Data Analytics Lab: Assignment-3
A Mathematical Essay on Naive Bayes Classifier

Arjav Singh
Metallurgical and Materials Engineering
Indian Institute of Technology Madras
Chennai, India
mm20b007@smail.iitm.ac.in

***Abstract*—In this study, the correlation between whether an individual makes over $50,000 a year and various factors such as age, educational qualification, marital status, occupation, race, and other factors is examined. The importance of these factors is modeled, and the income group of individuals is predicted using a Naive Bayes model.**

***Index Terms*—Introduction, Naive Bayes, Data & Problem, Conclusion**

I. Introduction

This study uses survey data from the 1994 Census database to conduct an empirical analysis of the factors influencing personal income. Education level is a crucial indicator, and classification is performed using the Naive Bayes model. It is discovered that several factors, including gender, age, education, and marital status, have a significant impact on personal income. Additionally, variations among different occupations are also explored. Naive Bayes is a classification technique founded on Bayes' Theorem, assuming conditional independence among predictors and that one particular feature in a class is unrelated to the presence of any other feature.

In this study, the income category of individuals is modeled based on education, age, socioeconomic factors, marital status, etc., using Naive Bayes. The process begins with the gathering, cleaning, and preparing data, followed by exploratory analysis. Subsequently, statistical models are constructed, and visualizations are generated to provide quantitative and visual evidence of the observed relationships. In the next section, the key principles underlying Naive Bayes are highlighted. Section 3 delves into the insights and observations derived from the data and models. Finally, in section 4, the salient features of the study are outlined, and potential avenues for further investigation are presented.

II. Naive Bayes

Naive Bayes is a mathematical technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, with the class labels being drawn from some finite set. All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. There are different types of Naive Bayes classifiers available, some of them are -

1) **Multinomial Naive Bayes**: In the case of a Multinomial Naive Bayes model, the samples (feature vectors) represent the frequencies at which certain events have been generated by a multinomial distribution with probabilities $(p_1, \dots, p_n)$, where p_i is the probability that event i occurs. The Multinomial Naive Bayes algorithm is typically preferred for data that follows a multinomial distribution. It is one of the standard algorithms used in text categorization and classification.
2) **Bernoulli Naive Bayes**: In the multivariate Bernoulli event model, features are independent boolean variables (binary variables) describing inputs. Like the multinomial model, this model is also commonly employed in document classification tasks, where binary term occurrence features are used instead of term frequencies.
3) **Gaussian Naive Bayes**: When dealing with continuous attribute values, an assumption is made that the values associated with each class follow a Gaussian or Normal distribution. For instance, consider training data containing a continuous attribute x. First, the data is segmented by class, and then the mean (μ_i) and variance (σ_i^2) of x in each class are computed. Suppose there is an observation value x_i. Then, the probability distribution of x_i given a class can be computed using the following equation:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

A. Model Structure

The Naïve Bayes Classifier utilizes Bayes' theorem to estimate the membership probabilities for each class, specifically the probability that a given record or data point belongs to a particular class. The class with the highest probability is considered the most likely class, often referred to as the Maximum A Posteriori (MAP) class.

The MAP for a hypothesis involving two events, A and B, is calculated as follows:

$$\text{MAP}(A) = \max(P(A|B)) \quad (1) \tag{1}$$

This can also be expressed as:

$$\text{MAP}(A) = \max\left(\frac{P(B|A) \cdot P(A)}{P(B)}\right) \quad (2) \qquad (2)$$

Simplifying further:

$$\text{MAP}(A) = \max(P(B|A) \cdot P(A)) \quad (3) \qquad (3)$$

Here, $P(B)$ represents the evidence probability, which is used for normalization purposes. It remains constant across calculations, so removing it does not affect the result.

The Naïve Bayes Classifier operates under the assumption that all features are mutually independent. In other words, the presence or absence of one feature does not influence the presence or absence of any other feature.

In real-world datasets, hypotheses are tested based on multiple items of evidence derived from various features. These calculations can become quite complex. To simplify this process, the feature independence assumption is applied to treat each item of evidence as independent, thereby simplifying the analysis.

B. Metrics for model evaluation

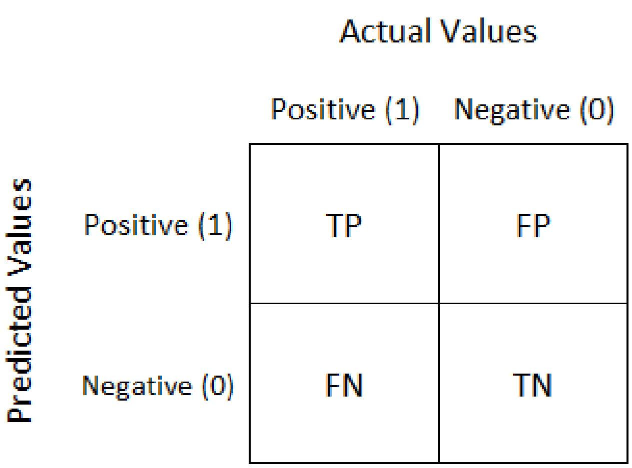


Fig. 1. Confusion Matrix.

1) **Confusion Matrix**: It is used to summarize the performance of a classification algorithm on a set of test data for which the true values are previously known. Sometimes it is also called an error matrix. Terminologies of the Confusion matrix (Figure 1) are:
   - **True Positive**: TP means the model predicted yes, and the actual answer is also yes.
   - **True negative**: TN means the model predicted no, and the actual answer is also no.
   - **False positive**: FP means the model predicted yes, but the actual answer is no.
   - **False negative**: FN means the model predicted no, but the actual answer is yes.

   The rates calculated using the Confusion Matrix are:

   a) **Accuracy**: (TP+TN/Total) tells about overall how classifier Is correct.
   b) **True positive rate**: TP/(actual yes) it says about how much time yes is predicted correctly. It is also called "sensitivity" or "recall."
   c) **False positive rate**: FP/(actual number) says how much time yes is predicted when the actual answer is no.
   d) **True negative rate**: TN/(actual number) says how much time no is predicted correctly, and the actual answer is also no. It is also known as "specificity."
   e) **Misclassification rate**: (FP+FN)/(Total) It is also known as the error rate and tells about how often our model is wrong.
   f) **Precision**: (TP/ (predicted yes)) If it predicts yes, then how often is it correct.
   g) **Prevalence**: (actual yes /total) how often yes condition actually occurs.
   h) **F1-score**: f1 score is defined as the weighted harmonic mean of precision and recall. The best achievable F1 score is 1.0, while the worst is 0.0. The F1 score serves as the harmonic mean of precision and recall. Consequently, the F1-score consistently yields lower values than accuracy measures since it incorporates precision and recall in its computation. When evaluating classifier models, it is advisable to employ the weighted average of the F1 score instead of relying solely on global accuracy.

2) **ROC curve (Receiver Operating Characteristic)**: The Receiver Operating Characteristic (ROC) curve is a useful tool for assessing a model's performance by examining the trade-offs between its True Positive (TP) rate, also known as sensitivity, and its False Negative (FN) rate, which is the complement of specificity. This curve visually represents these two parameters.
   The Area Under the Curve (AUC) metric to summarize the ROC curve concisely. The AUC quantifies the area under the ROC curve. In simpler terms, it measures how well the model can distinguish between positive and negative cases. A higher AUC indicates better classifier performance.
   In essence, AUC categorizes model performance as follows:

   - If AUC = 1, the classifier correctly distinguishes between all the Positive and Negative class points.
   - If 0.5¡ AUC ¡ 1, the classifier will distinguish the positive class value from the negative one because it finds more TP and TN than FP and FN.
   - If AUC = 0.5, the classifier cannot distinguish between positive and negative values.
   - If AUC =0, the classifier predicts all positive as negative and negative as positive.

III. Problem

The problem at hand is centered around predicting whether the income of a person is more than \$50,000 from the 1994 Census US income database. A naive Bayes classifier will be employed to predict the possibility for every person, incorporating various features such as age, relationship status, education, and other relevant factors for analysis.

A. Exploratory Data Analysis and Feature Generation

The training dataset employed in this study comprises 32,561 individuals and encompasses 14 distinct features. The interpretation of these features is as follows:

- *Age*: The individual's age, ranging from 17 to 90.
- *Workclass*: The employment category of the individual, which includes designations such as private, without-pay, state government, etc.
- *Fnlwgt*
- *Education*: The educational level of the individual.
- *Education Years*: The number of years of education completed by the individual.
- *Occupation*: The occupation of the individual.
- *Relationship*: The individual's role within the family.
- *Race*: The racial background to which the individual belongs.
- *Sex*: The gender of the individual.
- *Capital Gain, Loss*
- *Working Hours*: The average number of hours per week that the individual works.
- *Native Country*: The cultural or geographic background of the individual.

The dataset contains missing values denoted by "?" in the "Workclass" and "Occupation" features. Instead of discarding these entries, they are treated as a separate category due to the observation that removing them adversely impacts the model's performance. The primary objective is to predict the binary feature "Wage," which is equal to 1 if the individual earns an annual income greater than 50,000 dollars and 0 otherwise.

	Before resampling	After forward fill	After backward fill
Private	22696.0	24094.0	24056.0
Self-emp-not-inc	2541.0	2688.0	2701.0
Local-gov	2093.0	2204.0	2212.0
?	1836.0	nan	nan
State-gov	1297.0	1373.0	1373.0
Self-emp-inc	1116.0	1177.0	1182.0
Federal-gov	960.0	1002.0	1013.0
Without-pay	14.0	15.0	16.0
Never-worked	7.0	7.0	7.0

Fig. 2. Distribution of values before and after re-sampling of workclass feature.

The data is initially read into a pandas data frame, revealing a total of 32,561 data points and a total of 15 columns encompassing various person-related features. When the distributions of individuals who earn over 50K are visualized, it is observed that approximately 24.1% of the total population falls into this category (Figure 1). Among the 15 features, 9 are categorical, and 6 are numerical. Subsequently, an assessment is made to identify null values within the data, and it is determined that there are no NaN values. However, three columns, namely Workclass, Occupation, and Native Country, contain '?' marks in some data points, necessitating treatment. The initial step involves replacing these '?' with NaN values, then imputing them with the value preceded by them in each respective column, as the number of null values is very low since the effect of forward re-sampling and backward re-sampling had very low difference which can be observed in Figure 2, 3, and 4 hence 'forward fill' re-sampling method was opted.

The cardinality of each categorical feature is then examined, measuring the number of unique values each feature can assume. High cardinality can potentially lead to issues. It is observed that most features have no more than 7 attributes, with the Native Country feature having the highest number (Figure 2).

	Before resampling	After forward fill	After backward fill
Prof-specialty	4140.0	4386.0	4410.0
Craft-repair	4099.0	4364.0	4339.0
Exec-managerial	4066.0	4317.0	4287.0
Adm-clerical	3769.0	3981.0	3998.0
Sales	3650.0	3863.0	3867.0
Other-service	3295.0	3470.0	3493.0
Machine-op-inspct	2002.0	2134.0	2128.0
?	1843.0	nan	nan
Transport-moving	1597.0	1703.0	1686.0
Handlers-cleaners	1370.0	1471.0	1446.0
Farming-fishing	994.0	1038.0	1069.0
Tech-support	928.0	981.0	984.0
Protective-serv	649.0	683.0	689.0
Priv-house-serv	149.0	159.0	155.0
Armed-Forces	9.0	10.0	9.0

Fig. 3. Distribution of values before and after re-sampling of occupation feature.

B. Visualization and Feature Generation

The features are examined one by one, beginning with "Workclass." It is observed that the primary categories are government employees, private sector workers, self-employed individuals, those without pay, and individuals who have never worked. Each category exhibits a different higher income rate, with the highest rate among self-employed individuals and the lowest among those in the private sector.

Moving on to "Education," the primary classes are university level, school level, and postgraduate level. A trend emerges where lower levels of education correspond to lower incomes, while individuals with doctorates and professional school degrees earn the highest salaries.

In the case of "Marital Status," individuals who are married and have a spouse tend to have the highest incomes. In contrast, all other marital statuses are associated with a lower likelihood of high income.

Analyzing "Occupation," it is evident that the number of classes increases significantly. Each occupation class enjoys

	Before resampling	After forward fill	After backward fill
United-States	29169.0	29693.0	29693.0
Mexico	643.0	657.0	657.0
?	583.0	nan	nan
Philippines	198.0	200.0	200.0
Germany	137.0	141.0	141.0
Canada	121.0	124.0	124.0
Puerto-Rico	114.0	118.0	118.0
El-Salvador	106.0	109.0	109.0
India	100.0	101.0	101.0
Cuba	95.0	97.0	97.0
England	90.0	93.0	93.0
Jamaica	81.0	83.0	83.0
South	80.0	80.0	80.0
China	75.0	77.0	77.0
Italy	73.0	73.0	73.0
Dominican-Republic	70.0	74.0	74.0
Vietnam	67.0	72.0	72.0
Guatemala	64.0	66.0	66.0
Japan	62.0	63.0	63.0
Poland	60.0	60.0	60.0
Columbia	59.0	61.0	61.0
Taiwan	51.0	51.0	51.0
Haiti	44.0	45.0	45.0
Iran	43.0	43.0	43.0
Portugal	37.0	37.0	37.0
Nicaragua	34.0	34.0	34.0
Peru	31.0	31.0	31.0
France	29.0	29.0	29.0
Greece	29.0	30.0	30.0
Ecuador	28.0	28.0	28.0
Ireland	24.0	24.0	24.0
Hong	20.0	20.0	20.0
Cambodia	19.0	20.0	20.0
Trinadad&Tobago	19.0	19.0	19.0
Laos	18.0	19.0	19.0
Thailand	18.0	18.0	18.0
Yugoslavia	16.0	17.0	17.0
Outlying-US(Guam-USVI-etc)	14.0	14.0	14.0
Honduras	13.0	13.0	13.0
Hungary	13.0	13.0	13.0
Scotland	12.0	12.0	12.0
Holand-Netherlands	1.0	1.0	1.0

Fig. 4. Distribution of values before and after re-sampling of native country feature.

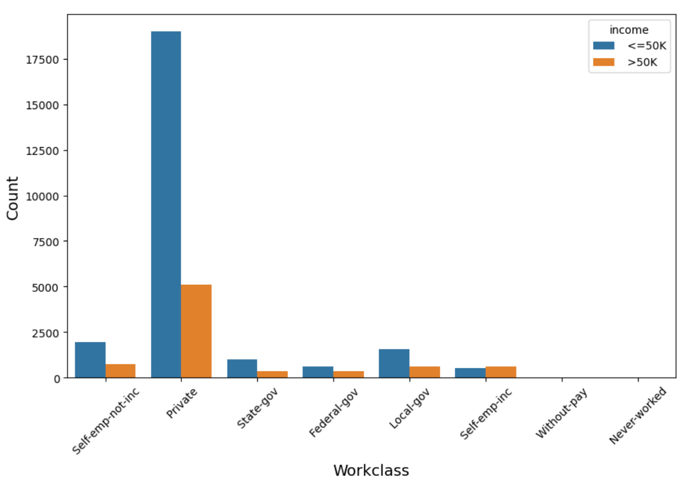


Fig. 5. Income variability over different work classes.

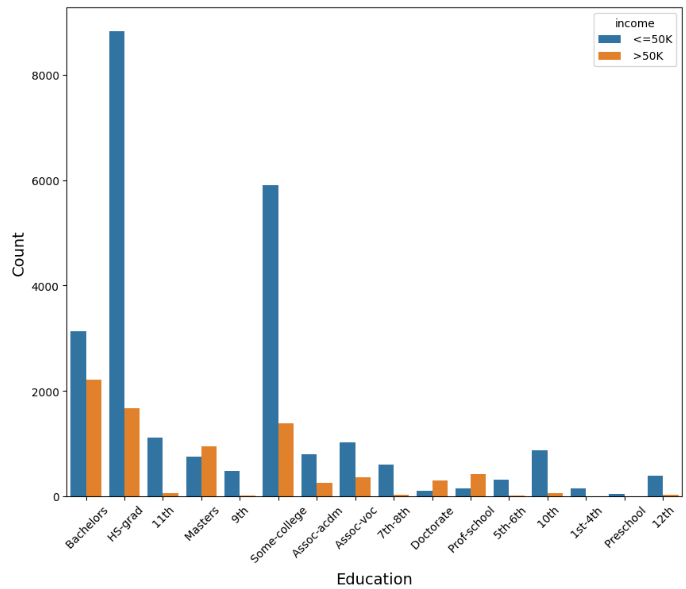


Fig. 6. Income variability over academic qualifications.

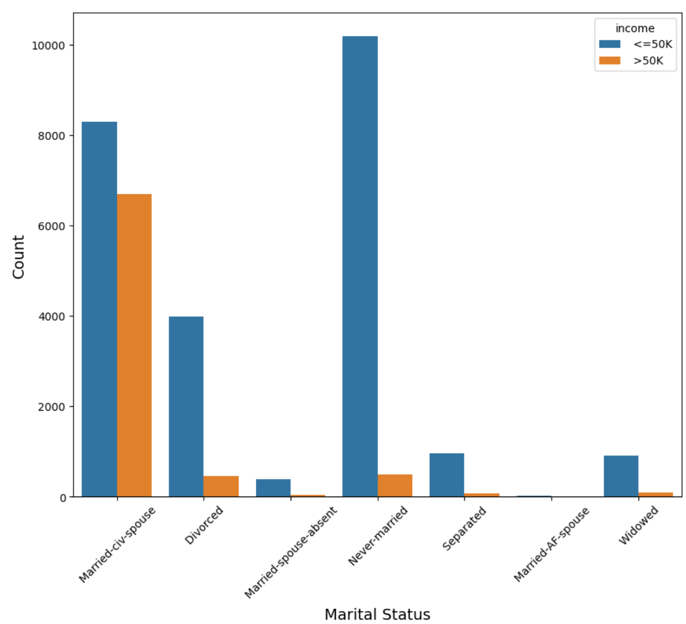


Fig. 7. Income distribution over marital status of an individual.

different income rates, with executive managers and professionals in specialized fields earning the highest incomes.

Exploring the ”Relationship” feature reveals that husbands and wives have the highest probability of having high incomes. In contrast, single individuals without families or those who are unmarried do not enjoy such high incomes.

In terms of ”Race,” there is a bias toward white individuals having higher salaries compared to others. A concerning observation is made when visualizing ”Sex,” with males having higher average incomes than females.

