Analyzing Income Inequality Syed Faizan

In this project we classify the US population, based on the given census data, into low and high income categories using a kNN Classifier.

```
# Import necessary libraries for EDA
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv(r'C:\Users\sfaiz\OneDrive\Desktop\ALY 6020 Project Module 1\adult.csv')


# Display the first few rows and basic info to understand structure
data.head()
```

```
   39         State-gov   77516  Bachelors  13       Never-married  \
0  50  Self-emp-not-inc   83311  Bachelors  13  Married-civ-spouse
1  38           Private  215646    HS-grad   9            Divorced
2  53           Private  234721       11th   7  Married-civ-spouse
3  28           Private  338409  Bachelors  13  Married-civ-spouse
4  37           Private  284582    Masters  14  Married-civ-spouse

        Adm-clerical  Not-in-family  White    Male  2174  0  40  \
0    Exec-managerial        Husband  White    Male     0  0  13
1  Handlers-cleaners  Not-in-family  White    Male     0  0  40
2  Handlers-cleaners        Husband  Black    Male     0  0  40
3     Prof-specialty           Wife  Black  Female     0  0  40
4    Exec-managerial           Wife  White  Female     0  0  40

   United-States  <=50K
0  United-States  <=50K
1  United-States  <=50K
2  United-States  <=50K
3           Cuba  <=50K
4  United-States  <=50K
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48841 entries, 0 to 48840
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   39              48841 non-null  int64
 1   State-gov       48841 non-null  object
```

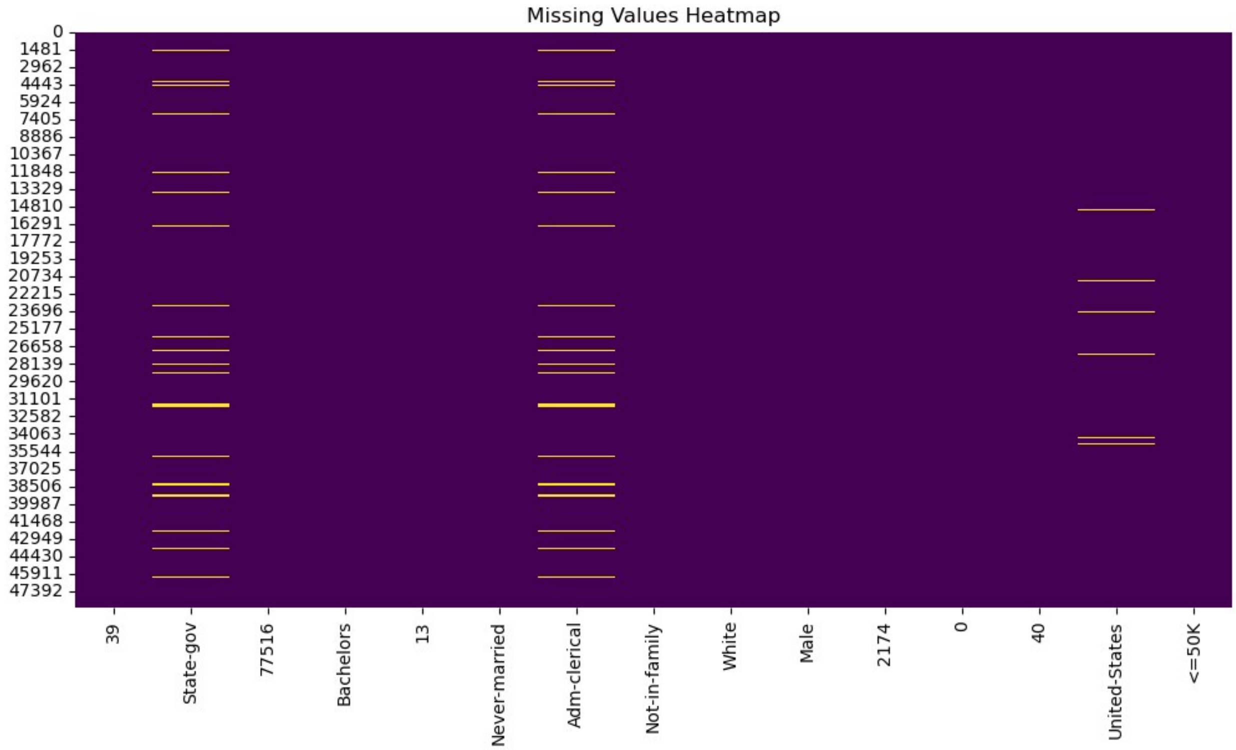
```
 2   77516           48841 non-null  int64
 3   Bachelors       48841 non-null  object
 4   13              48841 non-null  int64
 5   Never-married   48841 non-null  object
 6   Adm-clerical    48841 non-null  object
 7   Not-in-family   48841 non-null  object
 8   White           48841 non-null  object
 9   Male            48841 non-null  object
 10  2174            48841 non-null  int64
 11  0               48841 non-null  int64
 12  40              48841 non-null  int64
 13  United-States   48841 non-null  object
 14  <=50K           48841 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

```
data.describe()
```

```
                age        fnlwgt  education_num  capital_gain
capital_loss  \
count  48841.000000  4.884100e+04   48841.000000  48841.000000
48841.000000
mean      38.643578  1.896664e+05      10.078029   1079.045208
87.504105
std       13.710650  1.056039e+05       2.570965   7452.093700
403.008483
min       17.000000  1.228500e+04       1.000000      0.000000
0.000000
25%       28.000000  1.175550e+05       9.000000      0.000000
0.000000
50%       37.000000  1.781470e+05      10.000000      0.000000
0.000000
75%       48.000000  2.376460e+05      12.000000      0.000000
0.000000
max       90.000000  1.490400e+06      16.000000  99999.000000
4356.000000

       hours_per_week        income
count    48841.000000  48841.000000
mean        40.422391      0.239287
std         12.391571      0.426652
min          1.000000      0.000000
25%         40.000000      0.000000
50%         40.000000      0.000000
75%         45.000000      0.000000
max         99.000000      1.000000
```

