AMS 580 – Statistical Learning

Group Project: **Income Prediction using Machine Learning**

Group Number: **07**

Stony Brook University

Group Members

Aishwarya Bhanage aishwaryamahad.bhanage@stonybrook.edu	116556145
Rutika Kadam rutikaavinash.kadam@stonybrook.edu	116753960
Sakshi Shah sakshijanak.shah@stonybrook.edu	116727594
Sanjyot Amritkar sanjyotsatish.amritkar@stonybrook.edu	116483478
Tamali Halder tamali.halder@stonybrook.edu	116713494

Submitted to: **Prof. Weihao Wang**

Date: **April 25, 2025**

Contents

Project Overview

In this project, we aim to develop predictive machine learning models that can classify individuals into two income groups based on their demographic and employment characteristics, earning less than $50,000 or more than $50,000 annually.

The dataset used is the **Adult Income Dataset** derived from the 1994 U.S. Census Bureau database, later cleaned by Ronny Kohavi and Barry Becker for data mining research. It is publicly available through the UCI Machine Learning Repository.

Our key objective was to:

- Apply a variety of machine learning models to predict income category.
- Perform rigorous data preprocessing.
- Evaluate and compare models using standardized classification metrics.
- Select the best-performing model based on predictive accuracy and recall scores, with particular emphasis on identifying high-income individuals correctly.

Understanding income classification is not just a predictive task but has real-world significance, informing fields such as marketing, credit scoring, and public policy decision-making.

Dataset Explanation:

The dataset is split into two parts: 'train.csv' and 'test.csv'. The response variable is 'income', which has two categories: **'<=50K'** and **'>50K'**. The features can be broadly categorized into numerical andcategorical types:

Numerical features include:

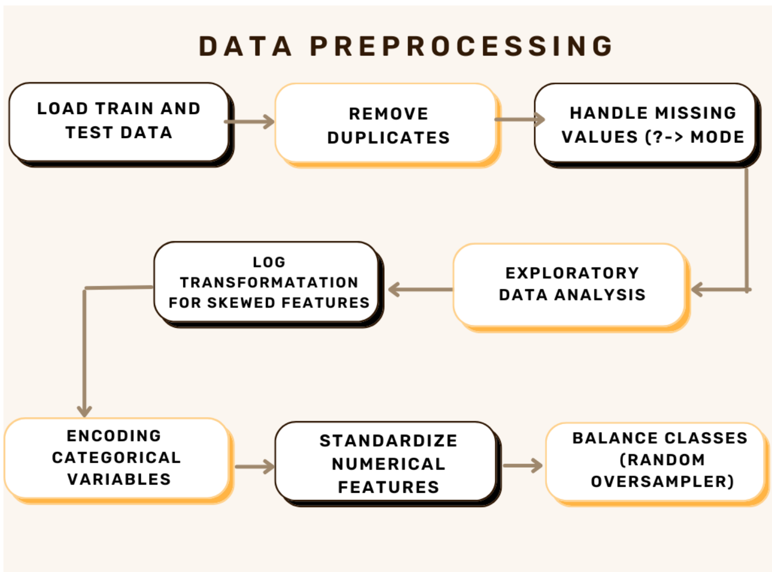
age, fnlwgt (final weight), education-num, capital-gain, capital-loss, hours-per-week.

Categorical features include:

workclass, education, marital-status, occupation, relationship, race, sex, native-country.

Data Preprocessing

Real-world data is rarely ready for machine learning models. Hence, extensive preprocessing was crucial to ensure data quality and model efficiency.



2.1 Data Cleaning

- **Missing Values Handling:**
  - "?" symbols representing missing data were identified.
  - Records with missing values in critical fields were dropped after careful assessment to avoid introducing bias.
- **Outlier Detection:**
  - Basic EDA plots (boxplots, histograms) revealed that the dataset had minor outliers, but no aggressive outlier removal was necessary.

2.2 Encoding Categorical Variables

  - Categorical variables (e.g., marital status, workclass, occupation) were Label Encoded using LabelEncoder from scikit-learn.

- Although one-hot encoding was an option, label encoding was preferred considering tree-based algorithms like Random Forest and XGBoost can naturally handle ordinal integer values.

2.3 Feature Scaling

- **Standardization**:
  - Continuous features like "age," "hours-per-week," and "education-num" were scaled using **StandardScaler** to have zero mean and unit variance.
  - This step was vital for distance-based models like K-Nearest Neighbors (KNN) and algorithms sensitive to feature magnitude like Support Vector Machine (SVM).

2.4 Handling Class Imbalance

- Upon EDA, we discovered that the <=$50K class was significantly overrepresented compared to the >$50K class.
- To address this imbalance:
  - We used **RandomOverSampler** from the *imbalanced-learn* package.
  - Synthetic oversampling ensured that both classes were nearly balanced without loss of information from undersampling.

2.5 Feature Selection

- **Correlation Analysis**:
  - Pearson correlation was computed for continuous variables.
  - Weakly correlated or redundant variables were considered for removal.
- **Feature Importance**:
  - Using Random Forest's feature importance scores, we identified the top contributing features (e.g., "education-num," "hours-per-week," "occupation," "relationship").

Exploratory Data Analysis

Exploratory Data Analysis (EDA) serves as the **first deep dive into the dataset**. It allows us to uncover underlying patterns, detect anomalies, test hypotheses, and check assumptions with the help of summary statistics and graphical representations.

Shape of the dataset: (32,561 rows, 15 columns)

This indicates that our dataset consists of 32,561 observations and 15 attributes

Columns:

['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income']

The dataset includes the following variables:

Feature Name	Description
Age	Age of the individual
workclass	Type of employer
fnlwgt	Final sampling weight
education	Highest level of education attained
education-num	Education level encoded as a numeric variable
marital-status	Marital status
occupation	Type of occupation
relationship	Family relationship
race	Race of the individual
sex	Gender of the individual
capital-gain	Income from investment sources other than wages/salary
capital-loss	Losses from investment sources
hours-per-week	Average working hours per week
native-country	Country of origin
income	Target variable (<=50K or >50K)

Feature Types

Upon inspecting the dataset, we categorized features into numerical and categorical based on their data types:

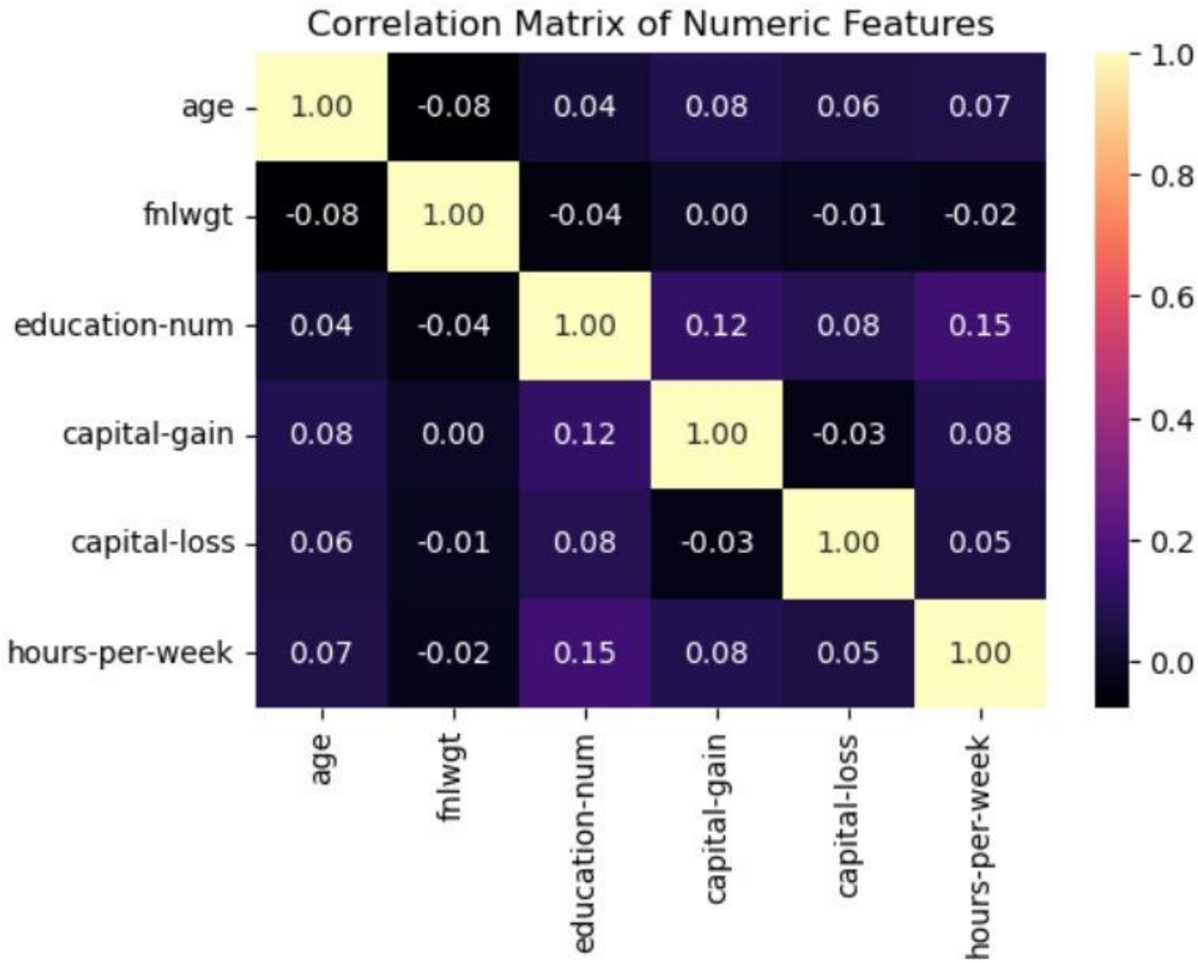
Feature	Data Type
age	int64
workclass	object
fnlwgt	int64
education	object
education-num	int64
marital-status	object
occupation	object
relationship	object
race	object
sex	object
capital-gain	int64
capital-loss	int64
hours-per-week	int64
native-country	object
income	object

Numeric columns (6): ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

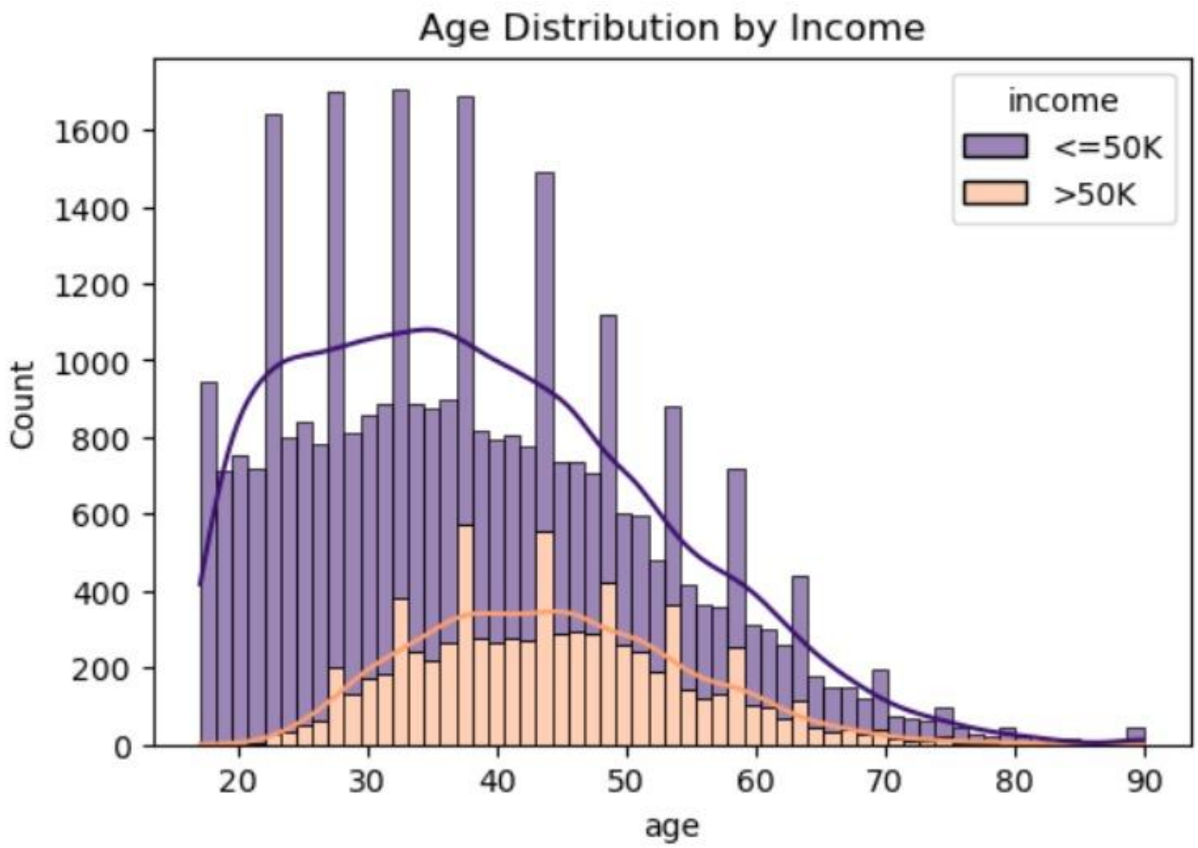
Categorical columns (9): ['workclass', 'education', 'marital-status', 'occupation', **'relationship', 'race', 'sex', 'native-country', 'income']**

Trends, correlations, and interesting relationships.

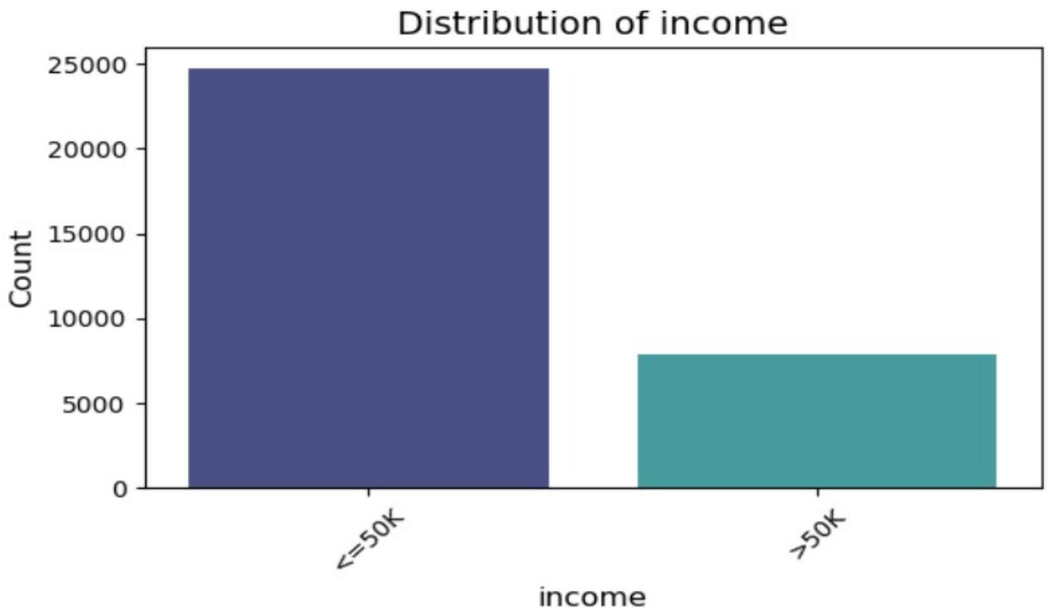
Correlation Matrix (Numeric variables only): Most numeric features in the dataset exhibit low correlation values, suggesting that the variables are largely independent of each other. This is good for modeling, as multicollinearity is less of a concern.



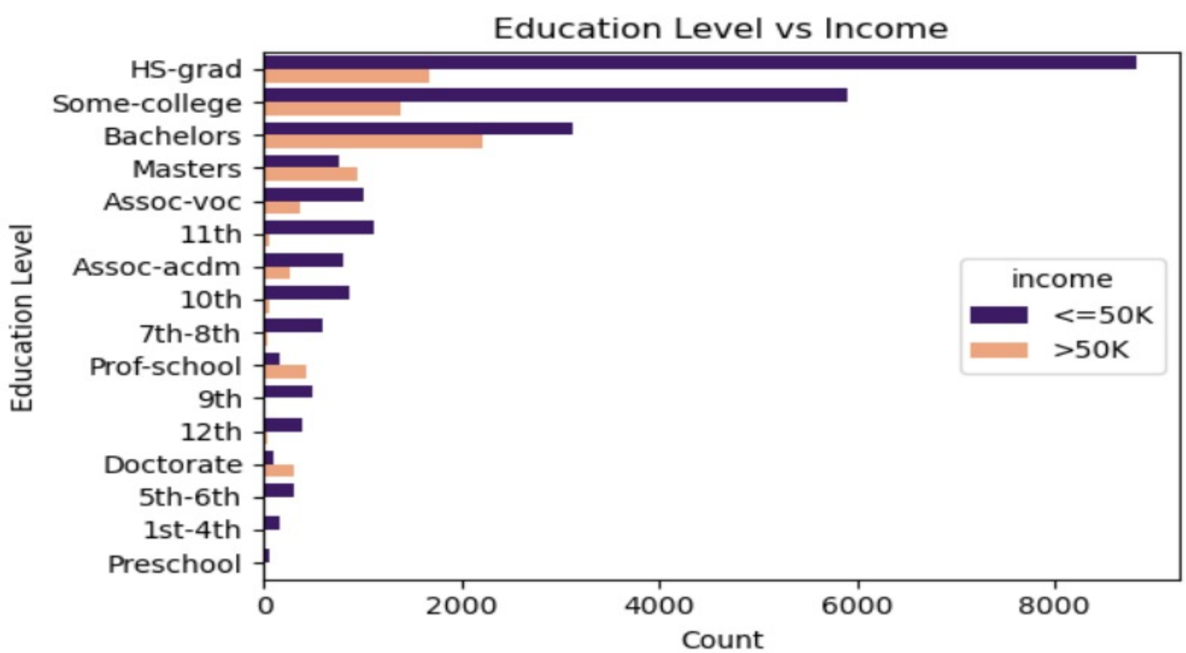
Age distribution by Income: The age distribution shows that individuals earning over $50K are mostly between 30 and 60, peaking around their 40s. Those earning $50K or less are more common across all ages, especially under 30. Income levels drop for both groups after age 60, likely due to retirement. The 40–50 range appears to be a key transition period toward higher income.