When exploring ”Age” using histograms and box plots, it becomes apparent that individuals in their 40s tend to have higher incomes. This can be attributed to the fact that most people do not earn well at the beginning of their careers, and as they grow old, in their 60s and 70s, they tend to lose more money than they earn.

Further investigation into ”Education-Num,” which represents the number of education levels, reveals a trend where individuals with higher education levels tend to have higher incomes. Similarly, ”Hours per Week (hourspw),” which represents the number of hours worked per week, indicates

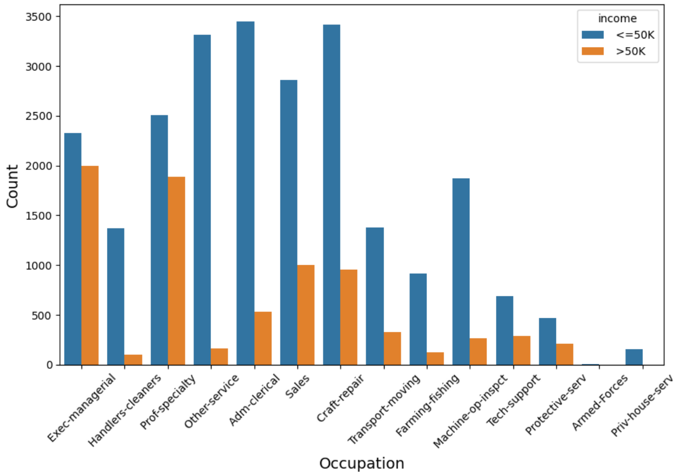


Fig. 8. Income distribution over different occupations.

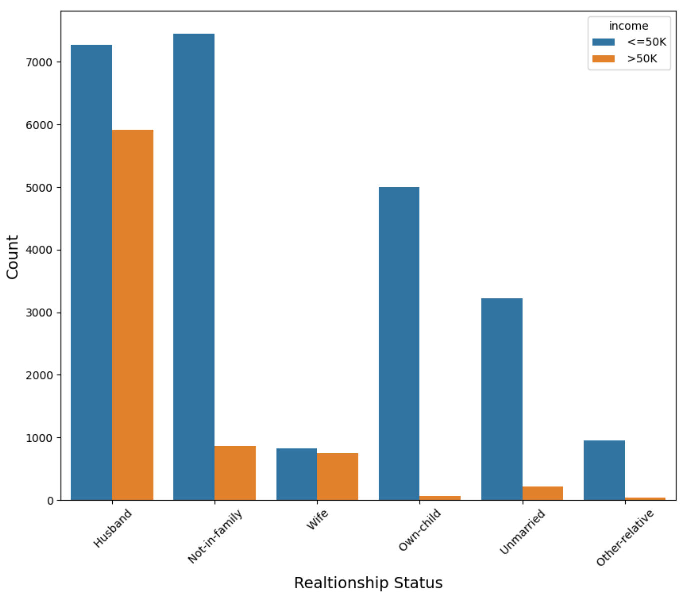


Fig. 9. Income over the relationship status of the individuals.

that individuals who work longer hours tend to have higher incomes.

Before moving to feature generation, Pearson correlation values were checked by converting all the categorical features into numerical ones by assigning each unique data point a number, and it was found that income is highly correlated to education-num, relationship, marital status, sex, age and number of hours per week.

With help of the above information new features were generated to improvise the modelling. The number of years of education is categorised into low, medium, and high based their values, similar categorization is implemented to hours per week. The occupation data was categorised into highskill if in managerial and specialty position and lowskill otherwise. Based on income distribution with respect to race it was found

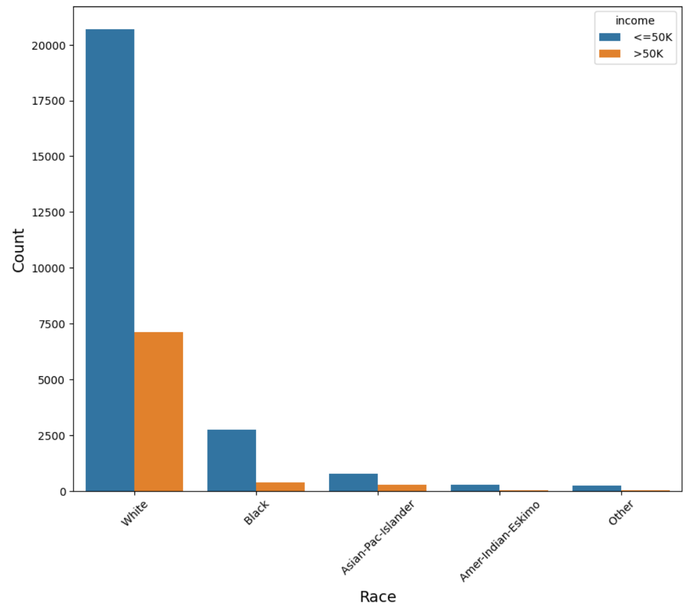


Fig. 10. Income variability over different races.

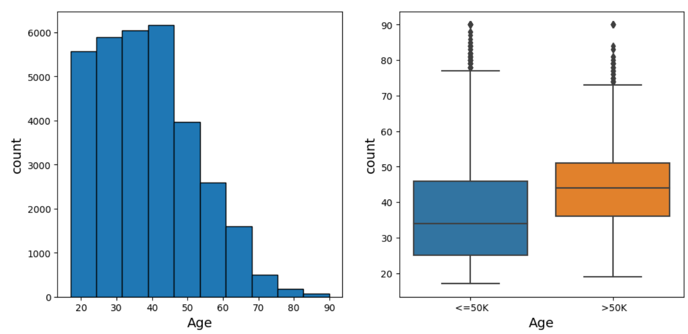


Fig. 11. Distribution of income over the age of the individuals.

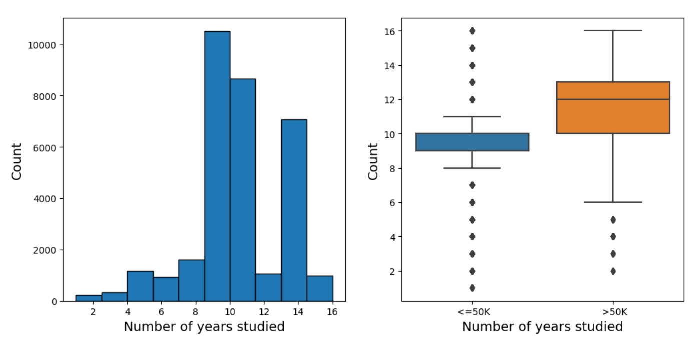


Fig. 12. Income vs number of years of education.

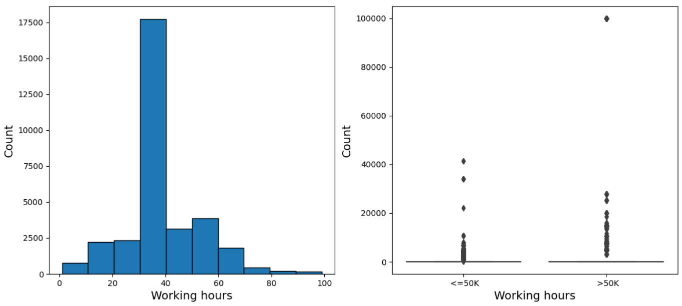


Fig. 13. Income distribution over the number of work hours.

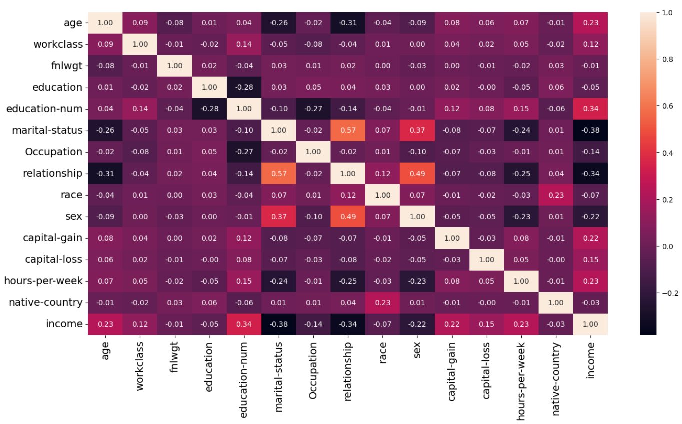


Fig. 14. Pearson Correlation heatmap.

that most of people who has more than $50000 income are white people hence race feature is categorised into white and other.

Finally, as part of postprocessing, we create dummies for the categorical variables to make them meaningful to the machine, also termed One Hot encoding.

C. Feature Selection

One hot encoding resulted in a very high dimensional data which is not suitable for model hence Variance threshold is utilized to reduce the dimension and choose the most relevant features. Variance Threshold is a univariate approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples.

D. Guassian Naive Bayes Classifier Modelling

The modeling begins by utilizing the Gaussian Naive Bayes classifier from the sklearn library. Initially, the data is split into training and validation sets to assess model performance on unseen data. To ensure feature compatibility, sklearn's robust scaler is applied. The subsequent step involves fine-tuning the model's hyperparameters, specifically the var smoothing parameter, through randomized search cross-validation from sklearn's model selection toolkit. Following training and predictions using the tuned model, % accuracy of 83.27% is achieved on the training set and 83.12% on the test dataset. The similarity in accuracy values between the test and training sets suggests no signs of overfitting.

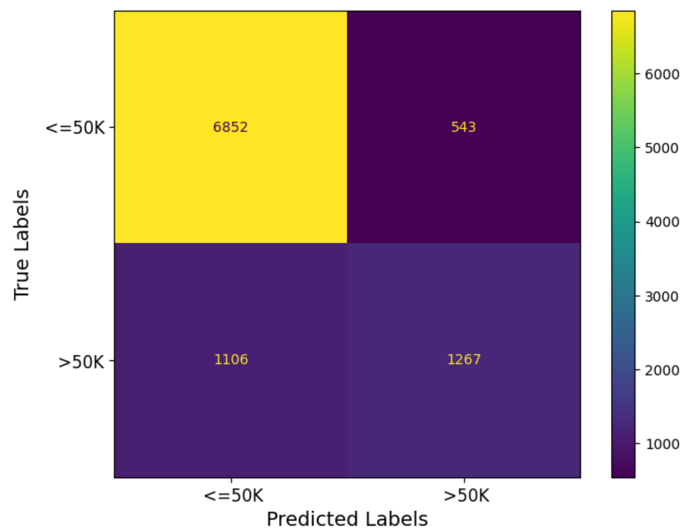


Fig. 15. Confusion Matrix.

Subsequently, evaluation metrics are examined, beginning with the Confusion Matrix. The analysis reveals 6,852 true positives, 1,267 true negatives, 543 false positives, and 1106 false negatives. Moving on to the classification report, an F1 score of 0.89 for incomes less than or equal to 50K and 0.61 for incomes greater than 50K is observed, with an accuracy of 0.83, a macro average of 0.76, and a weighted average of 0.83. These values indicate strong model performance.

TABLE I
CLASSIFICATION REPORT

Class	Precision	Recall	F1-Score	Support
<=50K	0.86	0.93	0.89	7395
>50K	0.70	0.53	0.61	2373
Accuracy			0.83	9768
Macro Avg	0.78	0.73	0.75	9768
Weighted Avg	0.82	0.83	0.82	9768

Next, the ROC curve is plotted, representing the false positive rate versus the true positive rate. The curve lies well above the y = x line, indicating good model discrimination, with an AUC value of 0.8843. Subsequently, variability in performance on testing and training datasets is assessed using 10-fold cross-validation. The mean accuracy is close to the original accuracy, with minimal deviation across folds, suggesting that the model's performance is not heavily reliant on the specific training data.

Finally, k-fold cross validation is done and it was found that the mean accuracy is close to the original one, and also there is not much deviation from the average for all the folds, thus it is clear that the model is not much reliant on the data on which it is being trained. Further more thresholds were tested to optimize accuracy on the test set. It is determined that a threshold of 0.8 yields the highest accuracy of 0.835.

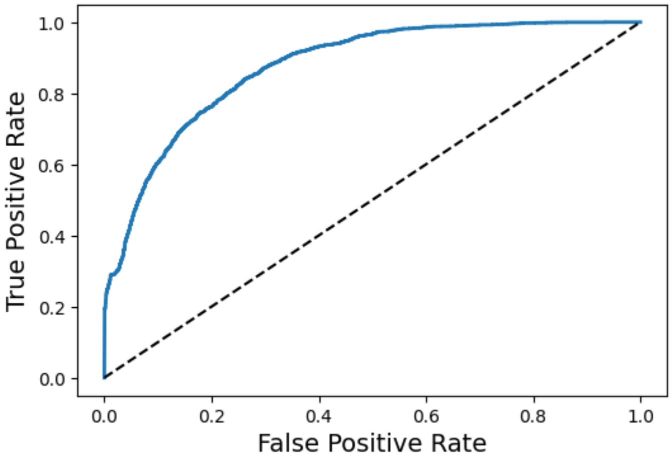


Fig. 16. ROC curve for Gaussian Naive Bayes Classifier for Predicting Salaries.

IV. Conclusion

In this study, it was observed that individuals who are male and aged (age ¿= 45) working longer hours are more inclined to earn annual wages exceeding $50K. Additionally, the data showed that most individuals typically undergo around 9 years of education. Still, those with more than 14 years of education and those who are self-employed are more likely to earn wages exceeding $50K annually. Furthermore, the analysis revealed that, on average, both women and men have similar educational levels, but women tend to work fewer hours and consequently receive lower salaries.

References

[1] "Naive Bayes Classifier," *Towards Data Science*, 2023. [Online]. Available: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c.

[2] "Naive Bayes Tutorial," *DataCamp*, 2023. [Online]. Available: https://www.datacamp.com/tutorial/naive-bayes-scikit-learn.

[3] "Naive Bayes classifier," *Wikipedia*, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

[4] "AUC-ROC Curve & Confusion Matrix Explained in Detail." [Online]. Available: https://www.kaggle.com/code/vithal2311/auc-roc-curve-confusion-matrix-explained-in-detail.

[5] Analytics Vidhya. "K-Fold Cross-Validation Technique and Its Essentials." [Online]. Available: https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/.

MM20B007 DAL Assignment 3

The key task is to determine whether a person makes over $50K a year.

```python
# Necessary Packages

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import table
import category_encoders as ce

from sklearn.feature_selection import VarianceThreshold
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler


path = '/content/drive/MyDrive/sem 7/EE5708/Assignment 3/adult.xlsx'

# Data

data = pd.read_excel(path)
df = data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   39               32560 non-null  int64
 1    State-gov       32560 non-null  object
 2   77516            32560 non-null  int64
 3    Bachelors       32560 non-null  object
 4   13               32560 non-null  int64
 5    Never-married   32560 non-null  object
 6    Adm-clerical    32560 non-null  object
 7    Not-in-family   32560 non-null  object
 8    White           32560 non-null  object
 9    Male            32560 non-null  object
 10  2174             32560 non-null  int64
 11  0                32560 non-null  int64
 12  40               32560 non-null  int64
```