```
data.describe(include=[object])
```

```
       workclass education      marital_status      occupation
relationship  \
count      48841     48841               48841           48841
48841
unique         8        16                   7              14
6
top      Private   HS-grad  Married-civ-spouse  Prof-specialty
Husband
freq       36705     15784               22379            8981
19716

         race    sex native_country
count   48841  48841          48841
unique      5      2             41
top     White   Male  United-States
freq    41761  32649          44688

# Re-load the dataset with explicit missing values handling to ensure
consistent `NaN` recognition
data = pd.read_csv(r'C:\Users\sfaiz\OneDrive\Desktop\ALY 6020 Project
Module 1\adult.csv', na_values=["NaN", "?", ""])

# Analyze missing values by checking the count of NaNs in each column
missing_values = data.isnull().sum()
missing_values_summary = missing_values[missing_values > 0]

# Visualize missing values with a heatmap to observe the distribution
across the dataset
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.heatmap(data.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()

# Display the summary of missing values for columns with NaNs
missing_values_summary
```




```
State-gov        2799
Adm-clerical     2809
United-States     857
dtype: int64
```

```python
# Impute missing values in categorical columns with the mode (most frequent value)
for column in ['State-gov', 'Adm-clerical', 'United-States']:
    mode_value = data[column].mode()[0]  # Calculate the mode
    data[column].fillna(mode_value, inplace=True)  # Fill missing values with the mode

# Verify that missing values are now handled
missing_values_post_imputation = data.isnull().sum()
missing_values_post_imputation_summary = missing_values_post_imputation[missing_values_post_imputation > 0]
missing_values_post_imputation_summary
```

```
Series([], dtype: int64)
```

```python
# Extensive Data Cleansing: Renaming columns for clarity, handling data types


# Step 1: Renaming columns for improved readability
# Mapping ambiguous column names to more descriptive names
data.columns = [
```

```
    'age',                   # '39'
    'workclass',             # 'State-gov'
    'fnlwgt',                # '77516'
    'education',             # 'Bachelors'
    'education_num',         # '13'
    'marital_status',        # 'Never-married'
    'occupation',            # 'Adm-clerical'
    'relationship',          # 'Not-in-family'
    'race',                  # 'White'
    'sex',                   # 'Male'
    'capital_gain',          # '2174'
    'capital_loss',          # '0'
    'hours_per_week',        # '40'
    'native_country',        # 'United-States'
    'income'                 # '<=50K'
]

# Step 2: Verify the new column names and display the first few rows
to confirm changes
renamed_data_head = data.head()

# Step 3: Check data types and convert them if necessary
# For example, if 'education_num' should be a categorical ordinal
feature, we may convert it accordingly
data['education_num'] = data['education_num'].astype(int)  # Ensure
'education_num' is integer
data['age'] = data['age'].astype(int)  # Ensure 'age' is integer



# Display cleaned data summary
cleaned_data_info = data.info()
renamed_data_head, cleaned_data_info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48841 entries, 0 to 48840
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48841 non-null  int32
 1   workclass       48841 non-null  object
 2   fnlwgt          48841 non-null  int64
 3   education       48841 non-null  object
 4   education_num   48841 non-null  int32
 5   marital_status  48841 non-null  object
 6   occupation      48841 non-null  object
 7   relationship    48841 non-null  object
 8   race            48841 non-null  object
 9   sex             48841 non-null  object
 10  capital_gain    48841 non-null  int64
```

```
 11  capital_loss    48841 non-null  int64
 12  hours_per_week  48841 non-null  int64
 13  native_country  48841 non-null  object
 14  income          48841 non-null  object
dtypes: int32(2), int64(4), object(9)
memory usage: 5.2+ MB

(   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

        marital_status         occupation   relationship   race
sex  \
 0  Married-civ-spouse    Exec-managerial        Husband  White
Male
 1            Divorced  Handlers-cleaners  Not-in-family  White
Male
 2  Married-civ-spouse  Handlers-cleaners        Husband  Black
Male
 3  Married-civ-spouse     Prof-specialty           Wife  Black
Female
 4  Married-civ-spouse    Exec-managerial           Wife  White
Female