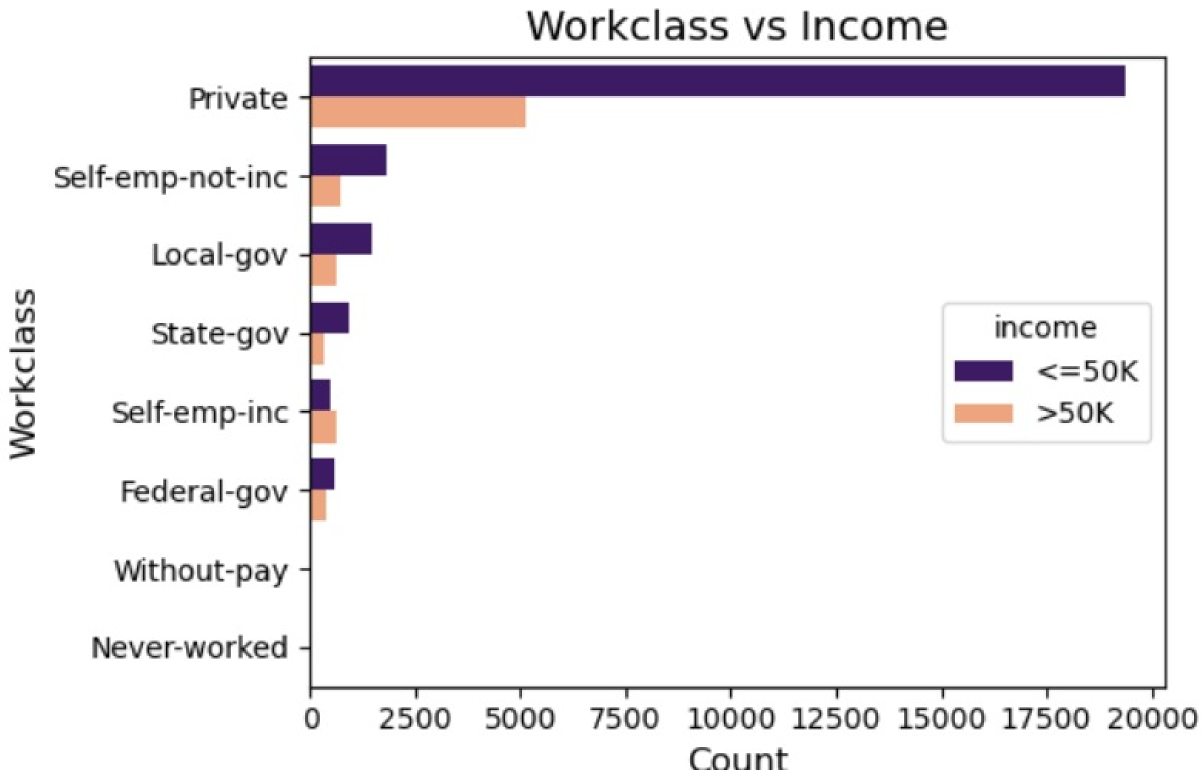


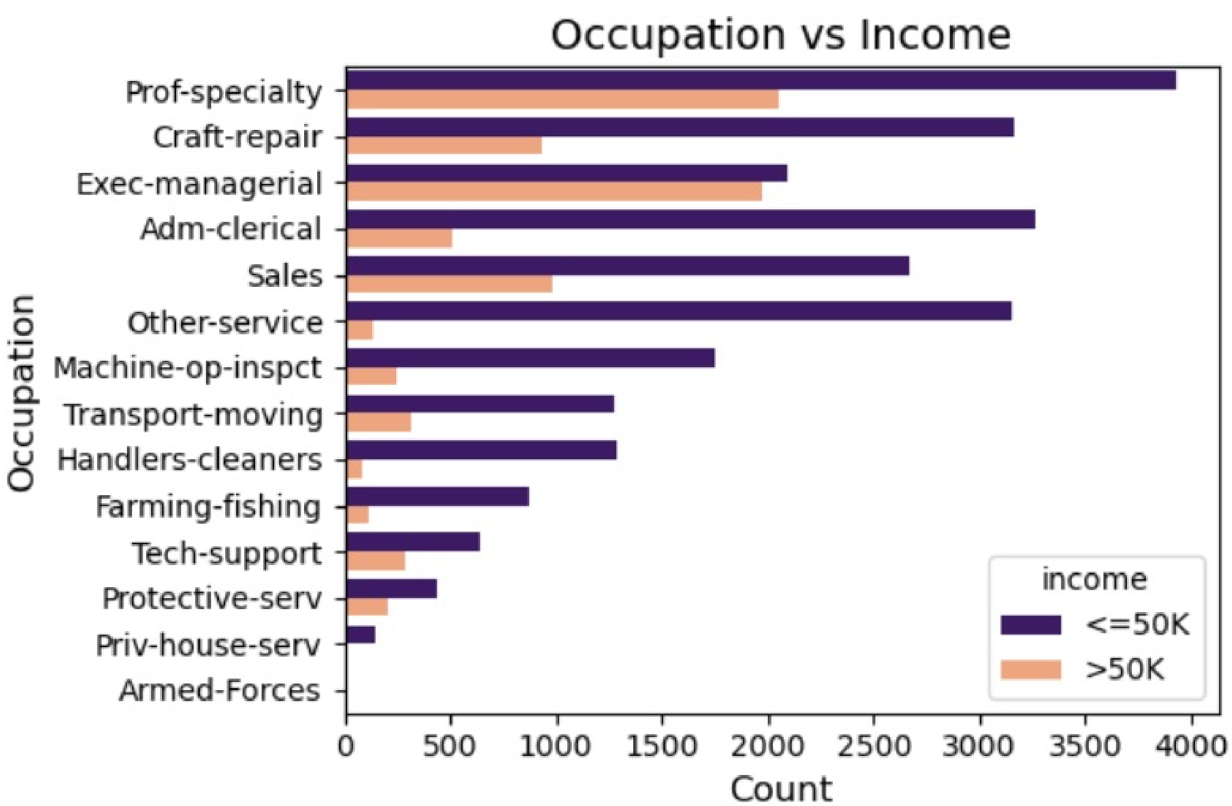
Income count distribution: The bar chart illustrates the distribution of income levels in the dataset, divided into two categories: individuals earning less than or equal to $50K and those earning more than $50K. It clearly shows that a significantly larger proportion of people fall into the lower income group (<=50K), with more than twice as many individuals compared to the higher income group (>50K). This indicates that the majority of the population represented in this data earns a modest income, highlighting income inequality or the broader economic distribution within the sample.

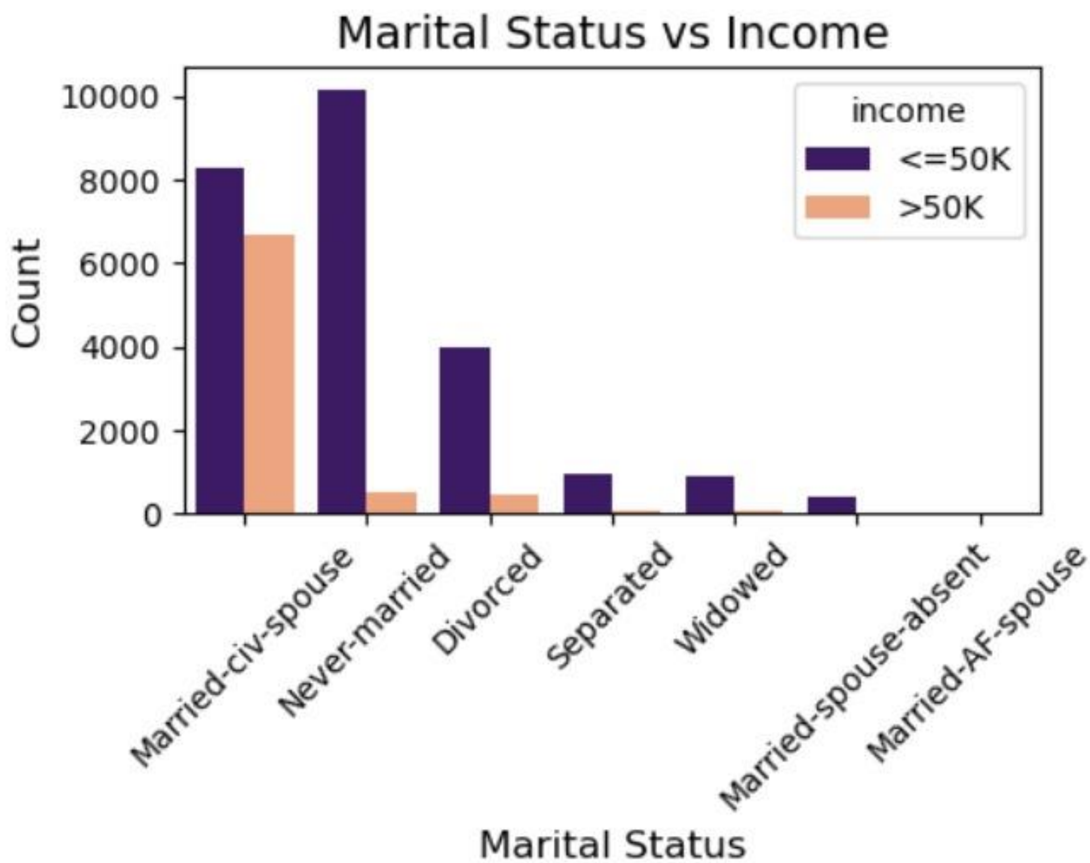


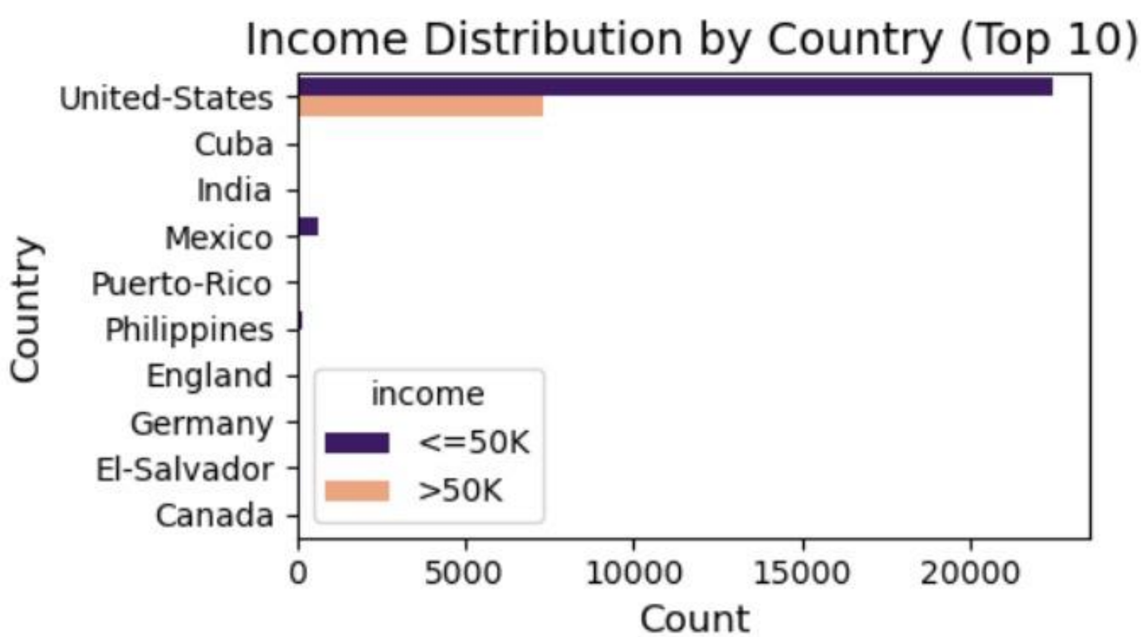
Education, workclass, occupation and marital status based on Income: Higher income levels are strongly associated with higher education (especially Bachelors and Masters), professional or managerial occupations, and employment in the private sector or self-employment. Additionally, being married, particularly to a civilian spouse, is linked to higher earnings compared to being single or divorced. These patterns highlight the significant influence of education, job type, work sector, and marital status on income potential.




Workclass vs Income
Private
Self-emp-not-inc
Local-gov
State-gov
Self-emp-inc
Federal-gov
Without-pay
Never-worked
Workclass
income
<=50K
>50K
0
2500
5000
7500
10000
12500
15000
17500
20000
Count

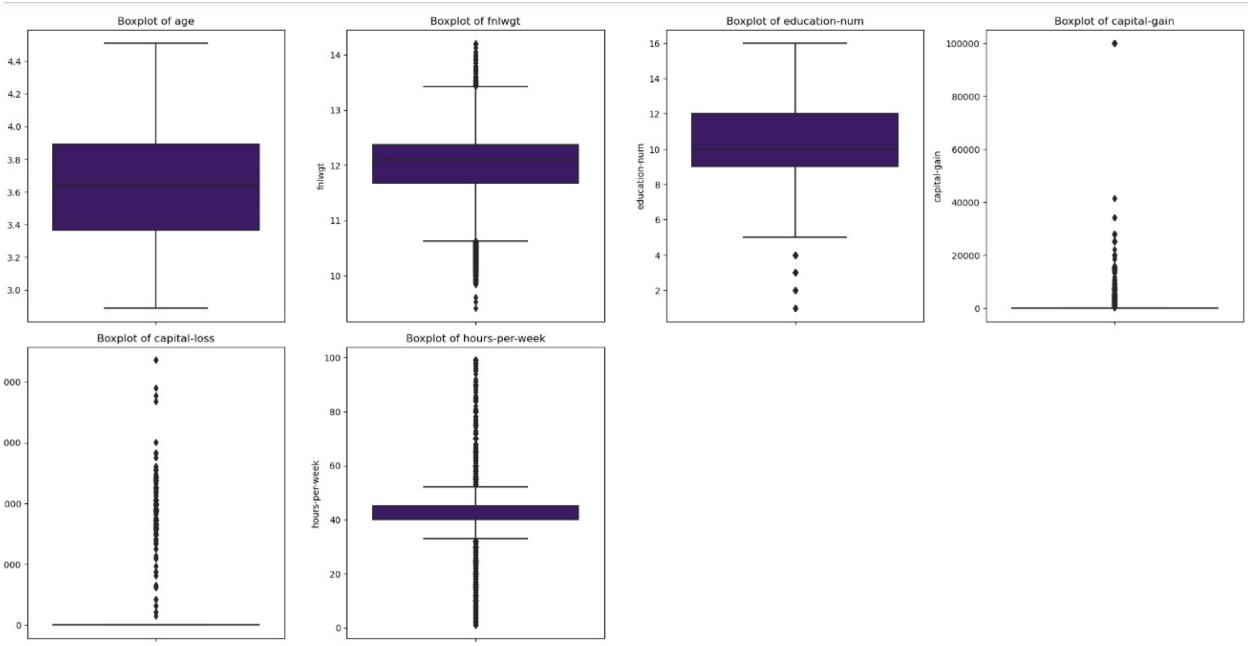

Occupation vs Income
Prof-specialty
Craft-repair
Exec-managerial
Adm-clerical
Sales
Other-service
Machine-op-inspct
Transport-moving
Handlers-cleaners
Farming-fishing
Tech-support
Protective-serv
Priv-house-serv
Armed-Forces
Occupation
income
<=50K
>50K
0
500
1000
1500
2000
2500
3000
3500
4000
Count


Marital Status vs Income
Count
10000
8000
6000
4000
2000
0
income
<=50K
>50K
Married-civ-spouse
Never-married
Divorced
Separated
Widowed
Married-spouse-absent
Married-AF-spouse
Marital Status


Income Distribution by Country (Top 10)
Country
United-States
Cuba
India
Mexico
Puerto-Rico
Philippines
England
Germany
El-Salvador
Canada
income
<=50K
>50K
0
5000
10000
15000
20000
Count

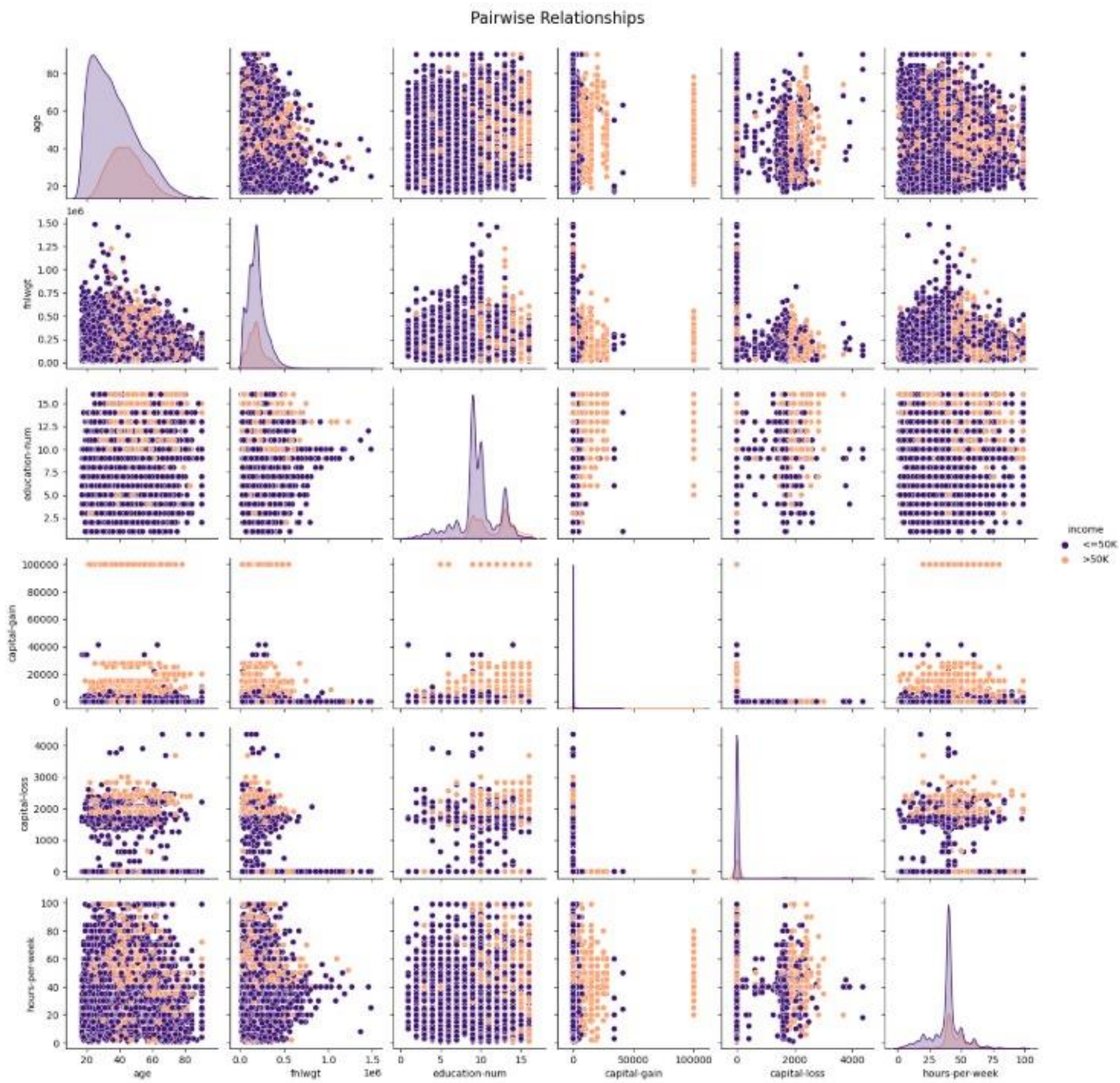
Outliers or unexpected values.

