Department of Computer Engineering

CSE454 - Data Mining

Fall 2022-2023

Project Report

21.01.2023

Demo url : https://youtu.be/JAAS6J__mNA

Abdurrahman Bulut

1901042258

CONTENTS

Introduction

In this project, I will be using a dataset of adult income from Kaggle to train a logistic regression classifier that can predict whether an individual earns more than $50,000 per year or less. The dataset has 48,842 data points, 14 independent variables, and 1 dependent variable. I will be coding the logistic regression model from scratch using Python and evaluating its performance using various metrics. This is a classic binary classification problem and logistic regression is a widely used method for solving such problems. The goal of this project is to gain a deeper understanding of logistic regression and its implementation, as well as to demonstrate the capabilities of machine learning in solving real-world problems. I will be exploring the data, analyzing the features, and comparing the performance of the logistic regression model to other possible solutions. By the end of this project, I will have a trained classifier that can accurately predict whether an individual earns more than $50,000 per year or less, based on their characteristics.

❖ Data Description

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). The prediction task is to determine whether a person makes over $50K a year.

❖ Feature Description

Categorical Attributes	Descriptions
workclass	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
marital-status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspect, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex	Female, Male
native-country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad Tobago, Peru, Hong, Holland-Netherlands.
Continuous Attributes	Descriptions
age	Age of an individual
fnlwgt	The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by the Population Division here at the Census Bureau.
capital-gain	continuous
capital-loss	continuous
hours-per-week	Individual's working hour per week

Analyze The Dataset

First 5 rows of the dataset:

```
df1.head(5)
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K

Last 5 rows of the dataset:

```
df1.tail(5)
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
48840	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
48841	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

The dataset consists of 48842 rows and 15 features.The target feature/class is “income” feature. This is our dependent variable. Other 14 variables are independent variables.

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              48842 non-null  int64
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64
 11  capital-loss     48842 non-null  int64
 12  hours-per-week   48842 non-null  int64
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

There is no null value for any feature.

Data Cleaning

Fixing the common “nan” values: Nan values were as ? in data. Hence I fixed this with the most frequent element(mode) in the entire dataset.

```
attrib, counts = np.unique(df1['workclass'], return_counts = True)
most_freq_attrib = attrib[np.argmax(counts, axis = 0)]
df1['workclass'][df1['workclass'] == '?'] = most_freq_attrib

attrib, counts = np.unique(df1['occupation'], return_counts = True)
most_freq_attrib = attrib[np.argmax(counts, axis = 0)]
df1['occupation'][df1['occupation'] == '?'] = most_freq_attrib

attrib, counts = np.unique(df1['native-country'], return_counts = True)
most_freq_attrib = attrib[np.argmax(counts, axis = 0)]
df1['native-country'][df1['native-country'] == '?'] = most_freq_attrib
```

Income column will be classified into 0’s and 1’s by looking at the condition.

```
df1['income']=df1['income'].map({'<=50K': 0, '>50K': 1, '<=50K.': 0, '>50K.': 1})
df1.head()
```

Duplicated values removed from the dataset.

```
dup=df1.duplicated().any()
```

```
print("Are there any duplicate Values in the data:", dup)
```

```
Are there any duplicate Values in the data: True
```

```
df1=df1.drop_duplicates()
df1
```

The Education category was a little bit complicated. I decreased the number of education level from 17 to 6.

```
df1['education'].replace('Preschool', 'dropout',inplace=True)
df1['education'].replace('10th', 'dropout',inplace=True)
df1['education'].replace('11th', 'dropout',inplace=True)
df1['education'].replace('12th', 'dropout',inplace=True)
df1['education'].replace('1st-4th', 'dropout',inplace=True)
df1['education'].replace('5th-6th', 'dropout',inplace=True)
df1['education'].replace('7th-8th', 'dropout',inplace=True)
df1['education'].replace('9th', 'dropout',inplace=True)
df1['education'].replace('HS-Grad', 'HighGrad',inplace=True)
df1['education'].replace('HS-grad', 'HighGrad',inplace=True)
df1['education'].replace('Some-college', 'CommunityCollege',inplace=True)
df1['education'].replace('Assoc-acdm', 'CommunityCollege',inplace=True)
df1['education'].replace('Assoc-voc', 'CommunityCollege',inplace=True)
df1['education'].replace('Bachelors', 'Bachelors',inplace=True)
df1['education'].replace('Masters', 'Masters',inplace=True)
df1['education'].replace('Prof-school', 'Masters',inplace=True)
df1['education'].replace('Doctorate', 'Doctorate',inplace=True)
```

```
] df1[['education', 'educational-num']].groupby(['education'], as_index=False).mean().sort_values(by='educational-num', ascending=False)
```

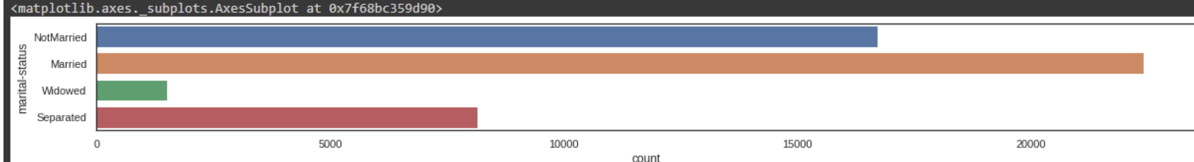
	education	educational-num
2	Doctorate	16.000000
4	Masters	14.238968
0	Bachelors	13.000000
1	CommunityCollege	10.362322
3	HighGrad	9.000000
5	dropout	5.617440

The Marital-status category also was a little bit complicated. I decreased the number of Marital-status from 7 to 4.

```
df1['marital-status'].replace('Never-married', 'NotMarried',inplace=True)
df1['marital-status'].replace(['Married-AF-spouse'], 'Married',inplace=True)
df1['marital-status'].replace(['Married-civ-spouse'], 'Married',inplace=True)
df1['marital-status'].replace(['Married-spouse-absent'], 'NotMarried',inplace=True)
df1['marital-status'].replace(['Separated'], 'Separated',inplace=True)
df1['marital-status'].replace(['Divorced'], 'Separated',inplace=True)
df1['marital-status'].replace(['Widowed'], 'Widowed',inplace=True)
```

```
fig = plt.figure(figsize=(20,2))
plt.style.use('seaborn-ticks')
sns.countplot(y="marital-status", data=df1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68bc359d90>
```

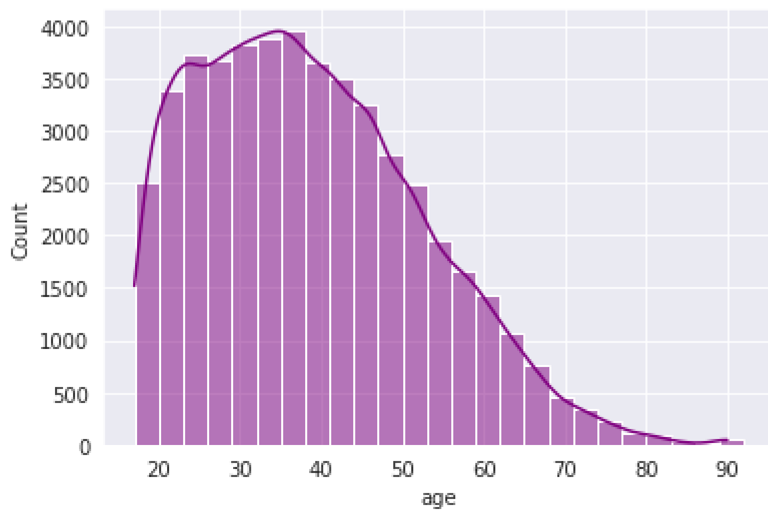


The Native-country category was a bit complicated. I decreased the number of Native-country from 42 to 5.