    capital_gain  capital_loss  hours_per_week native_country income
 0             0             0              13  United-States  <=50K
 1             0             0              40  United-States  <=50K
 2             0             0              40  United-States  <=50K
 3             0             0              40           Cuba  <=50K
 4             0             0              40  United-States
<=50K  ,
 None)
```

Dataset Update Summary

The dataset has been successfully cleaned and now has more descriptive column names, improving readability and interpretation:

Updated Column Names

- **age**: Age of the individual
- **workclass**: Type of employment
- **fnlwgt**: Final weight (a sampling weight, often useful in survey data)
- **education**: Highest level of education attained
- **education_num**: Numeric representation of education level (ordinal)

- **marital_status**: Marital status
- **occupation**: Job type
- **relationship**: Relationship status in the household
- **race**: Racial background
- **sex**: Gender
- **capital_gain** and **capital_loss**: Financial gain/loss
- **hours_per_week**: Weekly working hours
- **native_country**: Country of origin
- **income**: Target variable indicating income class (<=50K or >50K)

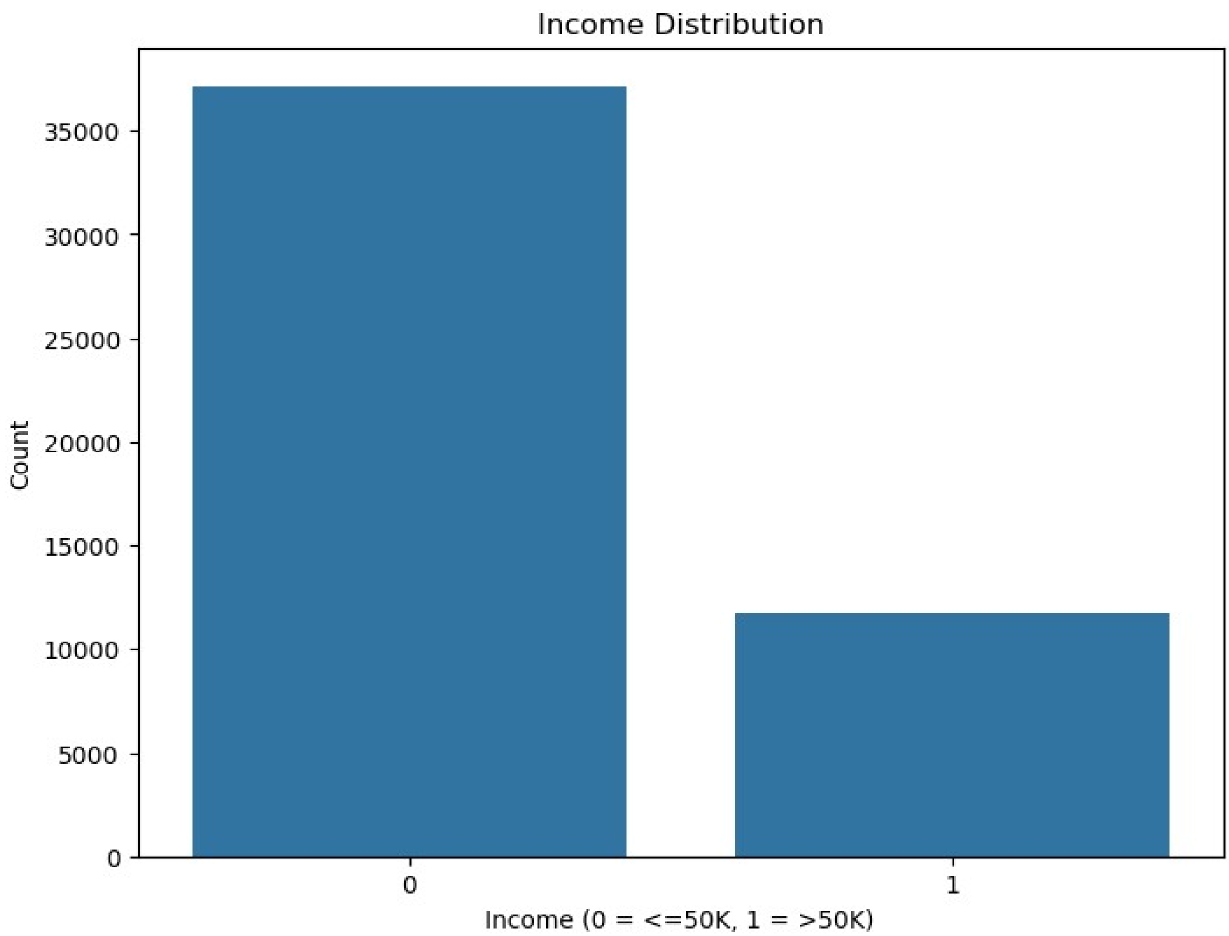
Data Types

All columns are now correctly typed, with numeric data stored as `int64` and categorical data as `object`.

```
# Step: Encode the Target Variable
# Convert the target variable 'income' to binary (0 for <=50K, 1 for
>50K)
data['income'] = data['income'].apply(lambda x: 1 if x == '>50K' else
0)

# Display the first few rows of the dataset to verify the target
encoding
data.head()
```

```
   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

       marital_status         occupation   relationship   race     sex
\
0  Married-civ-spouse    Exec-managerial        Husband  White    Male

1            Divorced  Handlers-cleaners  Not-in-family  White    Male

2  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male

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

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


   capital_gain  capital_loss  hours_per_week native_country  income
0             0             0              13  United-States       0
1             0             0              40  United-States       0
2             0             0              40  United-States       0
```

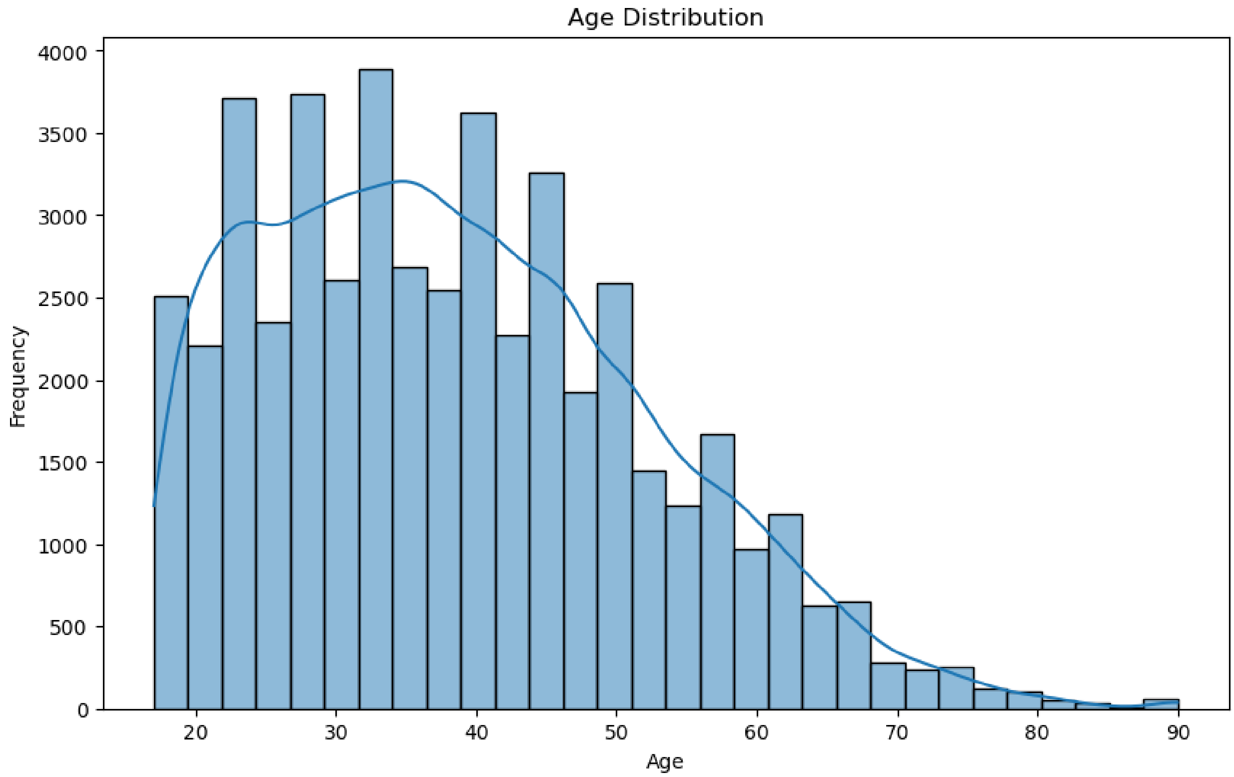
```
3                0                0                40          Cuba         0
4                0                0                40   United-States       0