Based on the boxplots of the numeric features, we observe several key patterns and potential outliers. **Age**, **fnlwgt**, **capital-gain**, **capital-loss**, and **hours-per-week** all show significant presence of outliers. Particularly, **capital-gain** and **capital-loss** are highly skewed, with most values being zero and a few extreme high values. **Fnlwgt** also has a long tail, suggesting a few individuals have disproportionately large weights. **Education-num** appears more normally distributed, though with slight skewness. **Hours-per-week** mostly clusters between 35 and 45, indicating standard full-time work, but with notable outliers above 70. These trends suggest some features may need transformation or outlier handling before modeling.



For future modeling steps (e.g., normality, linearity).

The histograms with KDE curves reveal the distributions of numeric variables. **Age** shows a right-skewed distribution, with a peak around 20–40 years. **Fnlwgt** is heavily right-skewed, indicating a majority of observations lie at lower values. **Education-num** appears discrete and multimodal, reflecting specific education levels. **Capital-gain** and **capital-loss** are extremely skewed to the right, with most values concentrated at zero and very few high values. **Hours-per-week** centers around 40, indicating full-time work is most common, but includes tails on both ends. These insights suggest the need for normalization or transformation for certain features before further analysis or modeling.



Modeling Approach

To build a high-performing income classification model, we adopted a progressive modeling strategy.
We started with Logistic Regression as the baseline model and gradually explored more complex algorithms, evaluating how each one performed compared to the base.

Each model's evaluation was accompanied by confusion matrices and calculated metrics:

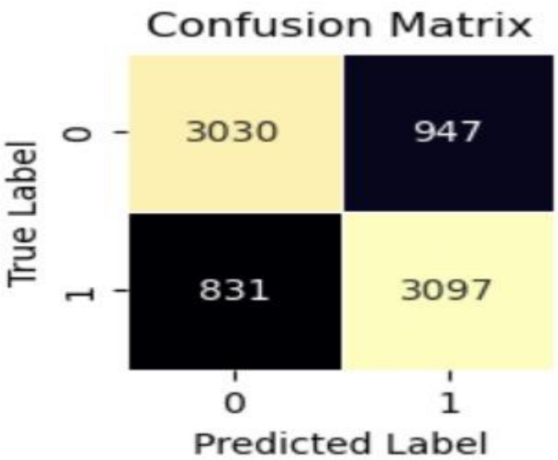
- **Accuracy**
- **Sensitivity** (Recall for the >50K class)
- **Specificity** (Recall for the <=50K class)

3.1 Logistic Regression (Baseline Model)

- Preprocessing: After handling missing values and applying log-transformation to skewed features, we standardized numerical features using StandardScaler to suit the assumptions of logistic regression (which is sensitive to scale).
- Label Encoding: All categorical variables were converted to numeric form using LabelEncoder, making them suitable for model input.
- Feature Selection: We applied forward feature selection using AIC (Akaike Information Criterion) via statsmodels.Logit to retain only the most informative predictors.
- Training: The model was trained using a balanced dataset (resampled via RandomOverSampler).
- Evaluation: Accuracy was evaluated using a train-test split and 10-fold cross-validation. The model yielded around 84.5% test accuracy—a modest but balanced performance across metrics.
- Insights: Logistic regression served as a strong baseline model due to its simplicity and interpretability. However, it lacked the complexity needed to capture non-linear relationships.

Performance:

- Accuracy: **77.51%**
- Sensitivity: **78.84%**
- Specificity: **76.19%**



Observation:

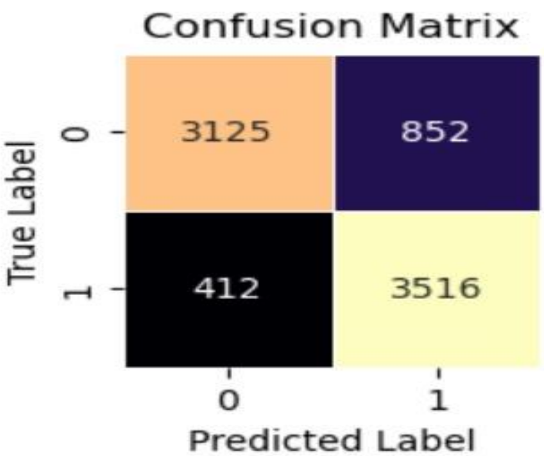
While Logistic Regression performed decently on overall accuracy, it **struggled to capture high-income individuals** (sensitivity was relatively low).

3.2 K-Nearest Neighbors (KNN)

- Preprocessing: Like logistic regression, KNN requires standardized features for distance-based calculations, so we ensured that all numerical features were scaled.
- Training & Evaluation: The model was trained on the resampled dataset. AIC-based forward feature selection was used to avoid the curse of dimensionality.
- Performance: While KNN showed higher training accuracy (almost 90%), its test accuracy (~84.13%) dropped, indicating possible overfitting and sensitivity to noisy features or imbalanced class boundaries.
- Insights: KNN's simplicity and non-parametric nature were useful, but performance degraded slightly due to the high-dimensional nature of the data and overlapping class distributions.

Performance:

- Accuracy: **84.01%**
- Sensitivity: **89.51%**
- Specificity: **78.58%**



Observation:

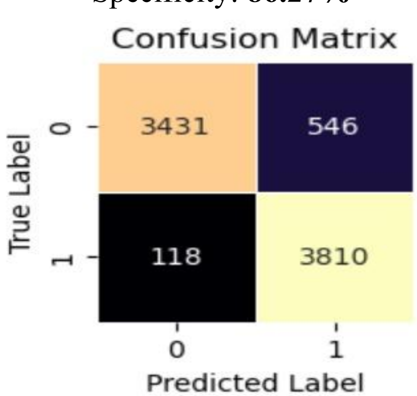
KNN slightly underperformed compared to Logistic Regression, especially in capturing the high-income class. It also exhibited longer prediction times due to instance-based computation.

3.3Decision Tree

- Training: A decision tree classifier was trained using the same selected features and balanced dataset.
- Performance: This model showed very high training accuracy (~95.87%) but poor test accuracy (~81.95%). Its lower CV score also confirmed overfitting.
- Insights: Decision Trees are prone to overfitting on complex datasets unless pruned or regularized. This justified exploring ensemble models next.

Performance:

- Accuracy: **91.60%**
- Sensitivity: **97.00%**
- Specificity: **86.27%**



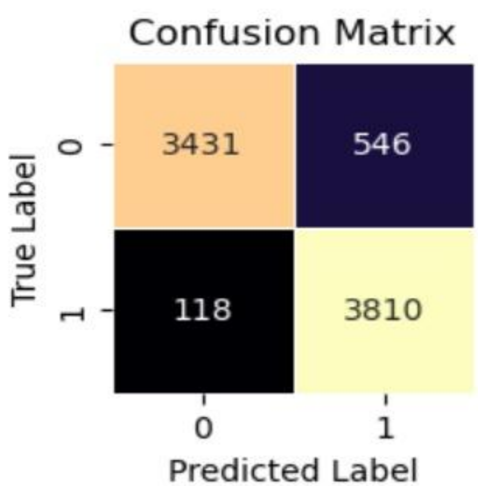
Observation:
While easily interpretable, the Decision Tree did not offer notable gains over Logistic Regression.

3.4 Random Forest

- Training: As an ensemble of decision trees, the Random Forest classifier was trained on the selected features after balancing the classes.
- Performance: It produced the highest test accuracy of 92.75%, along with excellent training (97.85%) and cross-validation scores.
- Feature Selection & Tuning:
- We applied Sequential Feature Selection to optimize inputs.
- Performed hyperparameter tuning using RandomizedSearchCV across n_estimators and max_depth.
- Insights: Random Forest emerged as the most stable and accurate model. It handles non-linearity well, reduces variance by averaging, and is less prone to overfitting compared to a single tree.

Performance:

- Accuracy: **92.75%**
- Sensitivity: **97.61%**
- Specificity: **87.96%**



Observation:

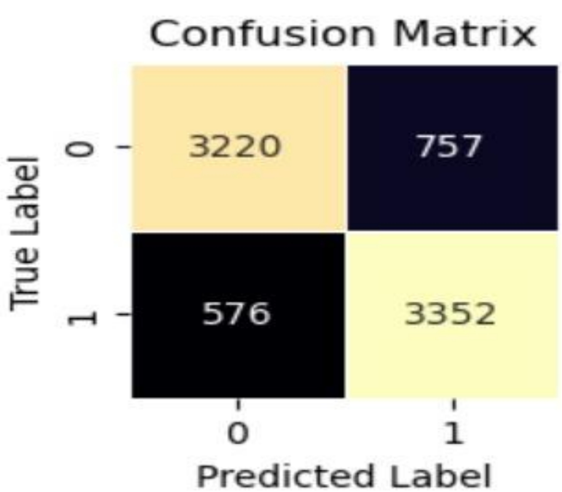
Random Forest is the **Final model achieving the best trade-off between sensitivity, specificity, and overall accuracy**. This final model was selected for deployment due to its consistent performance across multiple metrics and folds.

3.5 AdaBoost

- Training: Trained similarly using the AIC-selected features and balanced dataset.
- Performance: AdaBoost produced a test accuracy of 89.49%, slightly lower than XGBoost and Random Forest.
- Insights: AdaBoost works well on weak learners but is more affected by noisy data. Although it generalized reasonably well, it didn't outperform boosting or bagging counterparts.

Performance:

- Accuracy: **83.14%**
- Sensitivity: **85.34%**
- Specificity: **80.97%**



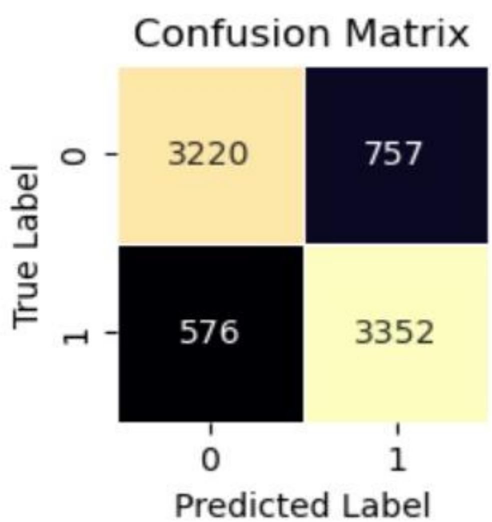
Observation:
AdaBoost marginally trailed Random Forest but still performed better than simpler classifiers.

3.6 Gradient Boosting

- Training: Like XGBoost, it is a boosting method, trained using forward-selected features.
- Performance: Gradient Boosting achieved a test accuracy of 91.60% with a strong CV score (88.24%), indicating good generalization.
- Insights: Although slower than Random Forest and slightly more complex, it performed robustly. However, its marginally lower accuracy made Random Forest the preferred model.

Performance:

- Accuracy: **83.73%**
- Sensitivity: **86.81%**
- Specificity: **80.69%**



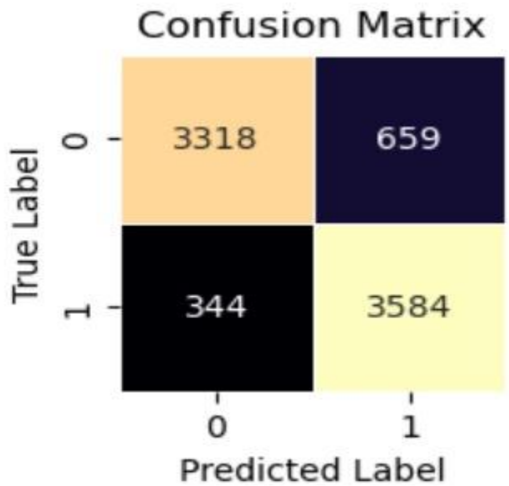
Observation:
Gradient Boosting offered further gains, especially in handling the high-income class better.

3.7 XGBoost

- Training: This gradient boosting model was trained with the same forward-selected features and class-balanced input.
- Performance: XGBoost showed high training accuracy (~94.31%) and a test accuracy of 91.85%, making it a strong contender.
- Insights: While it performed very well, the complexity of hyperparameter tuning, longer training time, and sensitivity to settings made it slightly less favorable than Random Forest for this specific dataset.

Performance:

- Accuracy: **87.31%**
- Sensitivity: **91.24%**
- Specificity: **83.43%**



Observation:
XGBoost provided a **significant improvement**, particularly boosting the sensitivity without sacrificing specificity.

Hyperparameter Tuning Strategy

While basic models such as Logistic Regression, Naive Bayes, and KNN require minimal parameter tuning, **ensemble models** like Random Forest, Gradient Boosting, and XGBoost **benefit significantly from hyperparameter optimization**.

To optimize model performance:

- We applied **RandomizedSearchCV**, an efficient strategy to sample a wide hyperparameter space without exhaustively checking every possible combination.
- This approach reduced computational cost while still identifying strong candidate configurations.

Cross-Validation Strategy:

- A **10-fold cross-validation** procedure was employed during hyperparameter tuning.
- The dataset was split into five parts; each model was trained on four parts and validated on the fifth, rotating across all splits.

- This technique ensures the model generalizes well and is not overfitted to specific data partitions.

By systematically optimizing hyperparameters and validating performance across multiple folds, we ensured that the selected models were not only highly accurate but also generalized well to unseen data.

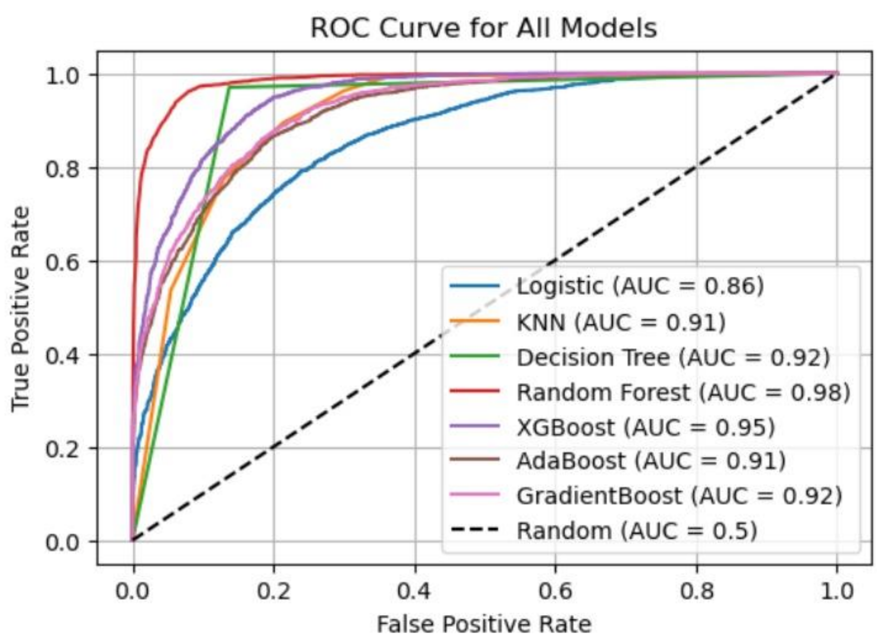
Results and Evaluation

4.1 Results

We experimented with a variety of machine learning models to identify the most effective classifier for predicting income level. Below is a summary of the models evaluated, including their training accuracy, cross-validation (CV) score, and test accuracy after applying forward feature selection.

	Model	Train Accuracy	CV Score	Test Accuracy
0	Logistic	77.4014	77.3974	77.5079
5	AdaBoost	82.9301	82.8323	83.1373
6	GradientBoost	83.8568	83.5813	83.7318
1	KNN	88.9363	84.7301	84.0101
4	XGBoost	89.5784	87.3539	87.3118
2	Decision Tree	99.9842	92.7332	91.6003
3	Random Forest	99.9842	93.6062	92.7514

4.2 Comparison of Models based on AUC



4.2 Final Chosen Model and Justification

After evaluating a wide range of classification algorithms—including logistic regression, decision trees, ensemble methods like AdaBoost and Gradient Boosting, and advanced models like XGBoost—we identified the **Random Forest Classifier** as the most suitable model for predicting income levels. This decision was based on its **superior performance metrics** across test accuracy, cross-validation score, sensitivity, and specificity.

Random Forest achieved the **highest test accuracy of approximately 92.75%**, significantly outperforming baseline models. This was further enhanced through:

- **Forward feature selection using Akaike Information Criterion (AIC)**, which optimized the input space to include only statistically significant predictors.
- **Hyperparameter tuning** with RandomizedSearchCV, specifically tuning the n_estimators (number of trees in the forest) and max_depth (depth of each tree), to reduce overfitting and improve generalization.

Why Random Forest?

- **Robust to Overfitting**: Unlike single decision trees, Random Forest aggregates predictions from multiple diverse trees using bagging (bootstrap aggregation), which minimizes variance and mitigates overfitting.
- **Handles High-Dimensional Data**: Random Forest excels at classification tasks with both high-dimensional and imbalanced datasets—a key consideration for our income dataset, which includes numerous categorical variables.
- **Implicit Feature Importance**: It ranks features by their importance in decision-making, which aligns well with our interpretability goals in a socio-economic dataset.
- **Versatility**: Random Forest does not assume any linear relationship between features and output, allowing it to model complex interactions, which is essential in real-world income prediction.
- **Resilient to Noisy Data and Imbalances**: Combined with oversampling, Random Forest naturally handles imbalances and is not overly sensitive to outliers.
- **Generalization Power**: Cross-validation and AIC-based selection ensured that the model generalized well across folds without overfitting.

Backed by strong empirical performance and robust theoretical foundations, Random Forest was chosen as the final model. It balances prediction accuracy with reliability and interpretability, making it a top-tier candidate for socio-demographic classification tasks such as income prediction.

Literature Survey

- https://aaai.org/papers/kdd96-033-scaling-up-the-accuracy-of-naive-bayes-classifiers-a-decision-tree-hybrid/
- https://ieeexplore.ieee.org/abstract/document/8489294
- https://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf
- https://dl.acm.org/doi/full/10.1145/3714334.3714341
- https://dl.acm.org/doi/abs/10.5555/2188385.2188395

Appendix

Team Contributions:

Name	SBU ID	Area of Contribution
Aishwarya Bhanage	116556145	Report & Literature Survey
Rutika Kadam	116753960	Model fitting
Sakshi Shah	116727594	Exploratory Data Analysis
Sanjyot Amritkar	116483478	Data Pre-processing
Tamali Halder	116713494	Report

Python Code – Starts from next page.

tedqotroy

April 26, 2025

0.0.1 Introduction to Dateset

This project involves building machine learning models to predict whether a person earns more than $50K per year based on demographic and employment-related attributes. The dataset originates from the 1994 Census Bureau database, extracted and cleaned by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics).

The dataset is available at: https://archive.ics.uci.edu/dataset/2/adult

The objective is to build a classification model using train.csv and evaluate its performance on test.csv.

