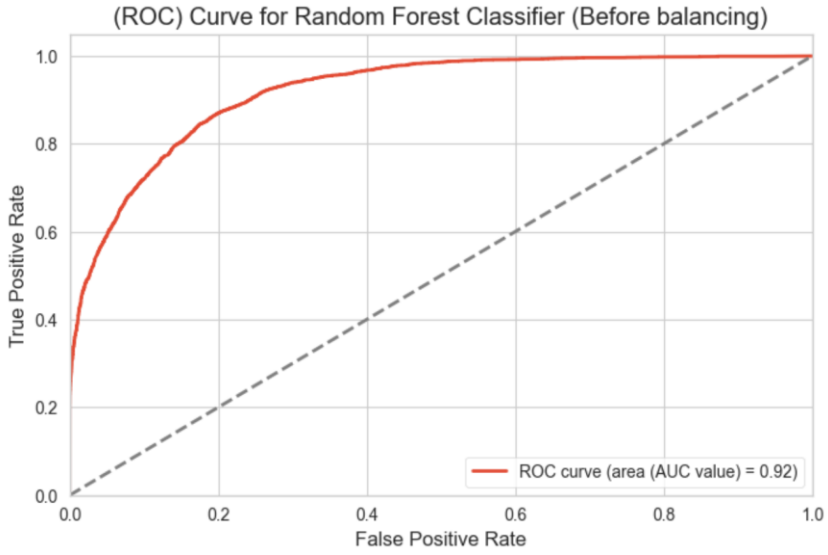
```
0.8718511705189449
```

get testing accuracy

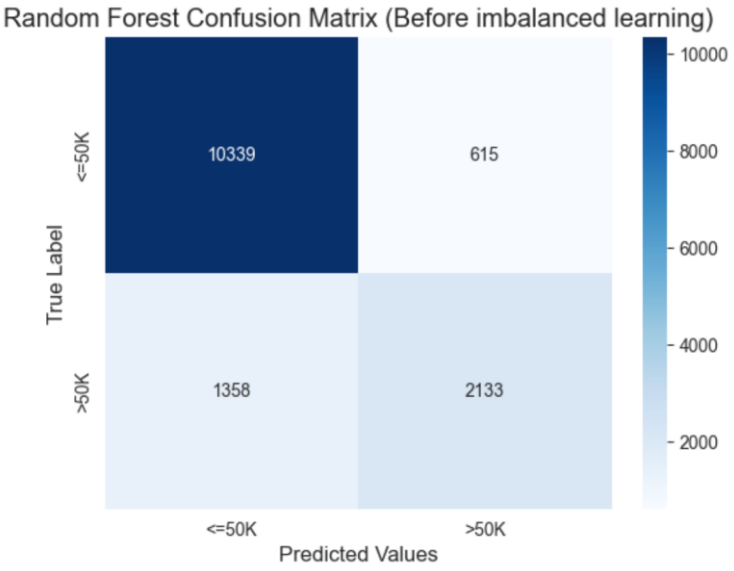
```
from sklearn.metrics import accuracy_score

accuracy_score(test_Y, model_random_forest.predict(test_X))
```

```
0.8634129456559363
```

When the ROC curve is plotted, there was an AUC value of 0.92. The below image shows how the ROC curve is plotted. However, AUC score 0.92 is a good score.



However, the confusion matrix is displayed with the help of seaborn heatmaps, and the classification report is generated. It shows that, comparatively, there is a low score for precision, recall, and f1-scores for income greater than 50k when compared with income less than or equal to 50k. It shows that it happens because the dataset is not properly balanced. The below images show the confusion matrix and the classification report for the random forest classification model when the dataset is imbalanced.



	<=50K	>50K	accuracy	macro avg	weighted avg
precision	0.883902	0.776201	0.863413	0.830051	0.857873
recall	0.943856	0.611000	0.863413	0.777428	0.863413
f1-score	0.912896	0.683763	0.863413	0.798330	0.857520
support	10954.000000	3491.000000	0.863413	14445.000000	14445.000000

Base Model built with Bernoulli Naïve Bayes

Before doing the imbalanced learning, naïve bayes model is trained using Bernoulli naïve bayes due to scikit learn official documentation shows that Bernoulli naïve bayes performs well compared to two other naïve bayes models (Multinomial naïve bayes and gaussian naïve bayes). When it is trained before doing the hyper parameterized optimization it has got a training accuracy of 0.79408 and testing accuracy of 0.7939.

Apply Bernouli Naive Bayes Classifier

```
from sklearn.naive_bayes import BernoulliNB
```

```
bernoulli_model = BernoulliNB()
```

```
bernoulli_model.fit(train_X, train_Y)
```

```
BernoulliNB()
```

get training accuracy

```
bernoulli_model.score(train_X, train_Y)
```

```
0.7940836127347713
```

get testing accuracy

```
from sklearn.metrics import accuracy_score

accuracy_score(test_Y, bernoulli_model.predict(test_X))
```

```
0.7939771547248182
```

The below image shows how the hyper parameterized optimization process is done for the Bernoulli naïve bayes to get a good accuracy by using grid search cv. However, compared to other models, Naïve Bayes-related algorithms consist of very few parameters to tune. The parameters that are used in the naïve bayes are "alpha","binarize","fit_prior" and"class_prior". When the parameters are optimized, there is a slight improvement in the accuracy levels, with a training accuracy of 0.79408 and a testing accuracy of 0.7939.

Check the model after parameterized optimization for Naive Bayes ¶

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import GridSearchCV

clf = BernoulliNB()

# Define the parameter grid to search
param_grid = {
              'alpha': [0.1, 0.5, 1.0, 2.0],  # Smoothing parameter
              'binarize': [0.0, 0.5, 1.0],  # Binarization threshold
              'fit_prior': [True, False],  # Whether to learn class prior probabilities
               'class_prior': [None, [0.5, 0.5]],  # Prior probabilities for the classes
             }

# Perform grid search with cross-validation
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(train_X, train_Y)

# Print the best parameters found
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'alpha': 0.1, 'binarize': 0.0, 'class_prior': None, 'fit_prior': True}
```

The below image shows the model created after parameters optimization for Bernoulli naïve bayes model when the data set is imbalanced.

```
bernoulli_model = BernoulliNB()
bernoulli_model.fit(train_X, train_Y)
```

```
BernoulliNB()
```

get training accuracy

```
bernoulli_model.score(train_X, train_Y)
```

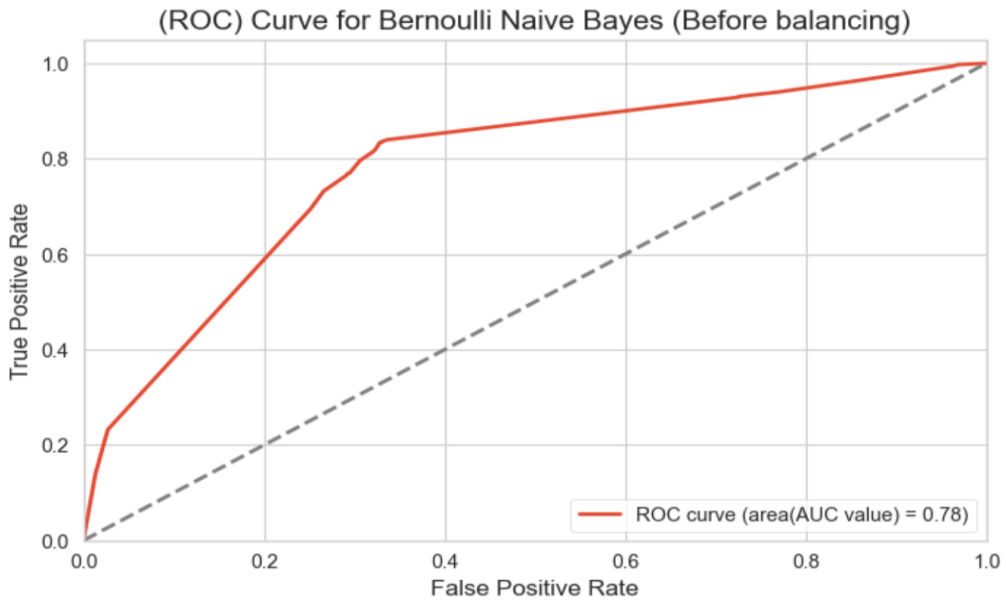
```
0.7940836127347713
```

get testing accuaracy

```
from sklearn.metrics import accuracy_score

accuracy_score(test_Y, bernoulli_model.predict(test_X))
```

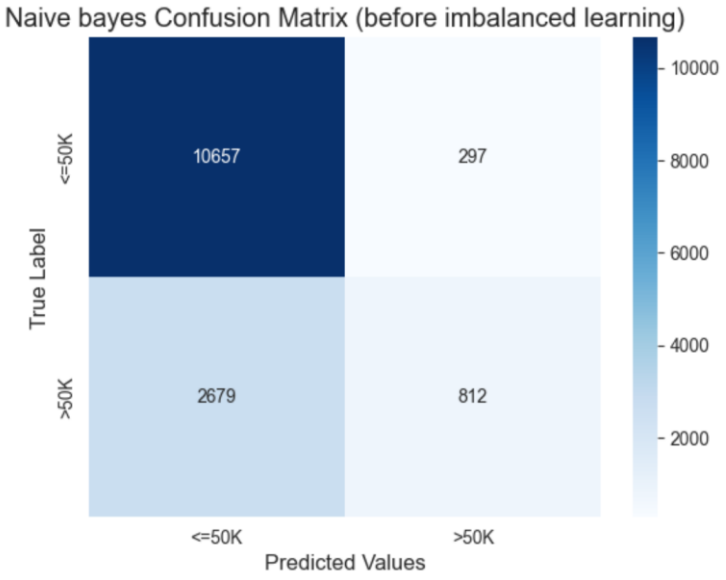
```
0.7939771547248182
```

When the ROC curve is plotted for the above Naïve Bayes model, it has an AUC value of 0.78. It is a good AUC value for the Naïve Bayes algorithm since, in general situations, it does not perform well compared to Random Forest Classifier algorithms when working with large datasets. The below image shows the ROC curve and its AUC value got for the above Bernoulli naïve bayes model.



However, when the confusion matrix and the classification report are generated, it shows that there is a high probability of identifying the class that is greater than 50k income as less than or equal to 50k income. In addition, there is a very low recall and F1-score for the class that has an income greater than 50k compared to the class that has an income less than or equals to

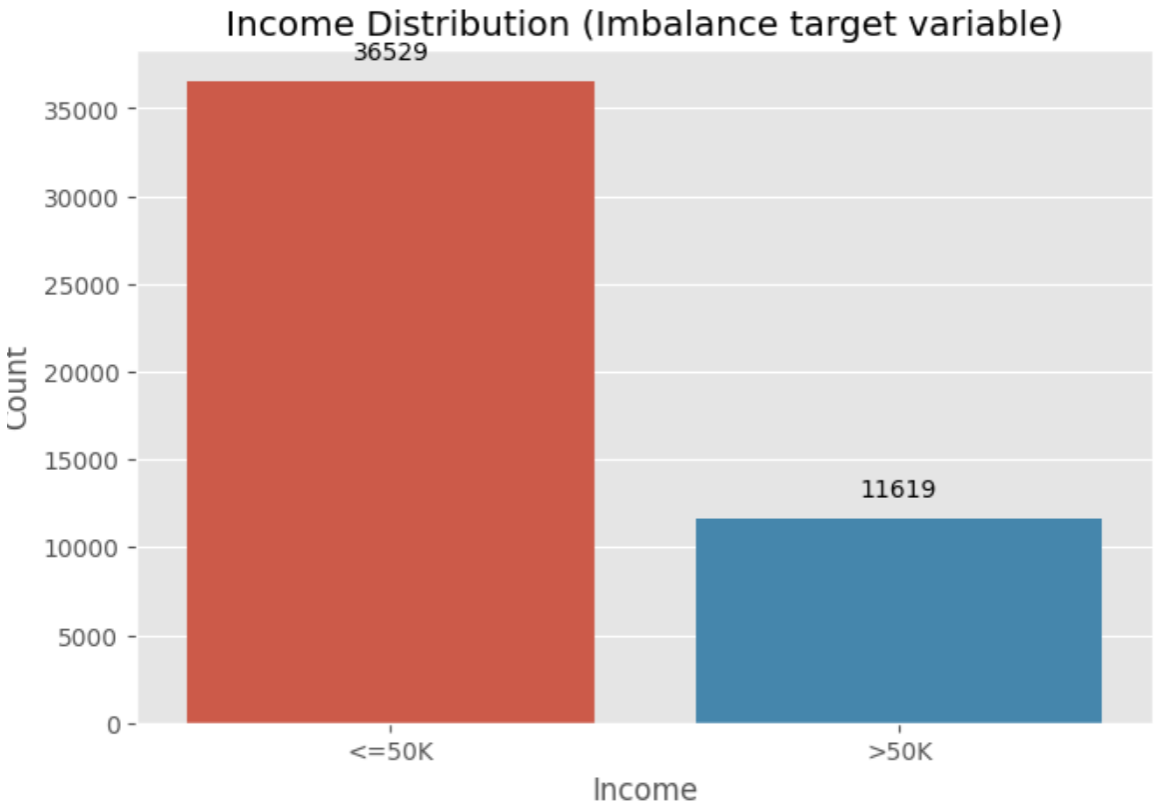
50k. Therefore, it shows that target variable of the dataset has to be balanced before training the models.



	<=50K	>50K	accuracy	macro avg	weighted avg
precision	0.799115	0.732191	0.793977	0.765653	0.782941
recall	0.972887	0.232598	0.793977	0.602742	0.793977
f1-score	0.877480	0.353043	0.793977	0.615262	0.750737
support	10954.000000	3491.000000	0.793977	14445.000000	14445.000000

Imbalanced Learning

The below bar chart shows that the target variable of the dataset is imbalanced. Income less than or equals to 50k class has 36529 records while income greater than 50k class has 11619 records. It has to be balanced to get better accuracies.



In machine learning, there are mainly two techniques used to balance datasets when working with textual data. They are, namely, undersampling and oversampling. Most of the time, oversampling performs better than undersampling techniques. In this problem, the oversampling method SMOTE is used to balance the dataset. SMOTE is an oversampling technique that works by generating synthetic examples of the minority class by interpolating between existing minority class examples. The below image shows how the SMOTE technique is used to handle the imbalanced dataset and how the shape of the dataset looks like after balancing the data of it.

Section 10 : Imbalanced Learning