# Plot distribution of the target variable 'income'
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='income')
plt.title('Income Distribution')
plt.xlabel('Income (0 = <=50K, 1 = >50K)')
plt.ylabel('Count')
plt.show()
```



This plot above shows the distribution of the income target variable, revealing class balance between <=50K (0) and >50K (1). Understanding the class balance is essential for classification tasks.

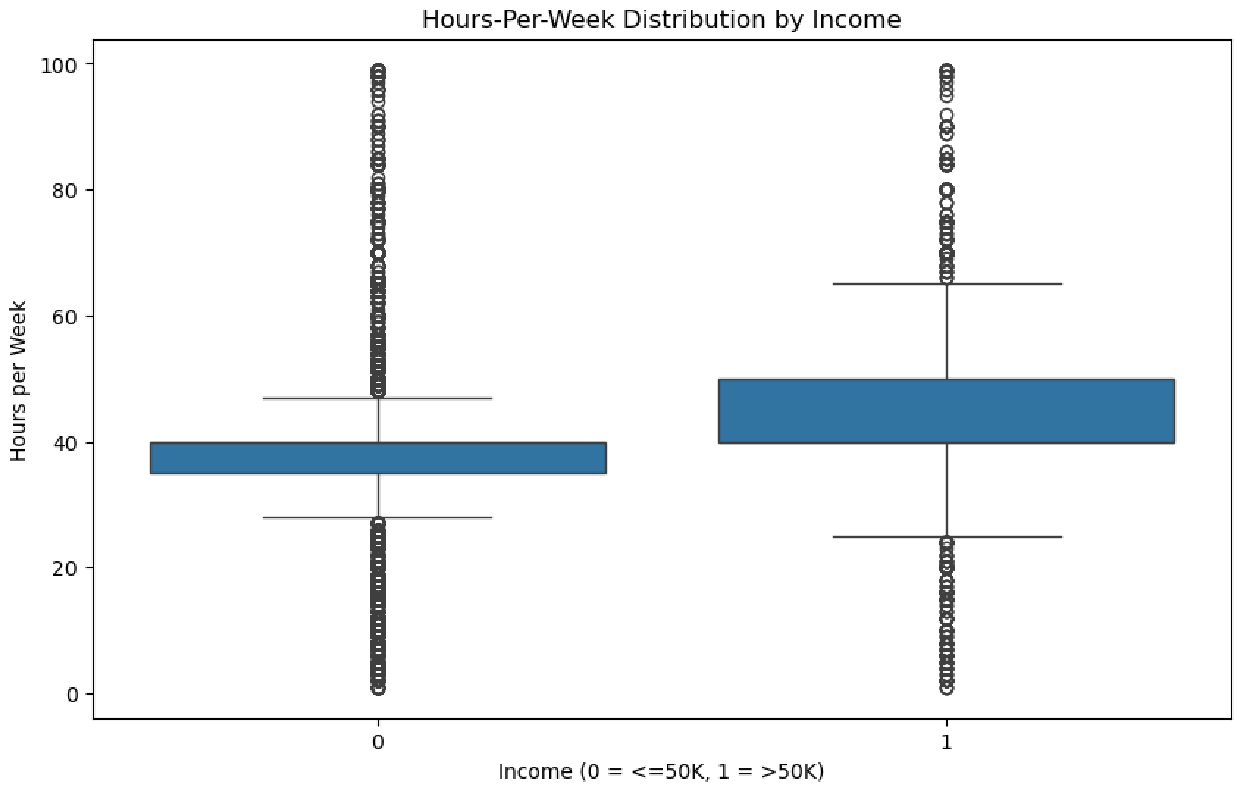
```
# Plot distribution of age
plt.figure(figsize=(10, 6))
sns.histplot(data['age'], kde=True, bins=30)
plt.title('Age Distribution')
plt.xlabel('Age')
```