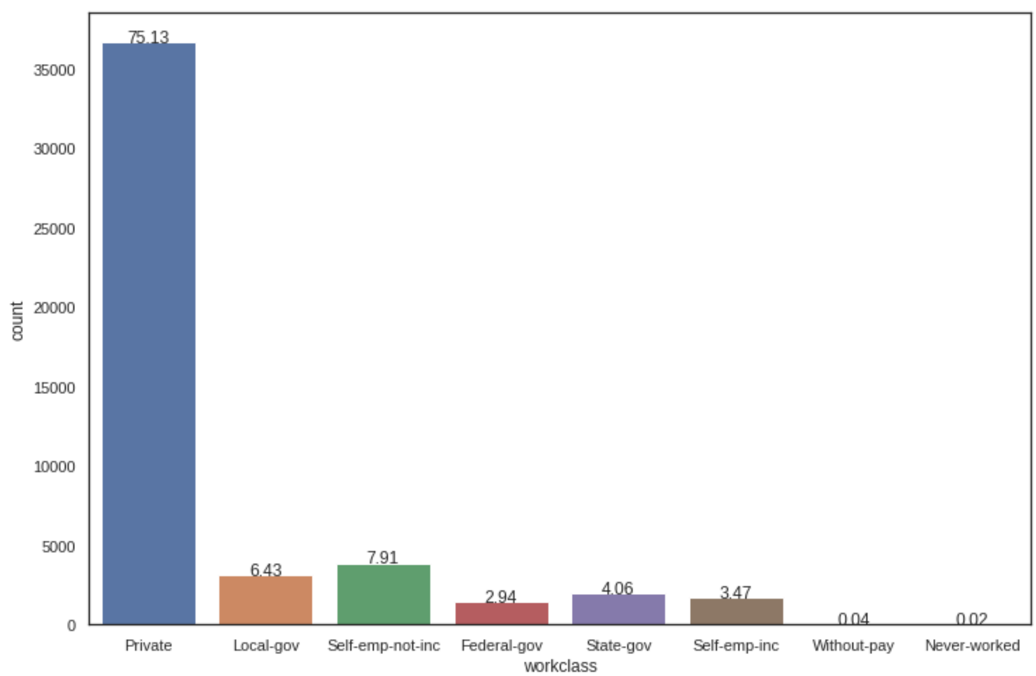
```
df1['native-country'].replace('Canada', 'North America',inplace=True)
df1['native-country'].replace('Cuba', 'North America',inplace=True)
df1['native-country'].replace('Dominican-Republic', 'North America',inplace=True)
df1['native-country'].replace('El-Salvador', 'North America',inplace=True)
df1['native-country'].replace('Guatemala', 'North America',inplace=True)
df1['native-country'].replace('Haiti', 'North America',inplace=True)
df1['native-country'].replace('Honduras', 'North America',inplace=True)
df1['native-country'].replace('Jamaica', 'North America',inplace=True)
df1['native-country'].replace('Mexico', 'North America',inplace=True)
df1['native-country'].replace('Nicaragua', 'North America',inplace=True)
df1['native-country'].replace('Outlying-US(Guam-USVI-etc)', 'North America',inplace=True)
df1['native-country'].replace('Puerto-Rico', 'North America',inplace=True)
df1['native-country'].replace('Trinadad&Tobago', 'North America',inplace=True)
df1['native-country'].replace('United-States', 'North America',inplace=True)

df1['native-country'].replace('Cambodia', 'Asia',inplace=True)
df1['native-country'].replace('China', 'Asia',inplace=True)
df1['native-country'].replace('Hong', 'Asia',inplace=True)
df1['native-country'].replace('India', 'Asia',inplace=True)
df1['native-country'].replace('Iran', 'Asia',inplace=True)
df1['native-country'].replace('Japan', 'Asia',inplace=True)
df1['native-country'].replace('Laos', 'Asia',inplace=True)
df1['native-country'].replace('Philippines', 'Asia',inplace=True)
df1['native-country'].replace('Taiwan', 'Asia',inplace=True)
df1['native-country'].replace('Thailand', 'Asia',inplace=True)
df1['native-country'].replace('Vietnam', 'Asia',inplace=True)

df1['native-country'].replace('Columbia', 'South America',inplace=True)
df1['native-country'].replace('Ecuador', 'South America',inplace=True)
df1['native-country'].replace('Peru', 'South America',inplace=True)

df1['native-country'].replace('England', 'Europe',inplace=True)
df1['native-country'].replace('France', 'Europe',inplace=True)
df1['native-country'].replace('Germany', 'Europe',inplace=True)
df1['native-country'].replace('Greece', 'Europe',inplace=True)
df1['native-country'].replace('Holand-Netherlands', 'Europe',inplace=True)
df1['native-country'].replace('Hungary', 'Europe',inplace=True)
df1['native-country'].replace('Ireland', 'Europe',inplace=True)
df1['native-country'].replace('Italy', 'Europe',inplace=True)
```

```
df1['native-country'].replace('Italy', 'Europe',inplace=True)
df1['native-country'].replace('Poland', 'Europe',inplace=True)
df1['native-country'].replace('Portugal', 'Europe',inplace=True)
df1['native-country'].replace('Scotland', 'Europe',inplace=True)
df1['native-country'].replace('Yugoslavia', 'Europe',inplace=True)

df1['native-country'].replace('South', 'Other',inplace=True)
df1['native-country'].replace('?', 'Other',inplace=True)
```

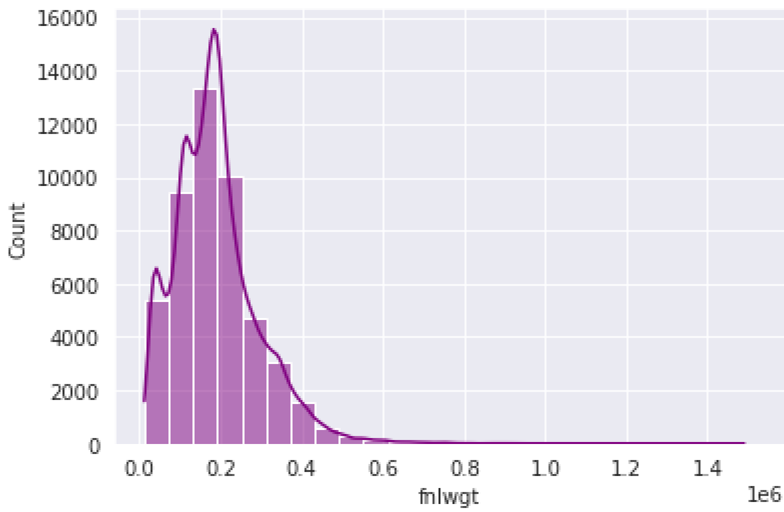
Exploratory Data Analysis - EDA

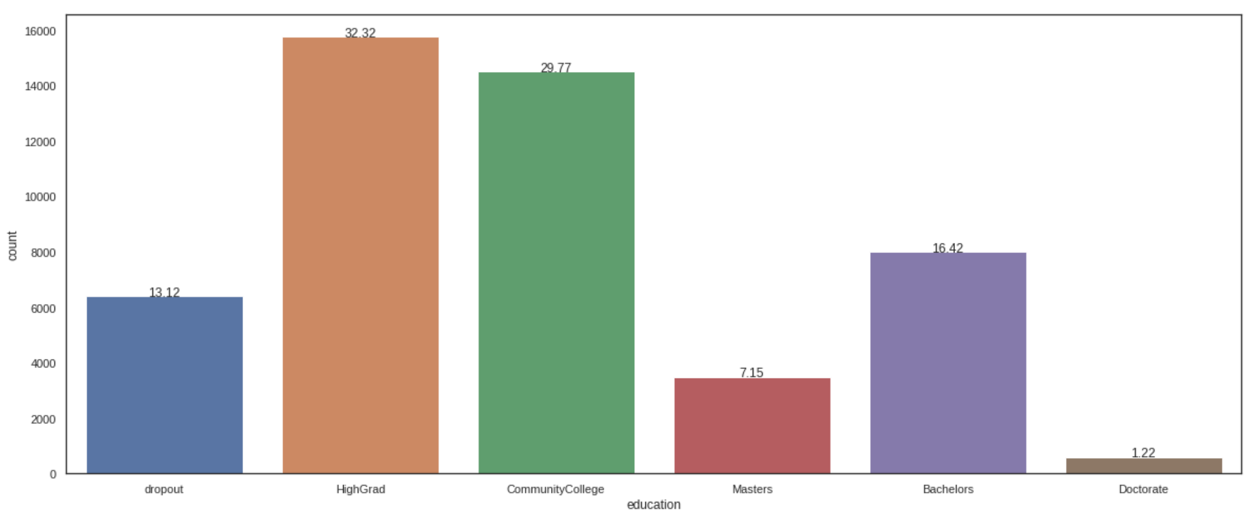


It is right-skewed (But this is totally fine as younger adults earn wages not the older ones). Minimum and Maximum age of the people is 17 and 90 respectively. "age" attribute is not symmetric.



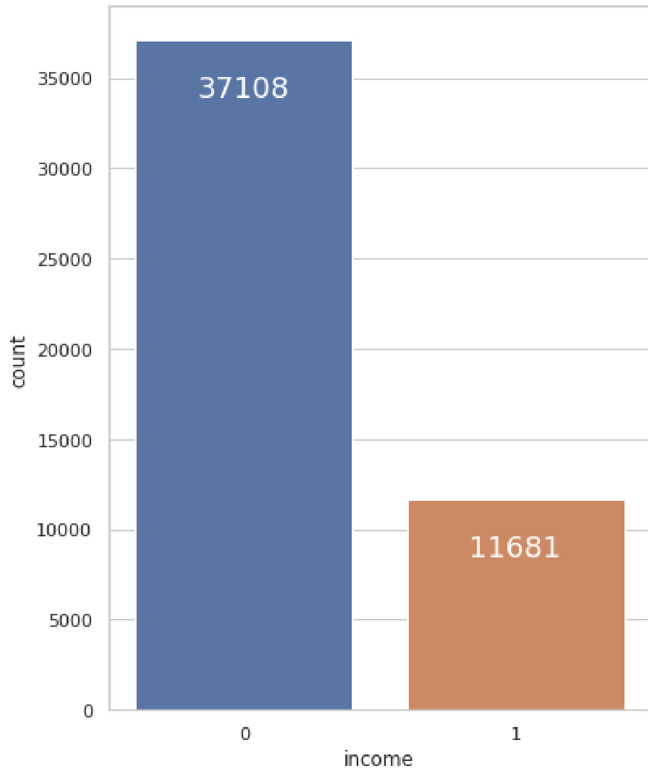
It is the workclass distribution graph. The private number is almost three-fourths of the total.