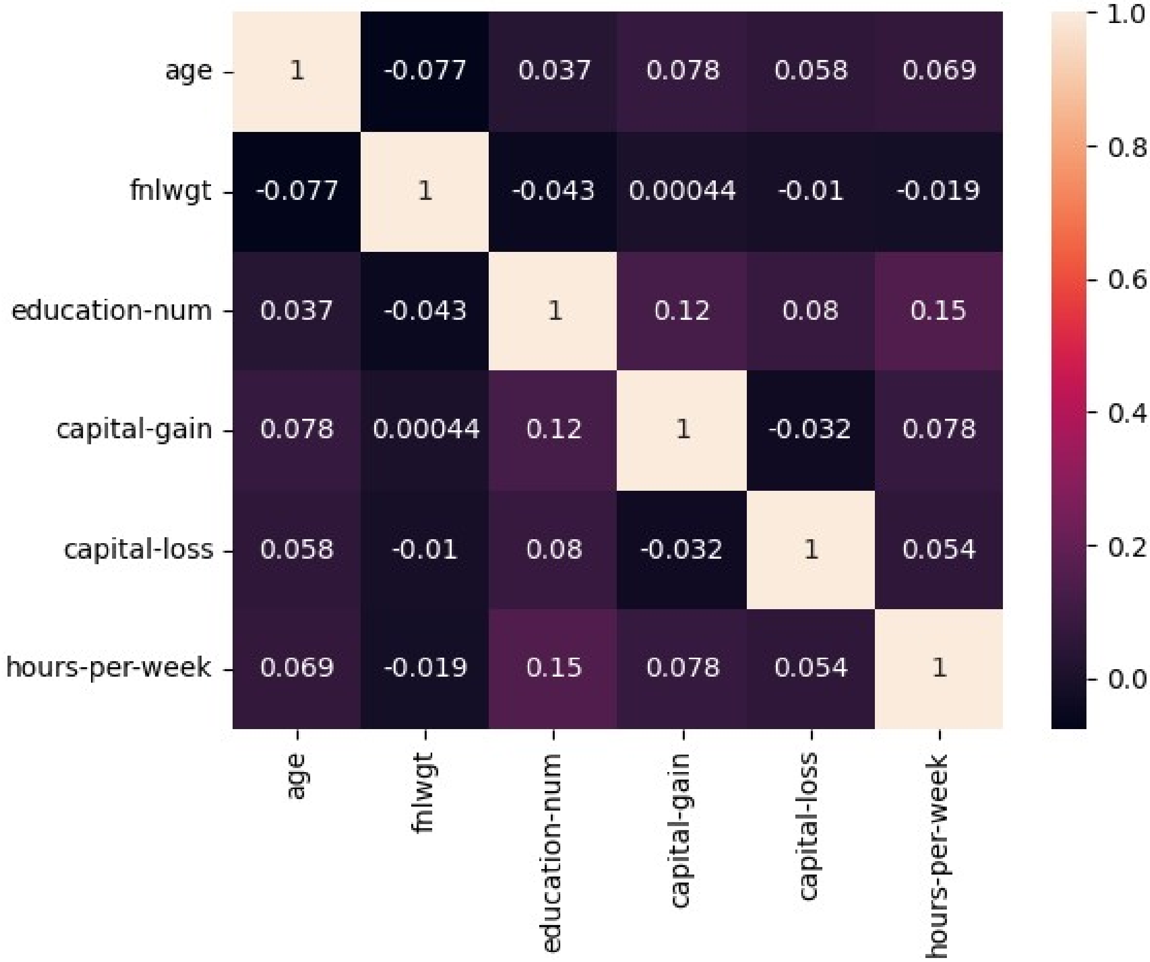
```
 13   United-States  32560 non-null  object
 14   <=50K          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
df.columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-
num', 'marital-status', 'Occupation', 'relationship', 'race', 'sex',
'capital-gain',
              'capital-loss', 'hours-per-week', 'native-country',
'income']

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32560 non-null  int64
 1   workclass       32560 non-null  object
 2   fnlwgt          32560 non-null  int64
 3   education       32560 non-null  object
 4   education-num   32560 non-null  int64
 5   marital-status  32560 non-null  object
 6   Occupation      32560 non-null  object
 7   relationship    32560 non-null  object
 8   race            32560 non-null  object
 9   sex             32560 non-null  object
 10  capital-gain    32560 non-null  int64
 11  capital-loss    32560 non-null  int64
 12  hours-per-week  32560 non-null  int64
 13  native-country  32560 non-null  object
 14  income          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
plt.plot(figsize = (10, 8))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot = True, xticklabels = True, yticklabels
= True)
```

```
<ipython-input-314-40074864e6d8>:2: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  corr_matrix = df.corr()

<Axes: >
```




```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32560 non-null  int64
 1   workclass       32560 non-null  object
 2   fnlwgt          32560 non-null  int64
 3   education       32560 non-null  object
 4   education-num   32560 non-null  int64
 5   marital-status  32560 non-null  object
 6   Occupation      32560 non-null  object
 7   relationship    32560 non-null  object
 8   race            32560 non-null  object
 9   sex             32560 non-null  object
 10  capital-gain    32560 non-null  int64
 11  capital-loss    32560 non-null  int64
 12  hours-per-week  32560 non-null  int64
```

```
 13  native-country  32560 non-null  object
 14  income          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
unique_classes = []
for cols in list(df.columns):
  if str(df[cols].dtypes) == 'object':
    unique_classes.append(df[cols].unique())

unique_classes
```

```
[array([' Self-emp-not-inc', ' Private', ' State-gov', ' Federal-gov',
        ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
        ' Never-worked'], dtype=object),
 array([' Bachelors', ' HS-grad', ' 11th', ' Masters', ' 9th',
        ' Some-college', ' Assoc-acdm', ' Assoc-voc', ' 7th-8th',
        ' Doctorate', ' Prof-school', ' 5th-6th', ' 10th', ' 1st-4th',
        ' Preschool', ' 12th'], dtype=object),
 array([' Married-civ-spouse', ' Divorced', ' Married-spouse-absent',
        ' Never-married', ' Separated', ' Married-AF-spouse', '
Widowed'],
       dtype=object),
 array([' Exec-managerial', ' Handlers-cleaners', ' Prof-specialty',
        ' Other-service', ' Adm-clerical', ' Sales', ' Craft-repair',
        ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
        ' Tech-support', ' ?', ' Protective-serv', ' Armed-Forces',
        ' Priv-house-serv'], dtype=object),
 array([' Husband', ' Not-in-family', ' Wife', ' Own-child', '
Unmarried',
        ' Other-relative'], dtype=object),
 array([' White', ' Black', ' Asian-Pac-Islander', ' Amer-Indian-
Eskimo',
        ' Other'], dtype=object),
 array([' Male', ' Female'], dtype=object),
 array([' United-States', ' Cuba', ' Jamaica', ' India', ' ?', '
Mexico',
        ' South', ' Puerto-Rico', ' Honduras', ' England', ' Canada',
        ' Germany', ' Iran', ' Philippines', ' Italy', ' Poland',
        ' Columbia', ' Cambodia', ' Thailand', ' Ecuador', ' Laos',
        ' Taiwan', ' Haiti', ' Portugal', ' Dominican-Republic',
        ' El-Salvador', ' France', ' Guatemala', ' China', ' Japan',
        ' Yugoslavia', ' Peru', ' Outlying-US(Guam-USVI-etc)', '
Scotland',
        ' Trinadad&Tobago', ' Greece', ' Nicaragua', ' Vietnam', '
Hong',
        ' Ireland', ' Hungary', ' Holand-Netherlands'], dtype=object),
 array([' <=50K', ' >50K'], dtype=object)]
```

```python
cols_with_missing_values = ['workclass', 'Occupation', 'native-country']
for items in cols_with_missing_values:
  print(df[items].value_counts())
```

```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: Occupation, dtype: int64
 United-States                 29169
 Mexico                          643
 ?                               583
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
```

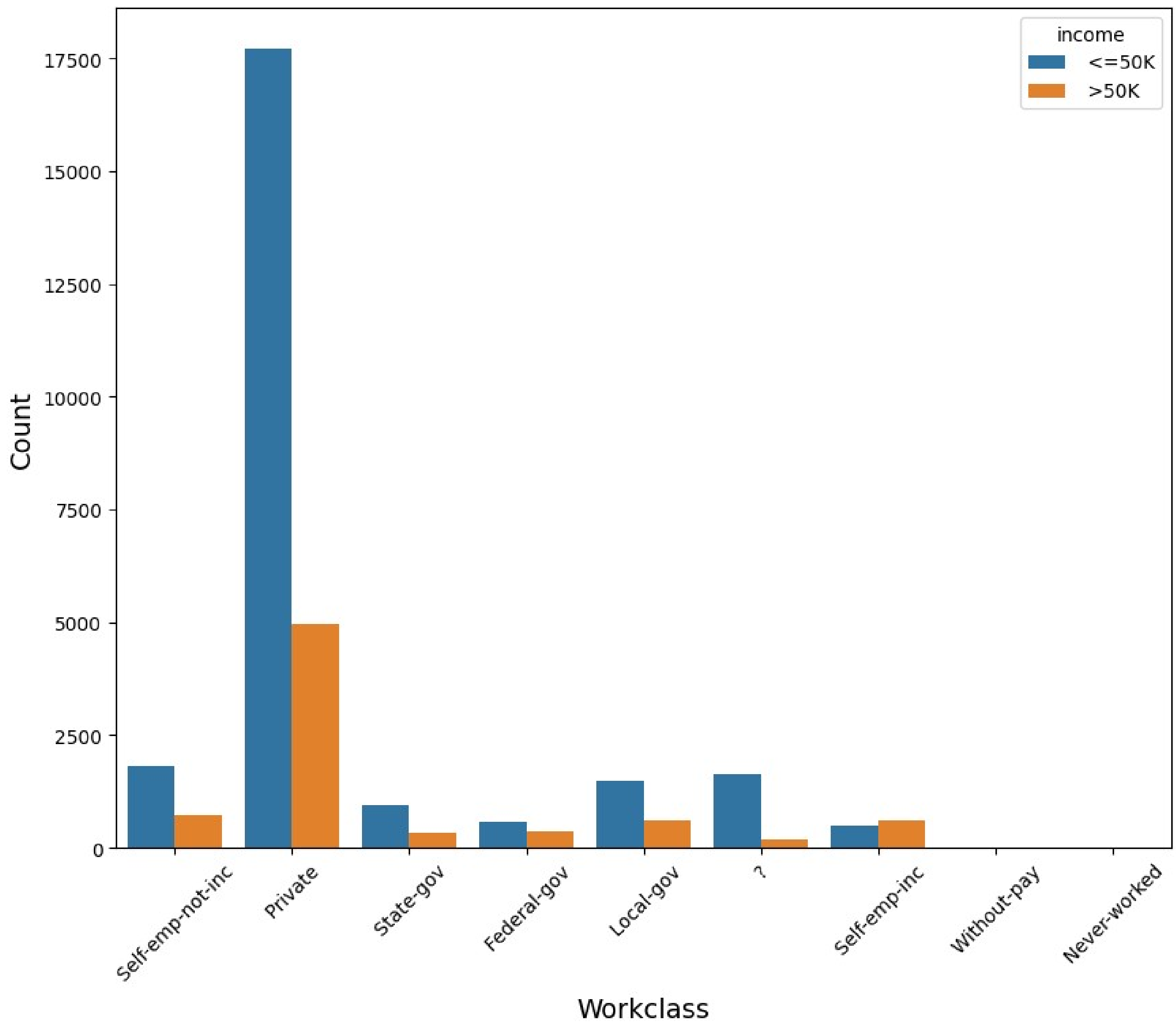
```
 Poland                                60
 Columbia                              59
 Taiwan                                51
 Haiti                                 44
 Iran                                  43
 Portugal                              37
 Nicaragua                             34
 Peru                                  31
 France                                29
 Greece                                29
 Ecuador                               28
 Ireland                               24
 Hong                                  20
 Cambodia                              19
 Trinadad&Tobago                       19
 Laos                                  18
 Thailand                              18
 Yugoslavia                            16
 Outlying-US(Guam-USVI-etc)            14
 Honduras                              13
 Hungary                               13
 Scotland                              12
 Holand-Netherlands                     1
Name: native-country, dtype: int64
```

1. We see there is a value '?' in some of the features, these features are 'workclass', 'Occupation', and 'native-country'.

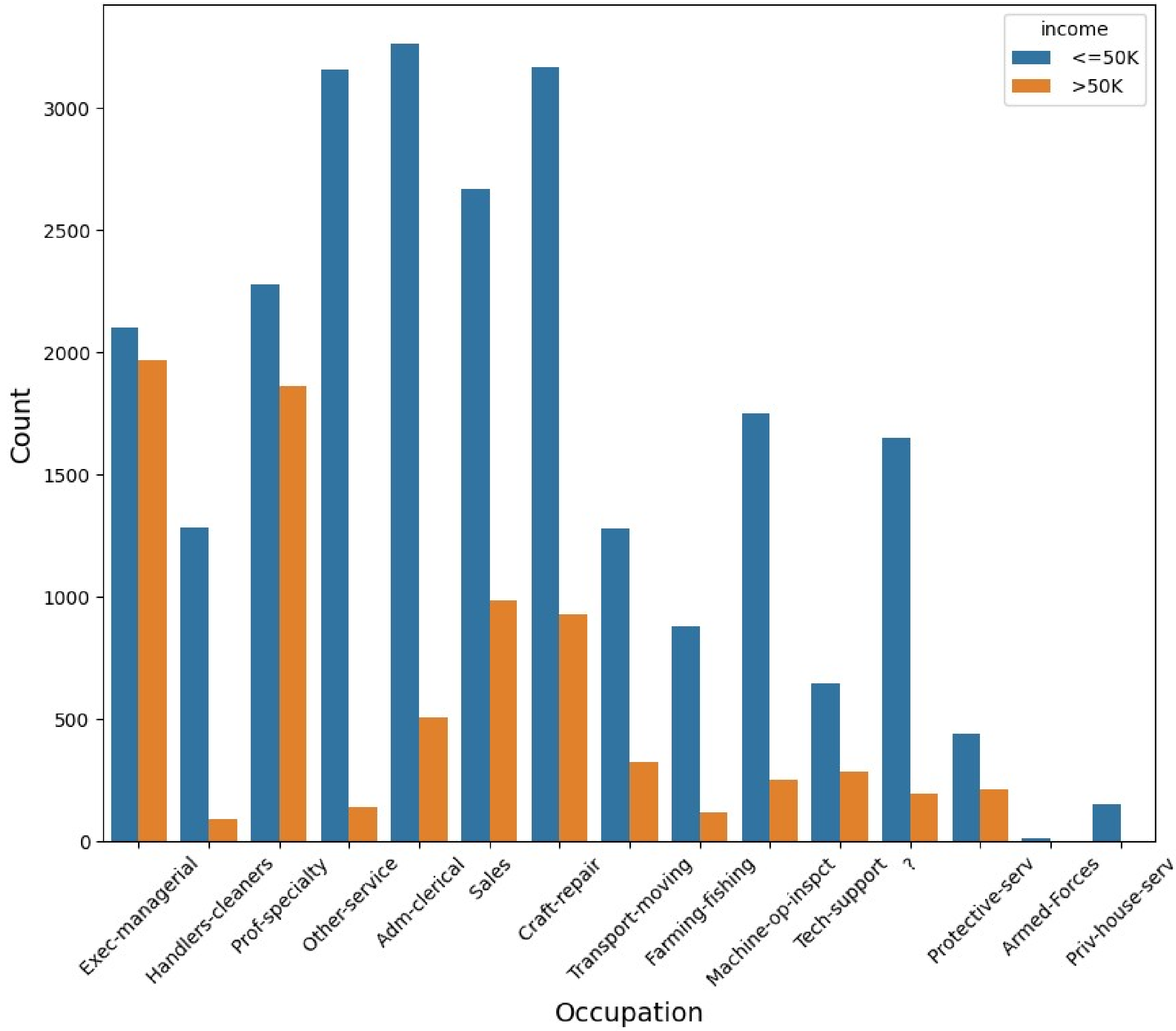
'workclass' has 1836 '?' values. 'Occupation' has 1843 '?' values. 'native-country' has 583 '?' values.

1. Rest of the features have unique features without any discrepancy.

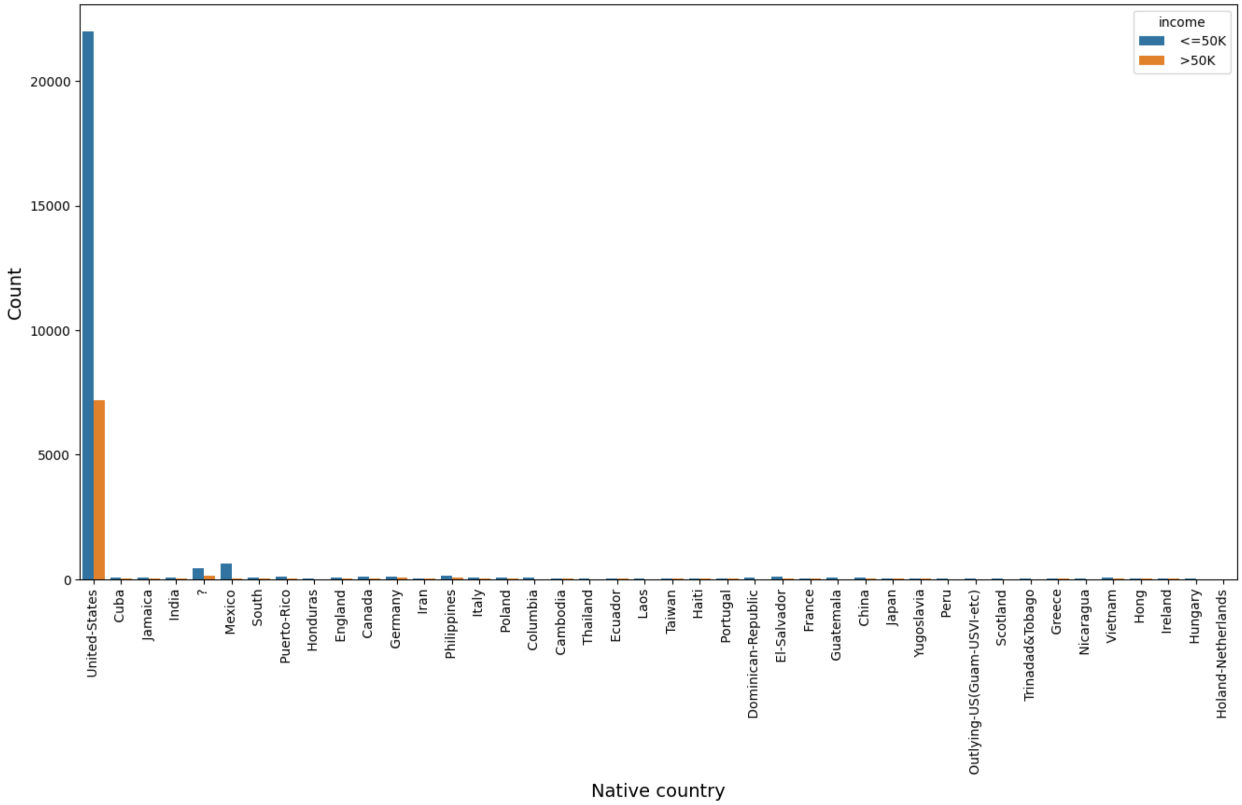
```
f, ax = plt.subplots(1, 1, figsize = (10, 8))
sns.countplot(x = 'workclass', data = df, hue = 'income', ax = ax)
plt.xticks(rotation = 45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Workclass', fontsize = 14)

Text(0.5, 0, 'Workclass')
```




```
f, ax = plt.subplots(1, 1, figsize = (10, 8))
sns.countplot(x = 'Occupation', data = df, hue = 'income', ax = ax)
plt.xticks(rotation = 45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Occupation', fontsize = 14)

Text(0.5, 0, 'Occupation')
```




```
f, ax = plt.subplots(1, 1, figsize = (16, 8))
sns.countplot(x = 'native-country', data = df, hue = 'income', ax =
ax)
plt.xticks(rotation = 90)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Native country', fontsize = 14)

Text(0.5, 0, 'Native country')
```




```
df
```

```
       age         workclass  fnlwgt   education  education-num  \
0       50  Self-emp-not-inc   83311   Bachelors             13
1       38           Private  215646     HS-grad              9
2       53           Private  234721        11th              7
3       28           Private  338409   Bachelors             13
4       37           Private  284582     Masters             14
...    ...               ...     ...         ...            ...
32555   27           Private  257302  Assoc-acdm             12
32556   40           Private  154374     HS-grad              9
32557   58           Private  151910     HS-grad              9
32558   22           Private  201490     HS-grad              9
32559   52      Self-emp-inc  287927     HS-grad              9

           marital-status          Occupation    relationship    race
\
0      Married-civ-spouse     Exec-managerial         Husband   White

1                Divorced   Handlers-cleaners   Not-in-family   White

2      Married-civ-spouse   Handlers-cleaners         Husband   Black

3      Married-civ-spouse      Prof-specialty            Wife   Black
```

```
4       Married-civ-spouse     Exec-managerial            Wife   White

...                    ...                 ...             ...     ...

32555   Married-civ-spouse        Tech-support            Wife   White

32556   Married-civ-spouse   Machine-op-inspct         Husband   White

32557              Widowed        Adm-clerical       Unmarried   White

32558        Never-married        Adm-clerical       Own-child   White

32559   Married-civ-spouse     Exec-managerial            Wife   White


           sex  capital-gain  capital-loss  hours-per-week  native-
country  \
0         Male             0             0              13   United-
States
1         Male             0             0              40   United-
States
2         Male             0             0              40   United-
States
3       Female             0             0              40
Cuba
4       Female             0             0              40   United-
States
...        ...           ...           ...             ...
...
32555   Female             0             0              38   United-
States
32556     Male             0             0              40   United-
States
32557   Female             0             0              40   United-
States
32558     Male             0             0              20   United-
States
32559   Female         15024             0              40   United-
States

       income
0       <=50K
1       <=50K
2       <=50K
3       <=50K
4       <=50K
...       ...
32555   <=50K
32556    >50K
32557   <=50K
```

```
32558    <=50K
32559     >50K

[32560 rows x 15 columns]
```

Taking care of '?' value.