```
plt.ylabel('Frequency')
plt.show()
```



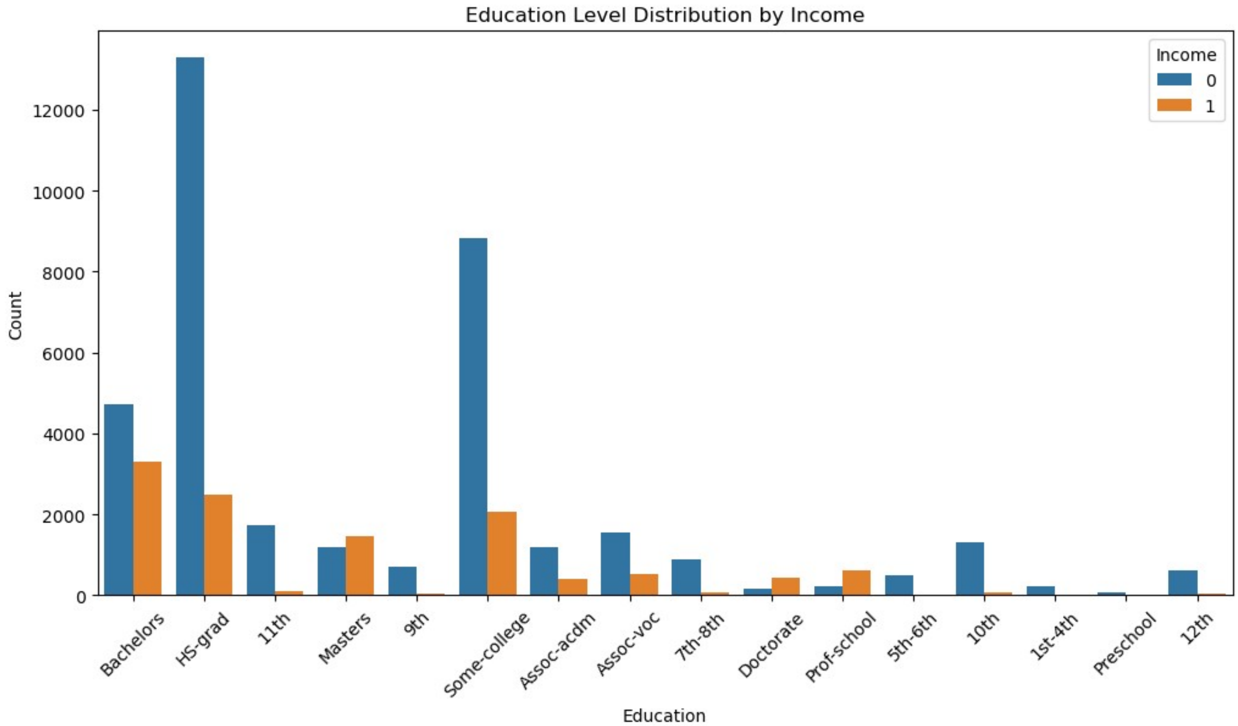
This histogram visualizes the age distribution in the dataset, helping us identify the most common age ranges and any potential outliers.

```
# Box plot for hours worked per week by income
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='income', y='hours_per_week')
plt.title('Hours-Per-Week Distribution by Income')
plt.xlabel('Income (0 = <=50K, 1 = >50K)')
plt.ylabel('Hours per Week')
plt.show()
```



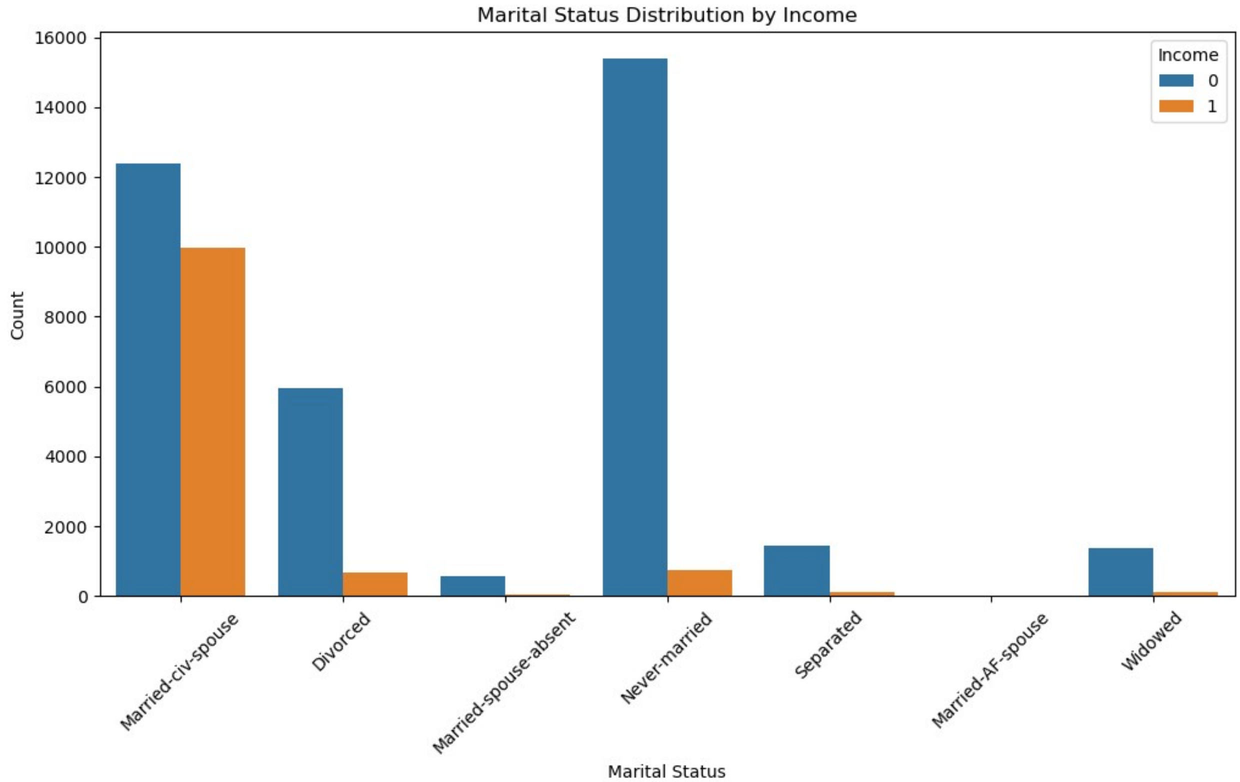
This box plot compares weekly working hours across income classes. It helps detect any differences in working hours associated with income level.