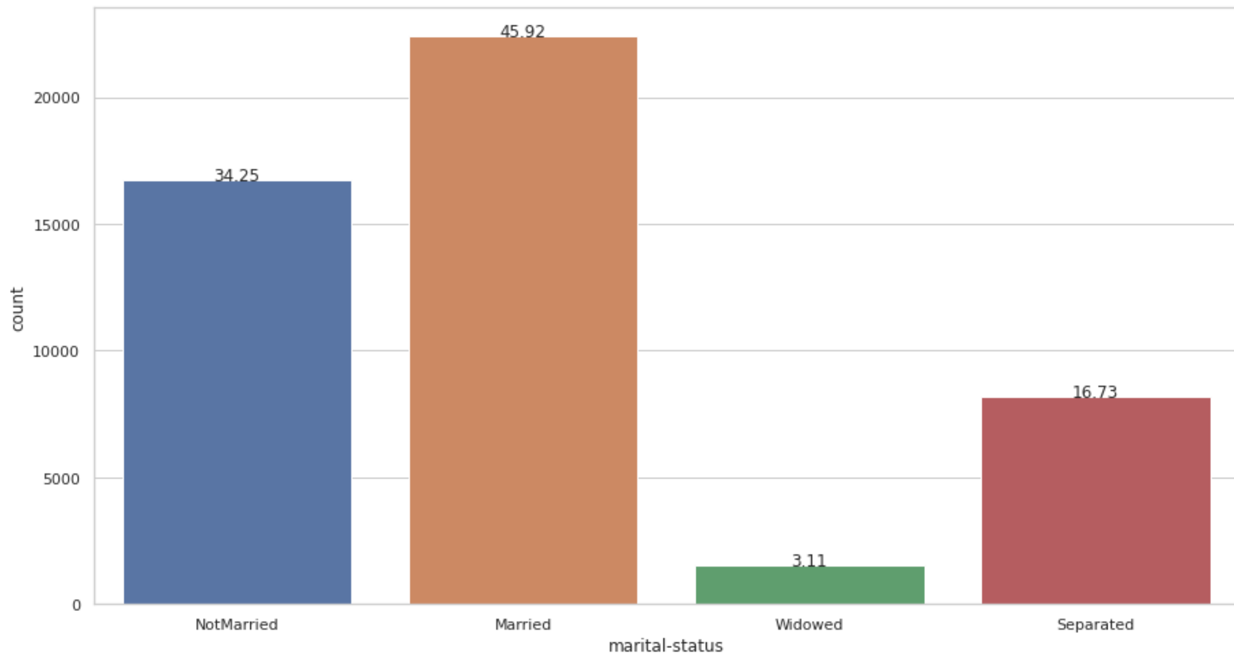


There are 6 unique categories present in the **education** attribute.

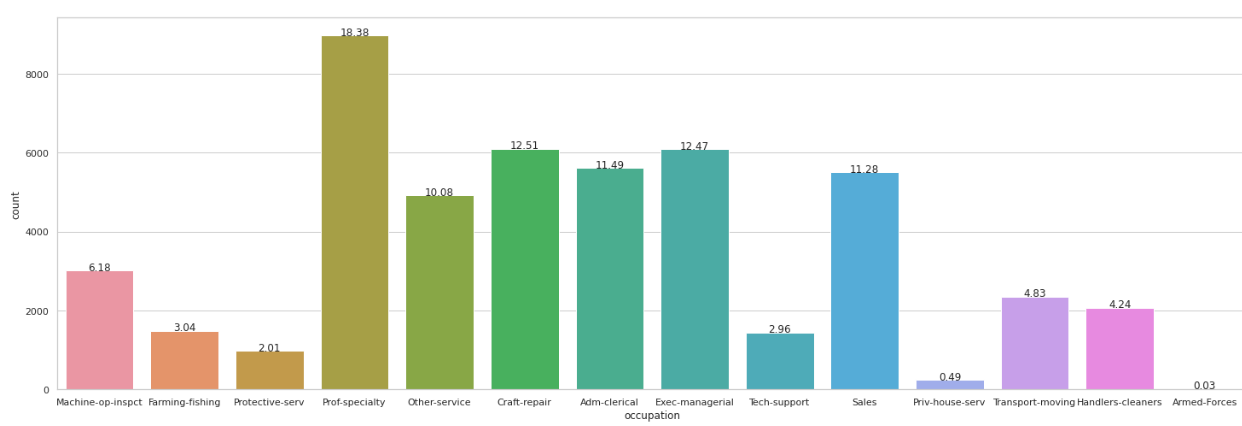
- *H-grad* has 32.32% of all the education attributes.
- *Doctorate* has 1.22% of all the education attributes.



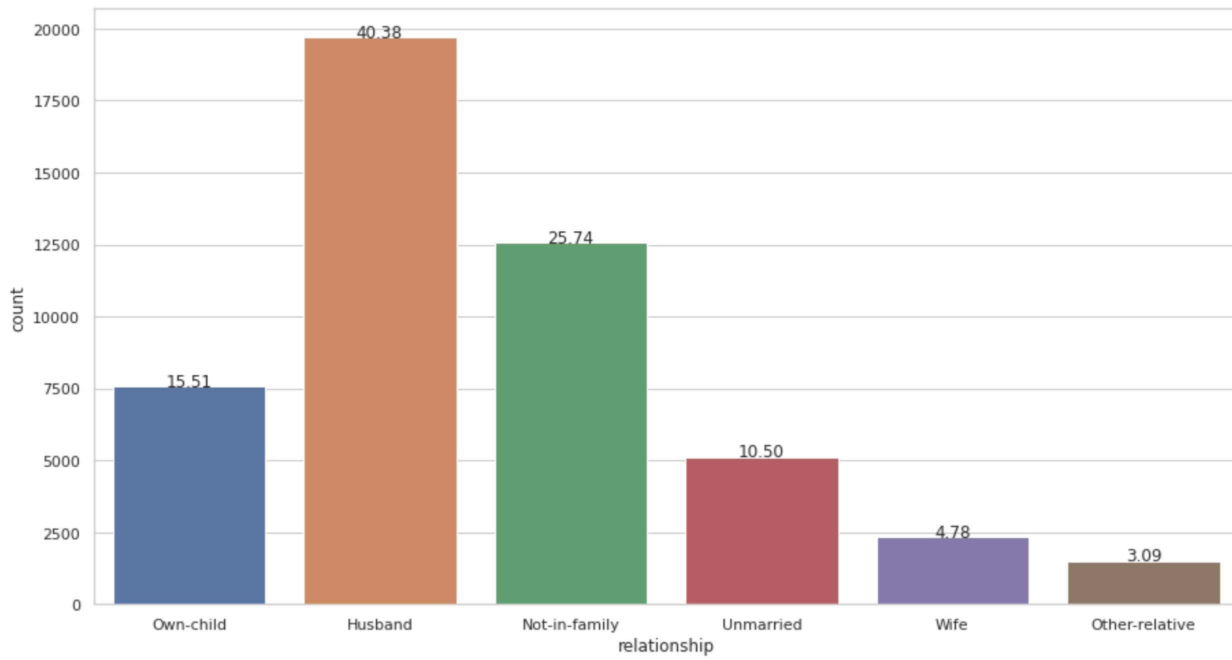
This dataset is not balanced , 11681 of them belong to income group 1 (who earns more than 50k) and 27108 fall under the income group 0 (who earns less than 50k).