Workclass

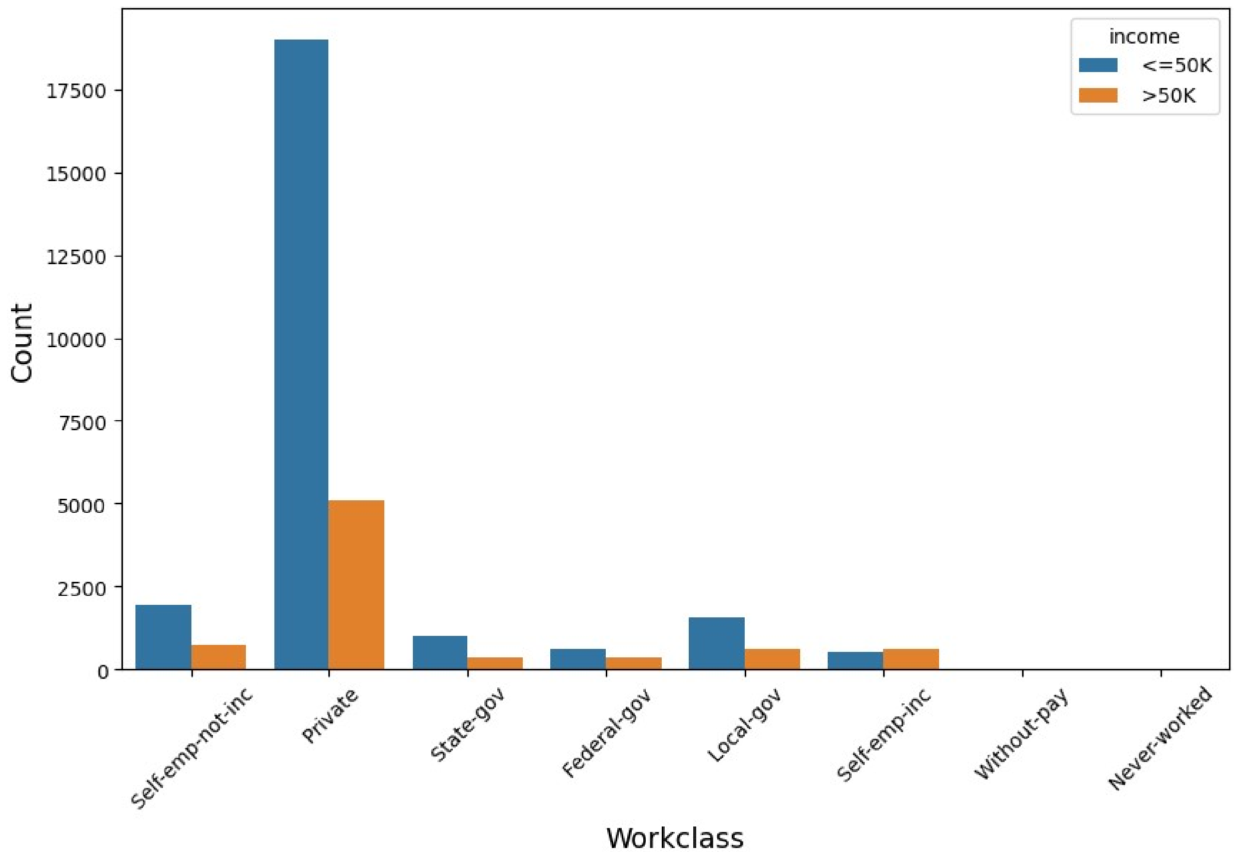
```
print('\n')
print('***************************************************************
**********************************************************************
*****************************')
workclass_before_resampling = df['workclass'].value_counts().to_dict()
workclass_before_resampling
```

```


**********************************************************************
**********************************************************************
**********************

{' Private': 22696,
 ' Self-emp-not-inc': 2541,
 ' Local-gov': 2093,
 ' ?': 1836,
 ' State-gov': 1297,
 ' Self-emp-inc': 1116,
 ' Federal-gov': 960,
 ' Without-pay': 14,
 ' Never-worked': 7}
```

```
df_workspace_ffill = df[['workclass', 'income']]
df_workspace_ffill['workclass'].replace(' ?', np.NaN, inplace = True)
df_workspace_ffill['workclass'].fillna(method = 'ffill', inplace = 
True)
df_workspace_ffill['workclass'].value_counts()

plt.figure(figsize = (10, 6))
ax = sns.countplot(x = 'workclass', data = df_workspace_ffill, hue = 
'income')
plt.xticks(rotation=45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Workclass', fontsize = 14)

print('\n')
print('***************************************************************
**********************************************************************
*****************************')
```

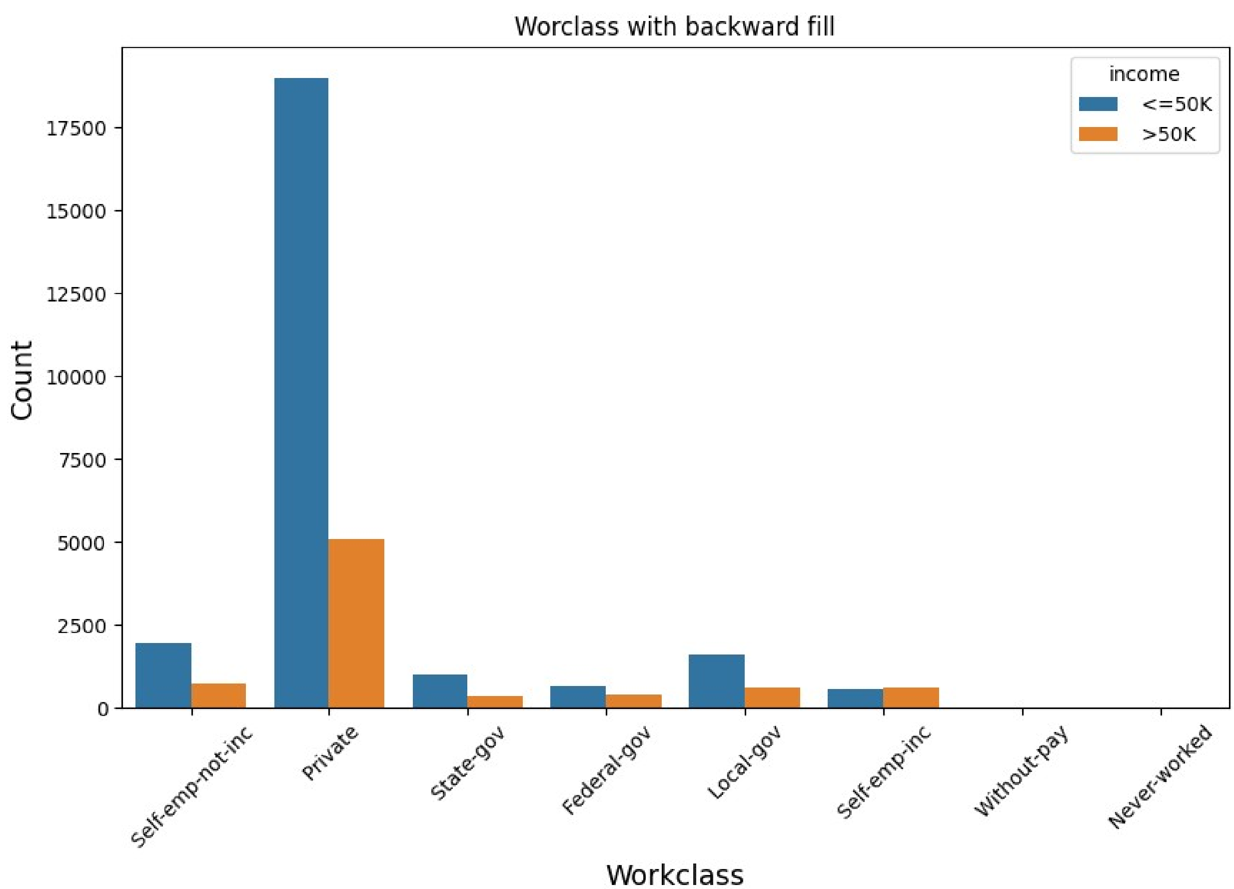
```
after_ffill_workclass =
df_workspace_ffill['workclass'].value_counts().to_dict()
```

```
<ipython-input-323-347ac2e3a97d>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_workspace_ffill['workclass'].replace(' ?', np.NaN, inplace =
True)
<ipython-input-323-347ac2e3a97d>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_workspace_ffill['workclass'].fillna(method = 'ffill', inplace =
True)



**********************************************************************
**********************************************************************
**********************
```




```
df_workspace_bfill = df[['workclass', 'income']]
df_workspace_bfill['workclass'].replace(' ?', np.NaN, inplace = True)
df_workspace_bfill['workclass'].fillna(method = 'bfill', inplace =
True)
df_workspace_bfill['workclass'].value_counts()

plt.figure(figsize = (10, 6))
ax = sns.countplot(x = 'workclass', data = df_workspace_bfill, hue =
'income')
plt.xticks(rotation=45)
plt.title('Worclass with backward fill')
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Workclass', fontsize = 14)

print('\n')
print('**********************************************************************
**********************************************************************
******************************')
after_bfill_workclass =
df_workspace_bfill['workclass'].value_counts().to_dict()

<ipython-input-324-eec52a7cad13>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_workspace_bfill['workclass'].replace(' ?', np.NaN, inplace =
True)
<ipython-input-324-eec52a7cad13>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_workspace_bfill['workclass'].fillna(method = 'bfill', inplace =
True)



**********************************************************************
**********************************************************************
**********************
```




```python
workclass_data_resampling = {
    'Before resampling': workclass_before_resampling,
```

```
    'After forward fill': after_ffill_workclass,
    'After backward fill': after_bfill_workclass
}

workclass_table = pd.DataFrame(workclass_data_resampling)

fig, ax = plt.subplots(figsize=(10, 6))  # Adjust the figsize as
needed
ax.axis("off")
tbl = table(ax, workclass_table, loc="center", cellLoc="center",
colWidths=[0.2] * len(workclass_table.columns))
tbl.auto_set_font_size(False)
tbl.set_fontsize(10)
tbl.scale(2, 2)  # Adjust the scale as needed
plt.show()
```

	Before resampling	After forward fill	After backward fill
Private	22696.0	24094.0	24056.0
Self-emp-not-inc	2541.0	2688.0	2701.0
Local-gov	2093.0	2204.0	2212.0
?	1836.0	nan	nan
State-gov	1297.0	1373.0	1373.0
Self-emp-inc	1116.0	1177.0	1182.0
Federal-gov	960.0	1002.0	1013.0
Without-pay	14.0	15.0	16.0
Never-worked	7.0	7.0	7.0

Occupation

```
print('\n')
print('***************************************************************
**********************************************************************
*****************************')
occupation_before_resampling =
df['Occupation'].value_counts().to_dict()
occupation_before_resampling
```

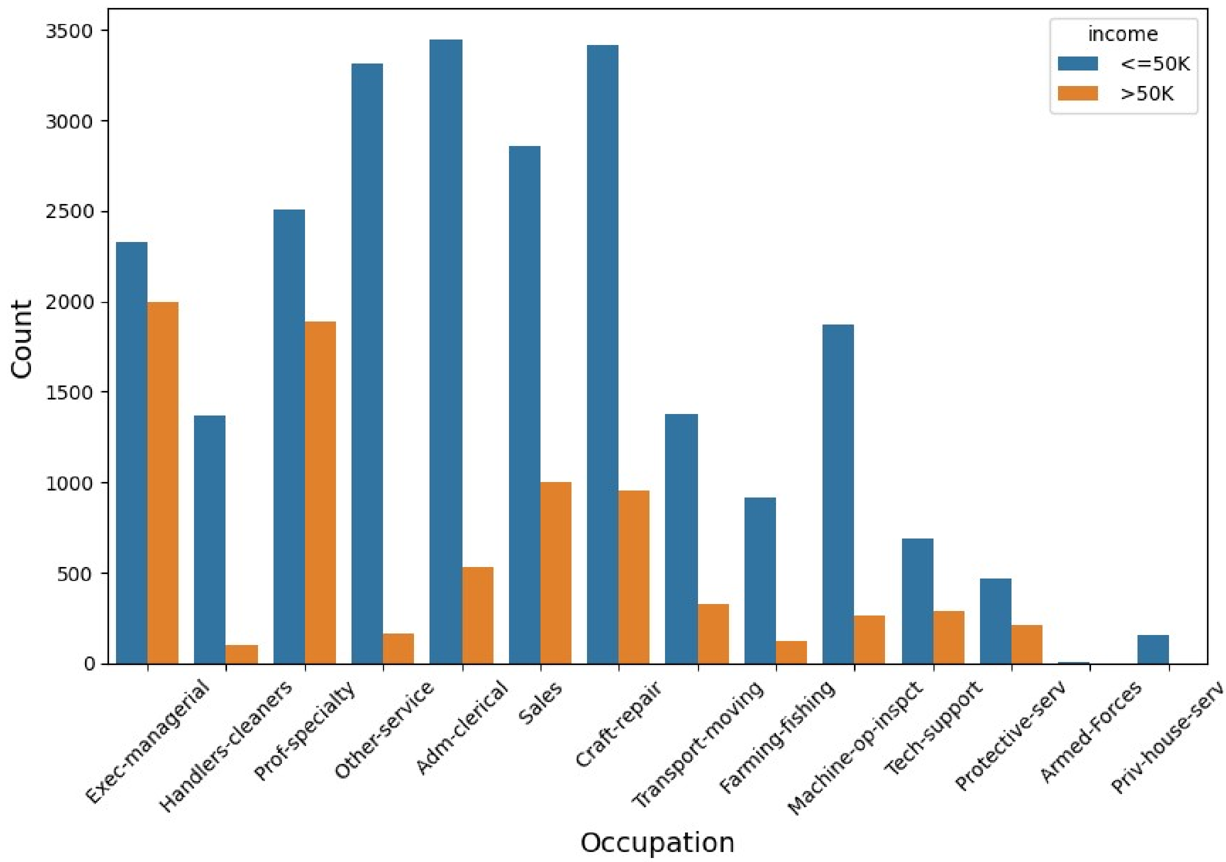
```


**********************************************************************
**********************************************************************
**********************
```

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

```
df_Occupation_ffill = df[['Occupation', 'income']]
df_Occupation_ffill['Occupation'].replace(' ?', np.NaN, inplace =
True)
df_Occupation_ffill['Occupation'].fillna(method = 'ffill', inplace =
True)
df_Occupation_ffill['Occupation'].value_counts()

plt.figure(figsize = (10, 6))
ax = sns.countplot(x = 'Occupation', data = df_Occupation_ffill, hue =
'income')
plt.xticks(rotation=45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Occupation', fontsize = 14)

print('\n')
print('**********************************************************************
**********************************************************************
******************************')
after_ffill_occupation =
df_Occupation_ffill['Occupation'].value_counts().to_dict()
```

```
<ipython-input-327-81ef09ff930c>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_Occupation_ffill['Occupation'].replace(' ?', np.NaN, inplace =
True)
<ipython-input-327-81ef09ff930c>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
```

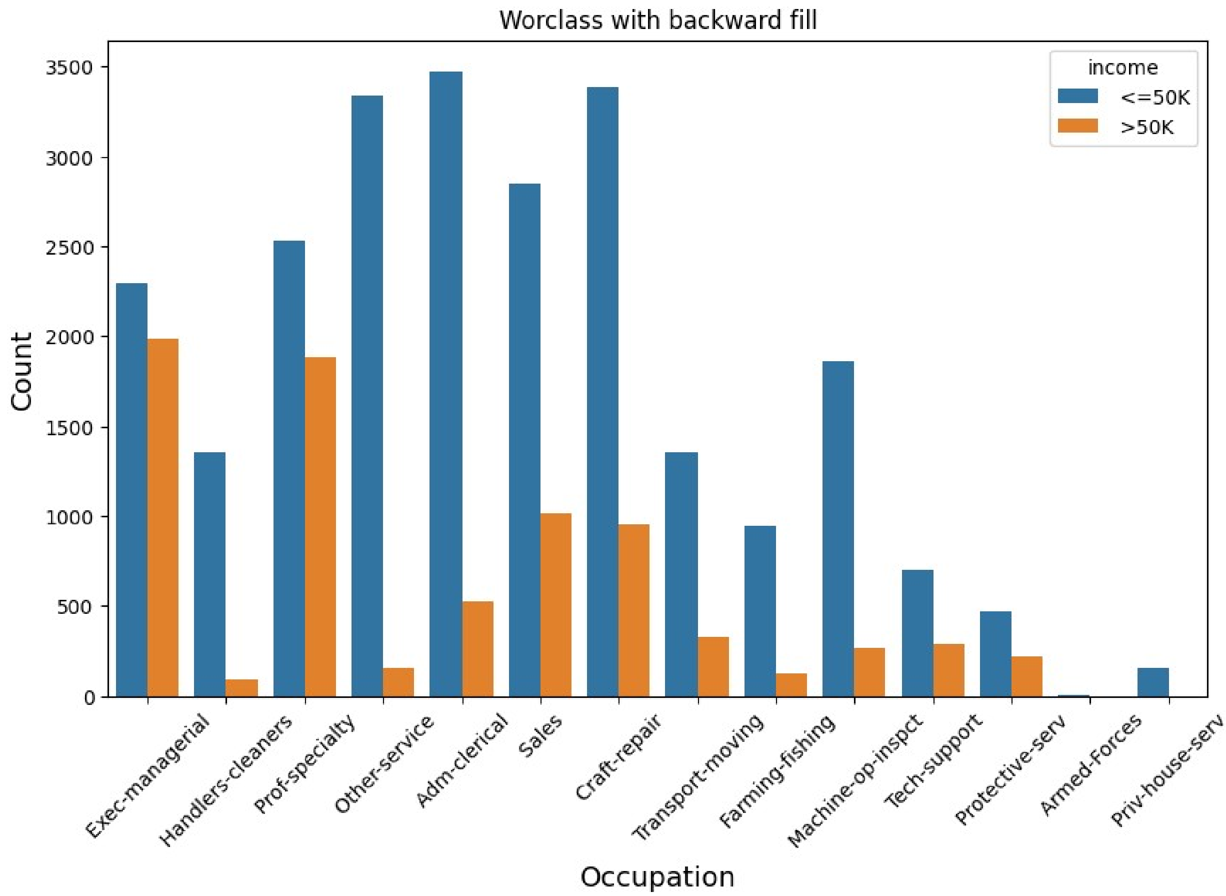
```
returning-a-view-versus-a-copy
  df_Occupation_ffill['Occupation'].fillna(method = 'ffill', inplace =
True)


************************************************************************
************************************************************************
**********************
```




```
df_Occupation_bfill = df[['Occupation', 'income']]
df_Occupation_bfill['Occupation'].replace(' ?', np.NaN, inplace =
True)
df_Occupation_bfill['Occupation'].fillna(method = 'bfill', inplace =
True)
df_Occupation_bfill['Occupation'].value_counts()

plt.figure(figsize = (10, 6))
ax = sns.countplot(x = 'Occupation', data = df_Occupation_bfill, hue =
'income')
plt.xticks(rotation=45)
plt.title('Worclass with backward fill')
plt.ylabel('Count', fontsize = 14)
```

```
plt.xlabel('Occupation', fontsize = 14)

print('\n')
print('*******************************************************************
**********************************************************************
******************************')
after_bfill_occupation =
df_Occupation_bfill['Occupation'].value_counts().to_dict()
```

```
<ipython-input-328-6fe3d0c84922>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_Occupation_bfill['Occupation'].replace(' ?', np.NaN, inplace =
True)
<ipython-input-328-6fe3d0c84922>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_Occupation_bfill['Occupation'].fillna(method = 'bfill', inplace =
True)



**********************************************************************
**********************************************************************
**********************
```




```
occupation_data_resampling = {
    'Before resampling': occupation_before_resampling,
    'After forward fill': after_ffill_occupation,
    'After backward fill': after_bfill_occupation
}

occupation_table = pd.DataFrame(occupation_data_resampling)

fig, ax = plt.subplots(figsize=(10, 6))  # Adjust the figsize as
needed
ax.axis("off")
tbl = table(ax, occupation_table, loc="center", cellLoc="center",
colWidths=[0.2] * len(occupation_table.columns))
tbl.auto_set_font_size(False)
tbl.set_fontsize(10)
tbl.scale(1.5, 1.5)  # Adjust the scale as needed
plt.show()
```