```
# Education level distribution by income class
plt.figure(figsize=(12, 6))
sns.countplot(data=data, x='education', hue='income')
plt.title('Education Level Distribution by Income')
plt.xlabel('Education')
plt.ylabel('Count')
plt.legend(title='Income')
plt.xticks(rotation=45)
plt.show()
```



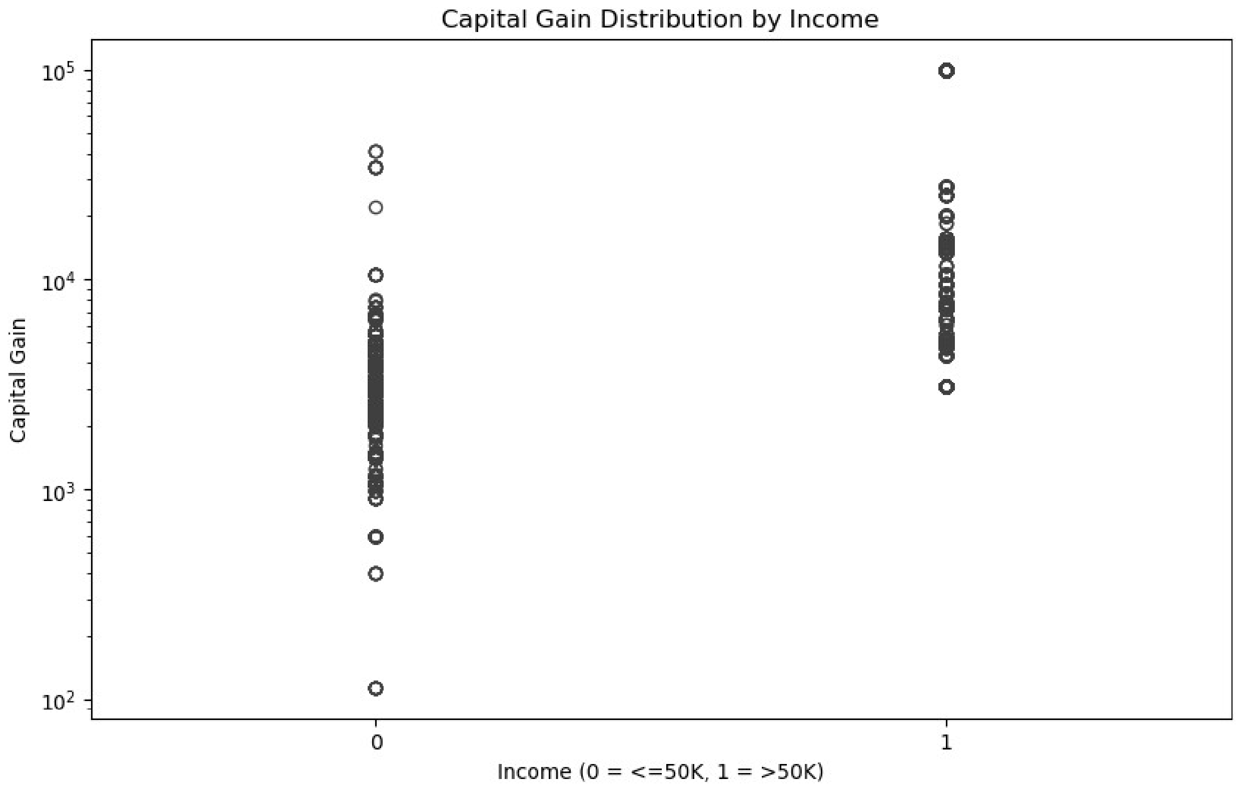
This plot shows the distribution of education levels across income classes, highlighting any education levels that correlate with higher income.