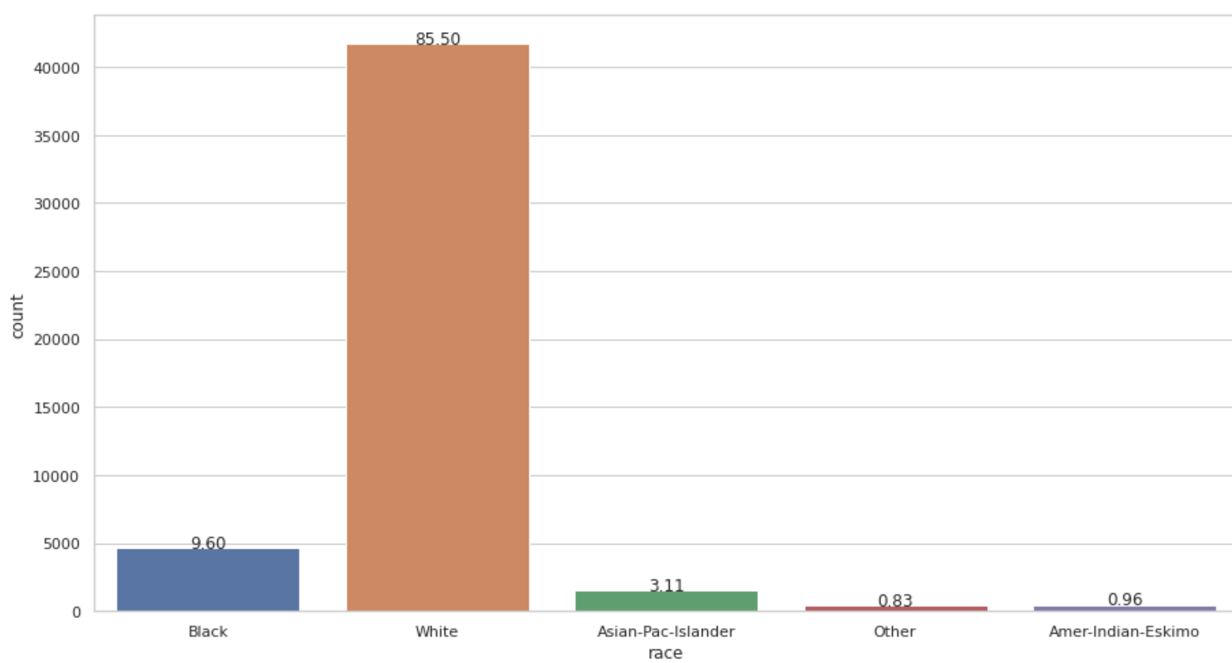
This *marital-status* attribute has 4 unique categories. *Married* maximum number of samples. *Widows* have a minimum number of samples.



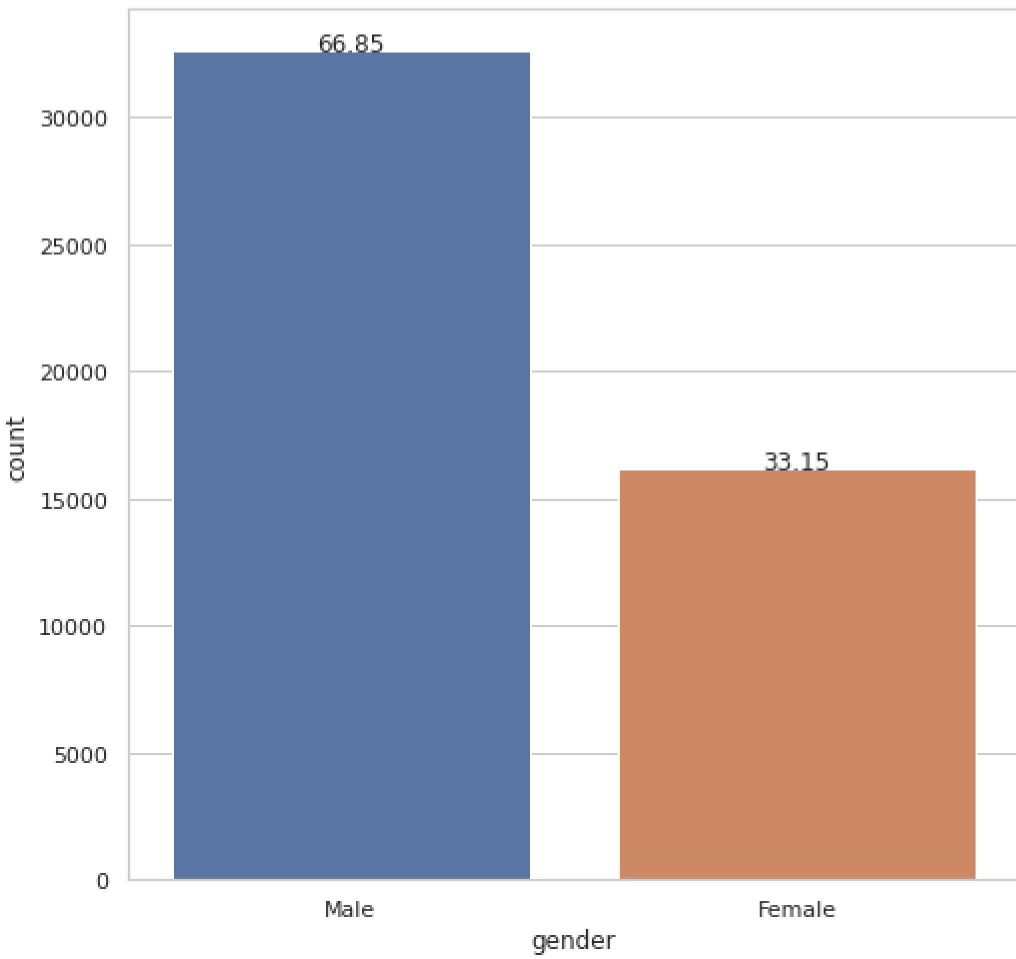
There are 14 unique categories present in the occupation attribute. Prof-specialty has the maximum count(8981) but Craft-repair, Exec-managerial and Adm-clerical Sales has a comparable number of observations. Armed-Forces has minimum samples in the occupation attribute.



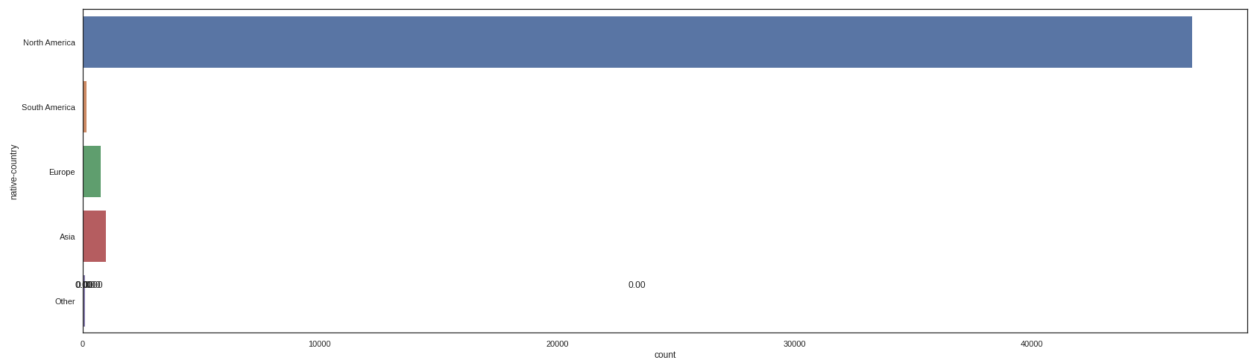
There are 6 unique categories in the relationship attribute. Husband has maximum percentage (40.38%) among all categories followed by not-in-family(25.74%)



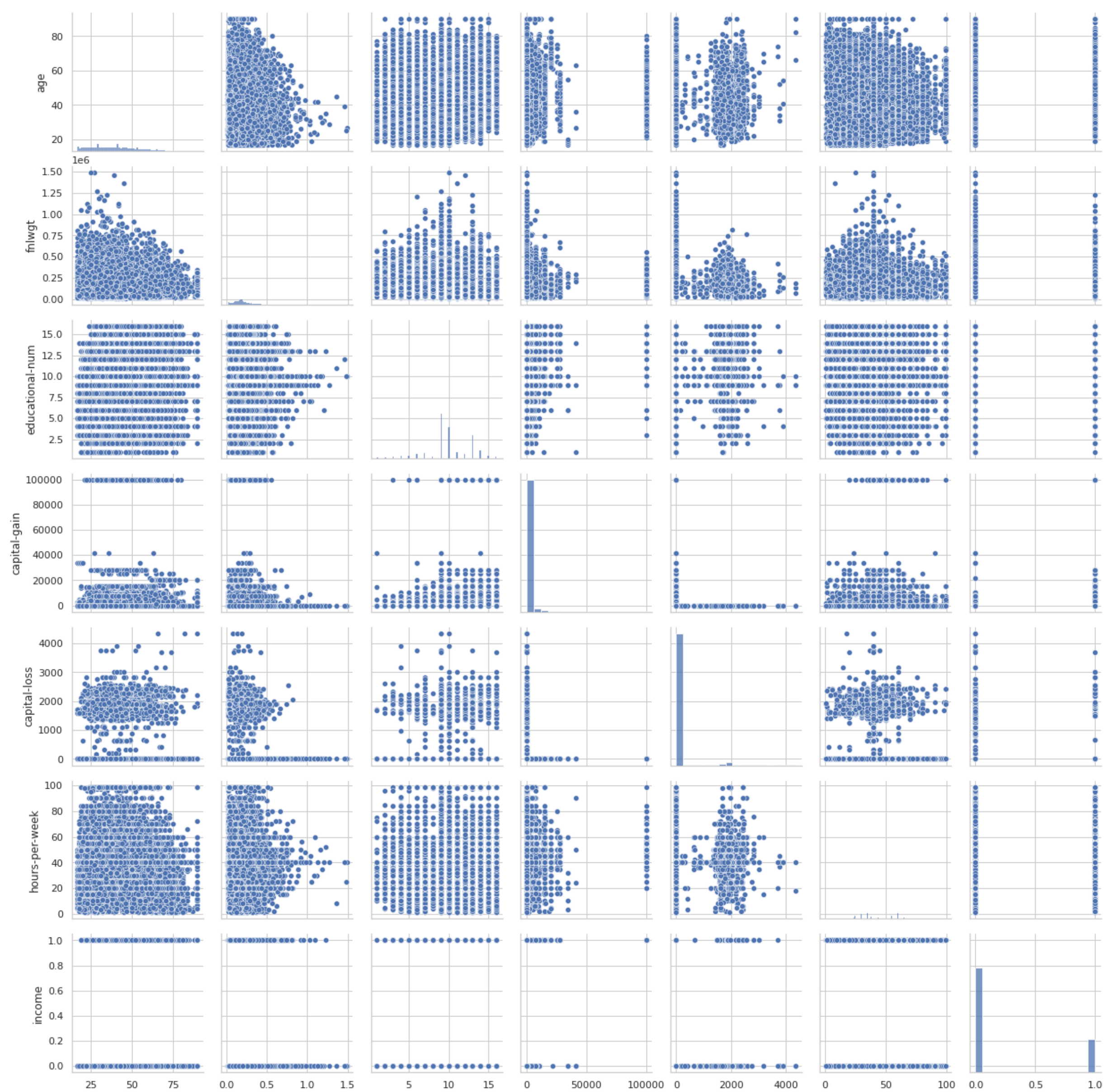
There are 5 unique categories in the race attribute. Most of them are "white" which is roughly 85.50%. This dataset is totally biased toward the "white" race. Second major race in the dataset is the "black" with just 9.60%.