	Before resampling	After forward fill	After backward fill
Prof-specialty	4140.0	4386.0	4410.0
Craft-repair	4099.0	4364.0	4339.0
Exec-managerial	4066.0	4317.0	4287.0
Adm-clerical	3769.0	3981.0	3998.0
Sales	3650.0	3863.0	3867.0
Other-service	3295.0	3470.0	3493.0
Machine-op-inspct	2002.0	2134.0	2128.0
?	1843.0	nan	nan
Transport-moving	1597.0	1703.0	1686.0
Handlers-cleaners	1370.0	1471.0	1446.0
Farming-fishing	994.0	1038.0	1069.0
Tech-support	928.0	981.0	984.0
Protective-serv	649.0	683.0	689.0
Priv-house-serv	149.0	159.0	155.0
Armed-Forces	9.0	10.0	9.0

Native country

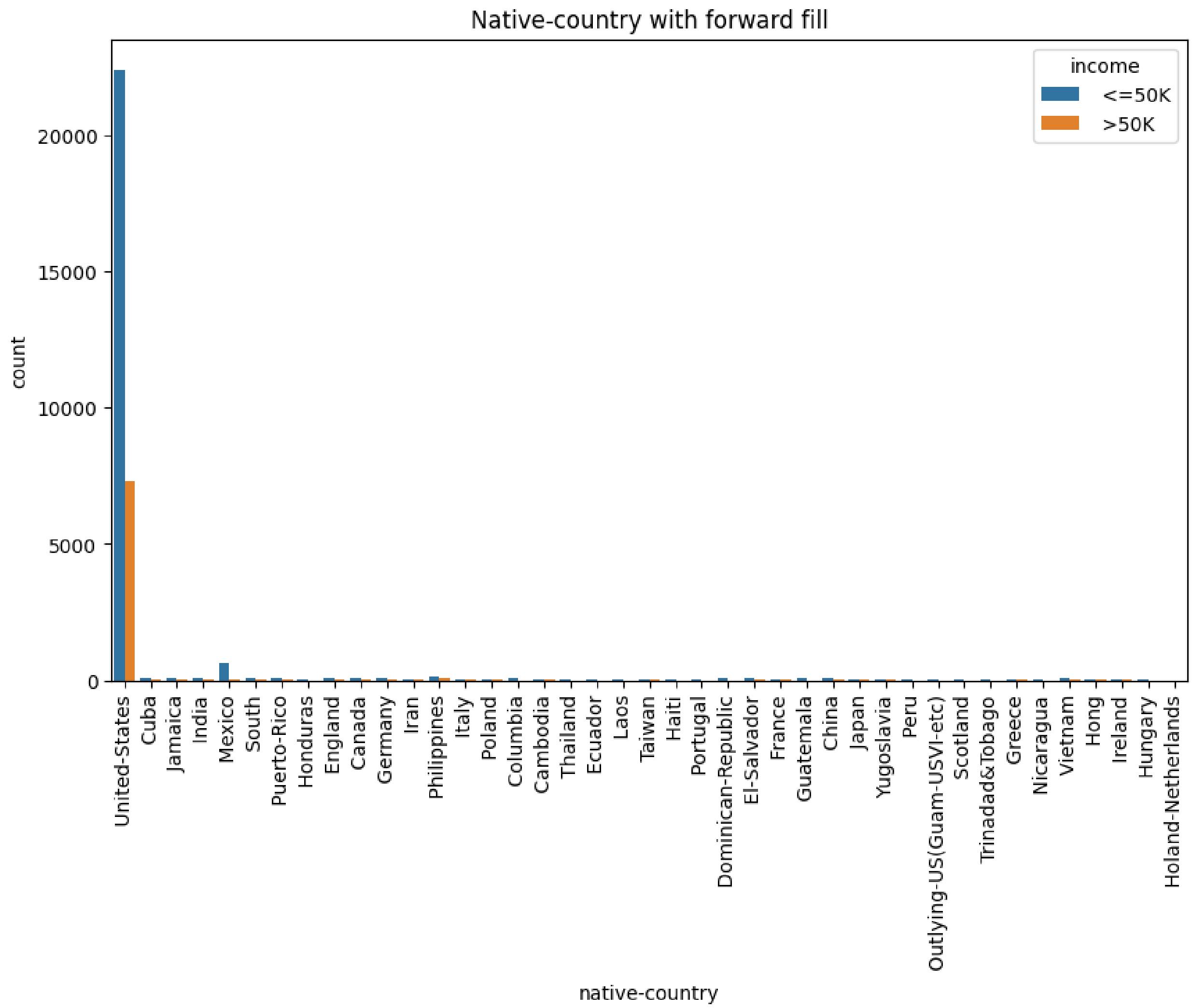
```
print('\n')
print('**********************************************************************
**********************************************************************
******************************')
native_before_resampling = df['native-
country'].value_counts().to_dict()
native_before_resampling
```

```
**********************************************************************
**********************************************************************
**********************

{' United-States': 29169,
 ' Mexico': 643,
 ' ?': 583,
 ' Philippines': 198,
 ' Germany': 137,
 ' Canada': 121,
 ' Puerto-Rico': 114,
 ' El-Salvador': 106,
 ' India': 100,
 ' Cuba': 95,
```

```
 ' England': 90,
 ' Jamaica': 81,
 ' South': 80,
 ' China': 75,
 ' Italy': 73,
 ' Dominican-Republic': 70,
 ' Vietnam': 67,
 ' Guatemala': 64,
 ' Japan': 62,
 ' Poland': 60,
 ' Columbia': 59,
 ' Taiwan': 51,
 ' Haiti': 44,
 ' Iran': 43,
 ' Portugal': 37,
 ' Nicaragua': 34,
 ' Peru': 31,
 ' France': 29,
 ' Greece': 29,
 ' Ecuador': 28,
 ' Ireland': 24,
 ' Hong': 20,
 ' Cambodia': 19,
 ' Trinadad&Tobago': 19,
 ' Laos': 18,
 ' Thailand': 18,
 ' Yugoslavia': 16,
 ' Outlying-US(Guam-USVI-etc)': 14,
 ' Honduras': 13,
 ' Hungary': 13,
 ' Scotland': 12,
 ' Holand-Netherlands': 1}

df_native_country_ffill = df[['native-country', 'income']]
df_native_country_ffill['native-country'].replace(' ?', np.NaN,
inplace = True)
df_native_country_ffill['native-country'].fillna(method = 'ffill',
inplace = True)
df_native_country_ffill['native-country'].value_counts()

plt.figure(figsize = (10, 6))
ax = sns.countplot(x = 'native-country', data =
df_native_country_ffill, hue = 'income')
plt.xticks(rotation=90)
plt.title('Native-country with forward fill')

print('\n')
print('**********************************************************************
**********************************************************************
******************************')
```

```
after_ffill_native = df_native_country_ffill['native-
country'].value_counts().to_dict()
```

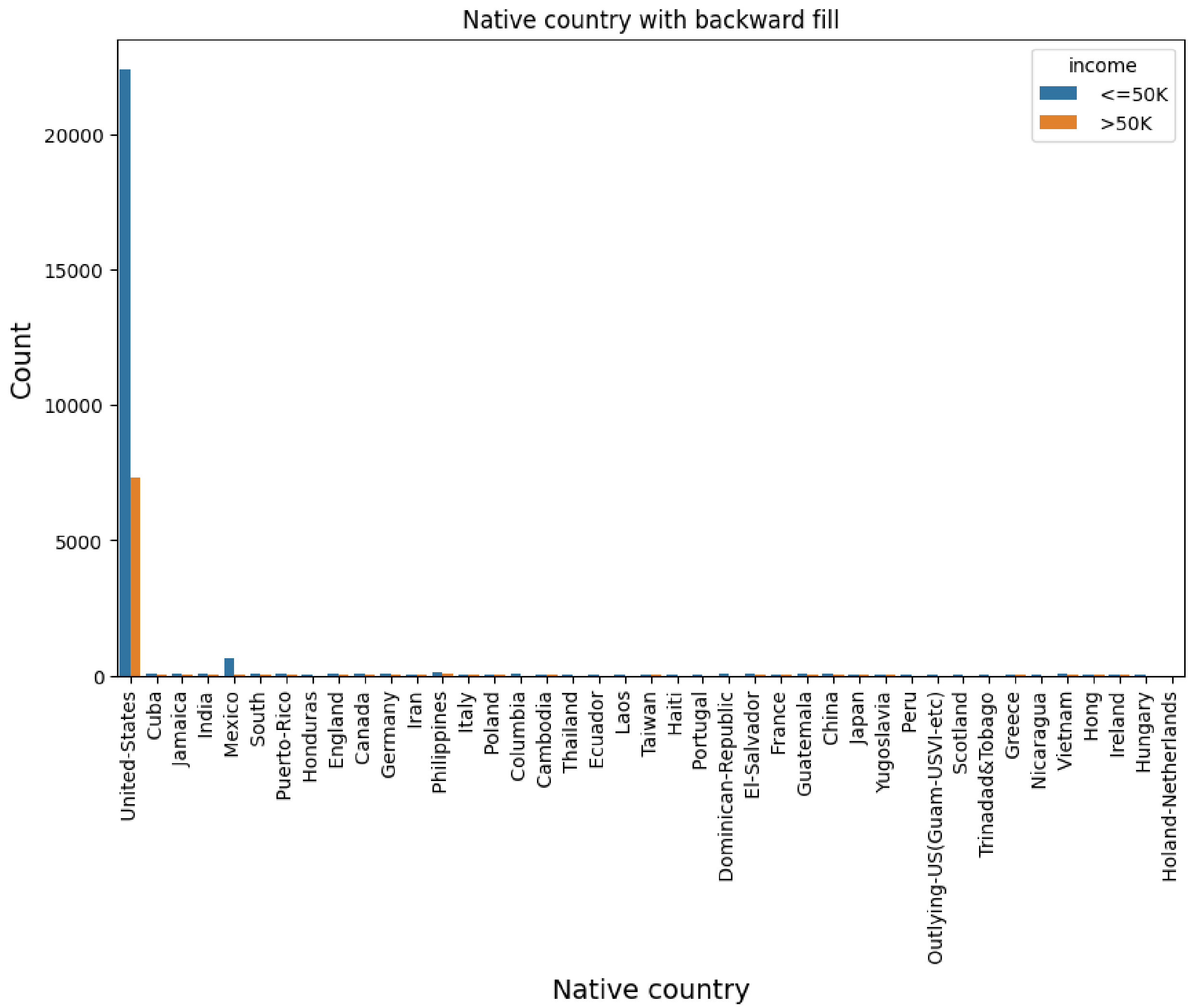
```
<ipython-input-331-f5fc5931a7a0>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_native_country_ffill['native-country'].replace(' ?', np.NaN,
inplace = True)
<ipython-input-331-f5fc5931a7a0>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_native_country_ffill['native-country'].fillna(method = 'ffill',
inplace = True)



**********************************************************************
**********************************************************************
**********************
```




```
df_native_country_bfill = df[['native-country', 'income']]
df_native_country_bfill['native-country'].replace(' ?', np.NaN,
inplace = True)
df_native_country_bfill['native-country'].fillna(method = 'bfill',
inplace = True)
df_native_country_bfill['native-country'].value_counts()

plt.figure(figsize = (10, 6))
ax = sns.countplot(x = 'native-country', data =
df_native_country_bfill, hue = 'income')
plt.xticks(rotation=90)
plt.title('Native country with backward fill')
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Native country', fontsize = 14)

print('\n')
print('*****************************************************************
*****************************************************************
```

```
****************************')
after_bfill_native = df_native_country_bfill['native-
country'].value_counts().to_dict()
```

```
<ipython-input-332-00abfc8eca65>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_native_country_bfill['native-country'].replace(' ?', np.NaN,
inplace = True)
<ipython-input-332-00abfc8eca65>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_native_country_bfill['native-country'].fillna(method = 'bfill',
inplace = True)



**********************************************************************
**********************************************************************
**********************
```




```
native_country_data_resampling = {
    'Before resampling': native_before_resampling,
    'After forward fill': after_ffill_native,
    'After backward fill': after_bfill_native
}

native_country_table = pd.DataFrame(native_country_data_resampling)

fig, ax = plt.subplots(figsize=(10, 6))  # Adjust the figsize as
needed
ax.axis("off")
tbl = table(ax, native_country_table, loc="center", cellLoc="center",
colWidths=[0.2] * len(native_country_table.columns))
tbl.auto_set_font_size(False)
tbl.set_fontsize(10)
tbl.scale(1.5, 1.5)  # Adjust the scale as needed

plt.show()
```

	Before resampling	After forward fill	After backward fill
United-States	29169.0	29693.0	29694.0
Mexico	643.0	657.0	652.0
?	583.0	nan	nan
Philippines	198.0	200.0	201.0
Germany	137.0	141.0	143.0
Canada	121.0	124.0	122.0
Puerto-Rico	114.0	118.0	119.0
El-Salvador	106.0	109.0	107.0
India	100.0	101.0	107.0
Cuba	95.0	97.0	97.0
England	90.0	93.0	91.0
Jamaica	81.0	83.0	82.0
South	80.0	80.0	82.0
China	75.0	77.0	77.0
Italy	73.0	73.0	74.0
Dominican-Republic	70.0	74.0	71.0
Vietnam	67.0	72.0	69.0
Guatemala	64.0	66.0	69.0
Japan	62.0	63.0	62.0
Poland	60.0	60.0	60.0
Columbia	59.0	61.0	61.0
Taiwan	51.0	51.0	51.0
Haiti	44.0	45.0	44.0
Iran	43.0	43.0	44.0
Portugal	37.0	37.0	38.0
Nicaragua	34.0	34.0	35.0
Peru	31.0	31.0	31.0
France	29.0	29.0	30.0
Greece	29.0	30.0	29.0
Ecuador	28.0	28.0	29.0
Ireland	24.0	24.0	24.0
Hong	20.0	20.0	20.0
Cambodia	19.0	20.0	19.0
Trinadad&Tobago	19.0	19.0	20.0
Laos	18.0	19.0	18.0
Thailand	18.0	18.0	18.0
Yugoslavia	16.0	17.0	16.0
Outlying-US(Guam-USVI-etc)	14.0	14.0	14.0
Honduras	13.0	13.0	14.0
Hungary	13.0	13.0	13.0
Scotland	12.0	12.0	12.0
Holand-Netherlands	1.0	1.0	1.0

Since there is no large variation with respect to forward fill resampling and backward fill resampling. Let's move ahead with forward fill resampling.

```
df['workclass'] = df_workspace_ffill['workclass']
df['Occupation'] = df_Occupation_ffill['Occupation']
df['native-country'] = df_native_country_ffill['native-country']
```

```
for cols in list(df.columns):
  if str(df[cols].dtypes) == 'object':
    print(df[cols].unique())
```

```
[' Self-emp-not-inc' ' Private' ' State-gov' ' Federal-gov' ' Local-
gov'
 ' Self-emp-inc' ' Without-pay' ' Never-worked']
[' Bachelors' ' HS-grad' ' 11th' ' Masters' ' 9th' ' Some-college'
 ' Assoc-acdm' ' Assoc-voc' ' 7th-8th' ' Doctorate' ' Prof-school'
 ' 5th-6th' ' 10th' ' 1st-4th' ' Preschool' ' 12th']
[' Married-civ-spouse' ' Divorced' ' Married-spouse-absent'
 ' Never-married' ' Separated' ' Married-AF-spouse' ' Widowed']
[' Exec-managerial' ' Handlers-cleaners' ' Prof-specialty'
 ' Other-service' ' Adm-clerical' ' Sales' ' Craft-repair'
 ' Transport-moving' ' Farming-fishing' ' Machine-op-inspct'
 ' Tech-support' ' Protective-serv' ' Armed-Forces' ' Priv-house-
serv']
[' Husband' ' Not-in-family' ' Wife' ' Own-child' ' Unmarried'
 ' Other-relative']
[' White' ' Black' ' Asian-Pac-Islander' ' Amer-Indian-Eskimo' '
Other']
[' Male' ' Female']
[' United-States' ' Cuba' ' Jamaica' ' India' ' Mexico' ' South'
 ' Puerto-Rico' ' Honduras' ' England' ' Canada' ' Germany' ' Iran'
 ' Philippines' ' Italy' ' Poland' ' Columbia' ' Cambodia' ' Thailand'
 ' Ecuador' ' Laos' ' Taiwan' ' Haiti' ' Portugal' ' Dominican-
Republic'
 ' El-Salvador' ' France' ' Guatemala' ' China' ' Japan' ' Yugoslavia'
 ' Peru' ' Outlying-US(Guam-USVI-etc)' ' Scotland' ' Trinadad&Tobago'
 ' Greece' ' Nicaragua' ' Vietnam' ' Hong' ' Ireland' ' Hungary'
 ' Holand-Netherlands']
[' <=50K' ' >50K']
```