```python
# Marital status distribution by income class
plt.figure(figsize=(12, 6))
sns.countplot(data=data, x='marital_status', hue='income')
plt.title('Marital Status Distribution by Income')
plt.xlabel('Marital Status')
plt.ylabel('Count')
plt.legend(title='Income')
plt.xticks(rotation=45)
plt.show()
```



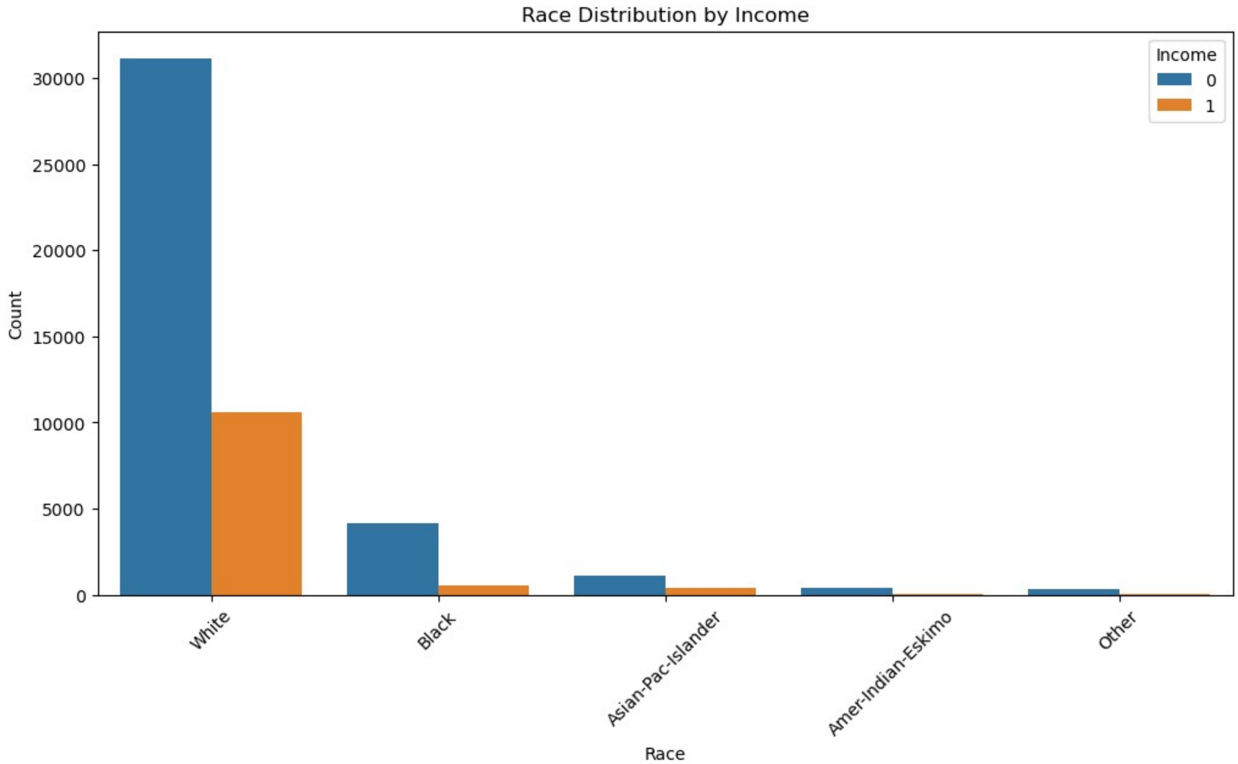
This plot shows marital status across income classes, helping understand if certain marital statuses are associated with higher or lower income levels.

```
# Capital gain distribution by income class (using log scale for
skewness)
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='income', y='capital_gain')
plt.title('Capital Gain Distribution by Income')
plt.xlabel('Income (0 = <=50K, 1 = >50K)')
plt.ylabel('Capital Gain')
plt.yscale('log')  # Log scale to handle skewness
plt.show()
```



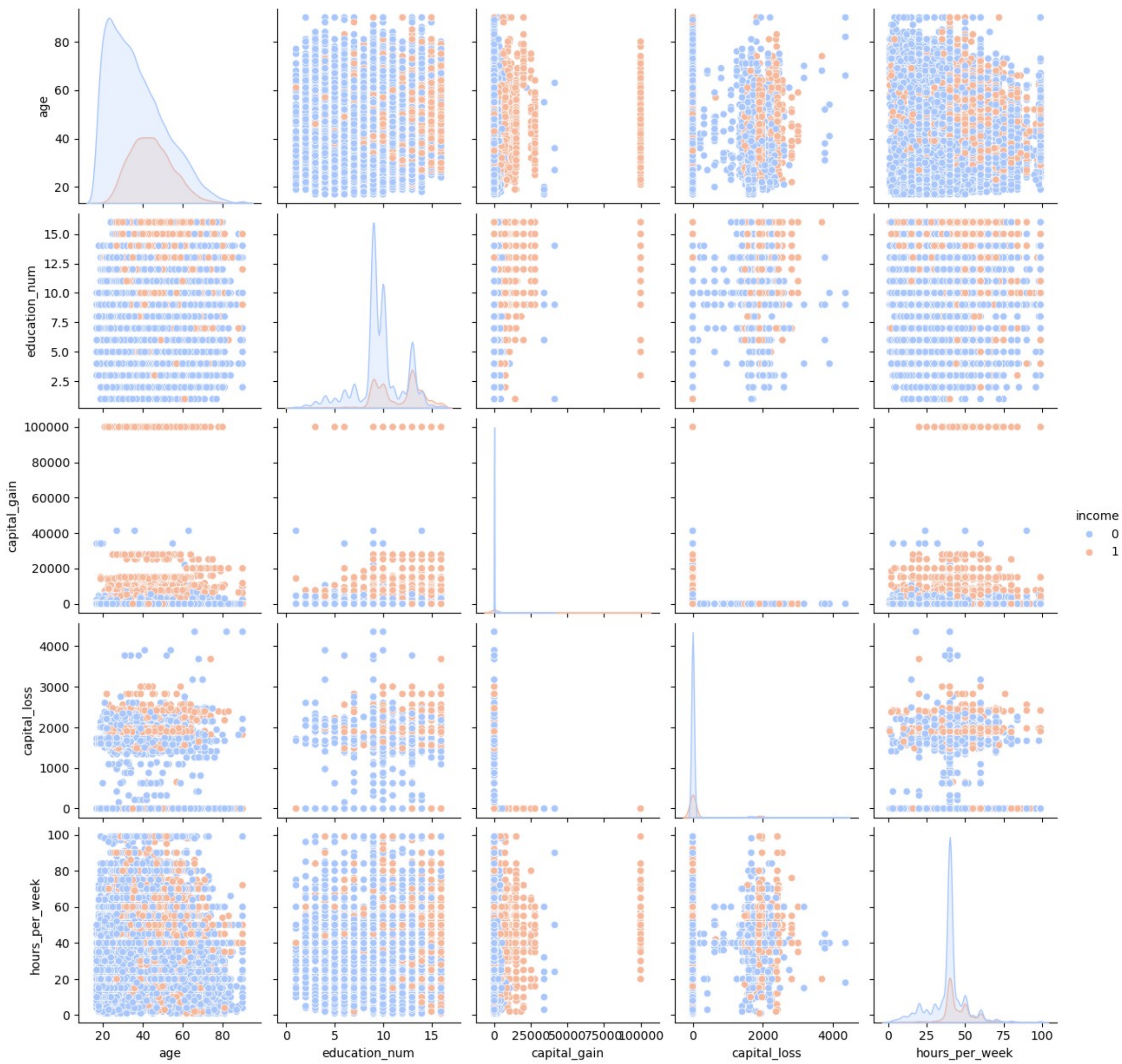
This box plot displays capital gain distribution across income levels, with a log scale to manage skewness. High capital gain values are often associated with higher income levels.