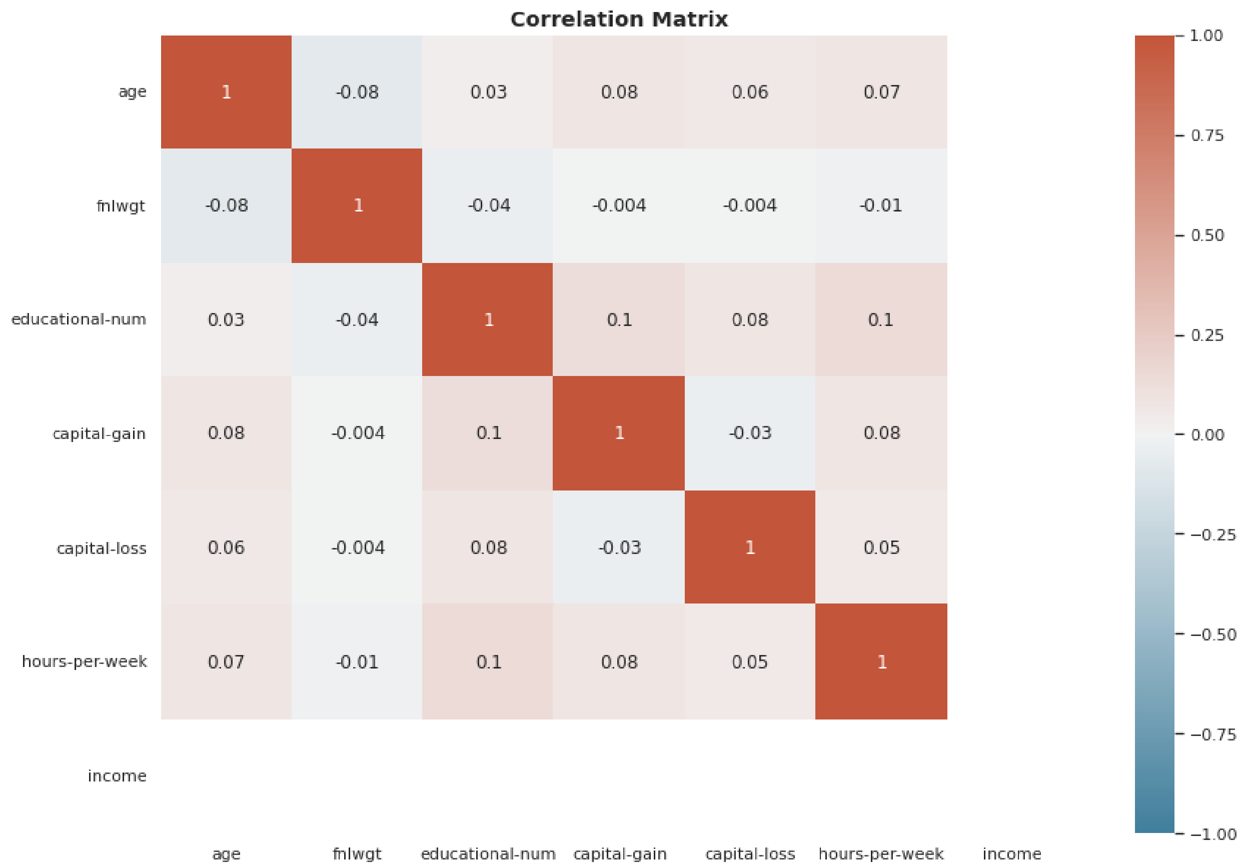


Gender has 2 unique categories(male and female). But the frequency of male is higher than the female categories. Distribution shows that this dataset is skewed toward the male with 66.85%.

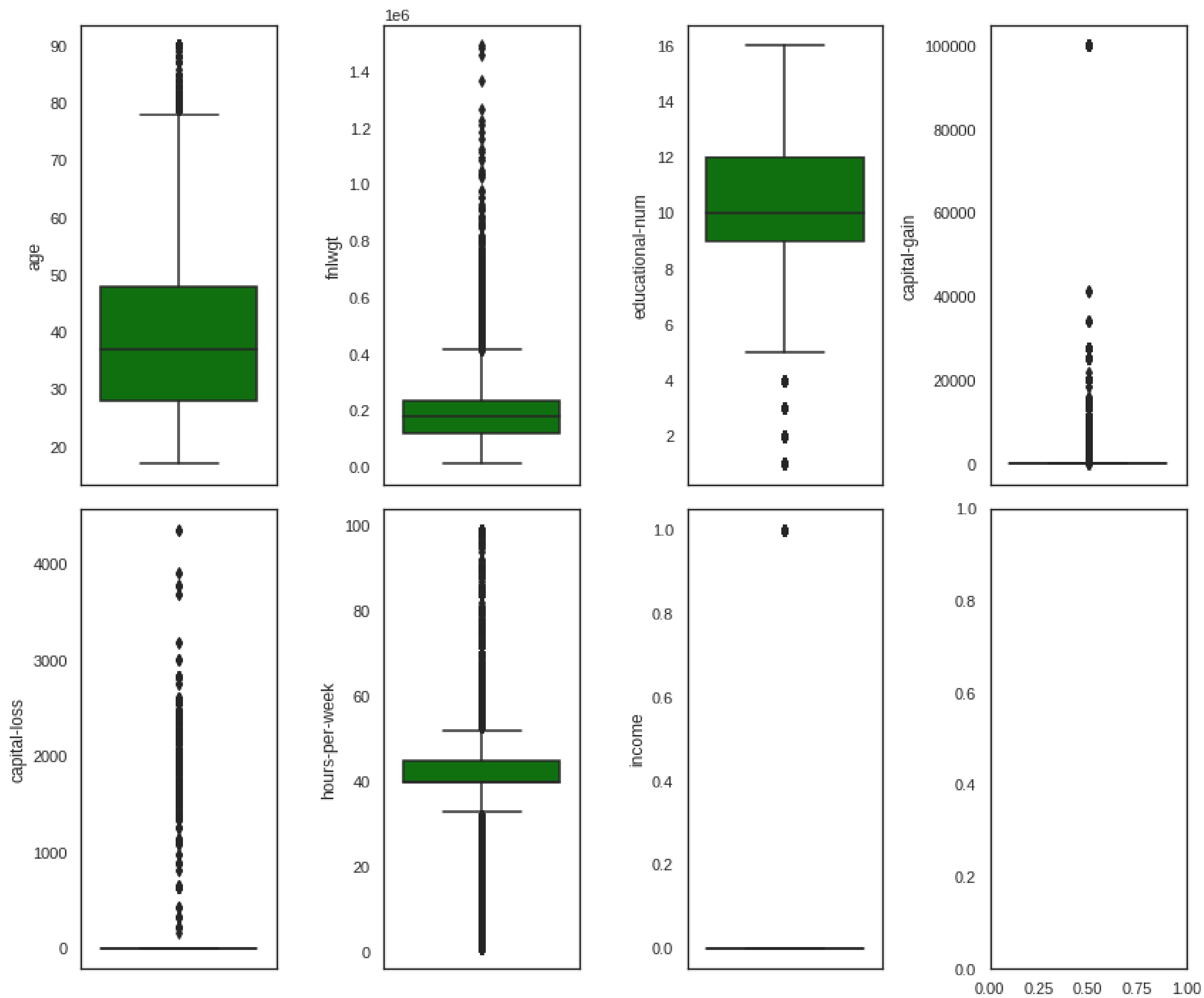


Native-country have 6 unique categories. But the frequency of North America is higher than All other categories. Distribution shows that this dataset is skewed toward North America.