Group Members: Aishwarya Bhanage | 116556145 | aishwaryamahad.bhanage@stonybrook.edu Rutika Kadam | 116753960 | rutikaavinash.kadam@stonybrook.edu
Sakshi Shah | 116727594 | sakshijanak.shah@stonybrook.edu
Sanjyot Amritkar | 116483478 | sanjyotsatish.amritkar@stonybrook.edu
Tamali Halder | 116713494 | tamali.halder@stonybrook.edu

[3]:
```
#imprting required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import cross_val_score, train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪GradientBoostingClassifier
from xgboost import XGBClassifier
import statsmodels.api as sm
from tabulate import tabulate
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,␣
↪recall_score
```

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

[55]:
```python
import os
print(os.getcwd())
```

```
C:\Users\Rutika\Statistical Learning Project\data
```

[56]:
```python
#reading train and test data
train = pd.read_csv(r"C:\Users\Rutika\Statistical Learning Project\data\train.
↪csv")
test = pd.read_csv(r"C:\Users\Rutika\Statistical Learning Project\data\test.
↪csv")
```

[57]:
```python
train.head()
```

[57]:
```
   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   49           Private  160187        9th              5

          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-spouse-absent      Other-service  Not-in-family  Black

      sex  capital-gain  capital-loss  hours-per-week  native-country  income
0    Male             0             0              13   United-States   <=50K
1    Male             0             0              40   United-States   <=50K
2    Male             0             0              40   United-States   <=50K
3  Female             0             0              40            Cuba   <=50K
4  Female             0             0              16         Jamaica   <=50K
```

[58]:
```python
test.head()
```

[58]:
```
   age   workclass  fnlwgt     education  education-num  \
0   39   State-gov   77516     Bachelors             13
1   37     Private  284582       Masters             14
2   23     Private  122272     Bachelors             13
3   32     Private  186824       HS-grad              9
4   30 Federal-gov   59951  Some-college             10

       marital-status          occupation   relationship    race      sex  \
```

```
0        Never-married         Adm-clerical   Not-in-family  White    Male
1   Married-civ-spouse      Exec-managerial            Wife  White  Female
2        Never-married         Adm-clerical       Own-child  White  Female
3        Never-married    Machine-op-inspct       Unmarried  White    Male
4   Married-civ-spouse         Adm-clerical       Own-child  White    Male

   capital-gain  capital-loss  hours-per-week  native-country  income
0          2174             0              40   United-States   <=50K
1             0             0              40   United-States   <=50K
2             0             0              30   United-States   <=50K
3             0             0              40   United-States   <=50K
4             0             0              40   United-States   <=50K
```

[59]:
```
#Add a column to identify the source
train['data_type'] = 'train'
test['data_type'] = 'test'
```

[60]:
```
#Combine datasets for consistent preprocessing
full_data = pd.concat([train, test], ignore_index=True)
```

[61]:
```
full_data.shape
```

[61]: (32561, 16)

[62]:
```
#checking columns
full_data.columns
```

[62]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'income', 'data_type'],
      dtype='object')
```

[63]:
```
#checking duplicate data
full_data.duplicated().sum()
```

[63]: 17

[64]:
```
#dropping duplicates
full_data=full_data.drop_duplicates()
```

[65]:
```
full_data.shape
```

[65]: (32544, 16)

[66]:
```
full_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32544 entries, 0 to 32560
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32544 non-null  int64
 1   workclass       32544 non-null  object
 2   fnlwgt          32544 non-null  int64
 3   education       32544 non-null  object
 4   education-num   32544 non-null  int64
 5   marital-status  32544 non-null  object
 6   occupation      32544 non-null  object
 7   relationship    32544 non-null  object
 8   race            32544 non-null  object
 9   sex             32544 non-null  object
 10  capital-gain    32544 non-null  int64
 11  capital-loss    32544 non-null  int64
 12  hours-per-week  32544 non-null  int64
 13  native-country  32544 non-null  object
 14  income          32544 non-null  object
 15  data_type       32544 non-null  object
dtypes: int64(6), object(10)
memory usage: 4.2+ MB
```

[67]:
```
full_data.describe()
```

[67]:
```
                age        fnlwgt  education-num  capital-gain  capital-loss  \
count  32544.000000  3.254400e+04   32544.000000  32544.000000  32544.000000
mean      38.582811  1.897798e+05      10.081828   1078.211775     87.349435
std       13.638327  1.055533e+05       2.571421   7387.179736    403.060513
min       17.000000  1.228500e+04       1.000000      0.000000      0.000000
25%       28.000000  1.178242e+05       9.000000      0.000000      0.000000
50%       37.000000  1.783560e+05      10.000000      0.000000      0.000000
75%       48.000000  2.370065e+05      12.000000      0.000000      0.000000
max       90.000000  1.484705e+06      16.000000  99999.000000   4356.000000

       hours-per-week
count    32544.000000
mean        40.436916
std         12.349961
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
```

[68]:
```
#checking missing values
full_data.isnull().sum()
```

[68]:
```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
income            0
data_type         0
dtype: int64
```

[69]:
```
#checking numeric and categoric columns in dataset
numeric_cols = [f for f in full_data.columns if full_data[f].dtype!='O']
print("We have {} numerical features in dataset: {}".
  ↪format(len(numeric_cols),numeric_cols))
categoric_cols = [f for f in full_data.columns if full_data[f].dtype=='O']
print("We have {} categorical faeatures in dataset: {}".
  ↪format(len(categoric_cols),categoric_cols))
```

```
We have 6 numerical features in dataset: ['age', 'fnlwgt', 'education-num',
'capital-gain', 'capital-loss', 'hours-per-week']
We have 10 categorical faeatures in dataset: ['workclass', 'education',
'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country',
'income', 'data_type']
```

[70]:
```
# Apply strip() across all categorical columns
full_data[categoric_cols] = full_data[categoric_cols].apply(lambda col: col.str.
  ↪strip())
```

[71]:
```
for col in categoric_cols:
    # get the unique categories in this column
    unique_vals = full_data[col].unique()
    count = len(unique_vals)

    print(f"There are {count} categories in '{col}' column")
    print(unique_vals.tolist(),"\n")
```

```
There are 9 categories in 'workclass' column
['Self-emp-not-inc', 'Private', 'State-gov', 'Federal-gov', 'Local-gov', '?',
'Self-emp-inc', 'Without-pay', 'Never-worked']

There are 16 categories in 'education' column
['Bachelors', 'HS-grad', '11th', '9th', 'Masters', 'Some-college', 'Assoc-acdm',
'Assoc-voc', '7th-8th', 'Doctorate', 'Prof-school', '5th-6th', '10th',
'1st-4th', 'Preschool', '12th']

There are 7 categories in 'marital-status' column
['Married-civ-spouse', 'Divorced', 'Married-spouse-absent', 'Never-married',
'Separated', 'Married-AF-spouse', 'Widowed']

There are 15 categories in 'occupation' column
['Exec-managerial', 'Handlers-cleaners', 'Prof-specialty', 'Other-service',
'Sales', 'Craft-repair', 'Transport-moving', 'Farming-fishing', 'Tech-support',
'?', 'Protective-serv', 'Machine-op-inspct', 'Adm-clerical', 'Priv-house-serv',
'Armed-Forces']

There are 6 categories in 'relationship' column
['Husband', 'Not-in-family', 'Wife', 'Own-child', 'Unmarried', 'Other-relative']

There are 5 categories in 'race' column
['White', 'Black', 'Asian-Pac-Islander', 'Amer-Indian-Eskimo', 'Other']

There are 2 categories in 'sex' column
['Male', 'Female']

There are 42 categories in 'native-country' column
['United-States', 'Cuba', 'Jamaica', 'India', '?', 'Mexico', 'South', 'Puerto-
Rico', 'Honduras', 'Iran', 'Philippines', 'England', 'Poland', 'Germany',
'Ecuador', 'Laos', 'Taiwan', 'Portugal', 'Dominican-Republic', 'El-Salvador',
'France', 'Haiti', 'Guatemala', 'China', 'Yugoslavia', 'Canada', 'Japan',
'Thailand', 'Peru', 'Scotland', 'Italy', 'Trinadad&Tobago', 'Greece',
'Nicaragua', 'Cambodia', 'Vietnam', 'Hong', 'Columbia', 'Ireland', 'Outlying-
US(Guam-USVI-etc)', 'Hungary', 'Holand-Netherlands']

There are 2 categories in 'income' column
['<=50K', '>50K']

There are 2 categories in 'data_type' column
['train', 'test']
```

Observations from above: Columns named 'workclass', 'occupation' and 'country' have '?' as category, so we will replace this '?' with mode of categories in that particular column.

[72]:
```
for col in categoric_cols:
    if '?' in full_data[col].values:
        # compute the mode (most frequent value) for this column
        mode_val = full_data[col].mode()[0]
        # replace all '?' entries with the mode
        full_data[col].replace('?', mode_val, inplace=True)
```

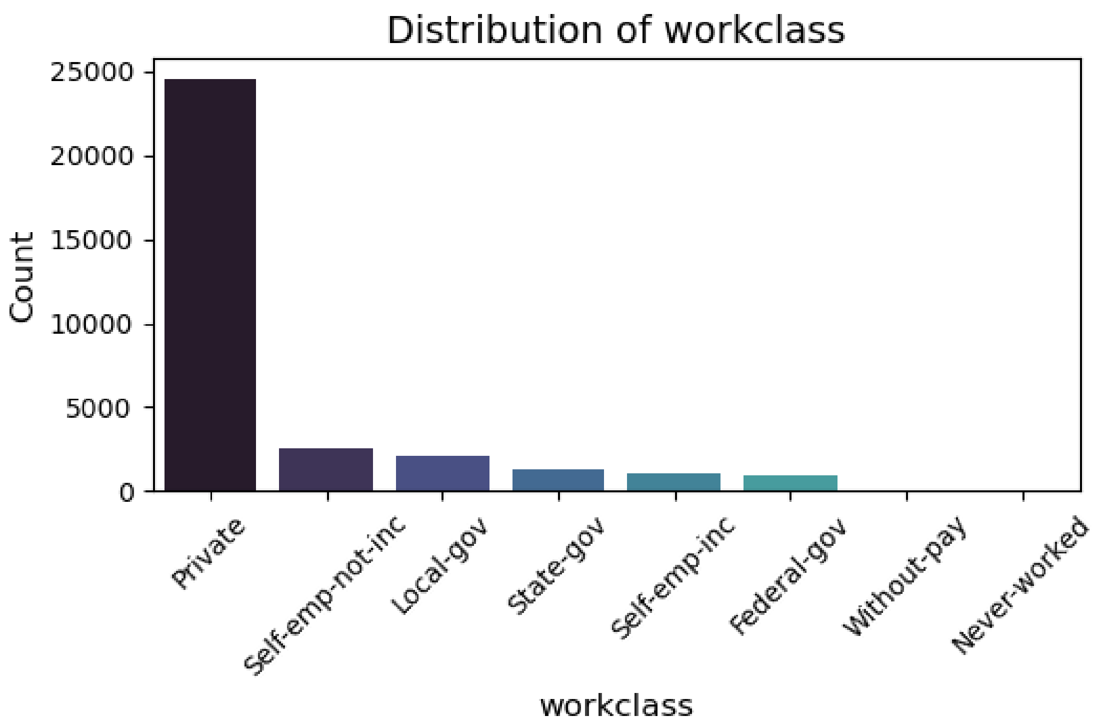
[73]:
```
#checking unique values in each column
full_data.nunique()
```

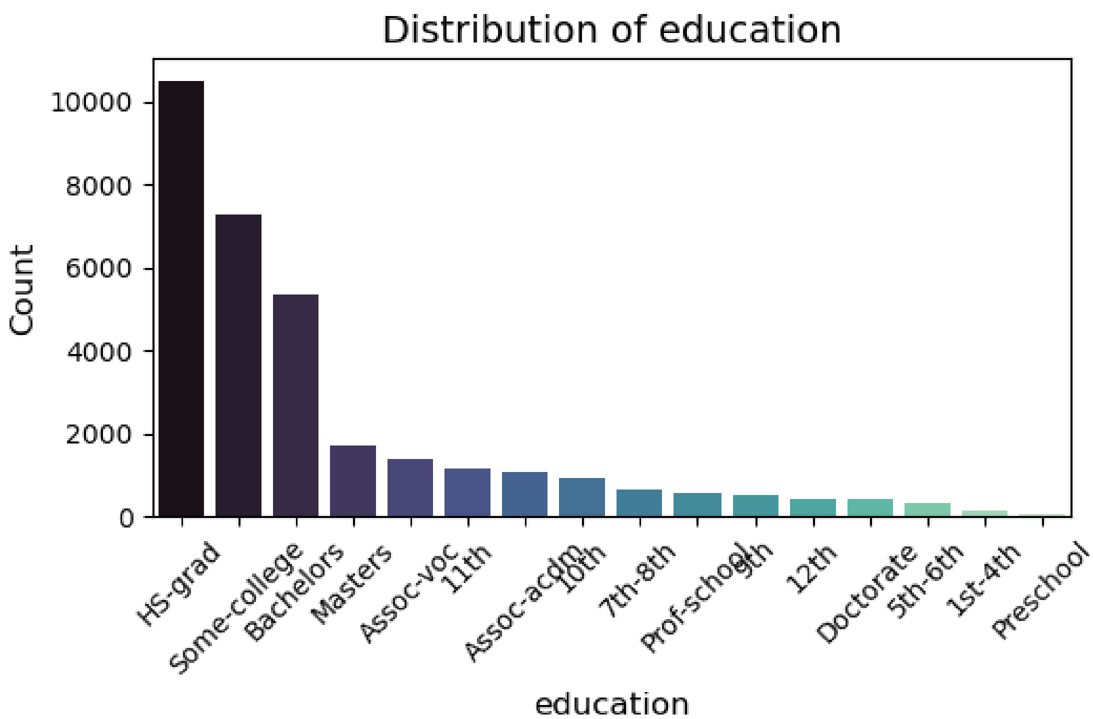
[73]:
```
age                  73
workclass             8
fnlwgt            21648
education            16
education-num        16
marital-status        7
occupation           14
relationship          6
race                  5
sex                   2
capital-gain        119
capital-loss         92
hours-per-week       94
native-country       41
income                2
data_type             2
dtype: int64
```

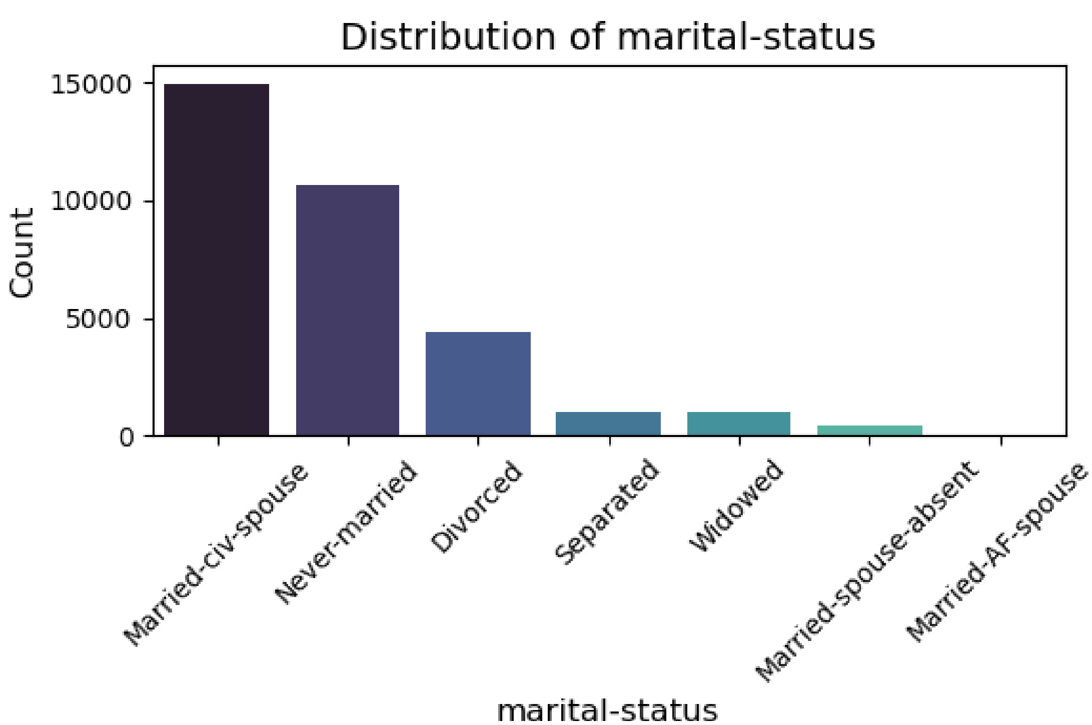
0.0.2 Exploratory Data Analysis

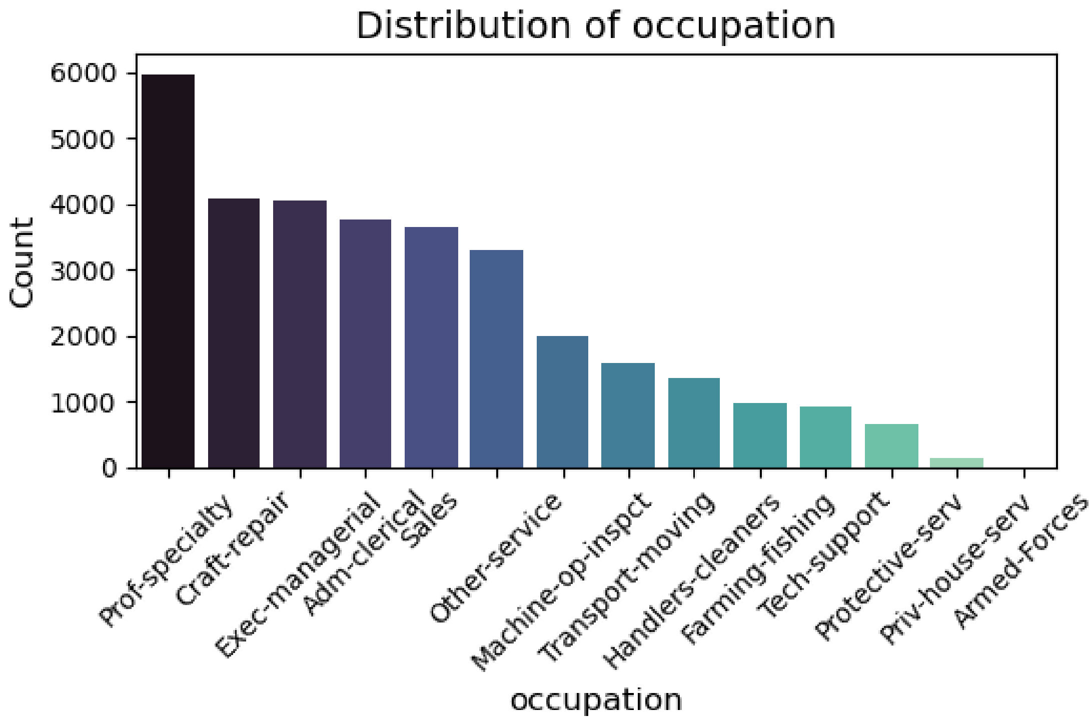
[77]:
```
# Define a color palette function for different colors
palette_colors = sns.color_palette("flare", 20)
for col in categoric_cols:
    plt.figure(figsize=(6, 4))