Exploratory Data Analysis

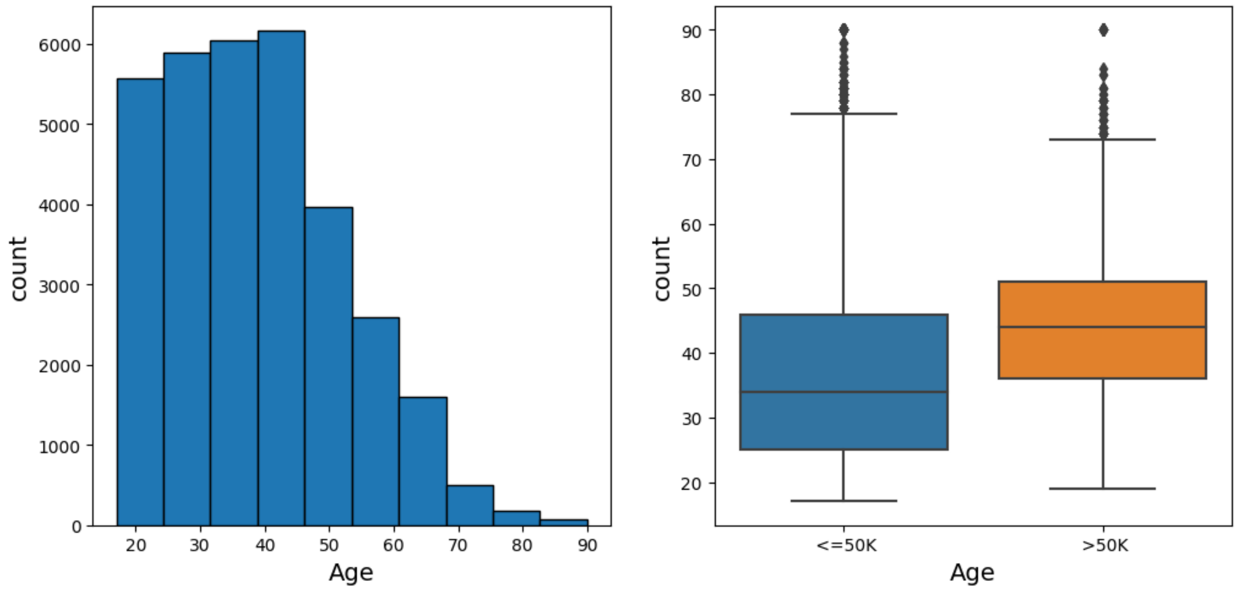
```
df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'Occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-
country',
       'income'],
      dtype='object')
```

Age vs Income

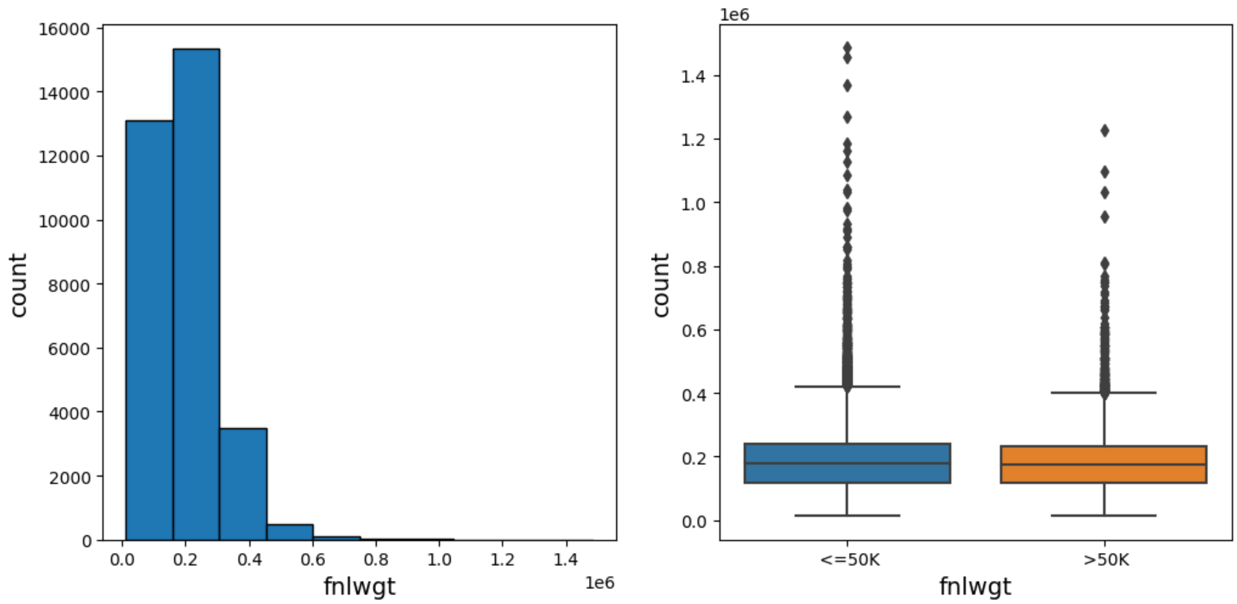
```
# Creating histograms and boxplot for numerical variables
f,ax=plt.subplots(1,2,figsize=(12,5.5))
ax[0].hist(df.age, edgecolor="black")
```

```
ax[0].set_xlabel('Age', fontsize = 14)
ax[0].set_ylabel('count', fontsize = 14)
sns.boxplot(x='income', y='age', data=df, ax=ax[1])
ax[1].set_xlabel('Age', fontsize = 14)
ax[1].set_ylabel('count', fontsize = 14)
plt.show()
```



fnlwgt vs income

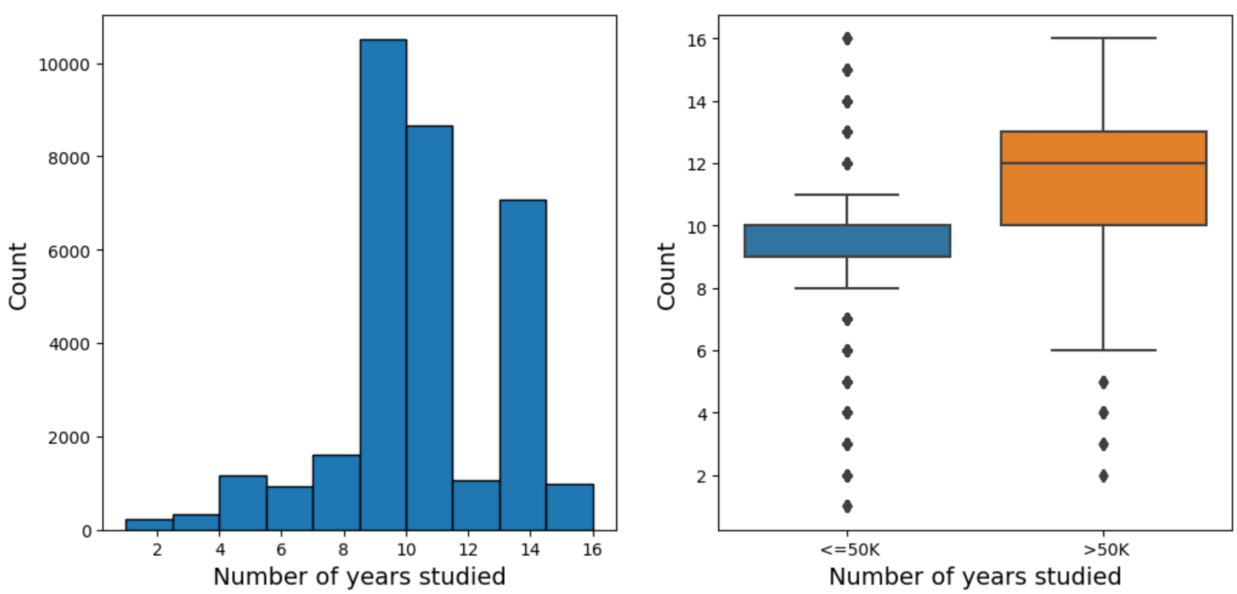
```
f,ax=plt.subplots(1,2,figsize=(12,5.5))
print(ax.shape)
ax[0].hist(df.fnlwgt, edgecolor="black")
ax[0].set_xlabel('fnlwgt', fontsize = 14)
ax[0].set_ylabel('count', fontsize = 14)
sns.boxplot(x='income', y='fnlwgt', data=df, ax=ax[1])
ax[1].set_xlabel('fnlwgt', fontsize = 14)
ax[1].set_ylabel('count', fontsize = 14)
plt.show()

(2,)
```



education num vs income

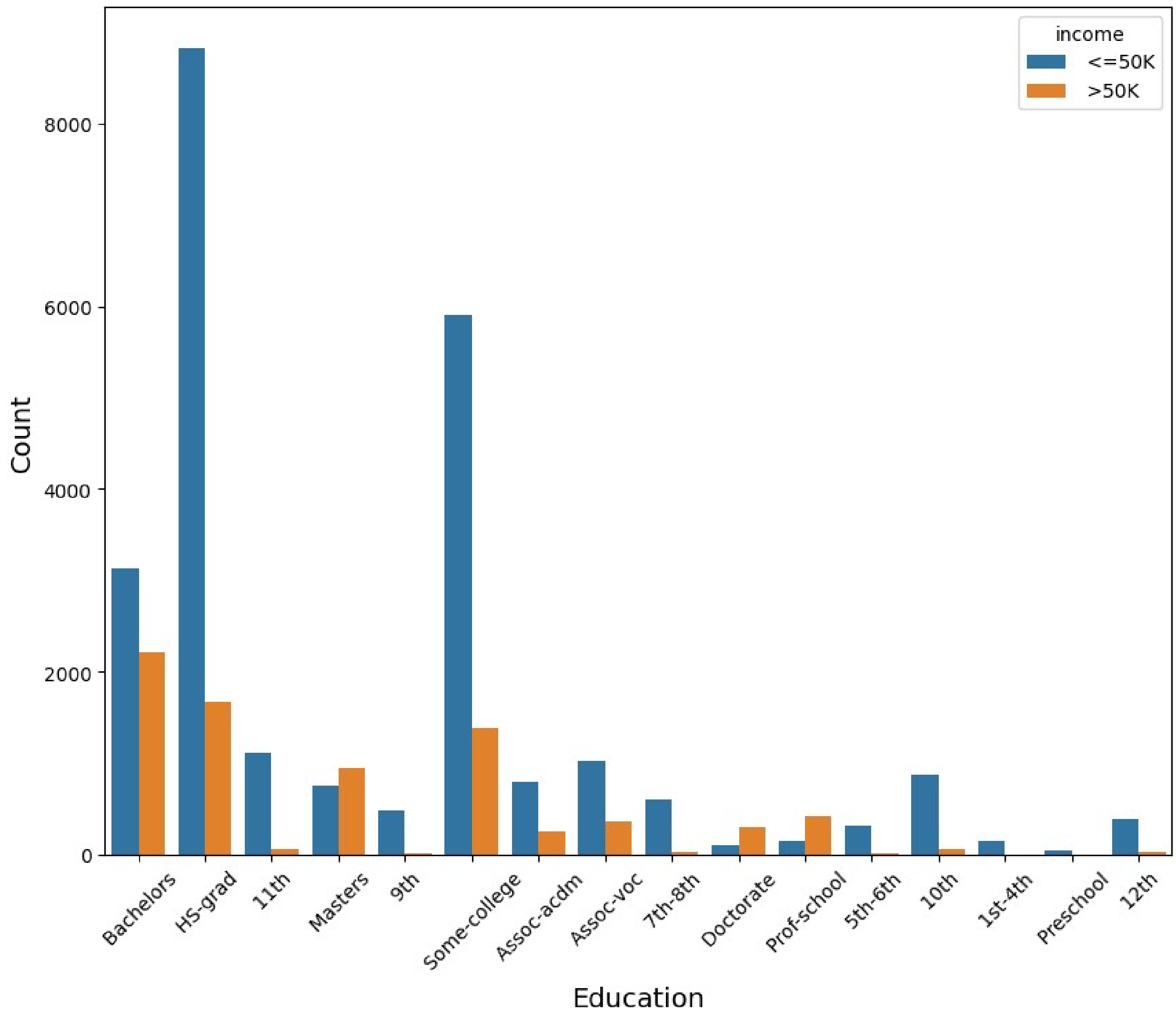
```
f,ax=plt.subplots(1,2,figsize=(12,5.5))
print(ax.shape)
ax[0].hist(df['education-num'], edgecolor="black")
ax[0].set_xlabel('Number of years studied', fontsize = 14)
ax[0].set_ylabel('Count', fontsize = 14)
sns.boxplot(x='income', y='education-num', data=df, ax=ax[1])
ax[1].set_xlabel('Number of years studied', fontsize = 14)
ax[1].set_ylabel('Count', fontsize = 14)
plt.show()
```

```
(2,)
```



education vs income

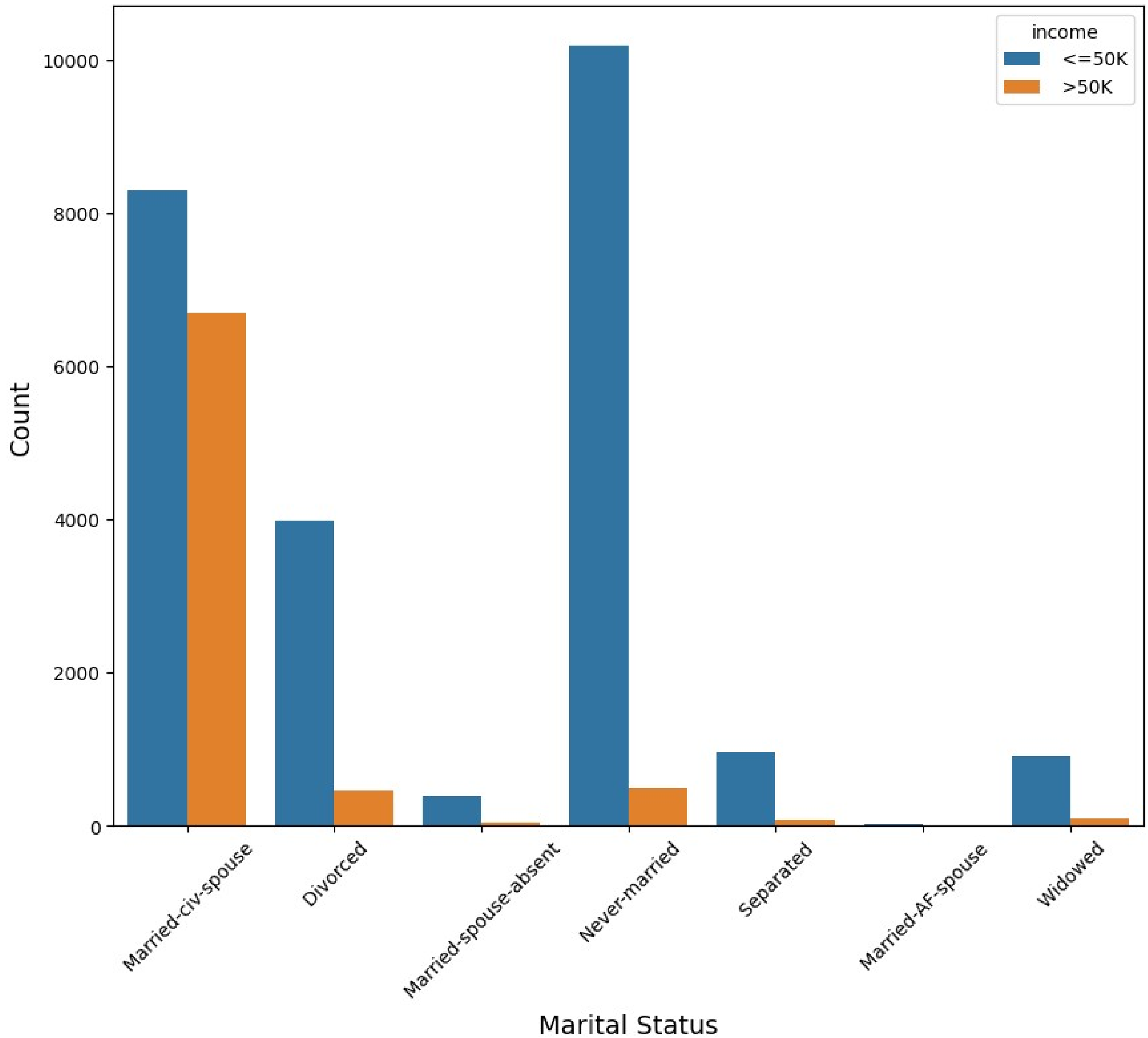
```
f,ax=plt.subplots(1,1,figsize=(10,8))
sns.countplot(x='education',data=df,hue='income', ax = ax)
plt.xticks(rotation=45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Education', fontsize = 14)

Text(0.5, 0, 'Education')
```