Correlation among the numeric variables. There is no strong correlation among the numeric attributes.There is neither strong positive nor strong negative correlation present in any variable .



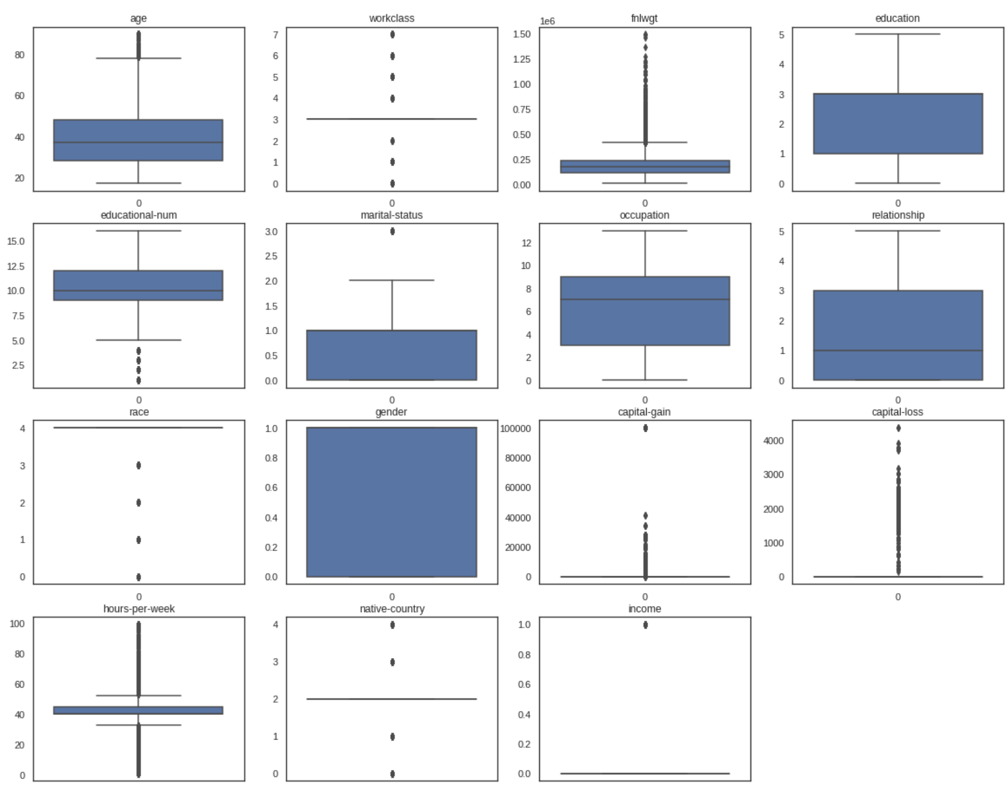
Boxplot notation of numeric variables. Boxplot is a method for graphically demonstrating the locality, spread and skewness groups of numerical data. It seems that capital-gain has some outliers.

Feature Encoding

I used LabelEncoder to encode categorical variables. It gives numbers in order for each category.

```
from sklearn.preprocessing import LabelEncoder
# Create an instance of the LabelEncoder class
le = LabelEncoder()

df1['workclass'] = le.fit_transform(df1['workclass'])
df1['education'] = le.fit_transform(df1['education'])
df1['marital-status'] = le.fit_transform(df1['marital-status'])
df1['occupation'] = le.fit_transform(df1['occupation'])
df1['relationship'] = le.fit_transform(df1['relationship'])
df1['race'] = le.fit_transform(df1['race'])
df1['gender'] = le.fit_transform(df1['gender'])
df1['native-country'] = le.fit_transform(df1['native-country'])
```

Boxplot representation after Label encoding



Building Logistic Regression Model

Logistic Regression is a statistical method that we use to fit a regression model when the response variable is binary. In simple words, it is a supervised learning algorithm used for predicting a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. It is a special case of linear regression where the target variable is categorical in nature. The probability of the binary outcome is modeled as a function of the independent variables using a logistic function, which is why it is called logistic regression.

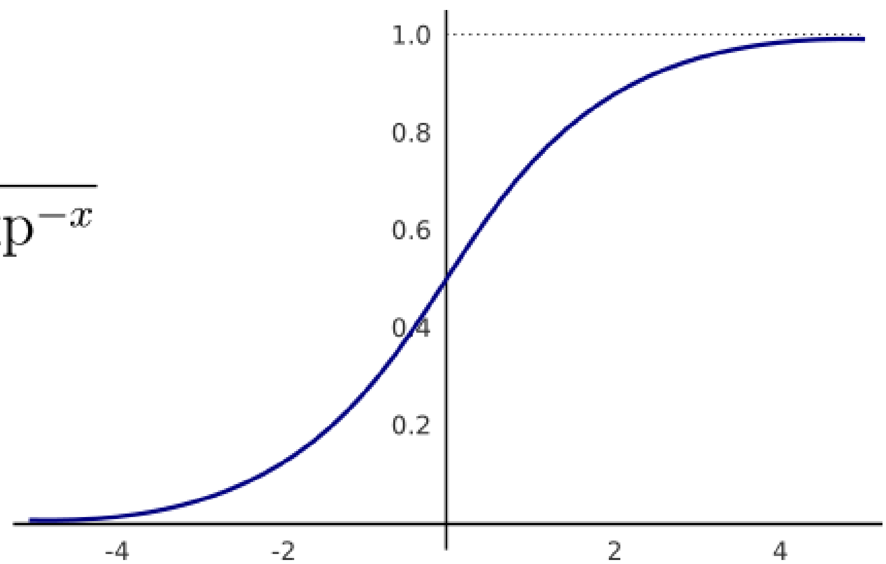
```
import numpy as np

class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=100000, fit_intercept=True):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) / y.size
            self.theta -= self.lr * gradient

    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X):
        return self.predict_prob(X).round()
```

__init__(self, lr=0.01, num_iter=100000, fit_intercept=True): This is the constructor method of the class. It sets the initial values of the learning rate (lr), number of iterations (num_iter) and the fit_intercept parameter.

__add_intercept(self, X): This is a helper method that adds an intercept term to the input data. It creates a column of ones with the same number of rows as the input data and concatenates it with the input data. This method is used when the fit_intercept parameter is set to True in the constructor.

__sigmoid(self, z): This is a helper method that applies the sigmoid function to the input data. The sigmoid function maps any input value to a value between 0 and 1. It's used to predict the probability of a sample belonging to the positive class.

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$



__loss(self, h, y) / cost function: This is a helper method that calculates the logistic loss function. The logistic loss function is used to measure the difference between the predicted probability and the true class. it is calculated by taking the mean of the negative log-likelihood of the true labels. This loss function is used in the fit method to update theta by the negative gradient of this function.

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x), y)$$

$$\boxed{\text{Cost}(h_\theta(\mathbf{x}), y) = -ylog(h_\theta(\mathbf{x})) - (1-y)log(1-h_\theta(\mathbf{x}))}$$

fit(self, X, y): This is the main method of the class that is used to train the model on the training data. It starts by adding an intercept term to the input data if the fit_intercept parameter is set to True. Then, it initializes the model's parameters (theta) with zeros. It uses the gradient descent algorithm to find the optimal values of the model's parameters that minimize the logistic loss function. The method performs the number of iterations specified by the num_iter parameter and updates the parameters in each iteration using the gradient of the loss function.

predict_prob(self, X): This method applies the trained model to the input data and returns the probability of a sample belonging to the positive class. It starts by adding an intercept term to the input data if the fit_intercept parameter is set to True, then it applies the sigmoid function to the dot product of the input data and the trained model's parameters.

predict(self, X): This method takes the probability returned by the predict_prob method and rounds it to the nearest integer to predict the class of a sample.

Model Results

Model test-train-scale-predict

```python
from sklearn.model_selection import train_test_split
y = df['income'].values
X = df.drop('income', axis=1).values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=24)
```

I splitted the data into tests and training. The 25% part of dataset is used as test.

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit the scaler to your data
scaler.fit(X_train)

# Scale the data
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Standard scaler is calculated by taking the mean of the negative log-likelihood of the true labels. This loss function is used in the fit method to update theta by the negative gradient of this function. Standard Scaler is implemented using the following formula.

```
x' = (x - mean(x)) / std(x)
```

Where x' is the standardized value, x is the original value, mean(x) is the mean of the feature and std(x) is the standard deviation of the feature. Standard Scaler can be implemented in python using the StandardScaler class from the sklearn.preprocessing module.
The Classification report of My Logistic Regression:

```
# evaluate the model performance
from sklearn.metrics import accuracy_score
print("Accuracy: ", accuracy_score(y_test, y_pred))
# generate the classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy:  0.8391404260604931
              precision    recall  f1-score   support

           0       0.86      0.94      0.90     12222
           1       0.73      0.53      0.62      3879

    accuracy                           0.84     16101
   macro avg       0.79      0.74      0.76     16101
weighted avg       0.83      0.84      0.83     16101
```

Outlier Detection

The Z-score, also known as standard score, is a measurement of how many standard deviations an observation or data point is from the mean of a dataset. It is calculated by subtracting the mean of the dataset from the value of the observation and dividing the result by the standard deviation of the dataset. The Z-score can be used to detect outliers in a dataset.