    # Get number of unique categories for that column
    num_categories = full_data[col].nunique()

    # Create a palette with unique colors for each bar
    palette = sns.color_palette("mako", num_categories)

    sns.countplot(
        data=full_data,
        x=col,
        order=full_data[col].value_counts().index,
        palette=palette
    )
```

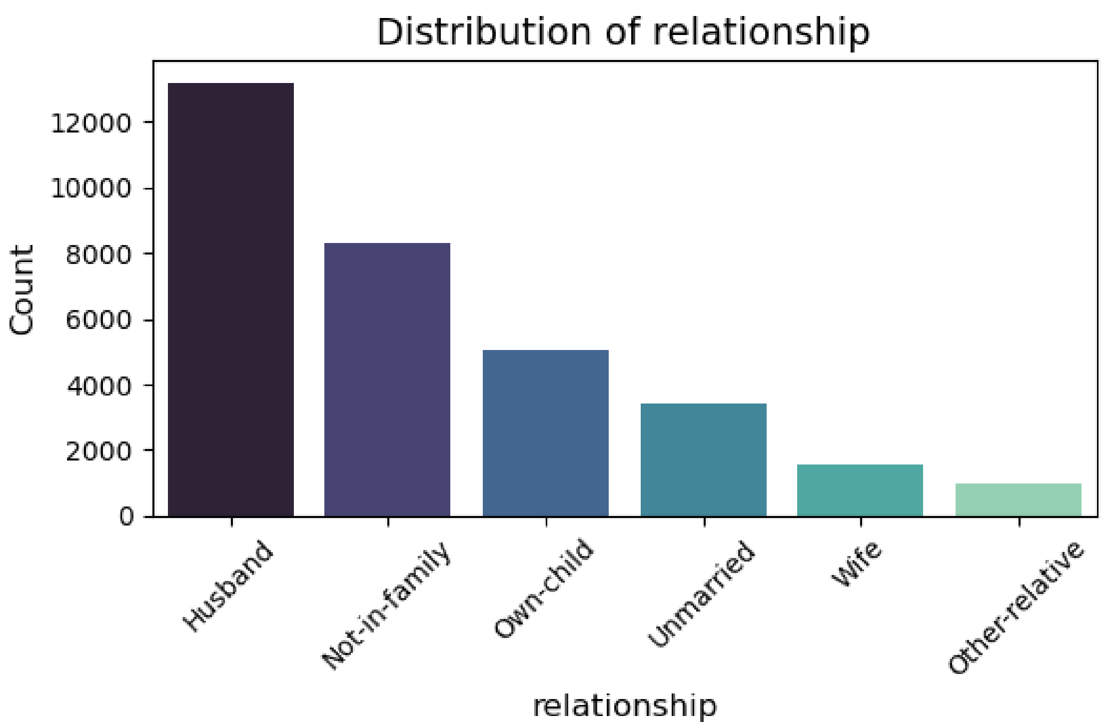
```
plt.xticks(rotation=45)
plt.title(f'Distribution of {col}', fontsize=14)
plt.xlabel(col, fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.tight_layout()
plt.show()
```

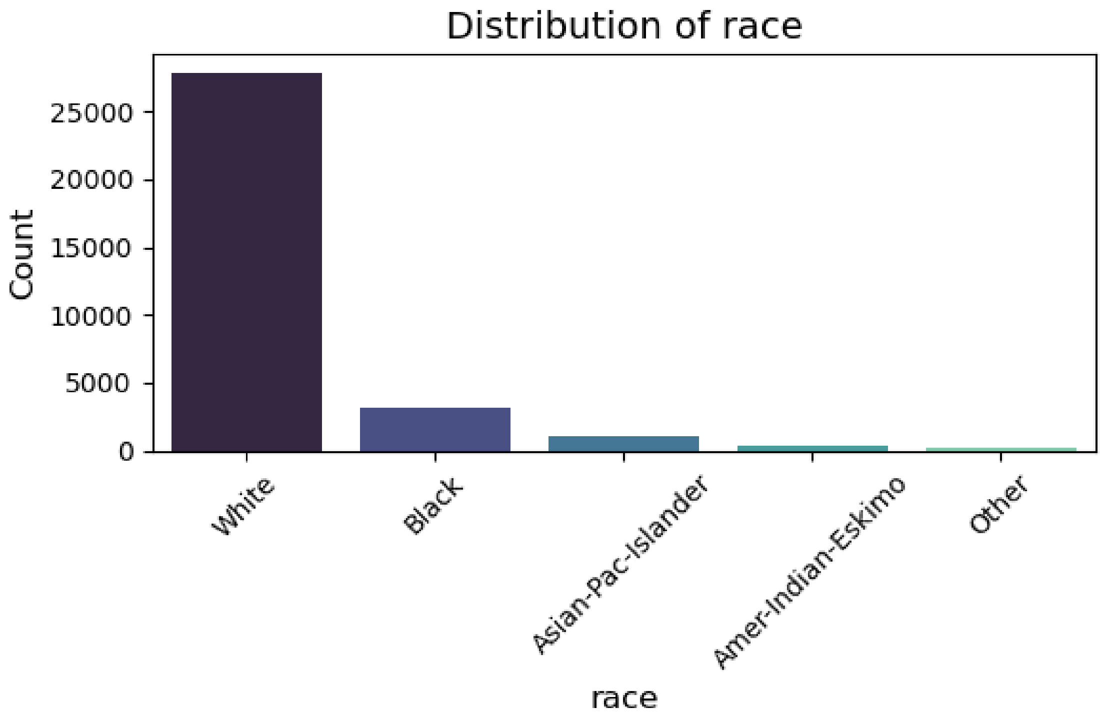


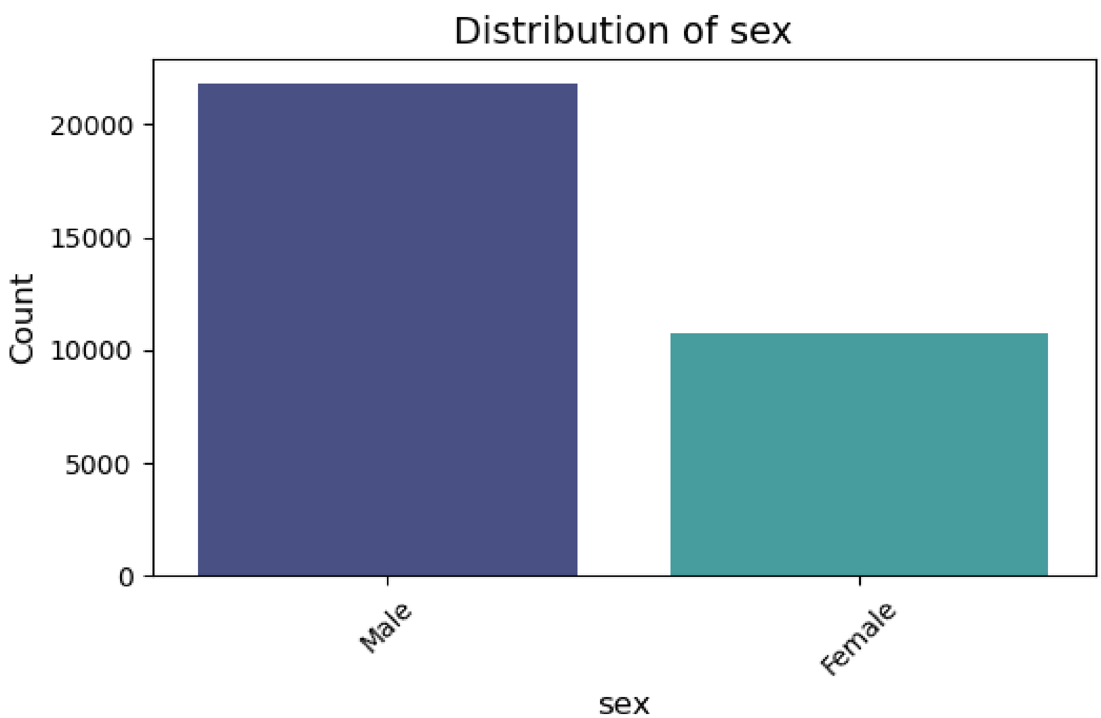

Distribution of education
Count
10000
8000
6000
4000
2000
0
HS-grad
Some-college
Bachelors
Masters
Assoc-voc
11th
Assoc-acdm
10th
7th-8th
Prof-school
9th
12th
Doctorate
5th-6th
1st-4th
Preschool
education

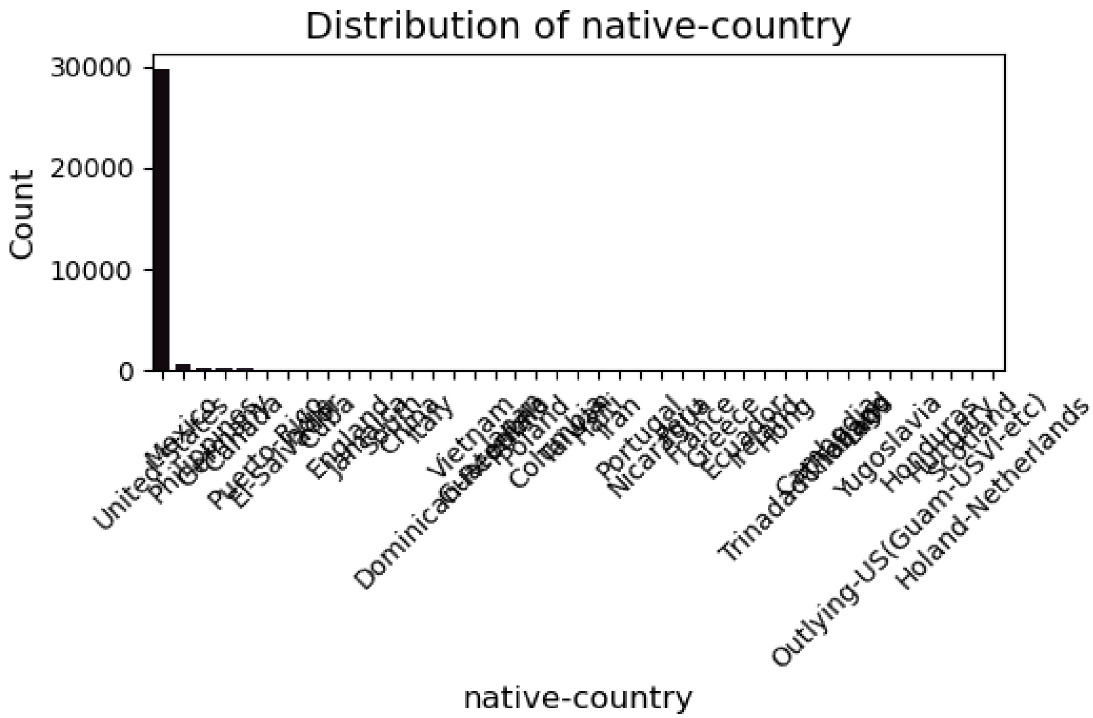

Distribution of marital-status
Count
15000
10000
5000
0
Married-civ-spouse
Never-married
Divorced
Separated
Widowed
Married-spouse-absent
Married-AF-spouse
marital-status

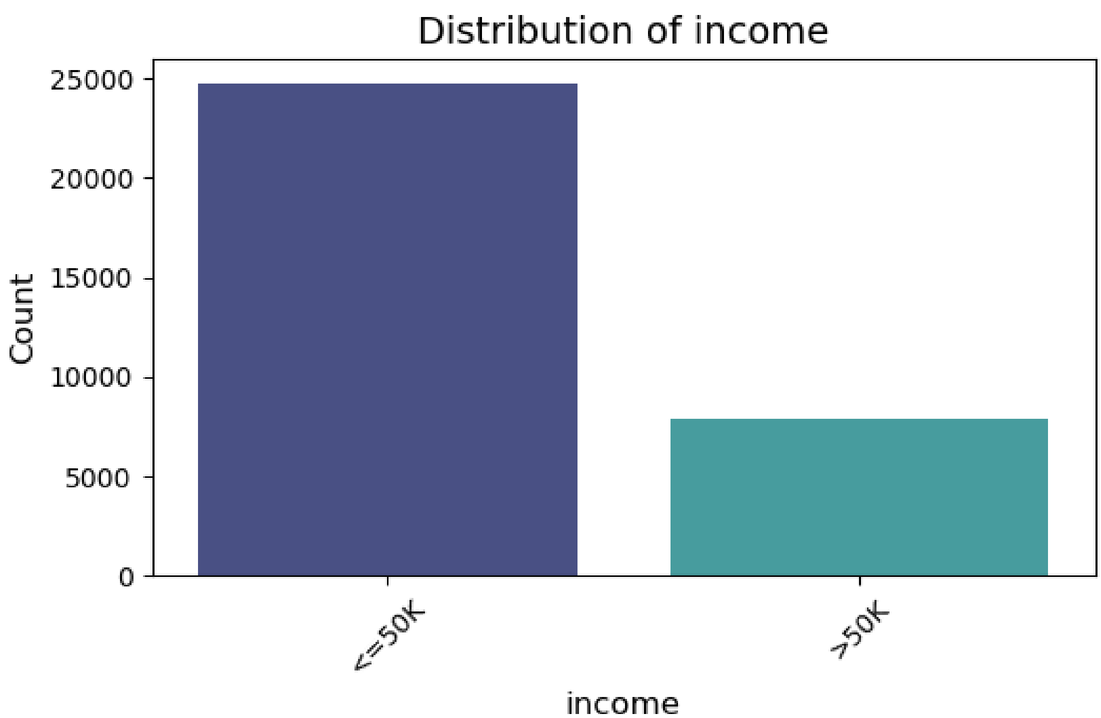

Distribution of occupation
Count
6000
5000
4000
3000
2000
1000
0
Prof-specialty
Craft-repair
Exec-managerial
Adm-clerical
Sales
Other-service
Machine-op-inspct
Transport-moving
Handlers-cleaners
Farming-fishing
Tech-support
Protective-serv
Priv-house-serv
Armed-Forces
occupation

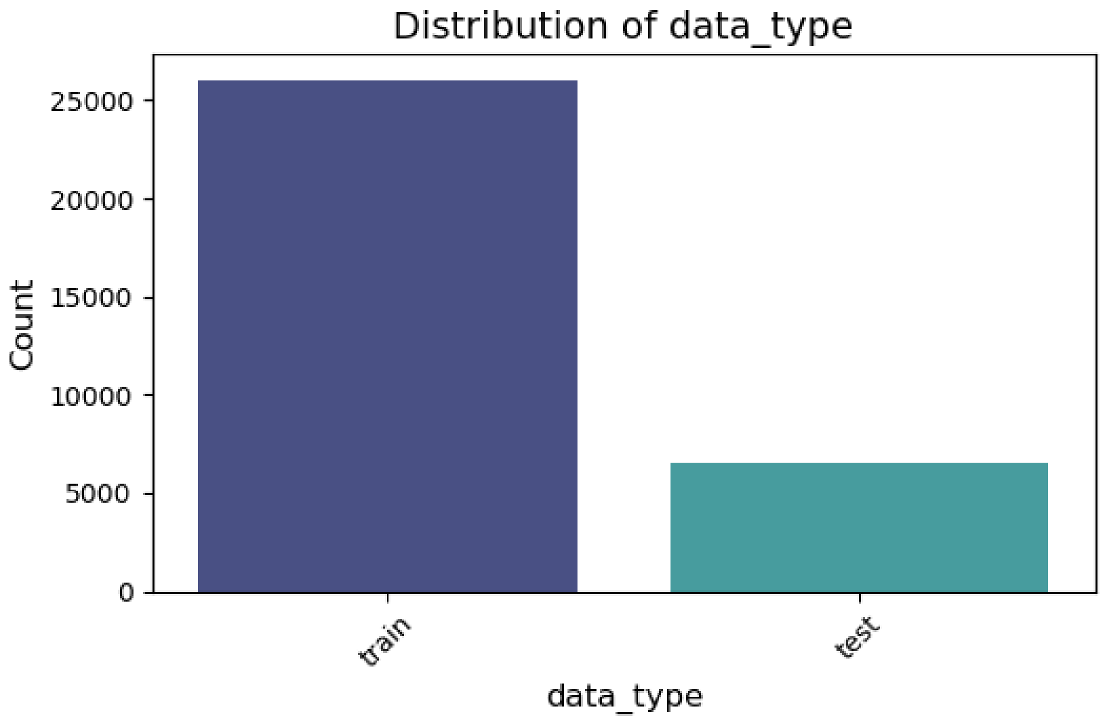

Distribution of relationship
Count
12000
10000
8000
6000
4000
2000
0
Husband
Not-in-family
Own-child
Unmarried
Wife
Other-relative
relationship


Distribution of race
Count
25000
20000
15000
10000
5000
0
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
race

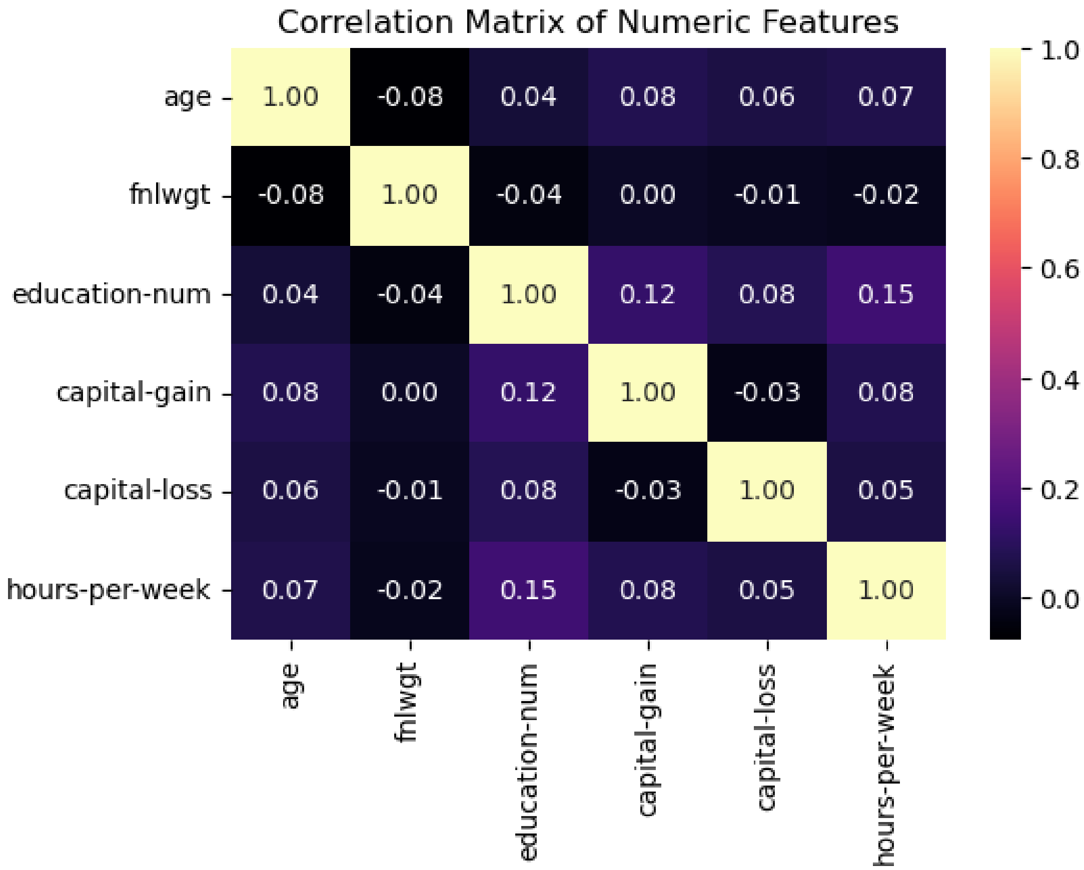

Distribution of sex
Count
20000
15000
10000
5000
0
Male
Female
sex


Distribution of native-country
30000
20000
10000
0
Count
United-States
Mexico
Canada
Puerto-Rico
England
China
Italy
Vietnam
Poland
Iran
Portugal
France
Greece
Ecuador
Yugoslavia
Honduras
Scotland
Outlying-US(Guam-USVI-etc)
Holand-Netherlands
native-country


Distribution of income
25000
20000
15000
10000
5000
0
Count
<=50K
>50K
income



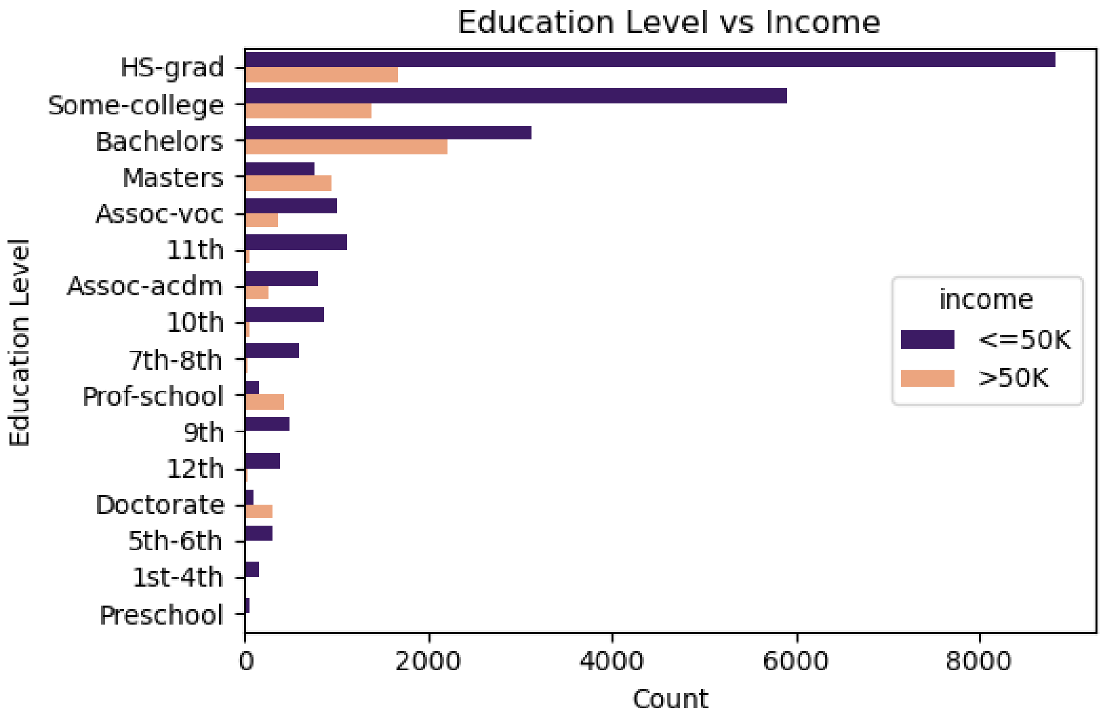
[108]:
```
#Compute and plot the correlation matrix for numeric varibales
plt.figure(figsize=(6, 4))
sns.heatmap(full_data.corr(), annot=True, cmap='magma', fmt=".2f")
plt.title("Correlation Matrix of Numeric Features")
plt.show()
```



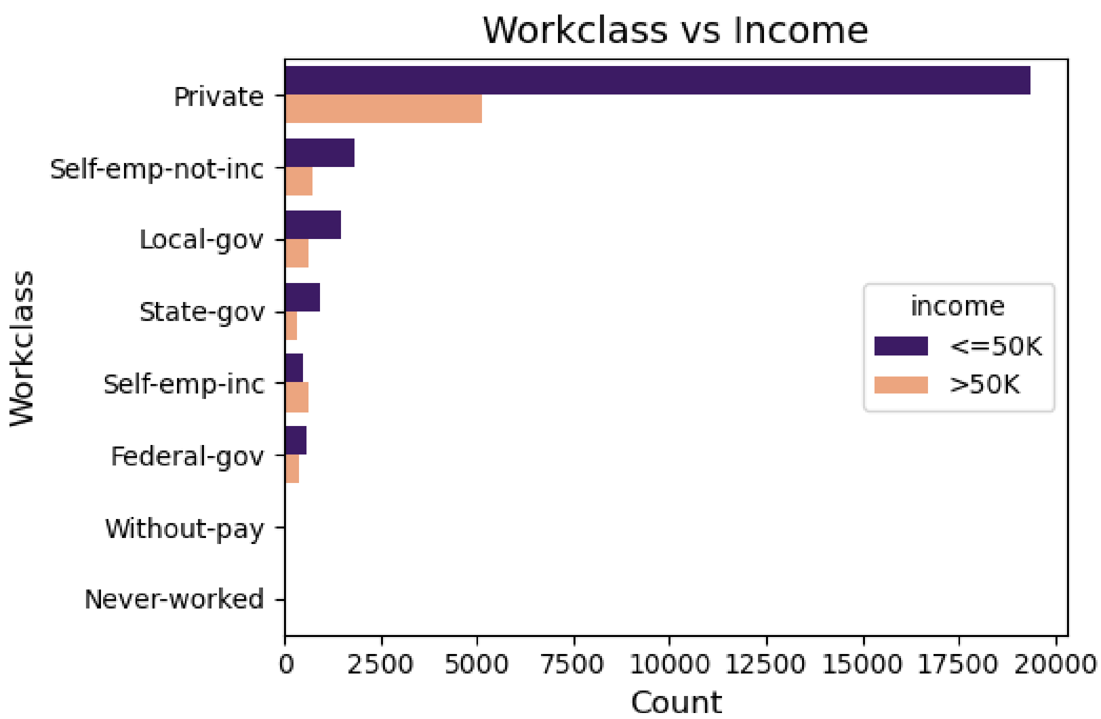
[109]:
```
#Age distribution by Income
plt.figure(figsize=(6, 4))
custom_palette = {'<=50K': '#3b0f70', '>50K': '#fe9f6d'}
sns.histplot(data=full_data, x='age', hue='income', multiple='stack', kde=True,
    palette=custom_palette)
plt.title("Age Distribution by Income")
plt.show()
```

[110]:

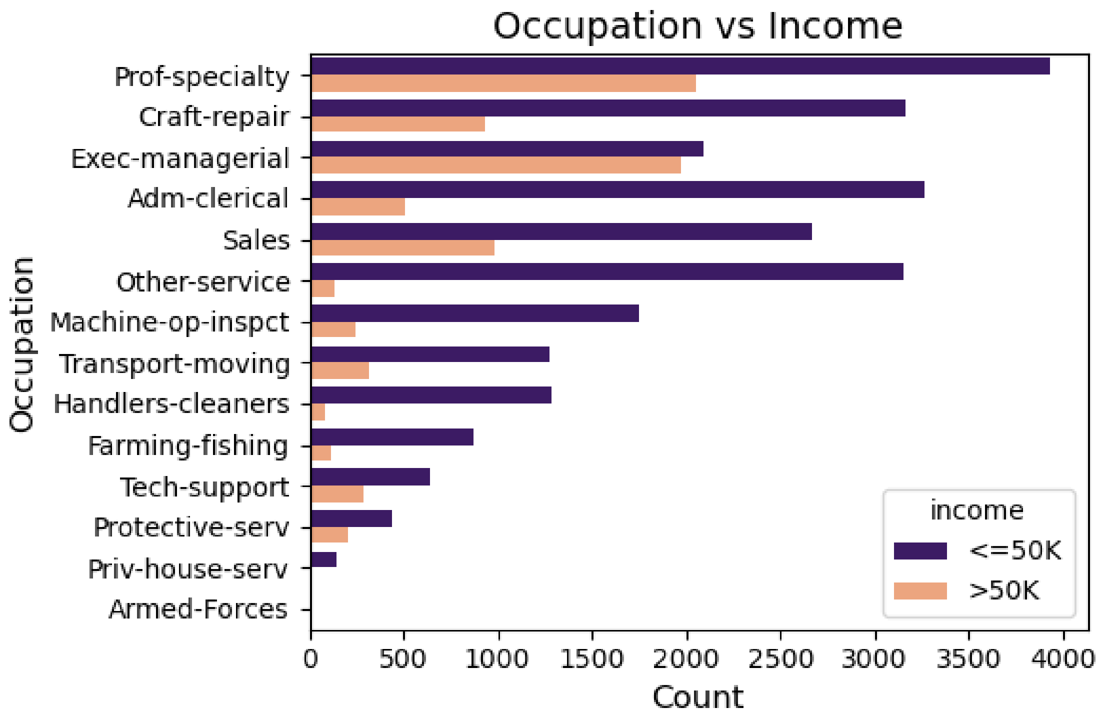
```
plt.figure(figsize=(6, 4))