```
Y.unique()
```

```
array(['<=50K', '>50K'], dtype=object)
```

Get the value count of target column ¶

```
Y.value_counts()
```

```
income
<=50K    36529
>50K     11619
Name: count, dtype: int64
```

Use SMOTE oversampling method to balance the dataset

```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, Y)
```

Check whether dataset is balanced, after applying SMOTE

```
y_resampled.value_counts()
```

```
income
<=50K    36529
>50K     36529
Name: count, dtype: int64
```

```
X_resampled.head()
```

	age	education-num	marital-status	occupation	relationship	race	capital-gain	capital-loss	hours-per-week	native-country
0	39	13	2	0	1	4	0.02174	0.0	40	1
1	50	13	1	3	0	4	0.00000	0.0	13	1
2	38	9	0	5	1	4	0.00000	0.0	40	1
3	53	7	1	5	0	2	0.00000	0.0	40	1
4	28	13	1	9	5	2	0.00000	0.0	40	0

Split the dataset into train and test

Then the dataset is split into training and testing sets (70% training, 30% testing) using the "train_test_split" method in the Scikit Learn library. Since there are approximately 72000 records in the dataset after oversampling, after splitting it, it consisted of 51140 records for the training dataset and 21918 records for the testing dataset.

```
from sklearn.model_selection import train_test_split
```

Split the dataset for training and testing with test size 30%

```
train_x, test_x, train_y, test_y = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)
```

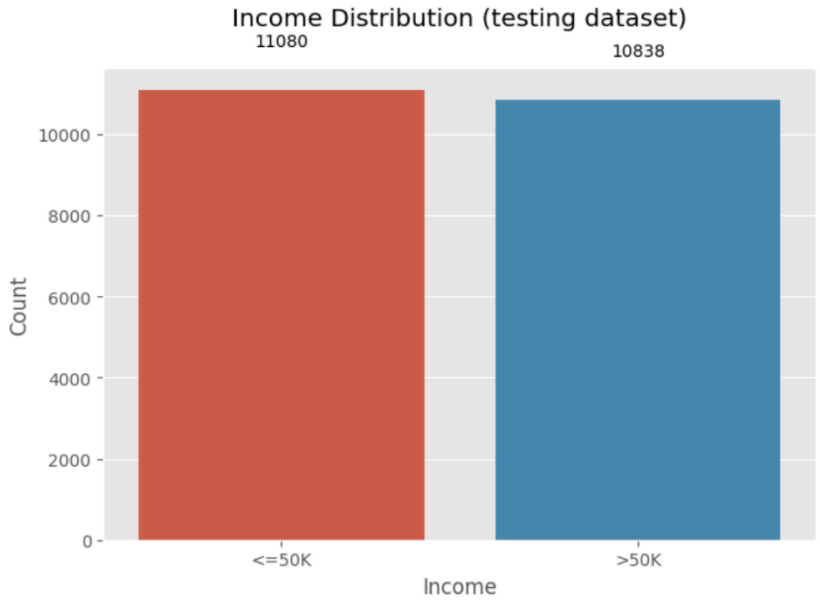
Check the training and testing data set shapes

```
train_x.shape, test_x.shape, train_y.shape, test_y.shape
```

```
((51140, 10), (21918, 10), (51140,), (21918,))
```

The below bar chart shows how the target variable of the training dataset is balanced after splitting the oversampled dataset. In the training dataset there are 25449 records of income less than or equals to 50k and 25691 records of people who have an income greater than 50k.

In addition, the below chart shows that the target variable of the testing dataset is also balanced after splitting the dataset into training and testing. The value count of income less than or equals to 50k in the testing data set is 11080 and the value count of the training dataset is 10838. It shows that the testing dataset is also approximately balanced.



This shows that after splitting the data into training and testing datasets, the target variable (income) is approximately balanced in both training and testing datasets. The important fact is that categories of the target variable in the training dataset has to be balanced to gain better performance.

Apply Random Forest Classifier (After doing imbalanced learning)

Before optimizing the parameters, the random forest classifier model was trained with its defa ult parameters. When it was trained with the default parameters, the random forest classifier model was still overfit. It has a training accuracy of 0.95981 and a testing accuracy of 0.8728 44. The below image shows the random forest classifier model trained with its accuracies before tuning the parameters.

```
from sklearn.ensemble import RandomForestClassifier
```

create a model and see how it works before tuning parameters

```
model = RandomForestClassifier(random_state=42)
model.fit(train_x, train_y)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

check the training score

```
model.score(train_x, train_y)
```

```
0.9598161908486508
```

check testing score

```
model.score(test_x, test_y)
```

```
0.8728442376129208
```

Hyperparameter Optimization for RandomForestClassifier

To optimize the random forest classifier model four main parameters changed and checked by using the randomized search cv method in scikit learn library. The parameters tuned are name ly, “n_estimators”, “max_features”, “min_samples_split” and “min_samples_leaf”. The descr iption of the tuned parameters is given below. (sklearn.ensemble.RandomForestClassifier, 20 24)

“n_estimators”: The number of decision trees in the random forest ensemble model. In this problem it has taken considered values in the range 5 to 49.

“max_features”: The number of features considered when looking for the best split. The “log 2", “auto”, “sqrt” values are considered for this.

“min_samples_split”: It defines the minimum number of samples required to split an interna l node.

“min_samples_leaf”: It defines the minimum number of samples required to have in the leaf node of trees.

The below image shows how the hyper parameters tuned for the random forest classifier.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
import random

# parameter grid
param_grid = {
    'n_estimators': list(range(5, 50)),
    'max_features': ['log2', 'auto', 'sqrt'],
    'min_samples_split': list(range(6, 15)),
    'min_samples_leaf': list(range(4, 15))
}

# Create a RandomForestClassifier instance
rf_classifier = RandomForestClassifier(n_jobs=5)

random.seed(42)

# Create RandomizedSearchCV instance
random_search = RandomizedSearchCV(estimator=rf_classifier, param_distributions=param_grid, n_iter=50, cv=5, n_jobs=5,
                                   random_state=42)

# Fit the model
random_search.fit(train_X, train_Y)

# Print best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

```
Best Parameters: {'n_estimators': 37, 'min_samples_split': 11, 'min_samples_leaf': 4, 'max_features': 'sqrt'}
Best Score: 0.8616147169739893
```

The cv parameter of the randomized search cv is responsible on the cross validation splitting strategy and “n_jobs” parameter defines the number of jobs that must be run in parallel. The randomized search cv method selects randomly 50 points and returns the best parameters in the above scenario due to the given iterations are equals to 50. It can miss the in-between points to tune the parameters. However, it can find out the best parameters in a short time. The given best parameters are as follows.

```
Best Parameters: {'n_estimators': 37, 'min_samples_split': 11, 'min_samples_leaf': 4, 'max_features': 'sqrt'}
Best Score: 0.8616147169739893
```

Train the model with tuned parameters

When the random forest classifier model is trained with the optimized parameters, it has got a training accuracy of 0.8916 and a testing accuracy of 0.8626. The below image shows how the models are trained with the optimized parameters.

Fit the Random Forest Model

```
model_forest.fit(train_x, train_y)
```

```
RandomForestClassifier(min_samples_leaf=4, min_samples_split=11,
                       n_estimators=37, n_jobs=5, random_state=42)
```

Get the training accuracy

```
model_forest.score(train_x, train_y)
```

```
0.8916112631990614
```

Get the testing accuracy ¶

```
model_forest.score(test_x, test_y)
```

```
0.8626699516379231
```

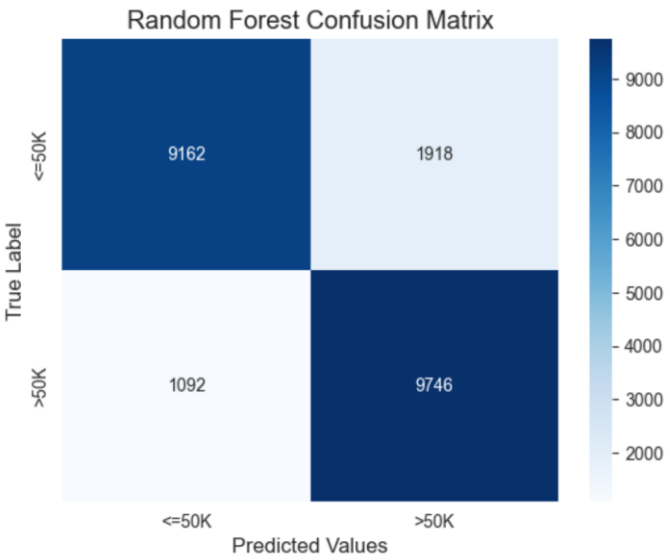
The above image shows that when the parameters are tuned the overfitting issue has resolved.

Evaluation metrices used to evaluate the random forest classifier model

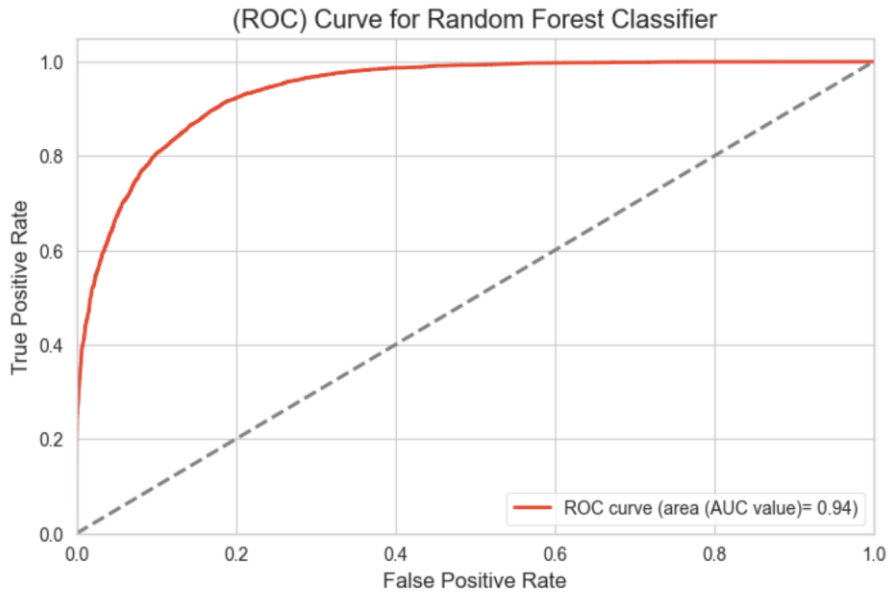
The ROC curve, AUC values, precision, recall, F1-score are the main evaluation metrices used to evaluate the random forest classification model since this model is built for a classification problem. The below image shows the classification report got for the random forest classifier model after its parameters are tuned.

	<=50K	>50K	accuracy	macro avg	weighted avg
precision	0.893505	0.835562	0.86267	0.864534	0.864854
recall	0.826895	0.899243	0.86267	0.863069	0.862670
f1-score	0.858911	0.866234	0.86267	0.862572	0.862532
support	11080.000000	10838.000000	0.86267	21918.000000	21918.000000

This classification report shows that precision, recall and f1-score for all categories have got a score greater than 80%. It shows this model works well for both categories income greater than 50K and income less than or equals to 50K.However, there is a slight difference in precision and recall values. The below image shows the confusion matrix got for the random forest classification model.