Outliers are data points that are significantly different from the rest of the data. They can be caused by measurement errors, data entry errors, or they can represent legitimate but rare events. Outliers can have a negative impact on the performance of some machine learning algorithms, so it's important to detect and handle them appropriately.

The Z-score is a useful measure for identifying outliers because it standardizes the data, so it's easier to compare the values of different features. A Z-score of more than 3 or less than -3 is usually considered as an outlier. It's worth noting that this threshold can be adjusted depending on the dataset.

It prints outlier counts by using the z-score method.

```
def get_outlier_counts(df, treshold):
    df = df.copy()
    #Get z-score for specified treshold. shitft and scale, ne kadar mean den uzaklar ı hesaplarız.
    treshold_z_score = stats.norm.ppf(treshold) #norm distribution. ppf: percent point funct. scipy içindfe bir istatik func. cdf nin tersi

    #get the z-scores for each value in track
    z_score_df = pd.DataFrame(np.abs(stats.zscore(df)), columns=df.columns)

    #compare df z-scores to the treshold, Return the count of outliers in each column
    return (z_score_df > treshold_z_score).sum(axis=0)
```

```
get_outlier_counts(df , 0.9999995)
```

```
age                  0
workclass            0
fnlwgt              94
education            0
educational-num      0
marital-status       0
occupation           0
relationship         0
race                 0
gender               0
capital-gain       247
capital-loss       366
hours-per-week       0
native-country    1156
income               0
dtype: int64
```

It removes outliers by a given treshold.

```
def remove_outliers(df, treshold):

    df1 = df.copy()

    #Get z-score for specified treshold
    treshold_z_score = stats.norm.ppf(treshold)

    #get the z-scores for each value in track and compare them to the treshold
    z_score_df = pd.DataFrame(np.abs(stats.zscore(df1)), columns=df1.columns)
    z_score_df["native-country"] = treshold_z_score  # treshold is not needed for native countries

    z_score_df = z_score_df > treshold_z_score
    # Get indicies of outliers
    outliers = z_score_df.sum(axis=1) # her rowdaki outliers toplamı
    outliers = outliers > 0

    outliers_indicies = df1.index[outliers]

    #Drop outliers
    df = df.drop(outliers_indicies, axis=0).reset_index(drop=True)

    return df
```

```
df = remove_outliers(df, 0.9999995)
```

The classification report after removing outliers.

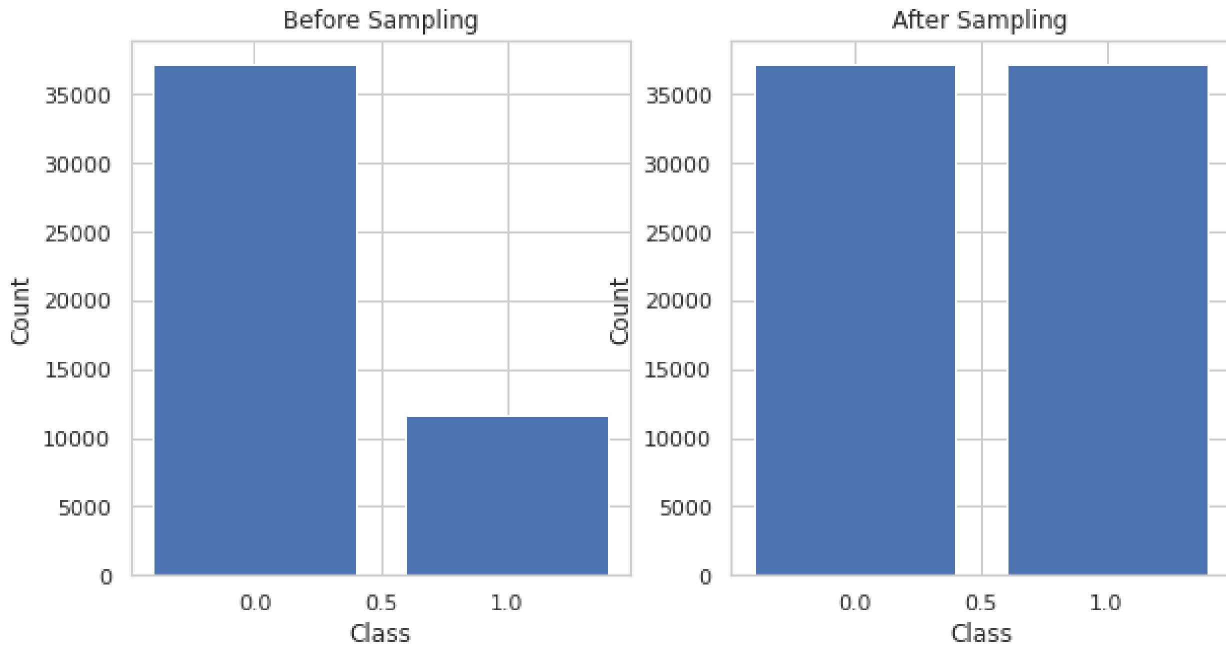
```
Accuracy:  0.84730053794802
              precision    recall  f1-score   support

           0       0.88      0.94      0.91     12169
           1       0.69      0.50      0.58      3260

    accuracy                           0.85     15429
   macro avg       0.78      0.72      0.74     15429
weighted avg       0.84      0.85      0.84     15429
```

Sampling

The target value of The Dataset's distribution is imbalanced. I used Combining oversampling and undersampling: This method involves applying both oversampling and undersampling techniques to the dataset. One of the most common techniques for combining oversampling and undersampling is called SMOTE + Tomek links.

```python
from imblearn.over_sampling import SMOTE

# apply SMOTE to the dataset
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
```

After Sampling:



Sampling decreased the accuracy. The classification report:

```
Accuracy:  0.8324322076193645
              precision    recall  f1-score   support

           0       0.86      0.79      0.83     12070
           1       0.81      0.87      0.84     11974

    accuracy                           0.83     24044
   macro avg       0.83      0.83      0.83     24044
weighted avg       0.83      0.83      0.83     24044
```

Principal Component Analysis - PCA

Principal Component Analysis (PCA) is a technique used to extract the most important features from a dataset, it is a linear dimensionality reduction technique. It reduces the dimensionality of the data by projecting it onto a lower-dimensional space while preserving as much of the original variance as possible. It is widely used in many fields including pattern recognition, image processing, and bioinformatics.

The basic idea behind PCA is to find a new coordinate system for the data such that the first axis (principal component) explains the most variance, the second axis explains the second most variance, and so on. PCA can be used to identify patterns in the data, reduce the noise in the data, or to visualize high-dimensional data.

```python
# Fit the scaler to your data
scaler.fit(X_train)

# Scale the data
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)


from sklearn.decomposition import PCA

# Create an instance of the PCA class
pca = PCA(n_components=13)

X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)


print(len(X_train_pca[0]))
print(X_test_pca)

print(pca.explained_variance_ratio_)
```

After outlier detection and PCA. Dimension reduced from 14 to 13. Dimension reduction is not well for our logistic regression model.

```
Accuracy:  0.8376433987944779
              precision    recall  f1-score   support

           0       0.86      0.95      0.90     12169
           1       0.69      0.42      0.52      3260

    accuracy                           0.84     15429
   macro avg       0.77      0.69      0.71     15429
weighted avg       0.82      0.84      0.82     15429
```

Other Classification Models

Results without PCA, Outlier detection, Samling:

```
LGBM Classifier Classification Accuracy: 0.8723060679460903
LGBM Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92     12222
           1       0.78      0.65      0.71      3879

    accuracy                           0.87     16101
   macro avg       0.84      0.80      0.81     16101
weighted avg       0.87      0.87      0.87     16101



XGBoost Classifier Classification Accuracy: 0.8636109558412521
XGBoost Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.95      0.91     12222
           1       0.80      0.58      0.67      3879

    accuracy                           0.86     16101
   macro avg       0.84      0.77      0.79     16101
weighted avg       0.86      0.86      0.86     16101
```

```
Decision Tree Classification Accuracy: 0.8125582261971306
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.87      0.88     12222
           1       0.61      0.62      0.62      3879

    accuracy                           0.81     16101
   macro avg       0.74      0.75      0.75     16101
weighted avg       0.81      0.81      0.81     16101



SVM Classification Accuracy: 0.796534376746786
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88     12222
           1       0.96      0.16      0.28      3879

    accuracy                           0.80     16101
   macro avg       0.87      0.58      0.58     16101
weighted avg       0.83      0.80      0.74     16101



Random Forest Classification Accuracy: 0.8559716787777156
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     12222
           1       0.74      0.62      0.68      3879

    accuracy                           0.86     16101
   macro avg       0.81      0.78      0.79     16101
weighted avg       0.85      0.86      0.85     16101
```