# Define custom colors
custom_palette = {
    '<=50K': '#3b0f70',    # Deep Purple
    '>50K': '#fe9f6d'      # Light Orange
}
sns.countplot(
    data=full_data,
    y='education',
    hue='income',
    order=full_data['education'].value_counts().index,
    palette=custom_palette
)
plt.title("Education Level vs Income")
plt.xlabel("Count")
plt.ylabel("Education Level")
plt.tight_layout()
plt.show()
```



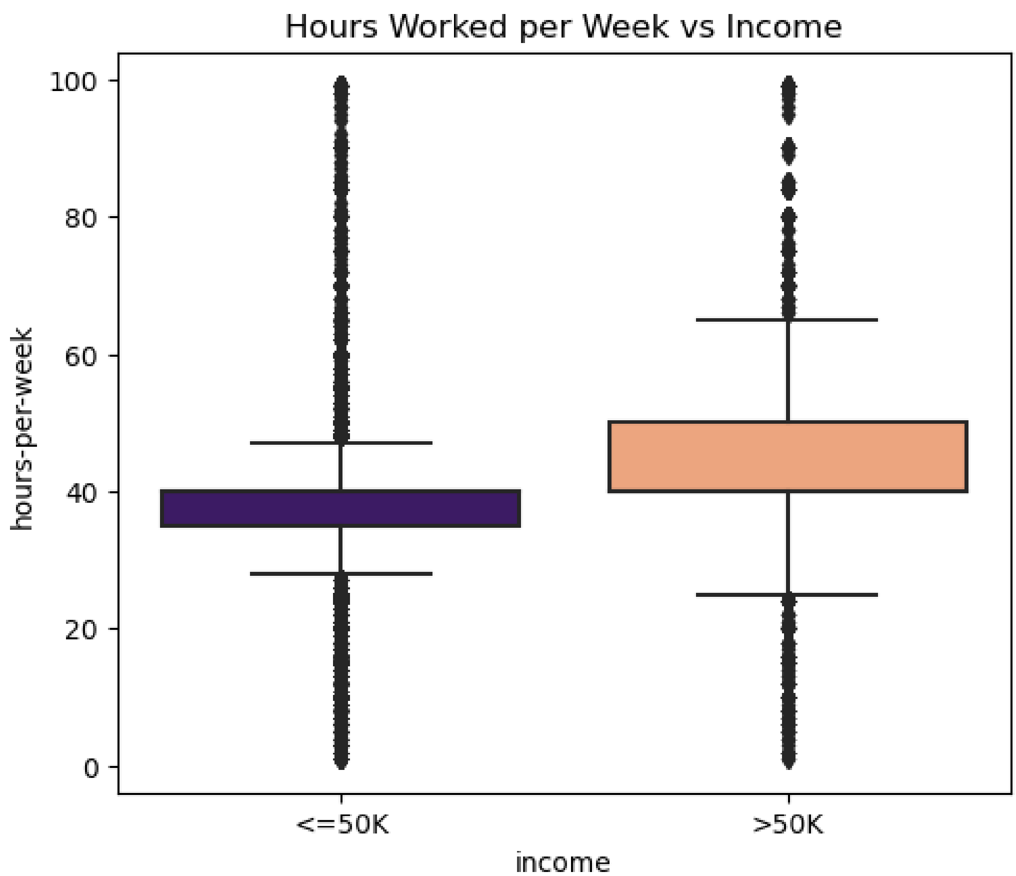
[111]:
```
plt.figure(figsize=(6, 4))
# Define custom colors
custom_palette = {
    '<=50K': '#3b0f70',    # Deep Purple
    '>50K': '#fe9f6d'      # Light Orange
}
sns.countplot(
    data=full_data,
    y='workclass',
    hue='income',
    order=full_data['workclass'].value_counts().index,
    palette=custom_palette
)
plt.title("Workclass vs Income", fontsize=14)
plt.xlabel("Count", fontsize=12)
plt.ylabel("Workclass", fontsize=12)
plt.tight_layout()
plt.show()
```



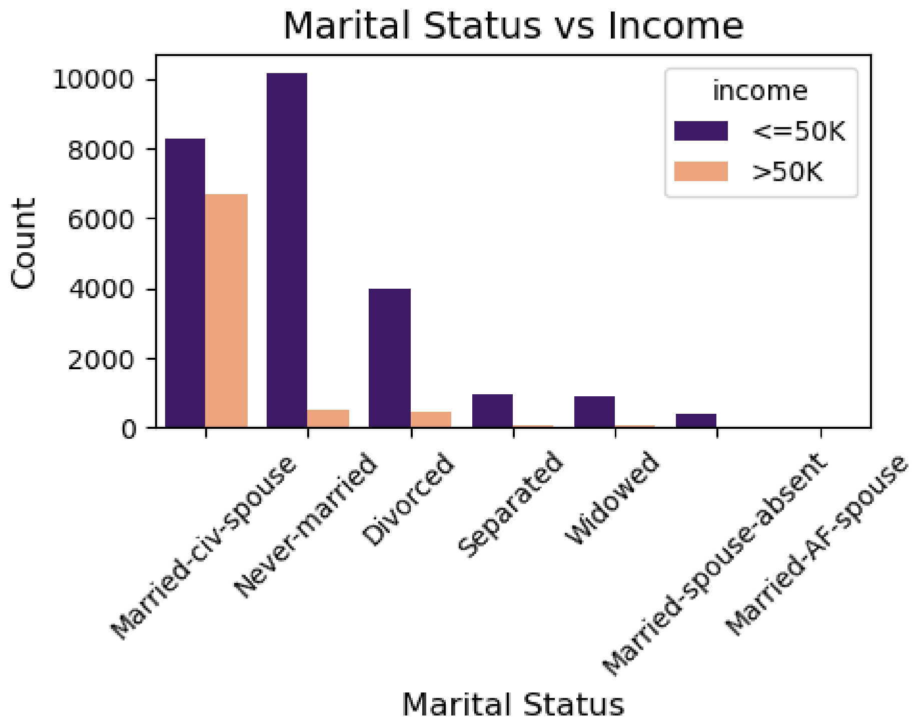
[112]:
```python
plt.figure(figsize=(6, 4))
# Define custom colors for income classes
custom_palette = {
    '<=50K': '#3b0f70',    # Deep Purple
    '>50K': '#fe9f6d'      # Light Orange
}
sns.countplot(
    data=full_data,
    y='occupation',
    hue='income',
    order=full_data['occupation'].value_counts().index,
    palette=custom_palette
)

plt.title("Occupation vs Income", fontsize=14)
plt.xlabel("Count", fontsize=12)
plt.ylabel("Occupation", fontsize=12)
plt.tight_layout()
plt.show()
```



[83]:
```
plt.figure(figsize=(6, 5))
# Define custom color palette
custom_palette = {
    '<=50K': '#3b0f70',
    '>50K': '#fe9f6d'
}
sns.boxplot(data=full_data, x='income',␣
 ↪y='hours-per-week',palette=custom_palette)
plt.title("Hours Worked per Week vs Income")
plt.show()
```

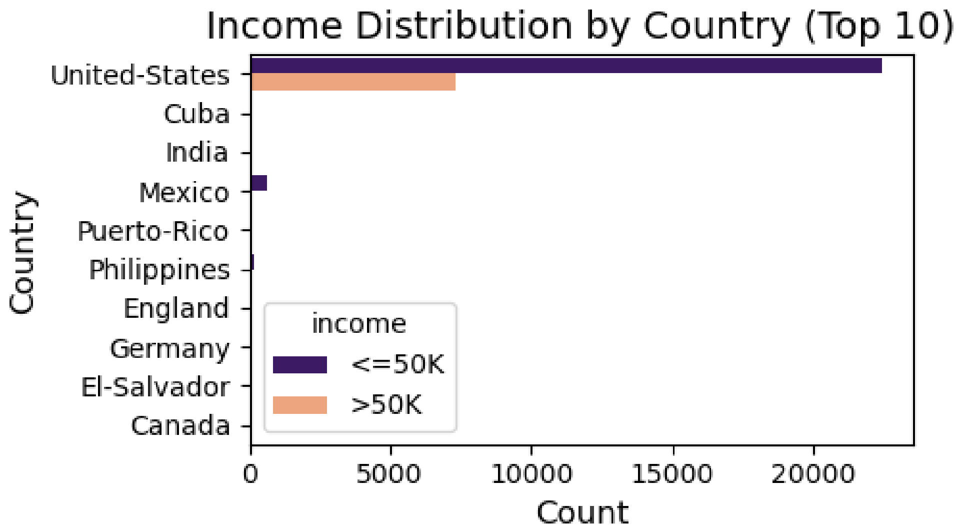


[78]:
```
plt.figure(figsize=(5, 4))
# Define custom color palette
custom_palette = {
    '<=50K': '#3b0f70',    # Deep Purple
    '>50K': '#fe9f6d'      # Light Orange
}
sns.countplot(
    data=full_data,
    x='marital-status',
    hue='income',
    order=full_data['marital-status'].value_counts().index,
    palette=custom_palette
)
plt.xticks(rotation=45)
plt.title("Marital Status vs Income", fontsize=14)
plt.xlabel("Marital Status", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.tight_layout()
plt.show()
```

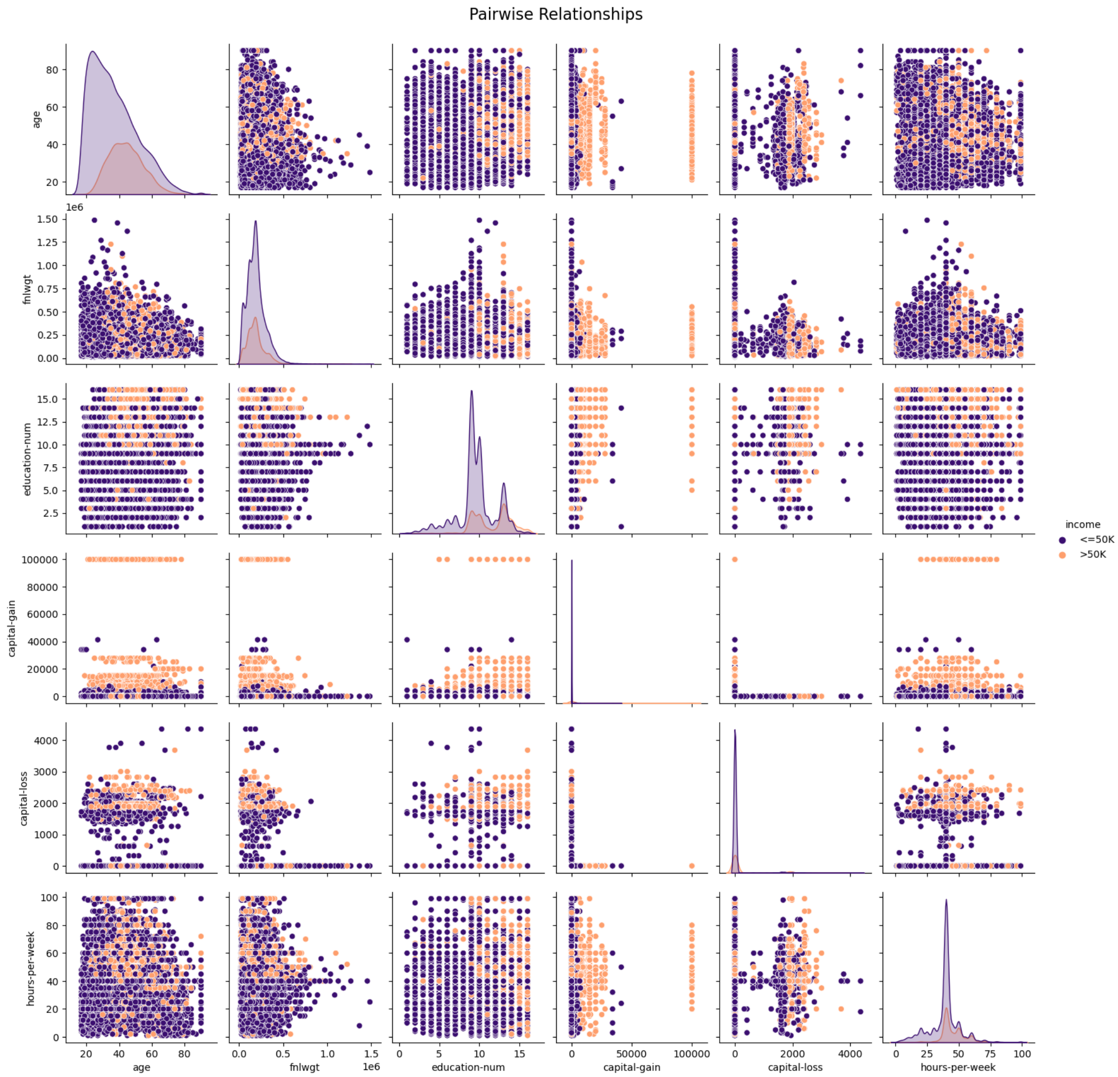


[81]:

```
# Get top 10 countries by count
top_countries = full_data['native-country'].value_counts().head(10).index
plt.figure(figsize=(5, 3))
# Define custom color palette
custom_palette = {
    '<=50K': '#3b0f70',    # Deep Purple
    '>50K': '#fe9f6d'      # Light Orange
}
sns.countplot(
    data=full_data[full_data['native-country'].isin(top_countries)],
    y='native-country',
    hue='income',
    palette=custom_palette
)
plt.title("Income Distribution by Country (Top 10)", fontsize=14)
plt.xlabel("Count", fontsize=12)
plt.ylabel("Country", fontsize=12)
plt.tight_layout()
plt.show()
```

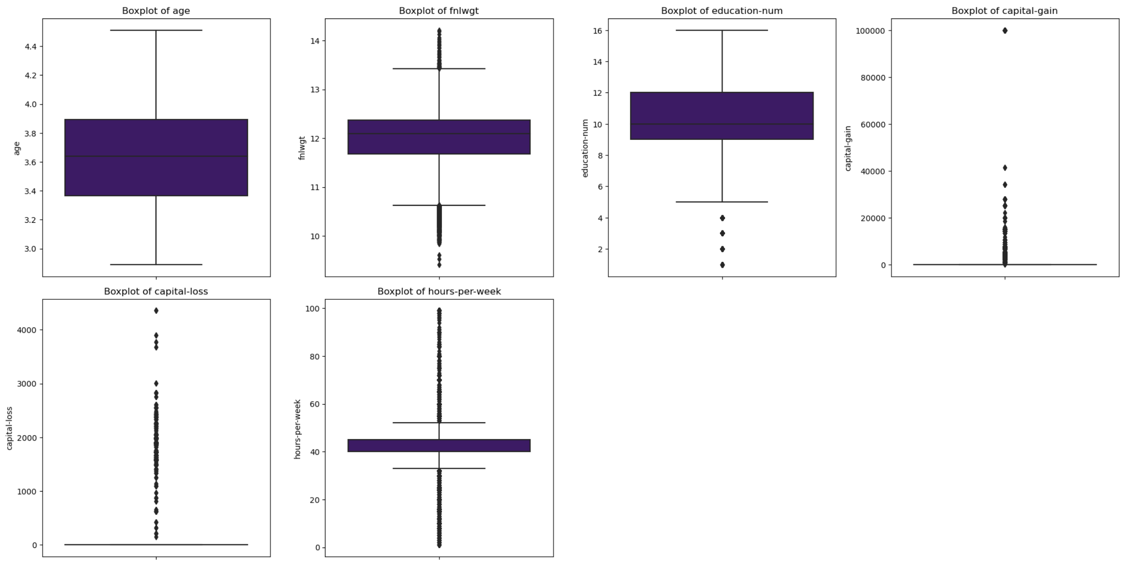


[115]:
```
# Define custom palette
custom_palette = {
    '<=50K': '#3b0f70',
    '>50K': '#fe9f6d'
}
# Create the pairplot
sns.pairplot(full_data[numeric_cols + ['income']], hue='income',␣
 ↪palette=custom_palette)

# Add a title with spacing
plt.suptitle("Pairwise Relationships", y=1.02, fontsize=16)
plt.show()
```



[122]:
```
# Create a grid of subplots: adjust nrows/ncols to fit your number of features
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(20, 15))  # 3x4 grid for up
 ↪to 12 plots
axes = axes.flatten()  # flatten to 1D array for easy indexing

index = 0
for col in numeric_cols:
    sns.boxplot(y=full_data[col].dropna(), ax=axes[index],color='#3b0f70')
    axes[index].set_title(f"Boxplot of {col}")
    axes[index].set_xlabel("")     # omit x-axis label
    axes[index].set_ylabel(col)
    index += 1

# If there are unused subplots, hide them
```

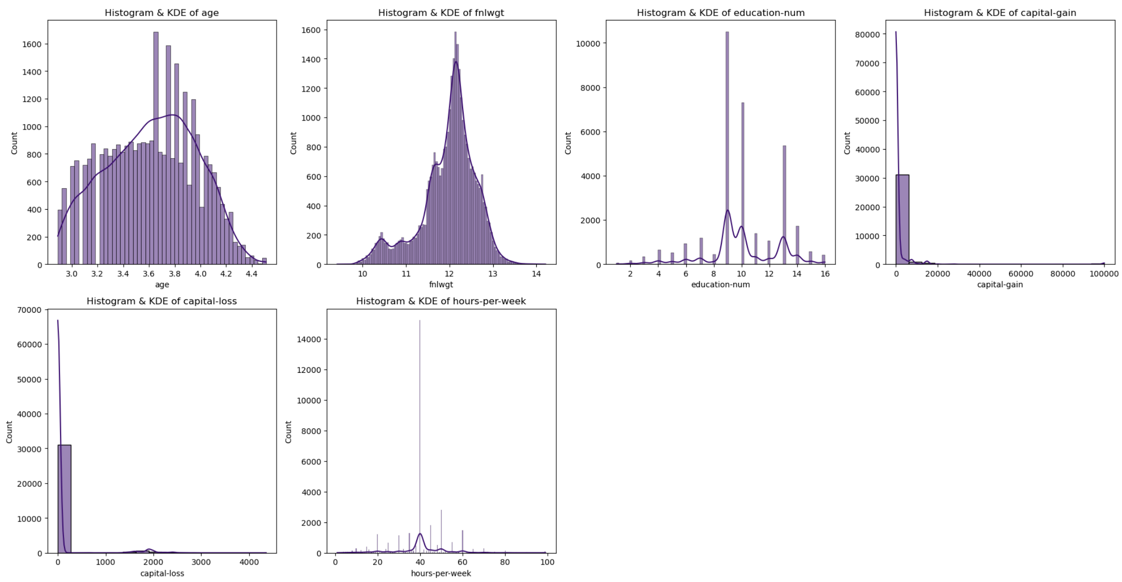
```
for ax in axes[index:]:
    ax.set_visible(False)

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



[117]:
```
#log-transforming 'age' and 'fnlwgt' to address skewness, reduce the effect of outliers, and stabilize variance
full_data['age']=np.log(1+full_data['age'])
full_data['fnlwgt']=np.log(1+full_data['fnlwgt'])
```

[121]:
```
# Create a grid of subplots: 3 rows × 4 columns (adjust if you have fewer/more features)
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(20, 15))
axes = axes.flatten()  # flatten to 1D array for easy indexing

index = 0
for col in numeric_cols:
    sns.histplot(data=full_data, x=col, kde=True, ax=axes[index], color='#3b0f70')
    axes[index].set_title(f"Histogram & KDE of {col}")
    axes[index].set_xlabel(col)
    axes[index].set_ylabel("Count")
    index += 1

# Hide any unused subplots
for ax in axes[index:]:
    ax.set_visible(False)
```

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



[56]:
```
#Label encoding for categorical columns except target variable data_type
#Loop through each column and encode if not excluded
for col in categoric_cols:
    le = LabelEncoder()
    full_data[col] = le.fit_transform(full_data[col].astype(str))
```

[57]:
```
#train_data split
x = full_data.drop(columns=['income'])
y = full_data['income']
```

[58]:
```
#Initialize the scaler
scaler = StandardScaler()
#Fit on the numerical columns and transform
x[numeric_cols] = scaler.fit_transform(x[numeric_cols])
```

[59]:
```
#Merge features (x) and target (y) into a single DataFrame
merged_data = pd.concat([x, y], axis=1)

#ptional: Check the shape or head of the combined data
print(merged_data.shape)
```

```
(32544, 16)
```

[60]:
```
#Filter rows where data_type == 'train' and save to train_df
train_df = merged_data[merged_data['data_type'] == 1]

#Filter rows where data_type == 'test' and save to test_df
test_df  = merged_data[merged_data['data_type'] == 0]

#Drop the helper column if you don't need it in the output
train_df = train_df.drop(columns='data_type')
test_df  = test_df.drop(columns='data_type')

#Write each subset back to its own CSV
train_df.to_csv(r'C:\Users\Rutika\Statistical Learning␣
↪Project\data\train_cleaned.csv', index=False)
test_df.to_csv(r'C:\Users\Rutika\Statistical Learning Project\data\test_cleaned.
↪csv',  index=False)
```

0.0.3 Model Training

Reading cleaned and preprocessed data.

[17]:
```
train_data = pd.read_csv(r"C:\Users\Rutika\Statistical Learning␣
↪Project\data\train_cleaned.csv")
test_data = pd.read_csv(r"C:\Users\Rutika\Statistical Learning␣
↪Project\data\test_cleaned.csv")
```

[18]:
```
train_data.shape
```

[18]:
```
(26033, 15)
```

[19]:
```
train_data.head()
```

[19]:
```
        age  workclass    fnlwgt  education  education-num  marital-status  \
0  0.895509          5 -1.036089          9       1.134865               2
1  0.129026          3  0.471863         11      -0.420718               0
2  1.058822          3  0.606254          1      -1.198510               2
3 -0.717464          3  1.186344          9       1.134865               2
4  0.838929          3  0.000483          6      -1.976302               3

   occupation  relationship  race  sex  capital-gain  capital-loss  \
0           3             0     4    1     -0.145959     -0.216719
1           5             1     4    1     -0.145959     -0.216719
2           5             0     2    1     -0.145959     -0.216719
3           9             5     2    0     -0.145959     -0.216719
4           7             1     2    0     -0.145959     -0.216719

   hours-per-week  native-country  income
0       -2.221654              38       0
```

```
1       -0.035378              38       0
2       -0.035378              38       0
3       -0.035378               4       0
4       -1.978734              22       0
```

[20]:
```python
test_data.shape
```

[20]:
```
(6511, 15)
```

[21]:
```python
test_data.head()
```

[21]:
```
        age  workclass    fnlwgt  education  education-num  marital-status  \
0  0.201364          6 -1.150401          9       1.134865               4
1  0.054809          3  0.911671         12       1.523761               2
2 -1.258165          3 -0.427765          9       1.134865               4
3 -0.348280          3  0.244382         11      -0.420718               4
4 -0.526913          0 -1.557817         15      -0.031822               2

   occupation  relationship  race  sex  capital-gain  capital-loss  \
0           0             1     4    1      0.148339     -0.216719
1           3             5     4    0     -0.145959     -0.216719
2           0             3     4    0     -0.145959     -0.216719
3           6             4     4    1     -0.145959     -0.216719
4           0             3     4    1     -0.145959     -0.216719

   hours-per-week  native-country  income
0       -0.035378              38       0
1       -0.035378              38       0
2       -0.845110              38       0
3       -0.035378              38       0
4       -0.035378              38       0
```

[22]:
```python
#train_data split
y_train = train_data['income']
x_train = train_data.drop(columns=['income'])
```

[23]:
```python
x_train.head()
```

[23]:
```
        age  workclass    fnlwgt  education  education-num  marital-status  \
0  0.895509          5 -1.036089          9       1.134865               2
1  0.129026          3  0.471863         11      -0.420718               0
2  1.058822          3  0.606254          1      -1.198510               2
3 -0.717464          3  1.186344          9       1.134865               2
4  0.838929          3  0.000483          6      -1.976302               3

   occupation  relationship  race  sex  capital-gain  capital-loss  \
0           3             0     4    1     -0.145959     -0.216719
```

```
1          5           1    4    1     -0.145959     -0.216719
2          5           0    2    1     -0.145959     -0.216719
3          9           5    2    0     -0.145959     -0.216719
4          7           1    2    0     -0.145959     -0.216719

   hours-per-week  native-country
0       -2.221654              38
1       -0.035378              38
2       -0.035378              38
3       -0.035378               4
4       -1.978734              22
```

[24]:
```
#test_data split
y_test = test_data['income']
x_test = test_data.drop(columns=['income'])
```

[25]:
```
x_test.head()
```

[25]:
```
        age  workclass    fnlwgt  education  education-num  marital-status  \
0  0.201364          6 -1.150401          9       1.134865               4
1  0.054809          3  0.911671         12       1.523761               2
2 -1.258165          3 -0.427765          9       1.134865               4
3 -0.348280          3  0.244382         11      -0.420718               4
4 -0.526913          0 -1.557817         15      -0.031822               2

   occupation  relationship  race  sex  capital-gain  capital-loss  \
0           0             1     4    1      0.148339     -0.216719
1           3             5     4    0     -0.145959     -0.216719
2           0             3     4    0     -0.145959     -0.216719
3           6             4     4    1     -0.145959     -0.216719
4           0             3     4    1     -0.145959     -0.216719

   hours-per-week  native-country
0       -0.035378              38
1       -0.035378              38
2       -0.845110              38
3       -0.035378              38
4       -0.035378              38
```

[26]:
```
#before lets check how many values for each class in target variable
y_train.value_counts()
```

[26]:
```
0    19761
1     6272
Name: income, dtype: int64
```

[27]:
```python
#Apply RandomOverSampler instead of SMOTE
oversample = RandomOverSampler(sampling_strategy='auto', random_state=42)
x_train, y_train = oversample.fit_resample(x_train, y_train)
```

[28]:
```python
y_train.value_counts()
```

[28]:
```
0    19761
1    19761
Name: income, dtype: int64
```

[79]:
```python
#Define a dictionary of models
models = {
    "Logistic": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
}
#Function to train, evaluate and cross-validate models, and return a DataFrame
def evaluate_models(X_train, Y_train, X_test, Y_test, model_dict):
    results = []

    for name, model in model_dict.items():
        model.fit(X_train, Y_train)
        Y_pred = model.predict(X_test)

        acc = round(accuracy_score(Y_test, Y_pred) * 100, 2)
        cv_score = round(np.mean(cross_val_score(model, X_train, Y_train,␣
 ↪cv=10)) * 100, 2)
        sensitivity = round(recall_score(Y_test, Y_pred, pos_label=1) * 100, 2)
        specificity = round(recall_score(Y_test, Y_pred, pos_label=0) * 100, 2)

        results.append({
            "Model": name,
            "Train Accuracy(%)": cv_score,
            "Test Accuracy(%)": acc,
            "Sensitivity(%)": sensitivity,
            "Specificity(%)": specificity
        })

    return pd.DataFrame(results)
#Call the function and store results in a DataFrame
results_df = evaluate_models(x_train, y_train, x_test, y_test, models)
```

[81]:
```python
#Sort by Test Accuracy
results_df = results_df.sort_values(by="Test Accuracy(%)", ascending=True).
 ↪reset_index(drop=True)
#Display results
```

```python
print(tabulate(results_df, headers='keys', tablefmt='grid'))
```

```
+----+---------------+--------------------+-------------------+--------------
---+------------------+
|    | Model         |   Train Accuracy(%) |   Test Accuracy(%) |
Sensitivity(%) |   Specificity(%) |
+====+===============+====================+===================+==============
===+==================+
|  0 | Logistic      |               77.3  |              76.98 |
78.44 |            76.51 |
+----+---------------+--------------------+-------------------+--------------
---+------------------+
|  1 | Decision Tree |               92.73 |              81.83 |
62.95 |            87.82 |
+----+---------------+--------------------+-------------------+--------------
---+------------------+
|  2 | XGBoost       |               88.1  |              83.27 |
81.51 |            83.84 |
+----+---------------+--------------------+-------------------+--------------
---+------------------+
|  3 | Random Forest |               94.06 |              85.32 |
69.32 |            90.39 |
+----+---------------+--------------------+-------------------+--------------
---+------------------+
```

0.0.4 Feature Selection and Modeling

Since we are not getting very good test accuracy for our models, now we will try doing Forward Feature Selection based on StepAIC, which will help us selecting the best subset of features for a classification model.