This confusion matrix shows that the random forest classification model performs well for the testing dataset in both categories. It shows that approximately 90% of the results of each category is predicted correctly. The below image shows the ROC curve and the AUC value got for the random forest classification model. It shows that it has an AUC value of 0.94. It depicts that this model performs well compared to the model created before handling the imbalanced dataset.



Check the models by passing some real values

It does not mean that models work perfectly by just evaluating them using evaluation matrices. Therefore, it must be checked by passing some values. The below images show some real values that has been passed and the categories these models have predicted.

Check an example less than or equals to 50K with Random Forest Classifier

```
data.loc[0]
```

```
age                          39
workclass             Government
fnlwgt                    77516
education         Bachelor degree
education-num                13
marital-status      Never-married
occupation           Adm-clerical
relationship        Not-in-family
race                        White
sex                          Male
capital-gain                 2174
capital-loss                    0
hours-per-week                 40
native-country      United-States
income                        <=50K
Name: 0, dtype: object
```

Get the normalized value of the capital gain

```
# Given original value
original_value = np.array([2174])

# Reshape the original value to match the input shape for transform
original_value_reshaped = original_value.reshape(-1, 1)

# Use the transform method to obtain the normalized value
normalized_value = (original_value_reshaped - capital_normalizer.data_min_) /
(capital_normalizer.data_max_ - capital_normalizer.data_min_)

normalized_value
```

```
array([[0.02174022]])
```

```
model_forest.predict([[39, 13, 3, 0, 1, 4, 0.02174022, 0, 39 , 1]])
```

```
array(['<=50K'], dtype=object)
```

Check an example more than 50K with Random Forest Classifier

```
data.loc[7]
```

```
age                          52
workclass         Self-emp-not-inc
fnlwgt                     209642
education              High School
education-num                   9
marital-status            Married
occupation         Exec-managerial
relationship               Husband
race                         White
sex                           Male
capital-gain                     0
capital-loss                     0
hours-per-week                  45
native-country       United-States
income                        >50K
Name: 7, dtype: object
```

```
model_forest.predict([[52, 13, 1, 3, 0, 4, 0, 0, 45, 1]])
```

```
array(['>50K'], dtype=object)
```

It shows that these values work perfectly, for the given input variables. Then the model has been saved and loaded again to check whether it works perfectly for the given input variables.

The below image shows how that model saving process is done.

Saving the Model and evaluate some values

```
import pickle

with open('random_forest_model.pkl', 'wb') as rf:
    pickle.dump(model_forest, rf)
```

Load model again and check for some values

```
with open('random_forest_model.pkl', 'rb') as rf:
    model_forest = pickle.load(rf)
```

```
model_forest.predict([[52, 13, 1, 3, 0, 4, 0, 0, 45, 1]])
```

```
array(['<=50K'], dtype=object)
```

Apply Naïve Bayes Algorithm (After doing imbalanced learning)

Do a cross validation to select the best naïve bayes model

The scikit learning official documentations show that Bernoulli Naïve Bayes is the best fitted model for binary classification problems. However, it will be better if a cross validation can be conducted to check the performances of different naïve bayes models and select the best model out of them. (Naive Bayes, 2024) The below picture shows the accuracy levels got for Bernoulli naïve bayes, Gaussian naïve bayes and Multinomial naïve bayes algorithms.

Section 13 :Check what is the best Naive Bayes Algorithm supports well by doing a cross validation

Basically Bernoulli Naive Bayes model performs well for binary classification, but this checks the other models as well

```
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# Define Naive Bayes models
gaussian_nb = GaussianNB()
multinomial_nb = MultinomialNB()
bernoulli_nb = BernoulliNB()

# Perform cross-validation and print accuracies
def perform_cross_validation(model, name):
    scores = cross_val_score(model, train_x, train_y, cv=5)
    print(f"Cross-validation accuracy for {name}: {scores.mean()}")
```

```
# Perform cross-validation for Gaussian Naive Bayes model
perform_cross_validation(gaussian_nb, "Gaussian Naive Bayes")
```

```
Cross-validation accuracy for Gaussian Naive Bayes: 0.6935862338678139
```

```
# Perform cross-validation for Multinomial Naive Bayes model
perform_cross_validation(multinomial_nb, "Multinomial Naive Bayes")
```

```
Cross-validation accuracy for Multinomial Naive Bayes: 0.7224872897927259
```

```
# Perform cross-validation for Bernoulli Naive Bayes model
perform_cross_validation(bernoulli_nb, "Bernoulli Naive Bayes")
```

```
Cross-validation accuracy for Bernoulli Naive Bayes: 0.7629644114196324
```

It shows that Bernoulli Naïve Bayes, Multinomial Naïve Bayes, and Gaussian Naïve Bayes algorithms have respectively got accuracies 0.7629, 0.7224 and 0.6935. It depicts that, Bernoulli Naïve bayes model performs well compared to other models for this binary classification problem.

Apply Bernoulli Naïve Bayes Model with tuned parameters

To get a better performance of Bernoulli Naïve Bayes model, a hyper parameterized optimization is done. 'alpha', 'binarize', 'fit_prior' and 'class_ prior' are the parameters tuned using grid search cv. Some information of the tuned parameters is given below.

"alpha": It is a smoothing parameter used for Laplace smoothing. It can be a positive value.

"binarize": It is the threshold value taken for binarizing the input features. This a value between 0 and 1.

"fit_prior": It is a Boolean calculation that indicates whether to learn class prior probabilities from the given training data.

"class_prior": It defines prior probabilities of the classes. It must be defined as a list of numbers and the summation of the numbers in the list should equals to 1.

The below image shows how the hyperparameters are tuned using the grid search cv method.

Section 14 :Bernoulli Naive Bayes Model (Since It is Binary Classification)

```
from sklearn.naive_bayes import BernoulliNB
```

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import GridSearchCV

clf = BernoulliNB()

# Define the parameter grid to search
param_grid = {
              'alpha': [0.1, 0.5, 1.0, 2.0],  # Smoothing parameter
              'binarize': [0.0, 0.5, 1.0],  # Binarization threshold
              'fit_prior': [True, False],  # Whether to learn class prior probabilities
               'class_prior': [None, [0.5, 0.5]],  # Prior probabilities for the classes
             }

# Perform grid search with cross-validation
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(train_x, train_y)

# Print the best parameters found
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'alpha': 0.1, 'binarize': 0.0, 'class_prior': None, 'fit_prior': True}
```

The given parameters from the grid search cv are “alpha” :0.1, “binarize”: 0.0, “class_prior”: None and “fit_prior”: True. The below image shows the model created by tuning the parameters

Assign the BernoulliNB model with parameters

```
model_bayes = BernoulliNB(alpha=0.1, binarize=0.0, class_prior=None, fit_prior=True)
```

Train bernouli Naive Bayes Model

```
model_bayes.fit(train_x, train_y)
```

```
BernoulliNB
BernoulliNB(alpha=0.1)
```

Get the training score of the Naive Bayes Model

```
model_bayes.score(train_x, train_y)
```

```
0.763042628079781
```

Get the testing score of the naive Bayes Model

```
model_bayes.score(test_x, test_y)
```

```
0.7612920886942239
```

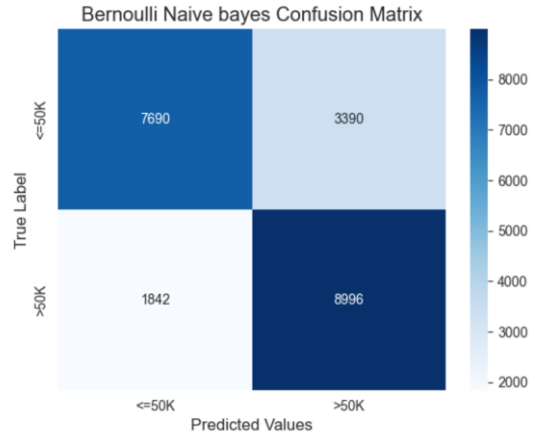
When the parameters are tuned and trained, it shows that accuracy levels have not much increased (There is a slight increment). Therefore, it shows that hyper parameterized optimization does not help a lot to improve the accuracy levels. However, this has considered as the finalized Bernoulli naïve bayes model since it performs better than the model trained without optimizing the parameters. This model consisted with a training accuracy of 0.76304 and a testing accuracy of 0.76129.

Evaluation methods used to evaluate the Bernoulli Naïve Bayes Model

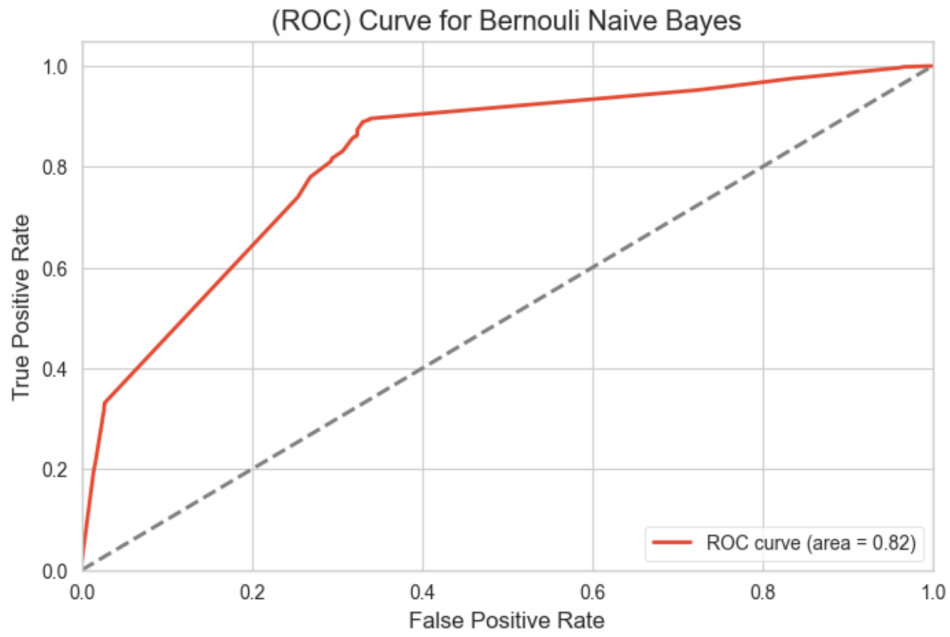
Since this model is created for a classification problem, as in the random forest classifier, ROC curve, AUC value, precision, recall and F1-score are the techniques used to evaluate the Bernoulli Naïve Bayes model. The below image shows the classification report got for this model.

	<=50K	>50K	accuracy	macro avg	weighted avg
precision	0.806756	0.726304	0.761292	0.766530	0.766974
recall	0.694043	0.830042	0.761292	0.762043	0.761292
f1-score	0.746167	0.774716	0.761292	0.760442	0.760284
support	11080.000000	10838.000000	0.761292	21918.000000	21918.000000

It shows that, all the precision, recall and F1-score values are greater than 0.69.In addition it shows that category that has an income greater than 50k performs better than category which has an income less than or equal to 50k with its recall and F1-score values, but the precision value of the income category less than or equal to 50k performs better than the category that has an income greater than 50k.However, random forest classifier model's all evaluation matrices performs better than the Bernoulli naïve bayes model. The below image shows the confusion matrix got for the Bernoulli naïve bayes model after its parameters are tuned.



This confusion matrix shows that income less than 50k category has given false predictions, compared to the income greater than 50k category. However, more than 70% of the values in the testing dataset has predicted correctly in the above confusion matrix. The below image shows the ROC curve got for the Bernoulli naïve bayes model after handling the imbalanced dataset. For this model, it has got an AUC value of 0.82. It depicts that this model has a better performance compared to the model created before handling the imbalanced dataset.



Check the models by passing some real values for the Bernoulli naïve bayes model.

As mentioned in the random forest model evaluation section, it is beneficial to check the model by giving input variables and see how model predicts the output. When the input variables are given and checked, the naïve bayes performed well as the random forest classification model with the same inputs. The below image shows the given inputs and the predicted values from the Bernoulli naïve bayes model.

Test some values with the model

```
X_resampled.loc[0], y_resampled[0]
```

```
(age               39.00000
 education-num     13.00000
 marital-status     2.00000
 occupation         0.00000
 relationship       1.00000
 race               4.00000
 capital-gain       0.02174
 capital-loss       0.00000
 hours-per-week    40.00000
 native-country     1.00000
 Name: 0, dtype: float64,
 '<=50K')
```

```
model_bayes.predict([[39, 13, 2, 0, 1, 4, 0.02174022, 0, 40, 1]])
```

```
array(['<=50K'], dtype='<U5')
```

```
X_resampled.loc[10], y_resampled[10]
```

```
(age              37.0
 education-num    10.0
 marital-status    1.0
 occupation        3.0
 relationship      0.0
 race              2.0
 capital-gain      0.0
 capital-loss      0.0
 hours-per-week   80.0
 native-country    1.0
 Name: 10, dtype: float64,
 '>50K')
```