After Outlier detection applied:

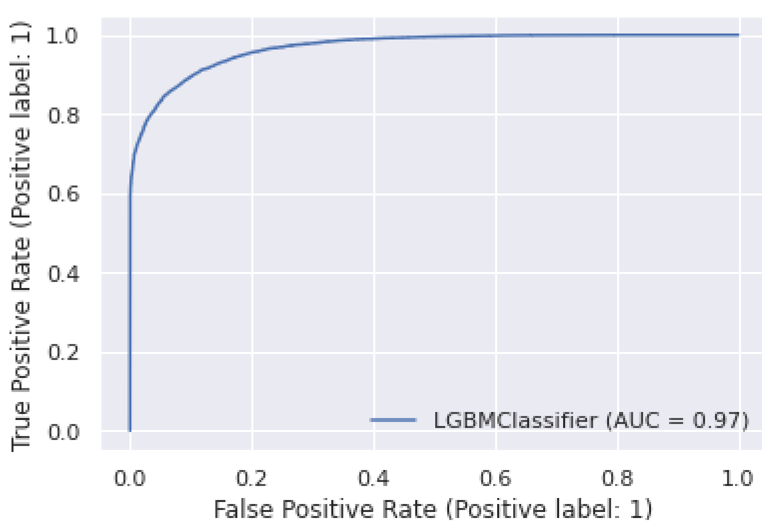
```
LGBM Classifier Classification Accuracy: 0.8741979389461404
LGBM Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92     12169
           1       0.74      0.62      0.67      3260

    accuracy                           0.87     15429
   macro avg       0.82      0.78      0.80     15429
weighted avg       0.87      0.87      0.87     15429



XGBoost Classifier Classification Accuracy: 0.8685592066887031
XGBoost Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92     12169
           1       0.75      0.57      0.65      3260

    accuracy                           0.87     15429
   macro avg       0.82      0.76      0.78     15429
weighted avg       0.86      0.87      0.86     15429
```

```
Decision Tree Classification Accuracy: 0.8051720785533735
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.87      0.88     12169
           1       0.54      0.57      0.55      3260

    accuracy                           0.81     15429
   macro avg       0.71      0.72      0.71     15429
weighted avg       0.81      0.81      0.81     15429



SVM Classification Accuracy: 0.8201438848920863
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.82      1.00      0.90     12169
           1       0.96      0.16      0.27      3260

    accuracy                           0.82     15429
   macro avg       0.89      0.58      0.58     15429
weighted avg       0.85      0.82      0.76     15429



Random Forest Classification Accuracy: 0.8536522133644436
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     12169
           1       0.68      0.58      0.62      3260

    accuracy                           0.85     15429
   macro avg       0.79      0.75      0.77     15429
weighted avg       0.85      0.85      0.85     15429
```

After Outlier detection and Sampling applied:

```
LGBM Classifier Classification Accuracy: 0.8976043919480952
LGBM Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.90      0.90     12070
           1       0.90      0.90      0.90     11974

    accuracy                           0.90     24044
   macro avg       0.90      0.90      0.90     24044
weighted avg       0.90      0.90      0.90     24044



XGBoost Classifier Classification Accuracy: 0.8806770919980037
XGBoost Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.86      0.88     12070
           1       0.86      0.90      0.88     11974

    accuracy                           0.88     24044
   macro avg       0.88      0.88      0.88     24044
weighted avg       0.88      0.88      0.88     24044
```

Roc Curve for LGBM:



```
Decision Tree Classification Accuracy: 0.8555980702046249
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.85      0.86     12070
           1       0.85      0.86      0.86     11974

    accuracy                           0.86     24044
   macro avg       0.86      0.86      0.86     24044
weighted avg       0.86      0.86      0.86     24044



SVM Classification Accuracy: 0.5937863916153718
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.98      0.71     12070
           1       0.93      0.20      0.33     11974

    accuracy                           0.59     24044
   macro avg       0.74      0.59      0.52     24044
weighted avg       0.74      0.59      0.52     24044



Random Forest Classification Accuracy: 0.8935701214440193
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.89      0.89     12070
           1       0.89      0.90      0.89     11974

    accuracy                           0.89     24044
   macro avg       0.89      0.89      0.89     24044
weighted avg       0.89      0.89      0.89     24044
```

After Outlier detection, Sampling and PCA applied.

```
LGBM Classifier Classification Accuracy: 0.8719846947263351
LGBM Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.88      0.87     12070
           1       0.88      0.87      0.87     11974

    accuracy                           0.87     24044
   macro avg       0.87      0.87      0.87     24044
weighted avg       0.87      0.87      0.87     24044



XGBoost Classifier Classification Accuracy: 0.8472799866910664
XGBoost Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.84      0.85     12070
           1       0.84      0.85      0.85     11974

    accuracy                           0.85     24044
   macro avg       0.85      0.85      0.85     24044
weighted avg       0.85      0.85      0.85     24044
```

```
Decision Tree Classification Accuracy: 0.8213275661287639
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.81      0.82     12070
           1       0.82      0.83      0.82     11974

    accuracy                           0.82     24044
   macro avg       0.82      0.82      0.82     24044
weighted avg       0.82      0.82      0.82     24044



SVM Classification Accuracy: 0.599151555481617
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.98      0.71     12070
           1       0.92      0.21      0.35     11974

    accuracy                           0.60     24044
   macro avg       0.74      0.60      0.53     24044
weighted avg       0.74      0.60      0.53     24044



Random Forest Classification Accuracy: 0.8776409915155549
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.88      0.88     12070
           1       0.88      0.87      0.88     11974

    accuracy                           0.88     24044
   macro avg       0.88      0.88      0.88     24044
weighted avg       0.88      0.88      0.88     24044
```

An ensemble model to the LGBM classifier:

```
from sklearn.ensemble import VotingClassifier

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=2)

# create an instance of the LGBM classifier
lgbm = LGBMClassifier()

# create an instance of the Random Forest classifier
rf = RandomForestClassifier()

# create an instance of the SVM classifier
svm = SVC(probability=True)

# create an ensemble model using the voting classifier
ensemble = VotingClassifier(estimators=[('lgbm', lgbm), ('rf', rf), ('svm', svm)], voting='soft')

# fit the ensemble model to the training data
ensemble.fit(X_train, y_train)

# predict the class labels for the test data
y_pred = ensemble.predict(X_test)

# evaluate the ensemble model
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
Accuracy: 0.8972716686075528
```

In this code, I first created an instance of the LGBM classifier and trained it on the training data. Then I created an instance of the Random Forest classifier and the SVM classifier and set the probability attribute of SVM to true. After that, I used the VotingClassifier function from the sklearn.ensemble library to create an ensemble model of the LGBM, Random Forest and SVM classifiers. The voting parameter is set to 'soft' which means the classifier will use predicted class probabilities to make a prediction.

Finally, I fitted the ensemble model to the training data and used it to make predictions on the test data. The accuracy of the ensemble model is then evaluated by comparing the predicted class labels to the true class labels of the test data using the accuracy_score function.

Results

In this income classification project, My goal was to predict whether an individual's salary was high or low based on various demographic and employment-related factors. To accomplish this, I used a variety of classification models including Logistic Regression(my implementation), LGBM, XGBoost, Decision Tree, SVM and Random Forest.

To improve the performance of my models, I first applied outlier detection methods to identify and remove any outliers in the dataset. I then used the SMOTE technique to oversample the minority class in order to balance the dataset.

After training and evaluating the performance of each model, I found that the LGBM classifier performed the best, achieving an overall accuracy of 0.8976 and F1-score of 0.90 on the test set. The precision and recall values of the LGBM classifier are also high.

Furthermore, the area under the Receiver Operating Characteristic (ROC) curve for LGBM classifiers was also high, indicating that it performed well in terms of distinguishing between the two classes.

In conclusion, the LGBM classifier is a suitable model for this income classification problem, with high accuracy and F1-score. The results from this study provide valuable insights into the factors that influence an individual's salary, and can be used to help predict future salaries based on demographic and employment-related factors.

Resources

- “Adult Income Dataeset”, http://archive.ics.uci.edu/ml/datasets/Adult, 1996

- “A logistic regression from scratch”, “Dennis Bakhuis”, https://towardsdatascience.com/a-logistic-regression-from-scratch-3824468b1f88, 2020

- “PCA (Principal Component Analysis) Temel Bileşenler Analizi”, “Gülcan Öğündür”, https://medium.com/@gulcanogundur/pca-principal-component-analysis-temel-bile%C5%9Fenler-analizi-bf9098751c62, 2020
- “Sampling Techniques— Statistical approach in Machine learning”, ”Suresha HP”, https://medium.com/analytics-vidhya/sampling-statistical-approach-in-machine-learning-4903c40ebf86, 2021
- “Scikit-learn developers”, https://scikit-learn.org/stable/modules/outlier_detection.html