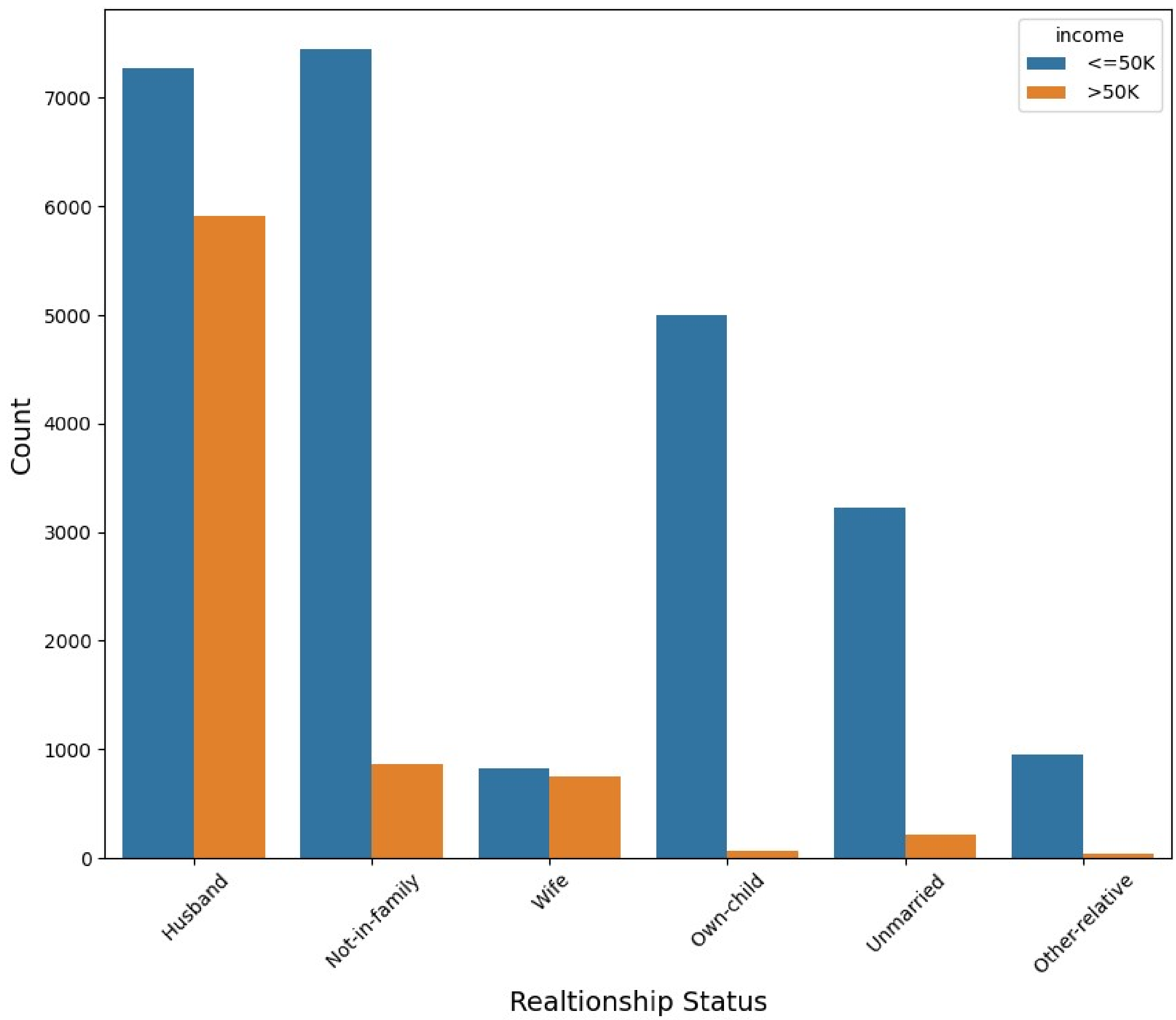
marital status vs income

```
f,ax=plt.subplots(1,1,figsize=(10,8))
sns.countplot(x='marital-status',data=df,hue='income', ax = ax)
plt.xticks(rotation=45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Marital Status', fontsize = 14)
```

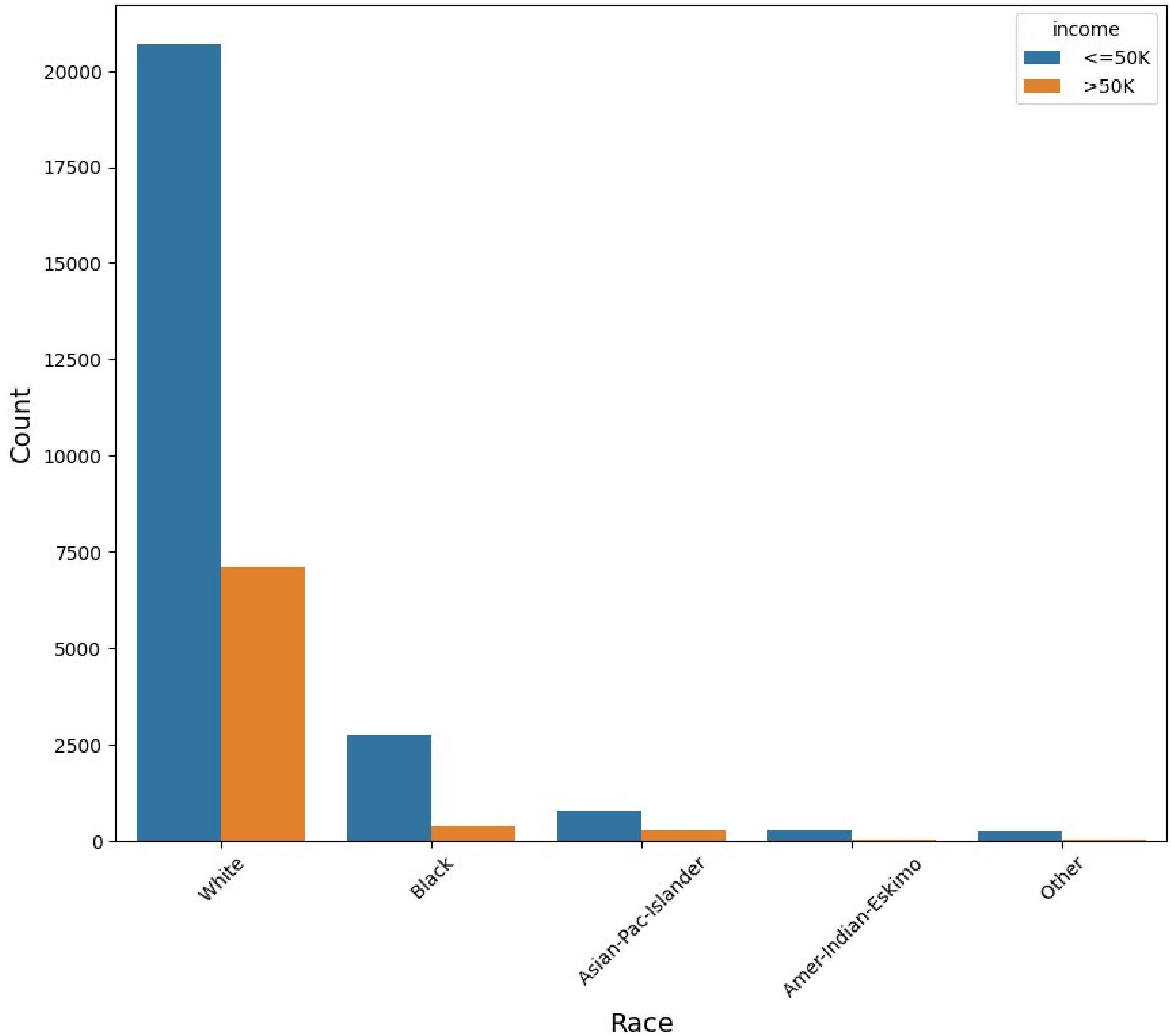
```
Text(0.5, 0, 'Marital Status')
```




```
f,ax=plt.subplots(1,1,figsize=(10,8))
sns.countplot(x='relationship',data=df,hue='income', ax = ax)
plt.xticks(rotation=45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Realtionship Status', fontsize = 14)

Text(0.5, 0, 'Realtionship Status')
```

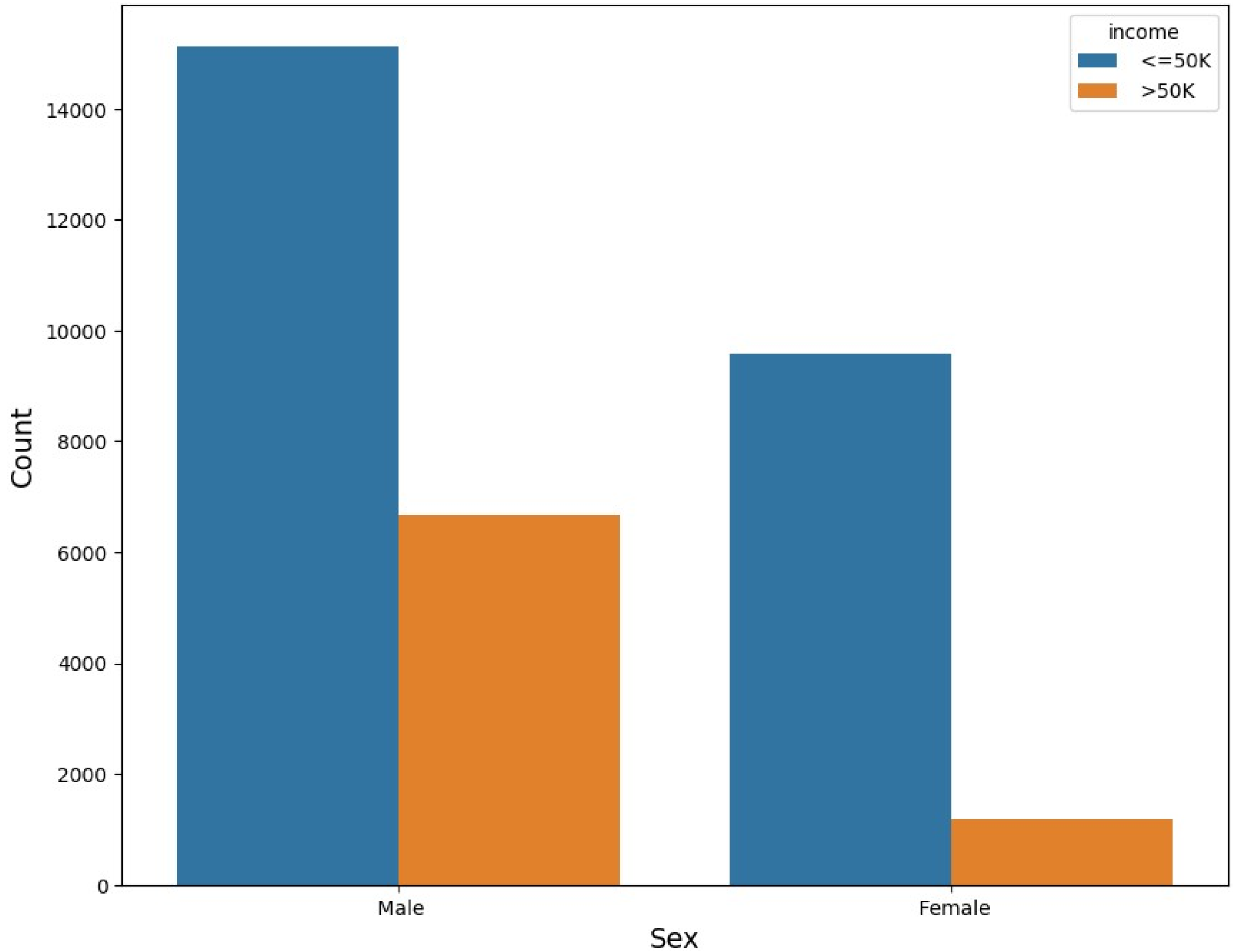



```
f,ax=plt.subplots(1,1,figsize=(10,8))
sns.countplot(x='race',data=df,hue='income', ax = ax)
plt.xticks(rotation=45)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Race', fontsize = 14)

Text(0.5, 0, 'Race')
```




```
f,ax=plt.subplots(1,1,figsize=(10,8))
sns.countplot(x='sex',data=df,hue='income', ax = ax)
plt.ylabel('Count', fontsize = 14)
plt.xlabel('Sex', fontsize = 14)
```

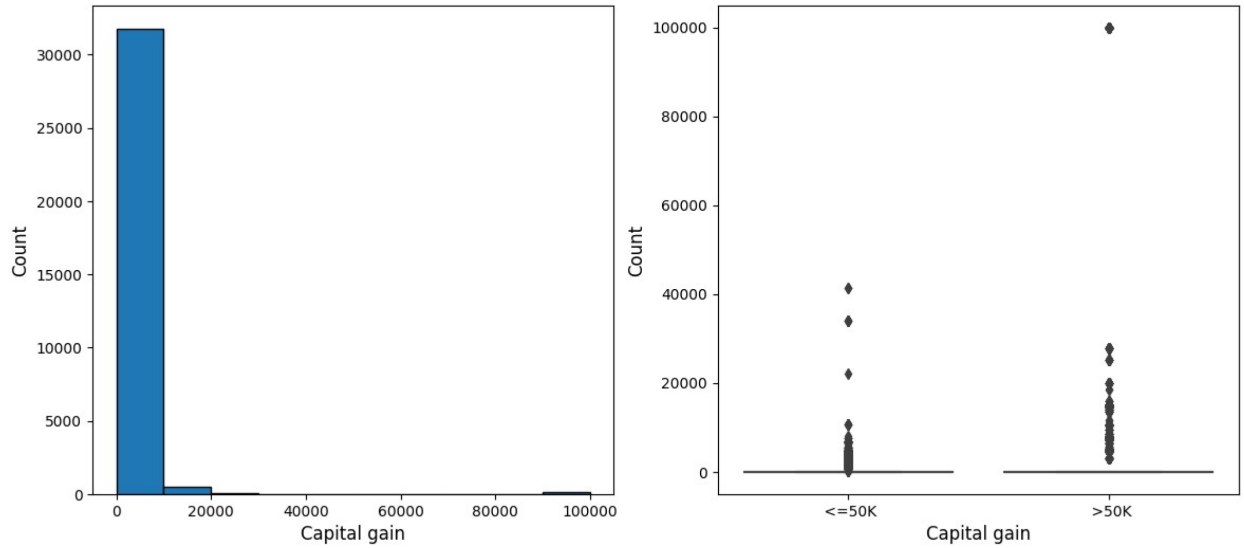
```
Text(0.5, 0, 'Sex')
```



Capital gain vs income

```
f,ax=plt.subplots(1,2,figsize=(14,6))
print(ax.shape)
ax[0].hist(df['capital-gain'], edgecolor="black")
ax[0].set_xlabel('Capital gain', fontsize = 12)
ax[0].set_ylabel('Count', fontsize = 12)
sns.boxplot(x='income', y='capital-gain', data=df, ax=ax[1])
ax[1].set_xlabel('Capital gain', fontsize = 12)
ax[1].set_ylabel('Count', fontsize = 12)
plt.show()
```

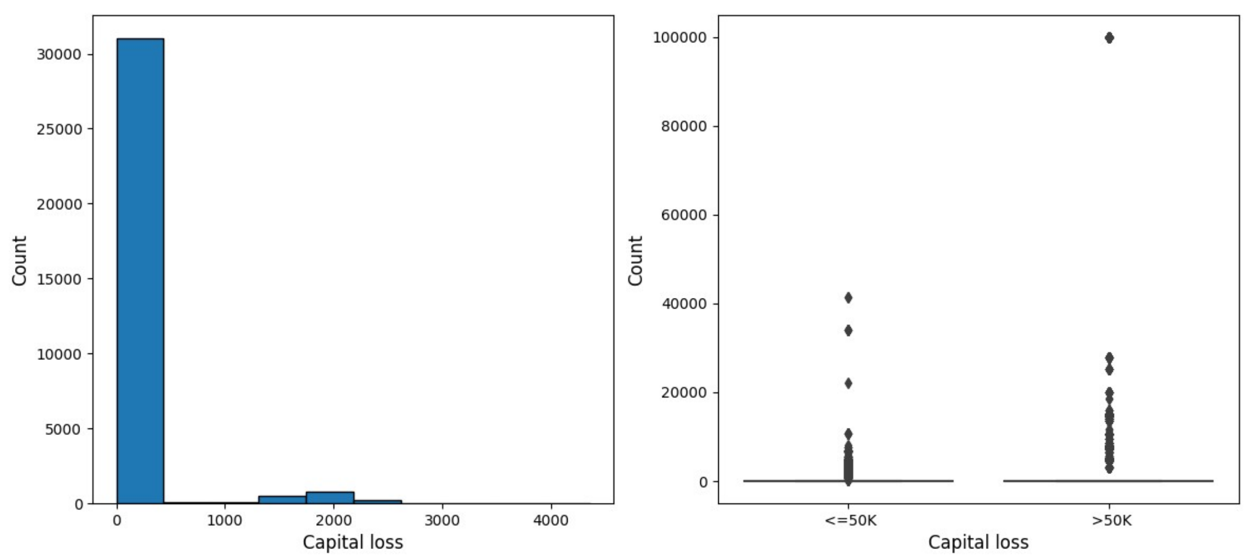
```
(2,)
```



Capital loss vs income

```
f,ax=plt.subplots(1,2,figsize=(14,6))
print(ax.shape)
ax[0].hist(df['capital-loss'], edgecolor="black")
ax[0].set_xlabel('Capital loss', fontsize = 12)
ax[0].set_ylabel('Count', fontsize = 12)
sns.boxplot(x='income', y='capital-gain', data=df, ax=ax[1])
ax[1].set_xlabel('Capital loss', fontsize = 12)
ax[1].set_ylabel('Count', fontsize = 12)
plt.show()
```

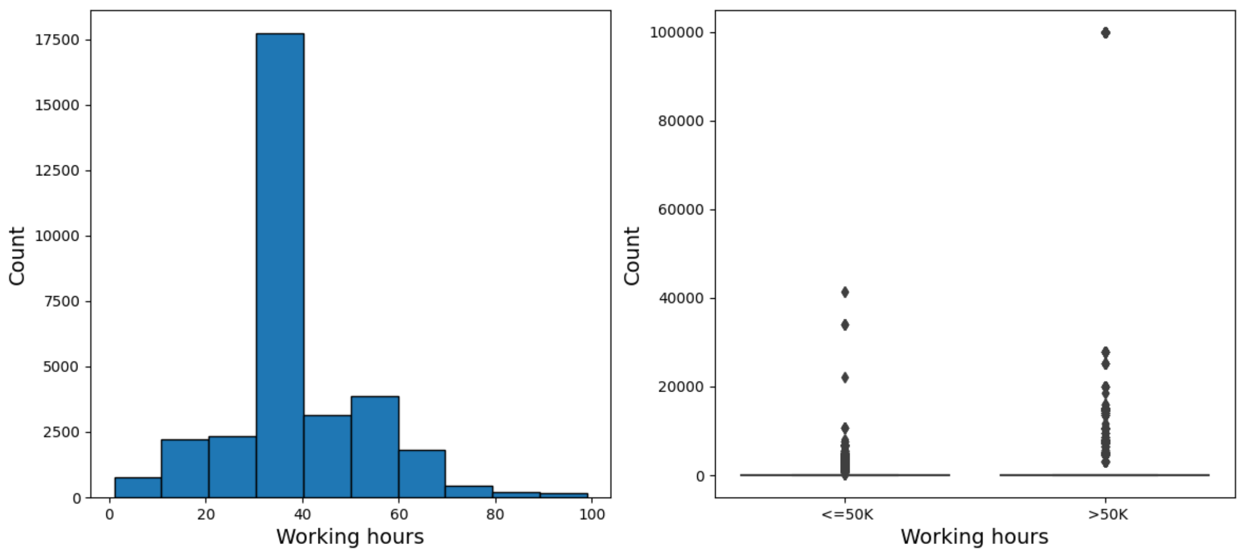
```
(2,)
```



hours per week vs income

```
f,ax=plt.subplots(1,2,figsize=(14,6))
print(ax.shape)
ax[0].hist(df['hours-per-week'], edgecolor="black")
ax[0].set_xlabel('Working hours', fontsize = 14)
ax[0].set_ylabel('Count', fontsize = 14)
sns.boxplot(x='income', y='capital-gain', data=df, ax=ax[1])
ax[1].set_xlabel('Working hours', fontsize = 14)
ax[1].set_ylabel('Count', fontsize = 14)
plt.show()
```

```
(2,)
```