```
model_bayes.predict([[37, 10, 1, 3, 0, 2, 0, 0, 80, 1]])
```

```
array(['>50K'], dtype='<U5')
```

Then Naïve bayes model is saved as a pickle file and loaded again to check whether it works properly when input data is given.

Save the Naive Bayes Model as a pickle file

```
import pickle

with open('naive_bayes_model.pkl', 'wb') as nb:
    pickle.dump(model_bayes, nb)
```

Load the Naive Bayes model and check for some values

```
with open('naive_bayes_model.pkl', 'rb') as nb:
    model_bayes = pickle.load(nb)
```

```
model_bayes.predict([[37, 10, 1, 3, 0, 2, 0, 0, 80, 1]])
```

```
array(['>50K'], dtype='<U5')
```

Experimental results and the Comparison of the final Random Forest Classification and Naïve Bayes models

	Random Forest Classifier		Bernoulli Naïve Bayes	
Category	<=50K	>50K	<=50K	>50K
Precision	0.893505	0.835562	0.806756	0.726304
Recall	0.826895	0.899243	0.694043	0.830042
F1-score	0.858911	0.866234	0.746167	0.774716
Accuracy	0.86267		0.761292	

Precision

The RandomForest classifier has achieved higher precision values for both classes that have an income greater than 50K and less than or equal to 50K compared to the Bernoulli Naïve Bayes classifier. In addition, it shows that the Random Forest classifier has performed well, especially for the class that has an income greater than 50K.

Recall

When the recall values are calculated, random forest classifier model has performed well in both the categories that has a income "<=50K" and ">50K".

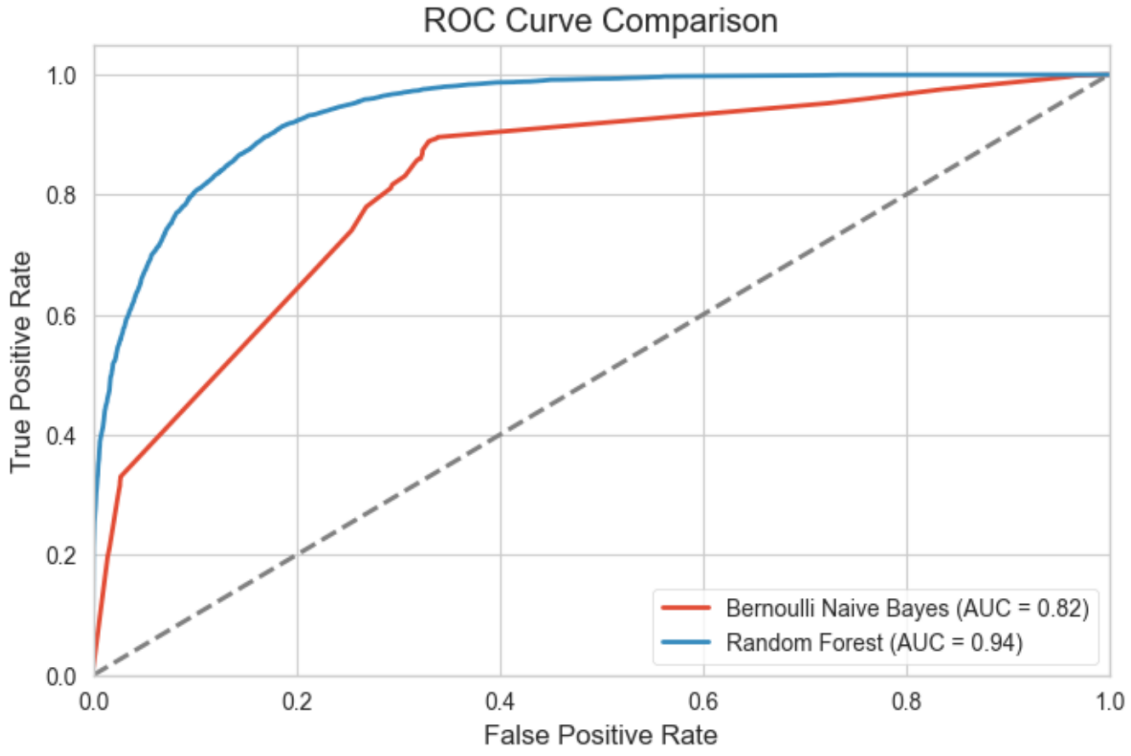
F1-Score

The F1-Score of the random forest classifier has performed well compared to the Bernoulli Naïve Bayes model in both income categories that has an income "<=50K" and ">50K". That calculation shows that Random Forest classifier has performed well in both precision and recall values.

Accuracy

The random forest classifier has performed well compared to the Bernoulli naïve bayes model. When the overall comparison done, it shows that random forest model performs well in all the evaluation matrices.

Roc curve created for the model comparison.



This AUC values and ROC curve depicts that Random Forest classifier model performs well compared to the Bernoulli Naïve Bayes model.

Limitations

Since the given dataset is imbalanced, imbalanced learning techniques had to be used and it has given good accuracies. However, if the data is initially balanced there was a possibility of getting better performance in both models. There were some abnormalities in the dataset when working hours per week is checked with respective to the age. If the false data points not given from the initial dataset, these models might perform well.

Possible Further Enhancements

This can be trained using some other classification-based models in machine learning and do a cross validation to check which model performs well. In addition, if required, classification based deep learning models can also be used to check which model performs well.

Git Repository details

The link for the git repository is given below. The code is there in the ipython notebook DataCleaningProcessModelTraining.ipnyb.

Git repository link - https://github.com/Lakshan2023/Machine-Learning-Coursework

The naïve_bayes_model.pkl and random_forest_model.pkl are the finalised models created for Bernoulli naïve bayes and random forest classifier.

References

More Americans Are Working Into Their 80s. What's Keeping Them in the Workforce. (2023, September 07). Retrieved from barrons: https://www.barrons.com/articles/working-in-your-eighties-retirement-f35ae81f

Naive Bayes. (2024, March 27). Retrieved from Scikit Learn: https://scikit-learn.org/stable/modules/naive_bayes.html

sklearn.ensemble.RandomForestClassifier. (2024, March 24). Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Appendix (Code)

```
## Section 1:  (Read and Describe Data)

#### Filter warnings make the consistency

import warnings
warnings.filterwarnings("ignore")

### Read the data

#### import the required libraries (numpy, pandas , seaborn, matplotlib)

import numpy as np
import pandas as pd
import seaborn as sns
import scipy as sc
from matplotlib import pyplot as plt
plt.style.use('ggplot')

#### read the dataset from UCI link

# install uci api call
!pip install ucimlrepo
from ucimlrepo import fetch_ucirepo

# fetch dataset
adult = fetch_ucirepo(id=2)

# data (as pandas dataframes)
X = adult.data.features
y = adult.data.targets
```

make the dataset available to display all the rows

pd.set_option('display.max_rows', None)

get the shape of the dataset

print the shape of the variable values
X.shape

print first five rows of the X dataset

#print first five elements of the vriables
X.head(5)

Check for unique columns, its data types, and null values

# Check for columns and its data types
X.info()

get the shape of the target variable dataset (48842 values)

print the shape of the y values
y.shape

print the first five values of target dataset

# print first five values of y dataset
y.head()

check how many null values are there in the target varible

get the information of the target column

```
y.info()

#### Join the target variable and the dataset

data = X.join(y)

#### get the shape of the joined dataset

data.shape

data.head()

data.info()

#### Check data types of each column

data.dtypes

#### Visualize how null values are distributed in all columns

null_mask = data.isnull()

plt.figure(figsize=(8, 6))
sns.heatmap(null_mask, cmap='viridis', cbar=False)
plt.title('Null Values Heatmap')
plt.show()

##### It shows columns workclass, occupation and native country has null values.

#### Check the sum of null values in all columns

data.isnull().sum()  # work class, occupation and native country has null values
```

Check for number of unique categories in each column

```
data.nunique()
```

get calculations for numerical data to understand the dataset

```
# descibe how data is distributed for integer data types
data.describe().T
```

```
# Things can be identified - Most of the ages are between late20's to 40's
# Capital gain and capital loss consists with mainly 0's
# Working hours per week is mostly close to 40 hours
```

Section 2 :(Null values and Abnormality values handelling in categorical data)

Check values in work class

Check unique values in the workclass

```
data["workclass"].unique()
```

Check how many ? contains in the workclass column

```
filter = data["workclass"] == "?"
data.loc[filter].shape
```

Check how many null values are there in workclass column

```
data[(data["workclass"].isnull() == True)].shape
```

Display sample rows that contain null values for workclass column

```
data[data["workclass"].isnull() == True].sample(5)

#### Display sample rows that contain ? values for workclass column

data[data["workclass"] == "?"].sample(5)

#### Replace all question mark values as null

data["workclass"].replace({"?": np.nan}, inplace=True)

#### Count of null values in workclass column after replacement

data["workclass"].isnull().sum()

#### Check the value count of categories in workclass column

data["workclass"].value_counts()

#### Visualization of workclass categories

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))

# Value counts of Work class categories
value_counts = data["workclass"].value_counts()

# Seaborn bar plot of workclass categories
sns.barplot(x=value_counts.values, y=value_counts.index, palette="hls")

# Add percentage annotations for class categories
```

```
total = len(data["workclass"])
for i, count in enumerate(value_counts.values):
    percentage = (count / total) * 100
    plt.text(count, i, f'{percentage:.2f}%', va='center', fontsize=14)

plt.title("Workclass Categories", fontsize=18)
plt.xlabel("Number of people", fontsize=16)
plt.ylabel("Category name", fontsize=16)

plt.show()
```

This shows that most of the workclass values contains with people who are working in the private sector. Therefore, the null values

have to be handled by replacing them with the mode category (Private Class).

```
####
#### Replace the null values in workclass with its mode

# Filled the null values of the workclass column with mode,
# since Private has a huge frequency
data["workclass"].fillna(data["workclass"].mode().iloc[0], inplace=True)

#### Unique values of workclass column after replacement

data["workclass"].value_counts()

## Check values in occupation

#### Check total null values in occupation class

data[data["occupation"].isnull() == True].shape[0]
```

```
#### Check value count of each unique occupation category

data["occupation"].value_counts()

#### Replace ? values of occupation column with null

data["occupation"].replace("?", np.nan, inplace=True)

#### Null values count of occupation column after replacement

data[data["occupation"].isnull() == True].shape[0]

#### Visualize Occupation value count before value replacements

plt.figure(figsize=(20, 10))

#  Value counts of Occupation categories
value_counts = data["occupation"].value_counts()

# Seaborn bar plot for occupation column before replacement
sns.barplot(x=value_counts.values, y=value_counts.index, palette="husl")

# calculate the percentages of each occupation category
total = len(data["occupation"])
for i, count in enumerate(value_counts.values):
    percentage = (count / total) * 100
    plt.text(count, i, f'{percentage:.2f}%', va='center', fontsize=14)

plt.title("Occupation before replacement", fontsize=16, color='black')
plt.xlabel("Number of people", fontsize=14, color='black')
plt.ylabel("Occupation Category", fontsize=14, color='black')
```

```
plt.show()

##### This bar chart shows that the “Prof-speciality” is the mode category, but “Craft repair” and “Exec Managerial” categories are close

##### to the value count of the mode category, therefore the missing values have to be replaced with “Prof-speciality”, “Craft repair” and

##### “Exec Managerial” categories randomly.

####

#### Replace Occupation columns with Prof-specialty, Craft-repair and Exec-managerial

###### This is done due to each occupation class consist with approximately same value counts

import random

import numpy as np

random.seed(2024)

occupation_categories = ["Prof-specialty", "Craft-repair", "Exec-managerial"]

data["occupation"] = data["occupation"].apply(lambda x:
random.choice(occupation_categories) if pd.isnull(x) else x)

#### Get the value counts of Occupation column after replacing null values

data["occupation"].value_counts()

#### Visualize Occupation value count after value replacements

plt.figure(figsize=(20, 10))

#  Value counts of Occupation categories
```

```
value_counts = data["occupation"].value_counts()

# Seaborn bar plot for occupation column after replacement
sns.barplot(x=value_counts.values, y=value_counts.index, palette="husl")

# calculate the percentages of each occupation category
total = len(data["occupation"])
for i, count in enumerate(value_counts.values):
    percentage = (count / total) * 100
    plt.text(count, i, f'{percentage:.2f}%', va='center', fontsize=14, color='black')

plt.title("Occupation after replacement", fontsize=18, color='black')
plt.xlabel("Number of people", fontsize=14, color='black')
plt.ylabel("Occupation Category", fontsize=14, color='black')

plt.show()

##### It shows that it has not impacted a lot on the initial distribution of the Occupation column by randomly replacing the categories
##### that consist with the highest value counts.
#####