[29]:
```python
#function for Forward Feature Selection based on the Akaike Information␣
↪Criterion(AIC).
def forward_selection_aic(X, y):
    remaining_features = list(X.columns)
    selected_features = []
    current_score, best_new_score = float('inf'), float('inf')

    while remaining_features:
        scores_with_candidates = []
        for candidate in remaining_features:
            model = sm.Logit(y, sm.add_constant(X[selected_features +␣
↪[candidate]])).fit(disp=0)
            aic = model.aic
            scores_with_candidates.append((aic, candidate))

        scores_with_candidates.sort()
        best_new_score, best_candidate = scores_with_candidates[0]
```

```
        if current_score > best_new_score:
            remaining_features.remove(best_candidate)
            selected_features.append(best_candidate)
            current_score = best_new_score
        else:
            break

    return selected_features
```

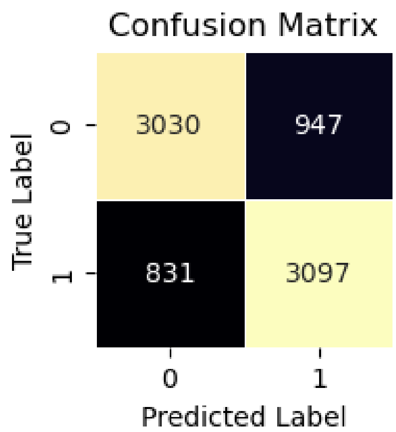
[51]:
```
models = {
    "Logistic": LogisticRegression(random_state=42),
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
    "AdaBoost": AdaBoostClassifier(random_state=42),
    "GradientBoost": GradientBoostingClassifier(random_state=42)
}

def classify(model, model_name, data, label):
    # Step 1: Feature Selection using AIC
    selected_features = forward_selection_aic(pd.DataFrame(data), label)
    print(f"\nModel: {model_name}")
    print("\nSelected Features:", selected_features)

    X_selected = pd.DataFrame(data)[selected_features]

    # Step 2: Split the selected data
    x_train, x_test, y_train, y_test = train_test_split(X_selected, label,␣
 ↪test_size=0.2, random_state=42)

    # Step 3: Train model
    model.fit(x_train, y_train)

    # Train Accuracy
    train_accuracy = accuracy_score(y_train, model.predict(x_train)) * 100
    print(f"\nTrain Accuracy: {train_accuracy:.2f}%")

    # Step 4: Cross-validation
    score = cross_val_score(model, X_selected, label, cv=10)
    print(f"\nCV Score: {np.mean(score) * 100:.2f}%")

    # Test Accuracy
    test_accuracy = accuracy_score(y_test, model.predict(x_test)) * 100
    print(f"\nTest Accuracy: {test_accuracy:.2f}%")
```

```
    # Confusion Matrix
    y_pred = model.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    #print(f"\nConfusion Matrix:\n{cm}")

    # Plot confusion matrix
    plt.figure(figsize=(2, 2))
    sns.heatmap(cm, annot=True, fmt='d', cmap='magma', cbar=False, linewidths=0.
↪5)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    # Sensitivity (Recall for Positive Class)
    y_pred = model.predict(x_test)
    sensitivity = recall_score(y_test, y_pred, pos_label=1) * 100
    print(f"\nSensitivity: {sensitivity:.2f}%")

    # Specificity (Recall for Negative Class)
    specificity = recall_score(y_test, y_pred, pos_label=0) * 100
    print(f"\nSpecificity: {specificity:.2f}%\n")

    # AUC-ROC Score
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(x_test)[:, 1]
    else:
        y_proba = model.decision_function(x_test)  # for models like SVM
    auc_score = roc_auc_score(y_test, y_proba)
    print(f"\nAUC ROC Score: {auc_score:.4f}\n")
    fpr, tpr, _ = roc_curve(y_test, y_proba)
  ␣
↪print("-------------------------------------------------------------------------------------
  ␣
↪print("-------------------------------------------------------------------------------------

    # Return all metrics except Confusion Matrix
    results = {
        "Model": model_name,
        "Train Accuracy": train_accuracy,
        "CV Score": np.mean(score) * 100,
        "Test Accuracy": test_accuracy,
        "FPR": fpr,
        "TPR": tpr,
        "AUC": auc_score
    }
```

```
    return results
```

[52]:
```
results_list = []
roc_data = []

for model_name, model in models.items():
    result = classify(model, model_name, x_train, y_train)
    roc_data.append((model_name, result["FPR"], result["TPR"], result["AUC"]))
    results_list.append(result)
```

```
Model: Logistic

Selected Features: ['education-num', 'age', 'capital-gain', 'sex', 'hours-per-week', 'marital-status', 'capital-loss', 'race', 'workclass', 'relationship', 'education', 'fnlwgt']

Train Accuracy: 77.40%

CV Score: 77.40%

Test Accuracy: 77.51%
```




```
Sensitivity: 78.84%

Specificity: 76.19%


AUC ROC Score: 0.8575

--------------------------------------------------------------------------------
```

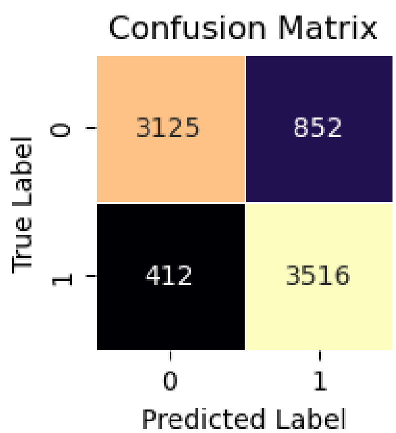

Model: KNN

Selected Features: ['education-num', 'age', 'capital-gain', 'sex', 'hours-per-week', 'marital-status', 'capital-loss', 'race', 'workclass', 'relationship', 'education', 'fnlwgt']

Train Accuracy: 88.94%

CV Score: 84.73%

Test Accuracy: 84.01%



Sensitivity: 89.51%

Specificity: 78.58%

AUC ROC Score: 0.9112

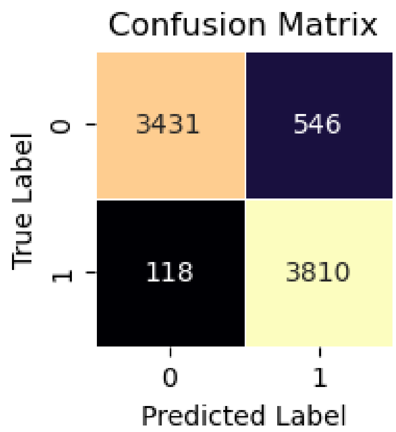
Model: Decision Tree

Selected Features: ['education-num', 'age', 'capital-gain', 'sex', 'hours-per-week', 'marital-status', 'capital-loss', 'race', 'workclass', 'relationship', 'education', 'fnlwgt']

Train Accuracy: 99.98%

CV Score: 92.73%

Test Accuracy: 91.60%



Sensitivity: 97.00%

Specificity: 86.27%

AUC ROC Score: 0.9165

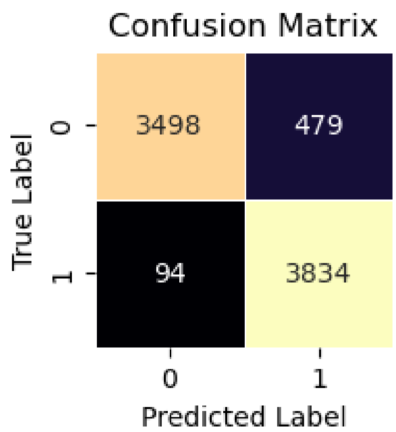
Model: Random Forest

Selected Features: ['education-num', 'age', 'capital-gain', 'sex', 'hours-per-week', 'marital-status', 'capital-loss', 'race', 'workclass', 'relationship', 'education', 'fnlwgt']

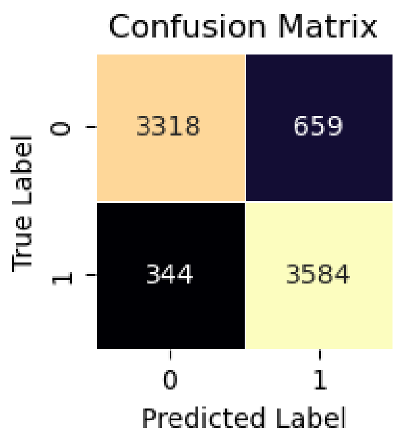
Train Accuracy: 99.98%

CV Score: 93.61%

```
Test Accuracy: 92.75%
```




```
Sensitivity: 97.61%

Specificity: 87.96%

AUC ROC Score: 0.9844

--------------------------------------------------------------------------------
-----------------------------
--------------------------------------------------------------------------------
-----------------------------

Model: XGBoost

Selected Features: ['education-num', 'age', 'capital-gain', 'sex', 'hours-per-
week', 'marital-status', 'capital-loss', 'race', 'workclass', 'relationship',
'education', 'fnlwgt']

Train Accuracy: 89.58%

CV Score: 87.35%

Test Accuracy: 87.31%
```



Sensitivity: 91.24%

Specificity: 83.43%

AUC ROC Score: 0.9478

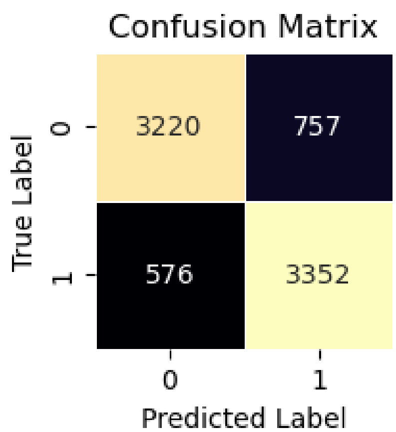
Model: AdaBoost

Selected Features: ['education-num', 'age', 'capital-gain', 'sex', 'hours-per-week', 'marital-status', 'capital-loss', 'race', 'workclass', 'relationship', 'education', 'fnlwgt']

Train Accuracy: 82.93%

CV Score: 82.83%

Test Accuracy: 83.14%



Sensitivity: 85.34%

Specificity: 80.97%

AUC ROC Score: 0.9148

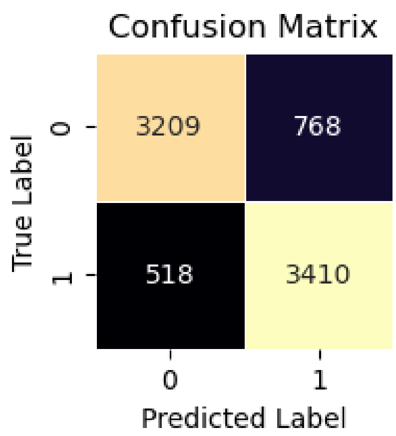
Model: GradientBoost

Selected Features: ['education-num', 'age', 'capital-gain', 'sex', 'hours-per-week', 'marital-status', 'capital-loss', 'race', 'workclass', 'relationship', 'education', 'fnlwgt']

Train Accuracy: 83.86%

CV Score: 83.58%

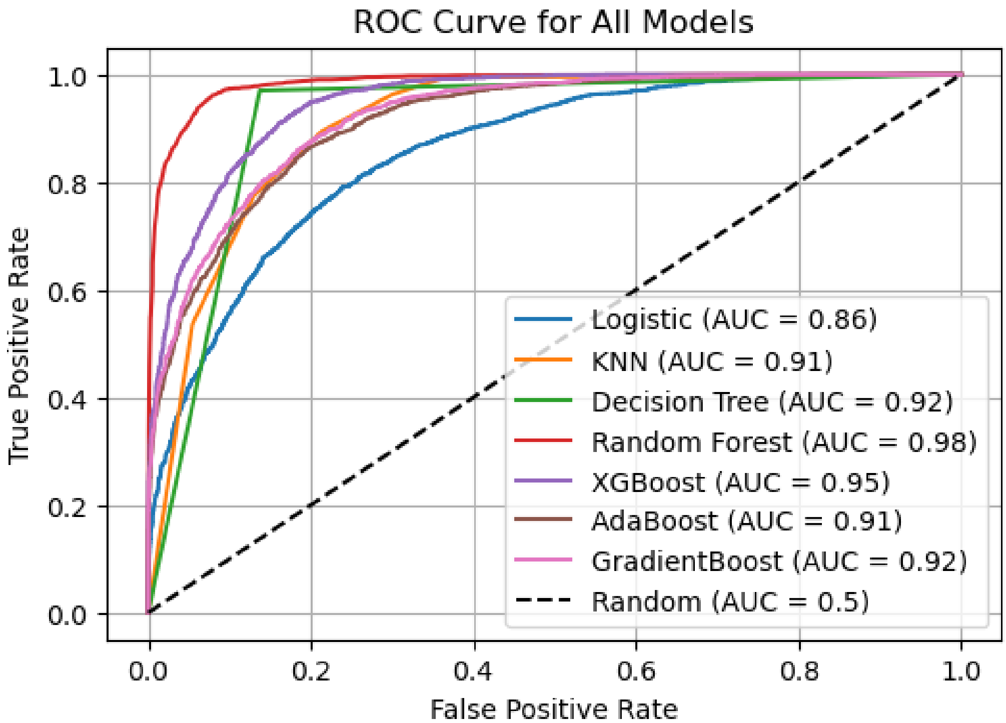
Test Accuracy: 83.73%




```
Sensitivity: 86.81%

Specificity: 80.69%


AUC ROC Score: 0.9220

--------------------------------------------------------------------------------
-----------------------------
--------------------------------------------------------------------------------
-----------------------------
```

[50]:

```
import matplotlib.pyplot as plt

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

for name, fpr, tpr, auc in roc_data:
    plt.plot(fpr, tpr, label=f'{name} (AUC = {auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random (AUC = 0.5)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for All Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



[36]:

```
#Convert the results to a DataFrame
results_df = pd.DataFrame(results_list)

#Reorganize columns for better display
results_df = results_df[["Model", "Train Accuracy", "CV Score", "Test␣
 ↪Accuracy"]]

#Sort by Test Accuracy in descending order
results_df = results_df.sort_values(by="Test Accuracy", ascending=True)

#Display the results as a table
print(tabulate(results_df, headers='keys', tablefmt='grid'))
```

```
+----+---------------+------------------+------------+-----------------+
|    | Model         |   Train Accuracy |   CV Score |   Test Accuracy |
+====+===============+==================+============+=================+
|  0 | Logistic      |          77.4014 |    77.3974 |         77.5079 |
+----+---------------+------------------+------------+-----------------+
|  5 | AdaBoost      |          82.9301 |    82.8323 |         83.1373 |
+----+---------------+------------------+------------+-----------------+
|  6 | GradientBoost |          83.8568 |    83.5813 |         83.7318 |
+----+---------------+------------------+------------+-----------------+
|  1 | KNN           |          88.9363 |    84.7301 |         84.0101 |
```

```
+----+---------------+------------------+------------+-----------------+
|  4 | XGBoost       |          89.5784 |    87.3539 |         87.3118 |
+----+---------------+------------------+------------+-----------------+
|  2 | Decision Tree |          99.9842 |    92.7332 |         91.6003 |
+----+---------------+------------------+------------+-----------------+
|  3 | Random Forest |          99.9842 |    93.6062 |         92.7514 |
+----+---------------+------------------+------------+-----------------+
```

0.0.5 Hyperparameter Tuning

We are tryin to tune 2 models below - 1. Random Forest and 2. XGBoost

Trying Hyperparameter Tuning on Random Forest

[27]:
```
selected_features = forward_selection_aic(x_train, y_train)
X_train_selected = x_train[selected_features]
X_test_selected = x_test[selected_features]
```

[39]:
```
# Define hyperparameter space
param_dist = {
    'n_estimators': [int(x) for x in np.linspace(40, 300, num=10)],
    'max_depth': [int(x) for x in np.linspace(40, 300, num=10)],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False],
    'max_features': ['sqrt', 'log2']
}
```

[40]:
```
rf_tuned = RandomForestClassifier(random_state=42)
```

[41]:
```
rf_cv = RandomizedSearchCV(
    estimator=rf_tuned, param_distributions=param_dist, cv=10, random_state=42)
```

[42]:
```
rf_cv.fit(X_train_selected, y_train)
```

[42]:
```
RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(random_state=42),
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [40, 68, 97, 126, 155, 184,
                                                      213, 242, 271, 300],
                                        'max_features': ['sqrt', 'log2'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [40, 68, 97, 126, 155,
                                                         184, 213, 242, 271,
                                                         300]},
                   random_state=42)
```

[43]:
```
rf_cv.best_score_
```

[43]: 0.9452717264395105

[44]:
```
rf_cv.best_params_
```

[44]:
```
{'n_estimators': 300,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 97,
 'bootstrap': False}
```

[45]:
```
rf_best = RandomForestClassifier(
    max_depth=97, n_estimators=300, min_samples_split=2,␣
 ↪min_samples_leaf=1,max_features='sqrt',bootstrap=False)
```

[46]:
```
rf_best.fit(X_train_selected, y_train)
```

[46]:
```
RandomForestClassifier(bootstrap=False, max_depth=97, max_features='sqrt',
                       n_estimators=300)
```

[47]:
```
Y_pred_rf_best = rf_best.predict(X_test_selected)
```

[57]:
```
print('Random Forest Classifier:')
print("Accuracy:", accuracy_score(y_test, Y_pred_xgb) * 100)
print("Confusion Matrix:\n", confusion_matrix(y_test, Y_pred_rf_best))
print("Sensitivity:", recall_score(y_test, Y_pred_rf_best, pos_label=1) * 100)
print("Specificity:", recall_score(y_test, Y_pred_rf_best, pos_label=0) * 100)
```

```
Random Forest Classifier:
Accuracy: 80.67885117493474
Confusion Matrix:
 [[4489  454]
 [ 578  990]]
Sensitivity: 63.13775510204081
Specificity: 90.81529435565446
```

Trying Hyperparameter Tuning on XGBoost

[54]:
```
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [3, 5, 7, 9, 12],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.5, 0.6, 0.7, 0.8, 1.0],
    'colsample_bytree': [0.5, 0.7, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2, 0.3],
    'min_child_weight': [1, 3, 5, 7],
    'scale_pos_weight': [1, 2, 5, 10],  # For imbalanced classes
    'reg_alpha': [0, 0.01, 0.1, 1],
```

```
    'reg_lambda': [0.1, 1, 10]
}
```

[55]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV

xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=123)

xgb_cv = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_dist,
    n_iter=12,                # try more for deeper search
    cv=10,
    verbose=1,
    random_state=123,
    n_jobs=-1
)


xgb_cv.fit(X_train_selected, y_train)

print("Best Parameters:", xgb_cv.best_params_)
```

```
Fitting 10 folds for each of 12 candidates, totalling 120 fits
Best Parameters: {'subsample': 0.8, 'scale_pos_weight': 2, 'reg_lambda': 0.1,
'reg_alpha': 0.1, 'n_estimators': 300, 'min_child_weight': 1, 'max_depth': 7,
'learning_rate': 0.1, 'gamma': 0.3, 'colsample_bytree': 0.9}
```

[56]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score

best_xgb = xgb_cv.best_estimator_
Y_pred_xgb = best_xgb.predict(X_test_selected)

print("Accuracy:", accuracy_score(y_test, Y_pred_xgb) * 100)
print("Confusion Matrix:\n", confusion_matrix(y_test, Y_pred_xgb))
print("Sensitivity:", recall_score(y_test, Y_pred_xgb, pos_label=1) * 100)
print("Specificity:", recall_score(y_test, Y_pred_xgb, pos_label=0) * 100)
```

```
Accuracy: 80.67885117493474
Confusion Matrix:
 [[3917 1026]
 [ 232 1336]]
Sensitivity: 85.20408163265306
Specificity: 79.24337446894599
```

[ ]:
```
```