Feature Generation

Created a new variable grouping according to education numbers ar high, med or low.

```
def bin_var(data, var, bins, group_names):
    bin_value = bins
    group = group_names
    data[var+'Cat'] = pd.cut(df[var], bin_value, labels=group)

bin_var(df, 'education-num', [0,6,11,16], ['Low', 'Medium', 'High'])

bin_var(df, 'hours-per-week', [0,35,40,60,100], ['Low', 'Medium',
'High','VeryHigh'])
```

Classifying the occupation into Highly Skilled and low Skilled

```
occu=pd.crosstab(df['Occupation'],df['income'],
margins=True).reset_index()
occu
```

income	Occupation	<=50K	>50K	All
0	Adm-clerical	3448	533	3981
1	Armed-Forces	9	1	10
2	Craft-repair	3412	952	4364
3	Exec-managerial	2323	1994	4317
4	Farming-fishing	915	123	1038
5	Handlers-cleaners	1373	98	1471
6	Machine-op-inspct	1871	263	2134
7	Other-service	3311	159	3470
8	Priv-house-serv	158	1	159
9	Prof-specialty	2502	1884	4386
10	Protective-serv	470	213	683
11	Sales	2861	1002	3863
12	Tech-support	691	290	981
13	Transport-moving	1375	328	1703
14	All	24719	7841	32560

```
#creating a function to categorize skill
import re
def occup(x):
    if re.search('managerial', x):
        return 'Highskill'
    elif re.search('specialty',x):
        return 'Highskill'
    else:
        return 'Lowskill'

# Creating the occupation category feature
df['Occupa_cat']=df.Occupation.apply(lambda x: x.strip()).apply(lambda
x: occup(x))
df['Occupa_cat'].value_counts()
```

```
Lowskill     23857
Highskill     8703
Name: Occupa_cat, dtype: int64
```

```
pd.crosstab(df['race'],df['income'], margins=True)
```

income	<=50K	>50K	All
race			
Amer-Indian-Eskimo	275	36	311
Asian-Pac-Islander	763	276	1039
Black	2737	387	3124
Other	246	25	271
White	20698	7117	27815
All	24719	7841	32560

```
# Creating race category feature
df['Race_cat']=df['race'].apply(lambda x: x.strip())
df['Race_cat']=df['Race_cat'].apply(lambda x: 'White' if x=='White'
else 'Other')
```

Encoding the categorical variables

```
# find categorical variables

categorical = [var for var in df.columns if df[var].dtype=='O']
print(categorical)
```

```
['workclass', 'education', 'marital-status', 'Occupation',
'relationship', 'race', 'sex', 'native-country', 'income',
'Occupa_cat', 'Race_cat']
```

```
# Find numerical variables

numerical = [var for var in df.columns if df[var].dtype !='O']
print(numerical)
```

```
['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss',
'hours-per-week', 'education-numCat', 'hours-per-weekCat']
```

```
categorical
```

```
['workclass',
 'education',
 'marital-status',
 'Occupation',
 'relationship',
 'race',
 'sex',
 'native-country',
 'income',
 'Occupa_cat',
 'Race_cat']
```

```
# import category encoders
# encode remaining variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['workclass',
 'education',
 'marital-status',
 'Occupation',
 'relationship',
 'race',
 'sex',
 'native-country',
 'Occupa_cat',
 'Race_cat', 'education-numCat', "hours-per-weekCat"])
```

```
df = encoder.fit_transform(df)
```

Feature Selection Using Variance Threshold

Variance Threshold is a univariate approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples. As an example, suppose that we have a dataset with boolean features, and we want to remove all features that are either one or zero (on or off) in more than 80% of the samples. Boolean features are Bernoulli random variables, and the variance of such variables is given by The below approach removes variable which have more than 80% values are either 0 or 1.

```
for items in list(df.columns):
  print(items)
```

```
age
workclass_1
workclass_2
workclass_3
workclass_4
workclass_5
workclass_6
workclass_7
workclass_8
fnlwgt
education_1
education_2
education_3
education_4
education_5
education_6
education_7
education_8
education_9
education_10
education_11
education_12
education_13
education_14
education_15
education_16
education-num
marital-status_1
marital-status_2
marital-status_3
marital-status_4
marital-status_5
marital-status_6
```

```
marital-status_7
Occupation_1
Occupation_2
Occupation_3
Occupation_4
Occupation_5
Occupation_6
Occupation_7
Occupation_8
Occupation_9
Occupation_10
Occupation_11
Occupation_12
Occupation_13
Occupation_14
relationship_1
relationship_2
relationship_3
relationship_4
relationship_5
relationship_6
race_1
race_2
race_3
race_4
race_5
sex_1
sex_2
capital-gain
capital-loss
hours-per-week
native-country_1
native-country_2
native-country_3
native-country_4
native-country_5
native-country_6
native-country_7
native-country_8
native-country_9
native-country_10
native-country_11
native-country_12
native-country_13
native-country_14
native-country_15
native-country_16
native-country_17
native-country_18
```

```
native-country_19
native-country_20
native-country_21
native-country_22
native-country_23
native-country_24
native-country_25
native-country_26
native-country_27
native-country_28
native-country_29
native-country_30
native-country_31
native-country_32
native-country_33
native-country_34
native-country_35
native-country_36
native-country_37
native-country_38
native-country_39
native-country_40
native-country_41
income
education-numCat_1
education-numCat_2
education-numCat_3
hours-per-weekCat_1
hours-per-weekCat_2
hours-per-weekCat_3
hours-per-weekCat_4
Occupa_cat_1
Occupa_cat_2
Race_cat_1
Race_cat_2
```

```
def variance_threshold_select(df, thresh=0.0, na_replacement=-999):
    df1 = df.copy(deep=True) # Make a deep copy of the dataframe
    selector = VarianceThreshold(thresh) # passing Threshold
    selector.fit(df1) # Fill NA values as VarianceThreshold cannot
deal with those
    df2 = df.loc[:,selector.get_support(indices=False)] # Get new
dataframe with columns deleted that have NA values
    return df2

# Setting a 80 percent threshold
df2=variance_threshold_select(df.drop('income', axis=1), thresh=.8* (1
- .8))
```

```
for items in list(df2.columns):
  print(items)
```

```
age
workclass_2
fnlwgt
education_2
education_6
education-num
marital-status_1
marital-status_4
relationship_1
relationship_2
sex_1
sex_2
capital-gain
capital-loss
hours-per-week
education-numCat_2
education-numCat_3
hours-per-weekCat_1
hours-per-weekCat_2
hours-per-weekCat_3
Occupa_cat_1
Occupa_cat_2
```

Modelling the Naive Bayes Classifier

```
model = GaussianNB()

# Splitting data into test train, using a 0.3 split
X = df2
y = df['income']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# check the shape of X_train and X_test
X_train.shape, X_test.shape
```

```
((22792, 22), (9768, 22))
```

```
# Scaling the training and test feature values

scaler = RobustScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Using Random Search method to find the best hyperparameters
from sklearn.model_selection import RandomizedSearchCV
gnb = GaussianNB()
```

```
param_grid = {
    'var_smoothing': np.logspace(0,-9, num=100)
}
CV_rfc = RandomizedSearchCV(estimator=gnb,
param_distributions=param_grid, cv= 5,random_state=1)
CV_rfc.fit(X_train, y_train)
print(CV_rfc.best_params_)
```

```
{'var_smoothing': 3.5111917342151273e-09}
```

```
#Defining the model with tuned hyperparameters
model = GaussianNB(var_smoothing=CV_rfc.best_params_['var_smoothing'])
# Fitting the model on to the train set
model.fit(X_train,y_train)
```

```
GaussianNB(var_smoothing=3.5111917342151273e-09)
```

```
predict_train = model.predict(X_train)

# Accuracy Score on train dataset
accuracy_train = accuracy_score(y_train,predict_train)
print('accuracy_score on train dataset : ', accuracy_train)

predict_test = model.predict(X_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(y_test,predict_test)
print('accuracy_score on test dataset : ', accuracy_test)
```

```
accuracy_score on train dataset :  0.8326605826605826
accuracy_score on test dataset :  0.8311834561834562
```

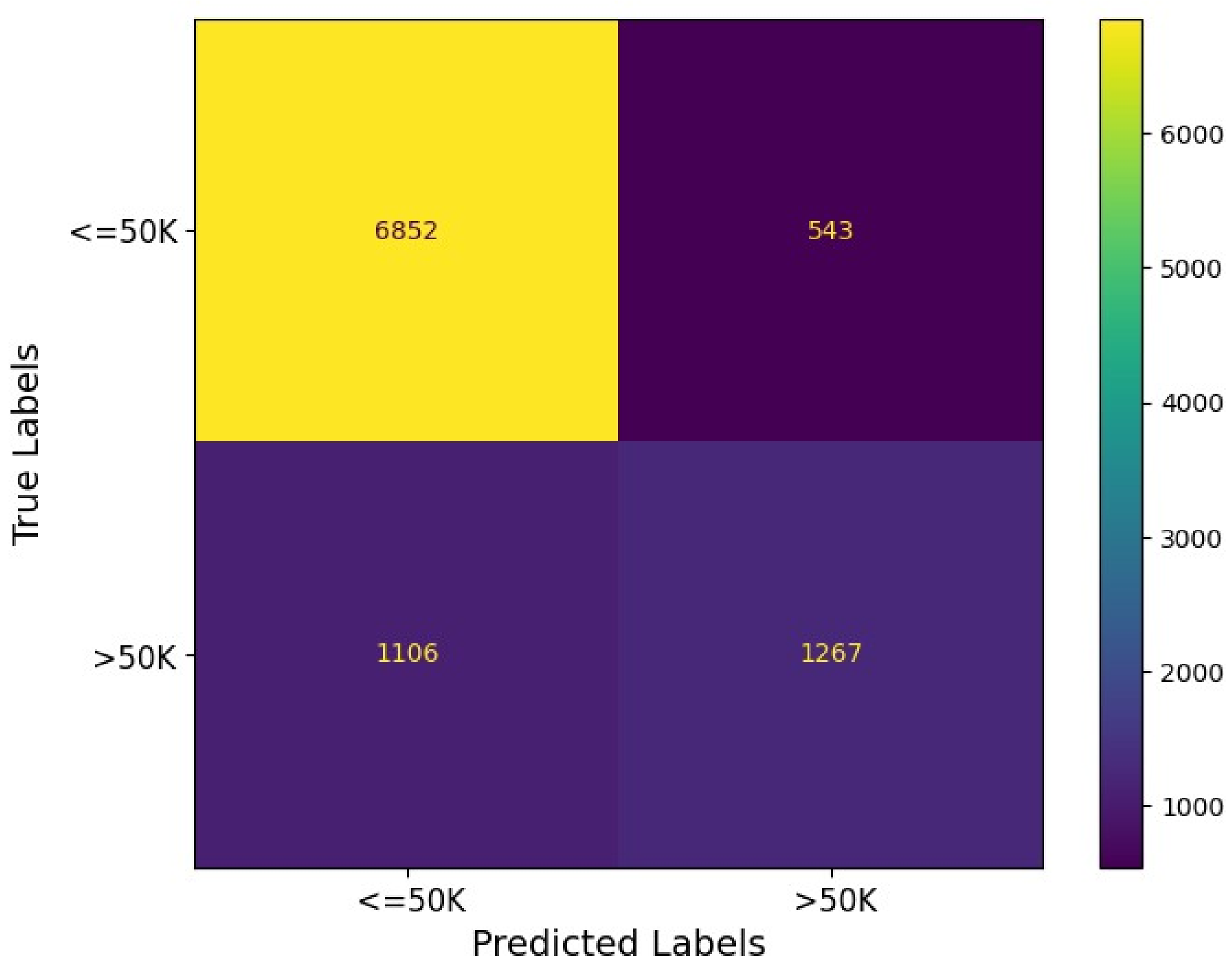
We see that the accuracy values for test and train are close, and thus no overfitting!

Confusion Matrix

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

y_pred = model.predict(X_test)

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Create the ConfusionMatrixDisplay
cmd = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=model.classes_)
```

```python
fig, ax = plt.subplots(figsize=(8, 6))  # Adjust the figure size as needed
cmd.plot(ax=ax)

ax.set_xlabel('Predicted Labels', fontsize=14)
ax.set_ylabel('True Labels', fontsize=14)
ax.tick_params(axis='both', which='both', labelsize=12)

plt.show()
```

```
[[6852  543]
 [1106 1267]]
```



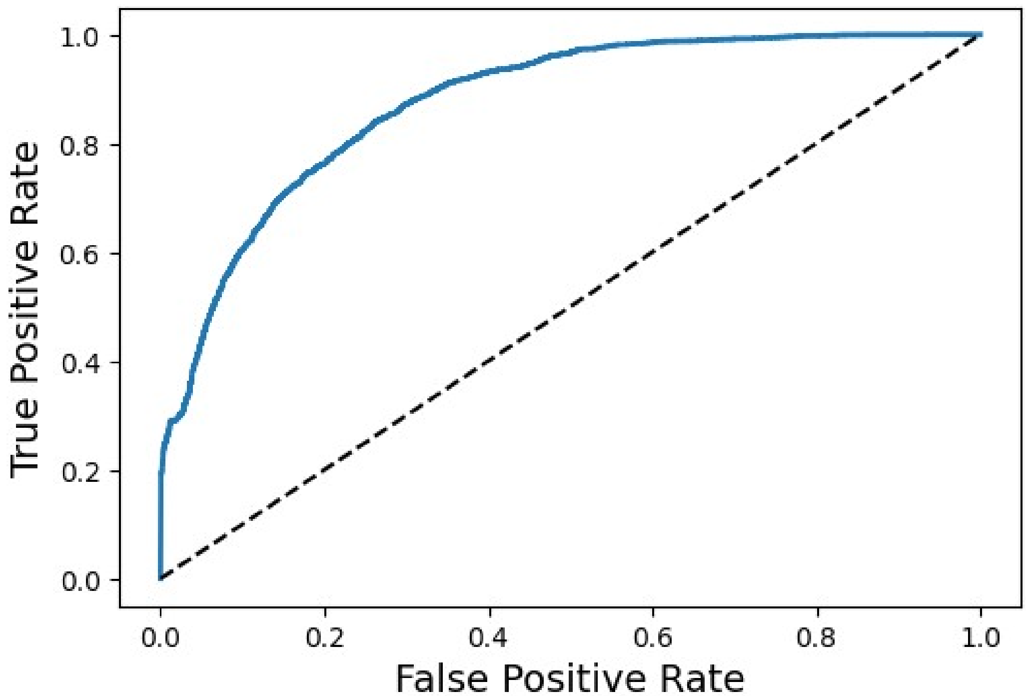

```python
print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
True Positives(TP) =  6852

True Negatives(TN) =  1267

False Positives(FP) =  543

False Negatives(FN) =  1106
```

```
#Prinitng the classification report using the sklearn metrics library
from sklearn.metrics import classification_report

print(classification_report(y_test, predict_test))
```

```
              precision    recall  f1-score   support

       <=50K       0.86      0.93      0.89      7395
        >50K       0.70      0.53      0.61      2373

    accuracy                           0.83      9768
   macro avg       0.78      0.73      0.75      9768
weighted avg       0.82      0.83      0.82      9768
```

```
#Getting predicted probabilities
pred_proba = model.predict_proba(X_test)[:, 1]
# plot ROC Curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, pred_proba, pos_label = '
>50K')

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.xlabel('False Positive Rate', fontsize = 14)

plt.ylabel('True Positive Rate', fontsize = 14)

plt.show()
```




```
# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, pred_proba)

print('ROC AUC : {:.4f}'.format(ROC_AUC))

ROC AUC : 0.8756
```

As our ROC AUC value is close to 1, we can say that our classifer model is working well !

k-Fold Cross Validation

```
# Applying 10-Fold Cross Validation

from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X_train, y_train, cv = 10,
scoring='accuracy')

print('Cross-validation scores:{}'.format(scores))

print('Cross-validation mean accuracy:{}'.format(scores.mean()))

Cross-validation scores:[0.83640351 0.82719298 0.83808688 0.825362
0.8372093  0.8332602
```

```
 0.81878017 0.83984204 0.825362   0.84379114]
Cross-validation mean accuracy:0.8325290216546193
```

We see that the mean accuracy is close to the original one, and also there is not much deviation from the average for all the folds, thus we can say our model is not much reliant on the data on which it is being trained.

Correlation Check

```
# def cat_to_num(col_data, col_name, class_lis ):
#   col_data[col_name] = col_data[col_name].apply(lambda x: class_lis.index(x) + 1)

# for cols in list(df.columns):
#   if str(df[cols].dtypes) == 'object':
#     cat_to_num(df, cols, list(df[cols].unique()))

# df.info()

# plt.figure(figsize=(16, 8))
# corr_matrix = df.corr()

# # Set the desired number of decimal places for the annotations
# decimal_places = 2  # Change this to your desired number of decimal places

# # Format the annotations with the desired number of decimal places
# ax = sns.heatmap(corr_matrix, annot=True, xticklabels=True, yticklabels=True,
#              annot_kws={"size": 10}, fmt=f'.{decimal_places}f')
# ax.tick_params(axis='both', which='both', labelsize = 14)

# plt.show()
```