## Check values in relationship

#### Check for unique values in relationship

data["relationship"].unique()

#### Value count of categories in relationship column

data["relationship"].value_counts()
```

Check values in Native Country

Check number of null values in native country column

```
data[data["native-country"].isnull() == True].shape
```

Value count of categories in native country column

```
data["native-country"].value_counts()
```

Number of ? values in native-country column

```
data[data["native-country"] == "?"].shape
```

Replace ? values in native-country column with null values

```
data["native-country"].replace("?", np.nan, inplace=True)
```

get the value count for native countries after replacing ? values

```
data["native-country"].value_counts()
# Most of the values are from us,
#then can replace the null and ? values with "United-States"
```

Fill the null values in native country column with United States(Mode)

```
data["native-country"].fillna(data["native-country"].mode().iloc[0], inplace=True)
```

Check the native-country value counts after replcing values

```
data["native-country"].value_counts()
```

Get a general overview of the data after handelling null values

Check whether there are any other columns with null values

```
data.isnull().sum() # null values solved
```

Check the number of unique values in each column

```
data.nunique()
```

Section 3: Feature Engineering for Categorical values

Check values in education

Unique values in education column

```
data["education"].unique()
```

Visualize Educational level categories before feature Engineering

```
# Set the fig size
plt.figure(figsize=(20, 6))

# Countplot for education level before feature engineering
ax = sns.countplot(x=data['education'], palette='husl')

# Add percentages for each bar
total = len(data['education'])
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
```

```
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=12, color='black')

# display the bar plot for educational level with labels
plt.title("Education Level before feature engineering", fontsize=20, color='black')
plt.xlabel("Education Level", fontsize=16)
plt.ylabel("Count", fontsize=16)

plt.show()

##### The categories ‘11th’, ‘9th’, ‘7th-8th’, ‘5th-6th’, ‘10th’, ‘1st-4th’, ‘12th’ and ‘Preschool’ are identified as categories that  falls within
##### the school or  below level. The categories, ‘Assoc-voc’, ‘Assoc-acdm’, ‘Some-college’, ‘Prof-school’ are identified under the
##### category of college level.  High school graduate, bachelor's, master’s and doctorate are considered as the other main
##### categories that falls under education level.

####
#### Do feature Engineering for Education Level by placing less categories

school_or_below = ['11th', '9th', '7th-8th', '5th-6th', '10th', '1st-4th', '12th', 'Preschool']
high_school = ['HS-grad']
college = ['Assoc-voc', 'Assoc-acdm', 'Some-college', 'Prof-school']
under_grad = ['Bachelors']
post_grad = ['Masters']
doctorate = ['Doctorate']

def change_education_values(value):
    # replace school level grades as school or below
    if value in school_or_below:
        return "School or below"
    # replace high school graduate as High School
    elif value in high_school:
```

```
        return "High School"
    # replace bachelor degree as Bachelors
    elif value in under_grad:
        return "Bachelor degree"
    # replace masters degree as masters
    elif value in post_grad:
        return "Masters Degree"
    # replace doctorate degree as Doctorate
    elif value in doctorate:
        return "Doctorate"
    # replace College level qualifications as College
    elif value in college:
        return "College"


data["education"] = data["education"].apply(change_education_values)

#### Get the value count for education after feature engineering

data["education"].value_counts()

#### Get the value count for education after feature engineering

plt.figure(figsize=(20, 6))

# Countplot for education level after feature engineering
ax = sns.countplot(x=data['education'], palette='Set1')

# Total number of data points in education level column
total = len(data['education'])

# display the percentages of values on top of each bar
```

```
for p in ax.patches:
    percentage = '{:.2f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=14, color="black")

# display the bar plot for educational level with labels after feature engineering
plt.title("Education Level after feature engineering", fontsize=18, color="black")
plt.xlabel("Education Level", fontsize=14, color="black")
plt.ylabel("Count", fontsize=14, color="black")

plt.show()
```

This barchart shows how each categorical values are distributed after feature engineering is done for the education level.

When the models are trained, its easy for the models to identify the categories in columns when it consist with less number of

categories.

```
####
#### Check education column consist with (48842) values after feature engineering

data["education"].value_counts().values.sum()
```

Check values in marital Status

Visualize Marital status column before feature engineering

```
plt.figure(figsize=(20, 8))

# Countplot for marital status before feature engineering
ax = sns.countplot(x=data['marital-status'], palette='Set1')
```

```
# Calculate total number of data points for martial-status
total = len(data['marital-status'])

# display the percentages of values on top of each bar
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=14)

# display the bar plot for Martail Status with labels before feature engineering
plt.title("Marital Status before feature engineering", fontsize=19, color="black")
plt.xlabel("Marital Status", fontsize=14, color="black")
plt.ylabel("Count", fontsize=14, color="black")

plt.show()

##### This bar chart shows that ‘married-spouce-absent’, ‘Married-AF-spouce’ and ‘Widowed’ categories consisted with less percentages
##### compared to other categories. Therefore, ‘Married-civ-spouse’, ‘Married-AF-spouse’ categories can be considered under married
##### category. Never Married and Divorced categories can be considered as two separated categories and ‘Married-spouse-absent’,
##### ‘Widowed’ and  ‘Separated’  status is considered under other categories of the marital status.

####
#### Unique Categories of marital status

data["marital-status"].unique()

#### Value counts for unique categories of marital status
```

```
data["marital-status"].value_counts()

#### Do feature engineering for marital-status with less categories

# civ spouce - civillian spouce, af-spouce - from armed forces
never_married = ['Never-married']
married = ['Married-civ-spouse', 'Married-AF-spouse']
divorced = ['Divorced']
# 'Married-spouse-absent' - (not living together)
other_status = ['Married-spouse-absent', 'Widowed', 'Separated']

def spouce_status_change_values(value):
    if value in never_married:
        return "Never-married"
    elif value in married:
        return "Married"
    elif value in divorced:
        return "Divorced"
    elif value in other_status:
        return "Other"

data["marital-status"] = data["marital-status"].apply(spouce_status_change_values)

#### get the value counts for marital-status after feature Engineering

data["marital-status"].value_counts()

#### Visualize the marital status column after feature Engineering

plt.figure(figsize=(20, 6))

# Countplot for marital status after feature engineering
```

```
ax = sns.countplot(x=data['marital-status'], palette='Set1')

# Calculate total number of data points for marital-status
total = len(data['marital-status'])

# display percentages for each marital status category after feature engineering
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=14)

# display the bar plot for Martail Status with labels after feature engineering
plt.title("Marital Status after feature engineering", fontsize=19, color="black")
plt.xlabel("Marital Status", fontsize=14, color="black")
plt.ylabel("Count", fontsize=14, color="black")

plt.show()
```

This bar chart shows after doing the feature engineering process, Marital status has deducted to four categories. Never married,

Married and Divorced are considered as the main categories and other categories renamed as Other.

```
##
## Feature Engineering for native country

#### Get the value count for native country column

data["native-country"].value_counts()
```

Since there is a huge gap between value counts in native country(US and other countries),

all other countries are replaced as other using a lambda function

```
data["native-country"] = data["native-country"].apply(lambda x : x if x == "United-States" else "Other")
```

Get the value counts of the native country after feature Engineering process

```
data["native-country"].value_counts()
```

Visualize the native country values after feature Engineering

```
# Calculate counts and percentages of native country categories
counts = data['native-country'].value_counts()
total = counts.sum()
percentages = [(count / total) * 100 for count in counts]

# Plot the count plot after feature engineering (Native country)
plt.figure(figsize=(8, 4.75))
sns.countplot(x=data['native-country'], palette='Set1')
plt.title("Native Country after feature engineering")
plt.xlabel("Native Country")

# Annotate bars with counts and percentages
for i, (count, percentage) in enumerate(zip(counts, percentages)):
    plt.text(i, count, f"({percentage:.2f}%)", ha='center', va='bottom')

plt.tight_layout()
plt.show()
```

This shows that more than 90 percent of the values of the dataset are from United States and 8.50% from other countries.

It shows that without dividing countries into regions, identifying it as United states or not is the best option based on the

```
##### value counts of the native country column.

## Check values in income

#### Check value counts of the target variable before feature engineering

data["income"].value_counts()

#### display head of data to check income

data.head(5)

#### Replace values less than 50k and greater than 50k in to one format

over_50k = [">50K", ">50K."]
less_equal_50k = ["<=50K", "<=50K."]

def income_classification_change(value):
    ## Assign income greater than 50k values
    if value in over_50k:
        return ">50K"
    ## Assign income less than or equals to 50K values
    elif value in less_equal_50k:
        return "<=50K"

### Assign the updated values to the income column
data["income"] = data["income"].apply(income_classification_change)

#### Unique values of target variable after feature engineering

data["income"].unique()
```

```
#### get the value counts of the target variable after feature engineering

data["income"].value_counts()

#### Visualize the income after feature engineering

plt.figure(figsize=(8, 4.75))

# Countplot for income class frequency
ax = sns.countplot(x=data['income'], palette='Set1')

# Calculate total number of data points for Income class
total = len(data['income'])

# calculate and display Percentages for each income class
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=10)

plt.title("Income Class Frequency")
plt.xlabel("Income class")
plt.ylabel("Count")

plt.show()
```

After done the feature engineering for income class, it shows that target variable income class is imbalanced in this dataset.

There are 76.1% of people who are having an income less than or equals to 50K and 23.9% people are from the category

who are having an income greater than 50K.

That show there is an imbalance status of the target variable in the data set

```
data["income"].value_counts()
```

Work Class Feature Engineering

display the value counts of workclass

```
data["workclass"].value_counts()
```

Function that is capable to identify percentage of each categories in columns

```
def calculate_value_counts(selected_column, data):
    # Calculating the value counts from the selected column
    value_counts = selected_column.value_counts()
    # Take the percentage
    percentage = value_counts / len(data) *100
    # display the value count and percentage in a dataframe
    result_df = pd.DataFrame({'value_counts': value_counts,'percentage':percentage})
    return result_df
```

Calculate income with greater than 50k for work class category percentages

```
high = calculate_value_counts(data[data["income"] == ">50K"]["workclass"], data)
high
```

Visualize work class category percentages for income greater than 50k

```
value_counts = high["value_counts"].values
percentage = high["percentage"].values
categories = high.index
```

```
plt.figure(figsize=(16, 8))
sns.barplot(x=categories, y=value_counts, palette='husl')

# display workclass category that has income greater than 50K
for i, (count, percent) in enumerate(zip(value_counts, percentage)):
    plt.text(i, count, f"{count}\n({percent:.2f}%)",
             ha='center', va='bottom', fontsize=12)

# display the barplot workclass category that has income greater than 50K
plt.title("work class categories for income greater than 50k", fontsize=20, color="black")
plt.xlabel("Work Class Category", fontsize=15, color="black")
plt.ylabel("Value Counts", fontsize=15, color="black")

plt.xticks(rotation=0, ha='center')

plt.tight_layout()
plt.show()

##### This bar chart shows that majority of people who are having an income greater than 50k are from Private work class
##### category. There are only few records of people who are earning more than 50k from above categories.

####
#### Calculate income with less than 50k for work class category percentages

low = calculate_value_counts(data[data["income"] == "<=50K"]["workclass"], data)
low

#### Visualize work class category percentages for income less than or equals 50k

value_counts = low["value_counts"].values
percentage = low["percentage"].values
```

```
categories = low.index

plt.figure(figsize=(16, 8))
sns.barplot(x=categories, y=value_counts, palette='husl')

# display workclass category that has income less than or equal to 50K
for i, (count, percent) in enumerate(zip(value_counts, percentage)):
    plt.text(i, count, f"{count}\n({percent:.2f}%)",
             ha='center', va='bottom', fontsize=12)

# display the barplot workclass category that has income less than or equal to 50K
plt.title("work class categories for income less than or equal to 50K", fontsize=20, color="black")
plt.xlabel("Work Class Category", fontsize=15, color="black")
plt.ylabel("Value Counts", fontsize=15, color="black")

plt.xticks(rotation=0, ha='center')

plt.tight_layout()
plt.show()

##### This bar chart also shows that people who are having an income less than or equals to 50k are from  Private work class
##### category.In addition, it shows that majority of people in this dataset are representing who are having an income less than
##### 50k category since out of 488842 records, private workclass that has an income less than or equals to 50k consist with
##### 29053 records.

###
### Check for abnormalities of  workclass column and remove them

#### Check whether never-worked people have working hours
```

```
#### This is an abnormality, this cannot be happen
data[data["workclass"] == "Never-worked"]["hours-per-week"]

#### Check number abnormality rows in the workclass (Never worked)

data[data["workclass"] == "Never-worked"]["hours-per-week"].shape

data[data["workclass"] == "Never-worked"].shape

#### Drop the never worked values

data = data[data["workclass"] != "Never-worked"]
data[data["workclass"] == "Never-worked"]["hours-per-week"]

#### Check the value count of workclass after dropping the values

data["workclass"].value_counts()

#### Function to do feature engineering for workclass

government = ["State-gov", "Local-gov", "Federal-gov"]
private = ["Private"]
self_employee_income = ["Self-emp-inc"]
self_employee_not_income = ["Self-emp-not-inc"]
other = ["Without-pay"]

def change_workclass(value):
    if value in government:
        return "Government"
    elif value in private:
        return "Private"
    elif value in self_employee_income:
```

```
        return "Self-emp-inc"
    elif value in self_employee_not_income:
        return "Self-emp-not-inc"
    elif value in other:
        return "Other"

#### Apply feature engineering function for workclass column

data["workclass"] = data["workclass"].apply(change_workclass)

#### Value counts of workclass column after feature engineering

data["workclass"].value_counts()

#### Visualize the workclass column after feature engineering

plt.figure(figsize=(11, 5))

# Countplot for work class after feature engineering
ax = sns.countplot(x=data['workclass'], palette='Set1')

# Calculate total number of data points in workclass column
total = len(data['workclass'])

# Display the percentages of workclass column after feature engineering
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=10)

plt.title("Work Class after Feature Engineering")
```

```
plt.xlabel("Work Class")
plt.ylabel("Count")

plt.show()
```

After considering "State-gov", "Local-gov" and "Federal-gov" as Government category, records from Government Category

consisted with 13.4% of the all records. Self employee income and not income categories considered as two seperate categories

due to it impacts for income based on the above two bar charts. All other work classes are considered as Other work classes.

####

Work class vs income after feature engineering

This shows that majority of groups earn less than 50k instead of people who have self employeement as income

```
plt.figure(figsize=(10, 5))
plt.title("Work class income status after feature Engineering")
sns.countplot(x="workclass", hue="income", data=data, palette="ocean")
plt.show()
```

The below count plot shows that majority of people have an income less than 50K, except the people who are doing

a self-employment.

Marital Status Analysis after feature engineering

Majority of people in all marital groups earn less than 50K

```
plt.figure(figsize=(10, 5))
plt.title("Marital status after feature Engineering")
sns.countplot(x="marital-status", hue="income", data=data, palette="ocean")
```

```
plt.show()

##### Majority of people in all the marital status categories are having an income less than 50k.
#####

### Education status Analysis after feature Engineering

##### The below bar chart shows that majority of groups earn less than 50k instead of people, who have Masters degrees
##### and Doctorate degree as income

plt.figure(figsize=(10, 5))
plt.title("income based on education status after feature Engineering")
sns.countplot(x="education", hue="income", data=data, palette="ocean")
plt.show()

## Analyse Dataset  with sex

#### This shows majority of both male and female people earn less than 50k as their income

plt.title("income vs sex")
sns.countplot(x="sex", hue="income", data=data, palette="ocean")
plt.show()

## Analyse dataset with Relationship

#### Visualize relationship categories

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

ax = sns.countplot(x="relationship", data=data, palette="ocean")
# Calculate total number of data points for Relationship class
```

```
total = len(data['relationship'])

# calculate and display Percentages for each Relationship class
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=10)

plt.title("Relationship", fontsize=18, color="black")
plt.xlabel("Relationship Category", color="black")
plt.ylabel("Count", color="black")

plt.show()
```

Majority of people from relationship categories are from Husband Relationship Category with a percentage of 40.4%.

```
####
#### Visualize income vs relationship

plt.figure(figsize=(10, 5))
plt.title("income vs relationship")
sns.countplot(x="relationship", hue="income", data=data, palette="ocean")
plt.show()
```

All the relationship categories having majority of people who are having an income less than or equals to 50K.

```
## Analyse dataset with Race

#### Get the value counts of Race column categories
data["race"].value_counts()
```

Visualize race categories

```
plt.figure(figsize=(12, 5))
ax = sns.countplot(x=data['race'], palette='Set1')

# Calculate total number of data points for race class
total = len(data['race'])

# calculate and display Percentages for each race class
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom', fontsize=10)

plt.title("Race", fontsize=16, color="black")
plt.xlabel("Race Category", fontsize=12, color="black")
plt.ylabel("Count", fontsize=12, color="black")
plt.show()
```

The majority of people from this dataset is representing from the "White" relationship category with a percentage of 85.5%.

Visualize income vs race

```
plt.figure(figsize=(10, 5))
plt.title("income vs race")
sns.countplot(x="race", hue="income", data=data, palette="ocean")
plt.show()
```

The majority of people from all the race categories are having an income less than or equals to 50k.

Analyse Age vs Income

Visualize age vs income

```
plt.figure(figsize=(20, 8))
plt.title("age vs income")
sns.countplot(x="age", hue="income", data=data, palette="ocean")
plt.show()
```

This plot shows that people who are earning a salary greater than 50k is plotted approximately normal. In addition it shows

the count of people who are earning a salary less than or equals to 50k dicreases from the age of 22.

Distribution of age

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

# Histogram for age distribution of the dataset
sns.histplot(data=data, x='age', bins=20, palette="husl", kde=True)

# Calculate the  mean age
mean_age = data['age'].mean()

# Majority of age based records are between 30's and 40's
plt.axvline(x=mean_age, color='red', linestyle='--', label=f'Mean Age: {mean_age:.2f}')
plt.title('Distribution of Age')
plt.xlabel('Age', fontsize=12, color="black")
plt.ylabel('Frequency', fontsize=12, color="black")
plt.legend()
plt.show()
```

When the age distribution is taken, majority of people are from 20 to 50 age limits.In addition, it shows that mean age

of people is 38.65.

Section 4: Outlier Detection of numerical values

plot numerical data using boxplots to see outliers

```
# Check whether data set has outliers
data.plot(kind="box", figsize=(15, 20), layout=(3, 3), subplots=True)
plt.show()
```

The above boxplots show the outliers detected for all numerical value columns.However, all the outlier values cannot be

dropped due to it impacts the dataset.This outliers are shown based on the inter quartile ranges. The meta data of the categories

are important to handle outliers. For example, it is possible to have people who work in their 80's as well.Therefore, these values

cannot be dropped.

Check how numerical data is distributed using histograms

```
data.hist(figsize=(12, 12), layout=(3, 3), sharex=False)
plt.show()
```

The above histograms show how each numerical column values are distributed in its ranges. However, it shows that capital-gain

and capital-loss columns not continuously distributed in the dataset.

####

Check how each column is corelated using a pairplot

```
sns.pairplot(data=data, hue="income")
```

It is difficult to identify income greater than 50k and income less than or equals to 50k values seperately in regions of

compared columns from the above pairplot.

Analyse an clean outliers detected

Average hours that people work per a week

```
data["hours-per-week"].mean()
```

Get a general idea on how hours per week is distributed

```
data["hours-per-week"].describe()
```

Visualize how working hours are distributed

```
plt.figure(figsize=(15,6))
sns.histplot(data["hours-per-week"], kde=True, bins=40)
plt.title("Distribution plot for working hours per week")
```

This hours per week histogram shows that, majority of people are working around 40 hours per week, It shows that there

are some people who work over 70 hours per week as well.Therefore, the age and the working hours has to be considered to find out the

abnormalities of the dataset to handle outliers. The people who work less hours also have to be considered to check abnormalities.

####

Check how many people who have over worked after 60

```
# people who have over worked after age 60
data[(data["age"] > 65) & (data["hours-per-week"] > 70)].shape
```

Drop the detected outliers

data = data[~((data["age"] > 65) & (data["hours-per-week"] > 70))]

people who work less than 10 hours a week in private and government

data[(data["hours-per-week"] < 10) & ((data["workclass"].isin(["Government", "Private"])))].shape

remove data working hours per week less than 10 and work class in (Government or private) workclasses

data = data[~((data["hours-per-week"] < 10) & ((data["workclass"].isin(["Government", "Private"]))))]

Get the shape of the data

data.shape

Check people who are married less than 18 years old in age

Can't remove this, due to USA allows marriage in 17 with certain laws

data[(data["age"] < 18) & (data["marital-status"] != "Never-married")]

print a sample of data after abnormalaties dropped

data.head(5)

Get the unique value count of each column

data.nunique()

Get the unique value count of each column

```
data["race"].value_counts()
```

Income Feature Engineering for numerical conversion

```
data["income"].value_counts()
```

Convert income less than or equals to 50k as 0 and greater than 50k as 1

```
data["income"] = data["income"].map({"<=50K" : 0, ">50K": 1})
```

After conversion get the value count

```
data["income"].value_counts()
```

Show the correlation of the numerical values with target value income

```
cor = data.corr(numeric_only=True, method="pearson")
cor
```

Visualize how each numerical column is corelated with each other

```
# heatmap to visualize numerical columns corelation
sns.heatmap(cor, annot=True)
```

Their is low correlation between final weight and income compared to other columns with income

get correlation with all numeric columns only with the target column

```
data.corr(numeric_only=True, method="pearson")["income"]
```

Analyse education-num

```
data["education-num"].value_counts().sort_index()
```

Get calculations of educational-num

```
data["education-num"].describe()
```

When educational number is greater or equal to 14, income is larger than 50K for majority of people

```
data.groupby("education-num")["income"].value_counts()
```

Visualize income vs educational-num

```
data["income"] = data["income"].map({0: "<=50K", 1: ">50K"})
plt.figure(figsize=(10, 4))
plt.title("income vs education-num")
sns.countplot(x="education-num", hue="income", data=data, palette="ocean")
plt.show()
```

This shows that when education number is greater than or equals to 14 most of the people are getting a salary greater than 50k

Analyse Hours per week with income

Visualize income vs hours per week

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20, 10))
plt.title("Income vs Hours-per-week")
sns.countplot(x="hours-per-week", hue="income", data=data, palette="ocean")
plt.xlabel("Hours-per-week")
```

```
plt.ylabel("Count")
plt.legend(title="pncome", loc="upper right")
plt.show()

## Analyse capital gain

#### Capital gain - Profit from selling assets for more than you paid

#### Get calculations of capital gain

data["capital-gain"].describe()

#### Visualize how capital gain is distributed among people

plt.figure(figsize=(10, 4))
data["capital-gain"].hist(bins=10)
plt.title("Capital Gain")
plt.xlabel("Capital gain value")
plt.ylabel("Count")

##### Most of the capital gain values in the dataset are 0. This can happen due to there is less number of people who are selling
##### their properties. If a person is able to sell a property than the expected amount, the capital gain is calculated.It shows their
##### is an outlier on the above histogram. Before dropping it, have to check whether  all outlier records are not from income
##### category that has an income less than or equals to 50k.

####

#### Check whether people who have capital gain greater than 80000 has income less than 50k

data[(data["capital-gain"] > 80000) & (data["income"] == "<=50K")].shape
```

Therefore can't detect them as outliers and drop the values

Visualize how many people have a zero capital gain or not

```
# Filter data
greater_than_0_count = (data["capital-gain"] > 0).sum()
equal_to_0_count = (data["capital-gain"] == 0).sum()

# Plotting
plt.bar(x=["0 capital gain", "Capital gain more than 0"],
        height=[equal_to_0_count, greater_than_0_count])
plt.xlabel("Capital Gain")
plt.ylabel("Count")
plt.title("Counts of Capital Gain")

# Annotate bars with counts
for i, count in enumerate([equal_to_0_count, greater_than_0_count]):
    plt.text(i, count, str(count), ha='center', va='bottom')

plt.show()
```

The above barchart shows that more than 90% of the people are not having a capital gain. (Their capital gain is 0).

Check how capital gain is distributed for people who's capital gain is more than 0

```
import matplotlib.pyplot as plt
import seaborn as sns
# Create the figure
plt.figure(figsize=(20, 9))

# Plot the histogram
```

```
sns.histplot(data=data[data['capital-gain'] != 0], x="capital-gain", hue="income", bins=30, palette='ocean', multiple='stack')


# Set labels and title
plt.xlabel("Capital Gain", fontsize=14)
plt.ylabel("Frequency", fontsize=14)
plt.title("Distribution of Capital Gain over 0 by Income", fontsize=16)


plt.show()


#### This shows that majority of people who have a capital gain have an income greater than 50k


## Capital Loss Analysis


data["capital-loss"].describe()


#### Visualize how capital loss is distributed among people


plt.figure(figsize=(10, 4))
data["capital-loss"].hist(bins=10)
plt.title("Capital Loss")
plt.ylabel("Count")


##### Most of the capital loss records are also 0. It happens due to a person will gain a capital loss when that person is not able,
##### to sell a property for the expected value.


####
#### Visualize how many people have a zero capital loss or not


# Filter data
greater_than_0_count = (data["capital-loss"] > 0).sum()
```

```
equal_to_0_count = (data["capital-loss"] == 0).sum()

# Plotting
plt.bar(x=["0 capital loss", "Capital loss more than 0"],
        height=[equal_to_0_count, greater_than_0_count])
plt.xlabel("Capital Loss")
plt.ylabel("Count")
plt.title("Counts of Capital Loss")

# Annotate bars with counts
for i, count in enumerate([equal_to_0_count, greater_than_0_count]):
    plt.text(i, count, str(count), ha='center', va='bottom')

plt.show()

##### Majority of people have got a capital loss of 0 (45938 values). There are only 2264 people, who have a capital
##### loss greater than 0.

####
#### Check how capital loss is distributed for people who's capital loss is more than 0

import matplotlib.pyplot as plt
import seaborn as sns

# Create the figure
plt.figure(figsize=(18, 6))

# Plot the histogram
sns.histplot(data=data[data['capital-loss'] != 0], x="capital-loss", hue="income", bins=30, palette='ocean', multiple="stack")

# Set labels and title
```

```
plt.xlabel("Capital Loss", fontsize=14)

plt.ylabel("Frequency", fontsize=14)

plt.title("Distribution of Capital Loss greater than 0 by Income", fontsize=16)


plt.show()


##### This histogram shows that people who have a capital loss greater than 0 can be found in both income categories

##### who have an income less than or equals 50k and greatet than 50k.


#  fnlwgt Analysis


data["fnlwgt"].describe().astype("int32")


#### Visualize how final weight is distributed


import matplotlib.pyplot as plt

import seaborn as sns


# Create the figure

plt.figure(figsize=(20, 10))


sns.histplot(data=data, x="fnlwgt", bins=20, hue="income", palette='ocean', multiple="stack")


plt.title("Final Weight", fontsize=16)

plt.xlabel("Value counts", fontsize=14)

plt.ylabel("Frequency", fontsize=14)

plt.show()


##### This histogram shows that majority of people who are having a capital loss (capital loss greater than 0) are from the category people

##### who earn less than  or equals to 50k.
```

Section 5 : Check duplicates of the dataset

Check for the duplicated values in the dataset

```
duplicates = data[data.duplicated(keep='first')]
```

check how many rows duplicated

```
duplicates.shape
```

display head rows of duplicates

```
duplicates.head()
```

Drop duplicates from the dataset

```
data.drop_duplicates(inplace=True, keep='first')
```

Get the shape of the dataset after removing data

```
data.shape
```

Section 6: Label Encoding Process

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

Encode labels in columns that consist with categorical data

```
from sklearn.preprocessing import LabelEncoder

final_data = data.copy()
```

```
encoders = {}

for column in final_data.columns:
    if final_data[column].dtype == 'object':
        encoder = LabelEncoder()
        final_data[column] = encoder.fit_transform(final_data[column])
        encoders[column] = encoder

final_data.head()

#### Display how each category in categorical columns has mapped with each encoded category

for column, encoder in encoders.items():
    print(f"{column}:")
    category_mapping = {label: category for label, category in enumerate(encoder.classes_)}
    print(category_mapping)

#### Put Standard Scaler to fnlwgt since it has values ranging in (12285 - 1490400)

from sklearn.preprocessing import StandardScaler

\begin{equation}
\begin{aligned}
\text{Standard Scaler formula:} \quad z &= \frac{x - \mu}{\sigma}
\end{aligned}
\end{equation}

scaler = StandardScaler()
final_data["fnlwgt"] = scaler.fit_transform(final_data[["fnlwgt"]])

#### Display final weight after fitting standard scaler
```

```
final_data["fnlwgt"].describe().astype("float64")

#### print first five values of final weight column

final_data["fnlwgt"].head()

### Normalization Process

#### Put Normalizer to capital gain

from sklearn.preprocessing import MinMaxScaler

#### describe data in capital gain before normalizing

final_data["capital-gain"].describe()

#### Apply the normalizer for capital-gain

capital_normalizer = MinMaxScaler()
final_data["capital-gain"] = capital_normalizer.fit_transform(final_data[["capital-gain"]])

#### Get the normalized value for example capital gain

capital_normalizer.data_max_, capital_normalizer.data_min_

# Given original value
original_value = np.array([96000])

# Reshape the original value to match the input shape for transform
original_value_reshaped = original_value.reshape(-1, 1)

# Use the transform method to obtain the normalized value
```

```
normalized_value = (original_value_reshaped - capital_normalizer.data_min_) /
(capital_normalizer.data_max_ - capital_normalizer.data_min_)

normalized_value

#### describe capital gain after normalization (capital gain)

final_data["capital-gain"].describe()

#### Put Normalizer for capital loss

from sklearn.preprocessing import MinMaxScaler

#### describe data in capital loss before normalizing

final_data["capital-loss"].describe()

#### Apply the normalizer for capital-loss

capital_loss_normalizer = MinMaxScaler()
final_data["capital-loss"] = capital_loss_normalizer.fit_transform(final_data[["capital-loss"]])

capital_loss_normalizer.data_max_ , capital_loss_normalizer.data_min_

#### describe capital gain after normalization (capital loss)
# Given original value
original_value = np.array([980])

# Reshape the original value to match the input shape for transform
original_value_reshaped = original_value.reshape(-1, 1)

# Use the transform method to obtain the normalized value
```

```
normalized_value = (original_value_reshaped - capital_loss_normalizer.data_min_) / (capital_loss_normalizer.data_max_ - capital_loss_normalizer.data_min_)

normalized_value

final_data["capital-loss"].describe()

## Section 7 :Check correlation with all columns

#### Get the pearson correlation for all columns with target column income

cor = final_data.corr(numeric_only=True, method="pearson")
cor["income"]

#### Plot the Pearson correlation of all the columns in a heatmap

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 10))
## plot the heatmap with two decimal point correlations
sns.heatmap(data=cor, annot=True, fmt=".2f")
plt.title("Heatmap for Pearson Correlation")
plt.show()

##### Since this is a supervised problem, correlation values are considered for the feature selection process

## Section 8 : Feature Selection process

#### assign the input column values to X and target variable column values to Y

X = final_data.drop('income', axis=1)
```

```
Y = final_data['income'].map({0: "<=50K", 1: ">50K"})

#### print first five values of both X and Y

X.head()

Y.head()

#### Drop columns based on low correlations and similar correlation patterns with other columns

X.drop(columns=["fnlwgt", "education", "sex", "workclass"], inplace=True)

#### Display first five column of variable dataset after feature reduction

X.head()

## Section 9.  Try to check models before doing imbalanced learning

### split the dataset

from sklearn.model_selection import train_test_split

train_X, test_X, train_Y, test_Y = train_test_split(X, Y, test_size=0.3, random_state=42)

train_X.shape, train_Y.shape, test_X.shape, test_Y.shape

### Apply random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

model_random_forest = RandomForestClassifier(random_state=42)
```

```
model_random_forest.fit(train_X, train_Y)

#### Get the training accuracy

model_random_forest.score(train_X, train_Y)

#### Get the testing Accuracy

from sklearn.metrics import accuracy_score

accuracy_score(test_Y, model_random_forest.predict(test_X))

##### This shows there is a Overfitting issue when the target variable is not balanced

#### Check whether it works properly by doing a parameterized optimization

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
import random

# parameter grid
param_grid = {
    'n_estimators': list(range(5, 50)),
    'max_features': ['log2', 'auto', 'sqrt'],
    'min_samples_split': list(range(6, 15)),
    'min_samples_leaf': list(range(4, 15))
}

# Create a RandomForestClassifier instance
rf_classifier = RandomForestClassifier(n_jobs=5)

random.seed(42)
```

```
# Create RandomizedSearchCV instance
random_search = RandomizedSearchCV(estimator=rf_classifier,
param_distributions=param_grid, n_iter=50, cv=5, n_jobs=5,
                                   random_state=42)

# Fit the model
random_search.fit(train_X, train_Y)

# Print best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)

#### Checked model after hyperparameterized optimization for RandomForest Classifier

model_random_forest = RandomForestClassifier(n_estimators=43,
n_jobs=5,min_samples_split= 8, min_samples_leaf=10, max_features='sqrt',
random_state=42)
model_random_forest.fit(train_X, train_Y)

#### get training accuracy

model_random_forest.score(train_X, train_Y)

#### get testing accuracy
from sklearn.metrics import accuracy_score

accuracy_score(test_Y, model_random_forest.predict(test_X))

#### get the roc and auc curve before doing imbalanced learning

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
# Convert labels to binary form
test_y_binary = (test_Y == '>50K').astype(int)

# Make predictions on the test set
predictions = model_random_forest.predict_proba(test_X)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(test_y_binary, predictions)

# Calculate the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 5))
sns.set_style("whitegrid")
plt.plot(fpr, tpr, lw=2, label='ROC curve (area (AUC value) = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('(ROC) Curve for Random Forest Classifier (Before balancing)')
plt.legend(loc="lower right")
plt.show()

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_Y, model_random_forest.predict(test_X))
labels = ["<=50K", ">50K"]
sns.heatmap(cm , annot=True , fmt=".0f", xticklabels=labels,
yticklabels=labels,cmap=plt.cm.Blues)
plt.xlabel("Predicted Values")
```

```
plt.ylabel("True Label")
plt.title("Random Forest Confusion Matrix (Before imbalanced learning)")

from sklearn.metrics import classification_report

report = pd.DataFrame(classification_report(test_Y, model_random_forest.predict(test_X), output_dict=True))
report

#### Now it gets a good accuracy, dataset is imbalanced,therefore there is a probability  for the low value count class to give wrong
#### predictions. For class that has an income greater than 50k has very low , precision, recall and f1-score values compaerd to the class that
#### has an income less than or equals to 50k. Therefore the dataset has to be balanced and train the models.

### Apply Bernouli Naive Bayes Classifier

from sklearn.naive_bayes import BernoulliNB

bernoulli_model = BernoulliNB()

bernoulli_model.fit(train_X, train_Y)

#### get training accuracy
bernoulli_model.score(train_X, train_Y)

#### get testing accuracy

from sklearn.metrics import accuracy_score

accuracy_score(test_Y, bernoulli_model.predict(test_X))
```

```
#### Check the model after parameterized optimization for Naive Bayes

from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import GridSearchCV

clf = BernoulliNB()

# Define the parameter grid to search
param_grid = {
            'alpha': [0.1, 0.5, 1.0, 2.0],  # Smoothing parameter
            'binarize': [0.0, 0.5, 1.0],  # Binarization threshold
            'fit_prior': [True, False],  # Whether to learn class prior probabilities
             'class_prior': [None, [0.5, 0.5]],  # Prior probabilities for the classes
           }

# Perform grid search with cross-validation
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(train_X, train_Y)

# Print the best parameters found
print("Best Parameters:", grid_search.best_params_)

bernoulli_model = BernoulliNB()
bernoulli_model.fit(train_X, train_Y)
#### get training accuracy

bernoulli_model.score(train_X, train_Y)

#### get testing accuaracy

from sklearn.metrics import accuracy_score
```

```
accuracy_score(test_Y, bernoulli_model.predict(test_X))

#### get the roc and auc curve before doing imbalanced learning

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Convert labels to binary form
test_y_binary = (test_Y == '>50K').astype(int)

# Make predictions on the test set
predictions = bernoulli_model.predict_proba(test_X)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(test_y_binary, predictions)

# Calculate the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 5))
sns.set_style("whitegrid")
plt.plot(fpr, tpr, lw=2, label='ROC curve (area(AUC value) = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('(ROC) Curve for Bernoulli Naive Bayes (Before balancing)')
plt.legend(loc="lower right")
plt.show()
```

```
cm = confusion_matrix(test_Y, bernoulli_model.predict(test_X))

cm


labels = ["<=50K", ">50K"]

sns.heatmap(cm , annot=True , fmt=".0f", xticklabels=labels, yticklabels=labels,
cmap=plt.cm.Blues)

plt.xlabel("Predicted Values")

plt.ylabel("True Label")

plt.title("Naive bayes Confusion Matrix (before imbalanced learning)")


report = pd.DataFrame(classification_report(test_Y, bernoulli_model.predict(test_X),
output_dict=True))

report


####  Even the parameters are optimized, the above confusion matrix show that, it predict
income greater than 50K class as income


####  less than or equals to 50k in most of the times. In addition, the recall, f1-score values of
income greater than 50k class is very low


####  compared to income less than or equals to 50k. Therefore, the dataset (target variable)
should be balanced and train the models.


## Section 10 : Imbalanced Learning


Y.unique()

#### Get the value count of target column


Y.value_counts()


#### Distribution of income data in the imbalanced dataset


import seaborn as sns

import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(7.5, 5))
# Given income data
income_counts = [36529, 11619]
income_categories = ['<=50K', '>50K']

# Create bar chart using Seaborn
ax = sns.barplot(x=income_categories, y=income_counts)

# Add value counts on top of each bar
for i, v in enumerate(income_counts):
    ax.text(i, v + 1000, str(v), ha='center', va='bottom')

# Add title and labels
plt.title('Income Distribution (Imbalance target variable)')
plt.xlabel('Income')
plt.ylabel('Count')

# Show plot
plt.show()

#### Use SMOTE oversampling method to balance the dataset

from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, Y)

#### Check whether dataset is balanced, after applying SMOTE

y_resampled.value_counts()

X_resampled.head()
```

Section 11 :Train test Split

```
from sklearn.model_selection import train_test_split

#### Split the dataset for training and testing with test size 30%

train_x, test_x, train_y, test_y = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

#### Check the training and testing data set shapes

train_x.shape, test_x.shape, train_y.shape, test_y.shape

#### Check whether training dataset is balanced

train_y.value_counts()

#### Distribution of the income target variable in the training dataset

import seaborn as sns
import matplotlib.pyplot as plt



plt.figure(figsize=(7.5, 5))
# Given income data
income_counts = [25449, 25691]
income_categories = ['<=50K', '>50K']

# Create bar chart using Seaborn
ax = sns.barplot(x=income_categories, y=income_counts)

# Add value counts on top of each bar
```

```
for i, v in enumerate(income_counts):
    ax.text(i, v + 1000, str(v), ha='center', va='bottom')

# Add title and labels
plt.title('Income Distribution (training dataset)', pad="20")
plt.xlabel('Income')
plt.ylabel('Count')

# Show plot
plt.show()

#### Check whether testing dataset is balanced

test_y.value_counts()

#### Distribution of the income target variable in the testing dataset

import seaborn as sns
import matplotlib.pyplot as plt



plt.figure(figsize=(7.5, 5))
# Given income data
income_counts = [11080, 10838]
income_categories = ['<=50K', '>50K']

# Create bar chart using Seaborn
ax = sns.barplot(x=income_categories, y=income_counts)

# Add value counts on top of each bar
for i, v in enumerate(income_counts):
    ax.text(i, v + 1000, str(v), ha='center', va='bottom')
```

```
# Add title and labels
plt.title('Income Distribution (testing dataset)', pad="25")
plt.xlabel('Income')
plt.ylabel('Count')

# Show plot
plt.show()

## Section 12:  Apply Random forest Classifier

from sklearn.ensemble import RandomForestClassifier

#### create a model and see how it works before tuning parameters

model = RandomForestClassifier(random_state=42)
model.fit(train_x, train_y)

#### check the training score

model.score(train_x, train_y)

#### check testing score
model.score(test_x, test_y)

#### Still it shows that model is overfitting

#### Use randomized search cv to find best parameters

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

np.random.seed(42)
```

```
# Define the parameter grid
param_grid = {
    'n_estimators': list(range(5, 50)),
    'max_features': ['log2', 'auto', 'sqrt'],
    'min_samples_split': list(range(6, 15)),
    'min_samples_leaf': list(range(4, 15))
}

# Create a RandomForestClassifier instance
rf_classifier = RandomForestClassifier(n_jobs=5)

# Create RandomizedSearchCV instance
random_search = RandomizedSearchCV(estimator=rf_classifier,
param_distributions=param_grid, n_iter=50, cv=5, n_jobs=5,
                                   random_state=42)

# Fit the model
random_search.fit(train_x, train_y)

# Print best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)

#### Assign the Random Forest Classifier model with parameters

model_forest = RandomForestClassifier(n_estimators=37, n_jobs=5, max_features="sqrt",
min_samples_split=11, min_samples_leaf=4,
                                      random_state=42)

#### Fit the Random Forest Model

model_forest.fit(train_x, train_y)
```

```
#### Get the training accuracy

model_forest.score(train_x, train_y)

#### Get the testing accuracy

model_forest.score(test_x, test_y)

### Test some values with the model

X_resampled.loc[0], y_resampled[0]

model_forest.predict([[39, 13, 2, 0, 1, 4, 0.02174, 0, 40, 1]])

X_resampled.loc[10], y_resampled[10]

model_forest.predict([[37, 10, 1, 3, 0, 2, 0, 0, 80, 1]])

for column, encoder in encoders.items():
    print(f"{column}:")
    category_mapping = {label: category for label, category in enumerate(encoder.classes_)}
    print(category_mapping)

#### Check an example less than or equals to 50K with Random Forest Classifier

data.loc[0]

#### Get the normalized value of the capital gain

# Given original value
original_value = np.array([2174])
```

```
# Reshape the original value to match the input shape for transform
original_value_reshaped = original_value.reshape(-1, 1)

# Use the transform method to obtain the normalized value
normalized_value = (original_value_reshaped - capital_normalizer.data_min_) /
(capital_normalizer.data_max_ - capital_normalizer.data_min_)

normalized_value

model_forest.predict([[39, 13, 3, 0, 1, 4, 0.02174022, 0, 39 , 1]])

#### Check an example more than 50K with Random Forest Classifier

data.loc[7]

model_forest.predict([[52, 13, 1, 3, 0, 4, 0, 0, 45, 1]])

### Evaluate the random forest model

from sklearn.metrics import classification_report
#### Evaluate Precision, Recall, f1-score with a classification report

classification_report = pd.DataFrame(classification_report(test_y,
model_forest.predict(test_x) , output_dict=True))
classification_report

#### When the dataset is balanced and trained, now it shows both classes (income >50K and
<=50k) has
#### precision, recall, f1-score values greater than 0.8. therefore, wrong prediction issue for
>50k class has solved

from sklearn.metrics import confusion_matrix
```

Get the confusion matrix for Random Forest Classifier Model

```
cm = confusion_matrix(test_y, model_forest.predict(test_x))

labels = ["<=50K", ">50K"]
sns.heatmap(cm , annot=True , fmt=".0f", xticklabels=labels, yticklabels=labels, cmap=plt.cm.Blues)
plt.xlabel("Predicted Values")
plt.ylabel("True Label")
plt.title("Random Forest Confusion Matrix")
```

ROC curve and AUC value for RandomForest Classifier

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Convert labels to binary form
test_y_binary = (test_y == '>50K').astype(int)

# Make predictions on the test set
predictions = model_forest.predict_proba(test_x)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(test_y_binary, predictions)

# Calculate the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 5))
sns.set_style("whitegrid")
plt.plot(fpr, tpr, lw=2, label='ROC curve (area (AUC value)= %0.2f)' % roc_auc)
```

```
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('(ROC) Curve for Random Forest Classifier')
plt.legend(loc="lower right")
plt.show()

### Saving the Model and evaluate some values

import pickle

with open('random_forest_model.pkl', 'wb') as rf:
    pickle.dump(model_forest, rf)

### Load model again and check for some values

with open('random_forest_model.pkl', 'rb') as rf:
    model_forest = pickle.load(rf)
model_forest.predict([[52, 13, 1, 3, 0, 4, 0, 0, 45, 1]])

### Section 13 :Check what is the best Naive Bayes Algorithm supports well by doing a cross validation

#### Basically Bernoulli Naive Bayes model performs well for binary classification, but this checks the other models as well

from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# Define Naive Bayes models
gaussian_nb = GaussianNB()
```

```
multinomial_nb = MultinomialNB()
bernoulli_nb = BernoulliNB()

# Perform cross-validation and print accuracies
def perform_cross_validation(model, name):
    scores = cross_val_score(model, train_x, train_y, cv=5)
    print(f"Cross-validation accuracy for {name}: {scores.mean()}")

# Perform cross-validation for Gaussian Naive Bayes model
perform_cross_validation(gaussian_nb, "Gaussian Naive Bayes")

# Perform cross-validation for Multinomial Naive Bayes model
perform_cross_validation(multinomial_nb, "Multinomial Naive Bayes")

# Perform cross-validation for Bernoulli Naive Bayes model
perform_cross_validation(bernoulli_nb, "Bernoulli Naive Bayes")
```

This shows that Bernoulli Naive Bayes Performs well in this Binary Classification Approach

Section 14 :Bernoulli Naive Bayes Model (Since It is Binary Classification)

```
from sklearn.naive_bayes import BernoulliNB

from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import GridSearchCV

clf = BernoulliNB()

# Define the parameter grid to search
param_grid = {
        'alpha': [0.1, 0.5, 1.0, 2.0],  # Smoothing parameter
```

```
    'binarize': [0.0, 0.5, 1.0],  # Binarization threshold
    'fit_prior': [True, False],  # Whether to learn class prior probabilities
     'class_prior': [None, [0.5, 0.5]],  # Prior probabilities for the classes
    }

# Perform grid search with cross-validation
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(train_x, train_y)

# Print the best parameters found
print("Best Parameters:", grid_search.best_params_)

#### Assign the BernoulliNB model with parameters

model_bayes = BernoulliNB(alpha=0.1, binarize=0.0, class_prior=None, fit_prior=True)

#### Train bernouli Naive Bayes Model

model_bayes.fit(train_x, train_y)

#### Get the training score of the Naive Bayes Model

model_bayes.score(train_x, train_y)

#### Get the testing score of the naive Bayes Model

model_bayes.score(test_x, test_y)

### Test some values with the model

X_resampled.loc[0], y_resampled[0]
```

```
model_bayes.predict([[39, 13, 2, 0, 1, 4, 0.02174022, 0, 40, 1]])

X_resampled.loc[10], y_resampled[10]

model_bayes.predict([[37, 10, 1, 3, 0, 2, 0, 0, 80, 1]])

for column, encoder in encoders.items():
    print(f"{column}:")
    category_mapping = {label: category for label, category in enumerate(encoder.classes_)}
    print(category_mapping)

#### Check an example less than or equals to 50K for Naive Bayes Model

data.loc[0]

model_bayes.predict([[39, 13, 3, 0, 1, 4, 0.02174022, 0, 40 , 1]])

#### Check an example greater than 50K for Naive Bayes Model

data.loc[7]
model_bayes.predict([[52, 13, 1, 3, 0, 4, 0, 0, 45, 1]])

### Evaluate with matrices

from sklearn.metrics import accuracy_score

#### Get the testing Accuracy Score

accuracy_score(test_y, model_bayes.predict(test_x))

#### Evaluate Precision, Recall, f1-score with a classification report
```

```
from sklearn.metrics import classification_report

classification_report = pd.DataFrame(classification_report(test_y,
model_bayes.predict(test_x) , output_dict=True))
classification_report

#### Get the confusion matrix of the Naive Bayes Model

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_y, model_bayes.predict(test_x))
cm

labels = ["<=50K", ">50K"]
sns.heatmap(cm , annot=True , fmt=".0f", xticklabels=labels, yticklabels=labels,
cmap=plt.cm.Blues)
plt.xlabel("Predicted Values")
plt.ylabel("True Label")
plt.title("Bernoulli Naive bayes Confusion Matrix")

### Evaluate the Bernouli Naive Bayes Model with ROC and AUC curves
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Convert labels to binary form
test_y_binary = (test_y == '>50K').astype(int)

# Make predictions on the test set
predictions = model_bayes.predict_proba(test_x)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(test_y_binary, predictions)
```

```
# Calculate the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 5))
sns.set_style("whitegrid")
plt.plot(fpr, tpr, lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], lw=2, linestyle='--', color='gray')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('(ROC) Curve for Bernouli Naive Bayes')
plt.legend(loc="lower right")
plt.show()

### Save the Naive Bayes Model as a pickle file

import pickle

with open('naive_bayes_model.pkl', 'wb') as nb:
    pickle.dump(model_bayes, nb)

### Load the Naive Bayes model and check for some values

with open('naive_bayes_model.pkl', 'rb') as nb:
    model_bayes = pickle.load(nb)

model_bayes.predict([[37, 10, 1, 3, 0, 2, 0, 0, 80, 1]])

### Section 15 : (Comparison of Naive Bayes and Random Forest models through ROC curves)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc

# Convert labels to binary form
test_y_binary = (test_y == '>50K').astype(int)

# Make predictions on the test set
predictions_bayes = model_bayes.predict_proba(test_x)[:, 1]
predictions_forest = model_forest.predict_proba(test_x)[:, 1]

# Calculate the ROC curve and AUC for Bernoulli Naive Bayes
fpr_bayes, tpr_bayes, thresholds_bayes = roc_curve(test_y_binary, predictions_bayes)
roc_auc_bayes = auc(fpr_bayes, tpr_bayes)

# Calculate the ROC curve and AUC for Random Forest
fpr_forest, tpr_forest, thresholds_forest = roc_curve(test_y_binary, predictions_forest)
roc_auc_forest = auc(fpr_forest, tpr_forest)

# Plot the ROC curves
plt.figure(figsize=(8, 5))
sns.set_style("whitegrid")
sns.lineplot(x=fpr_bayes, y=tpr_bayes, lw=2, label=f'Bernoulli Naive Bayes (AUC =
{roc_auc_bayes:.2f})')
sns.lineplot(x=fpr_forest, y=tpr_forest, lw=2, label=f'Random Forest (AUC =
{roc_auc_forest:.2f})')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc="lower right")
```

```
plt.show()
```

The above ROC curve plots shows that, Random Forest Classifier model performs better than the Bernoulli Naive

Bayes model for the testing data.