```
# Race distribution by income class
plt.figure(figsize=(12, 6))
sns.countplot(data=data, x='race', hue='income')
plt.title('Race Distribution by Income')
plt.xlabel('Race')
plt.ylabel('Count')
plt.legend(title='Income')
plt.xticks(rotation=45)
plt.show()
```



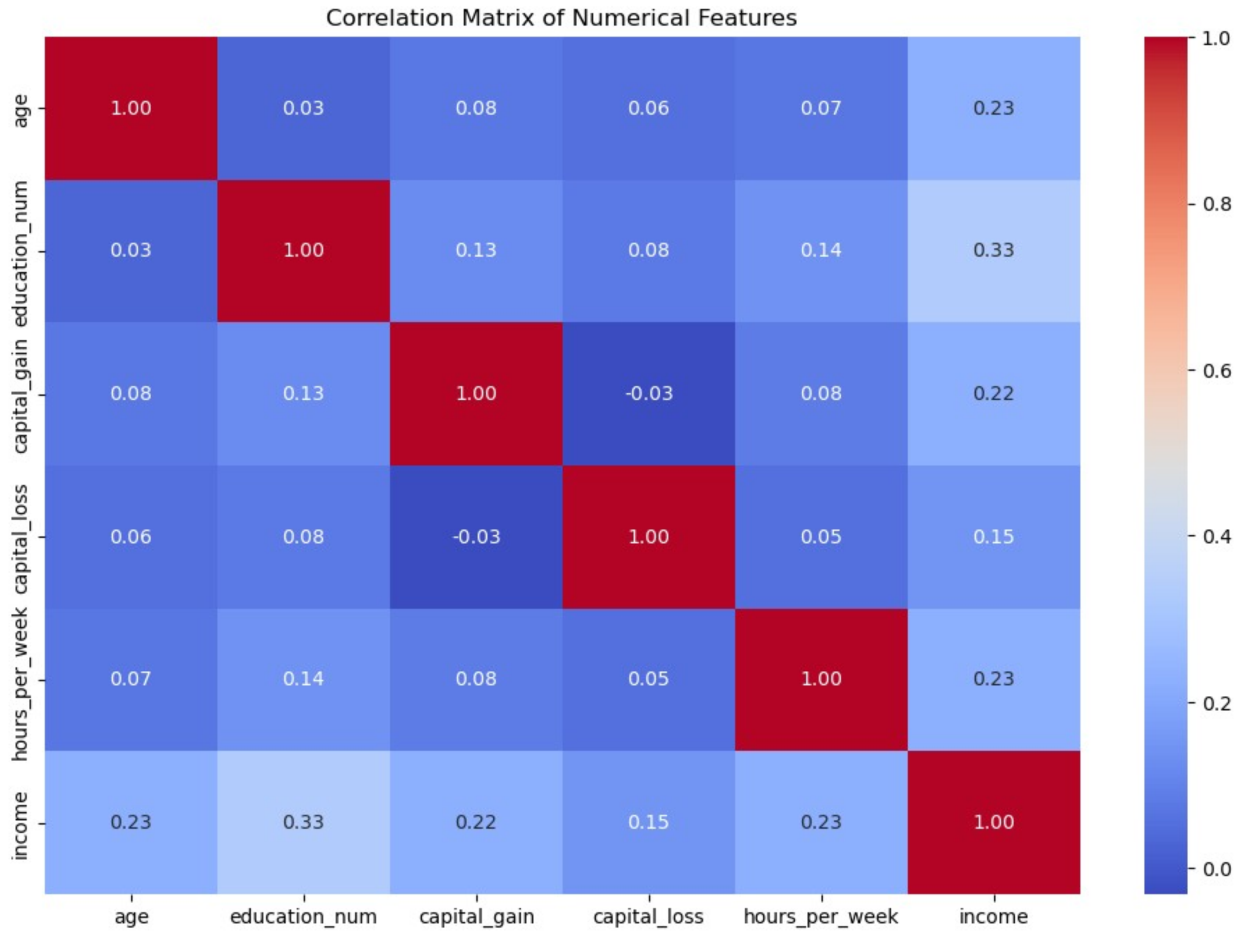
This count plot reveals the racial demographics of income classes, highlighting any demographic differences in income distribution.

```
# Pair plot for key numerical features to examine relationships
numeric_columns = ['age', 'education_num', 'capital_gain',
'capital_loss', 'hours_per_week', 'income']
sns.pairplot(data[numeric_columns], hue='income', palette='coolwarm')
plt.show()
```



This pair plot visualizes relationships between numerical features, colored by income class. It helps identify feature clusters and any separation that may assist in classification.

```
# Correlation matrix for numerical features
correlations = data[numeric_columns].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



This correlation matrix shows relationships between numerical features. It's useful for identifying highly correlated features and understanding their influence on the target variable.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Select only numerical columns ('income' is the target variable)
numerical_features = ['age', 'fnlwgt', 'education_num',
'capital_gain', 'capital_loss', 'hours_per_week']
X = data[numerical_features]
y = data['income']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Scale the numerical features for KNN
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

This cell selects only numerical columns for the feature set (X), and splits it into training and testing sets. It then scales the features to standardize them for KNN.

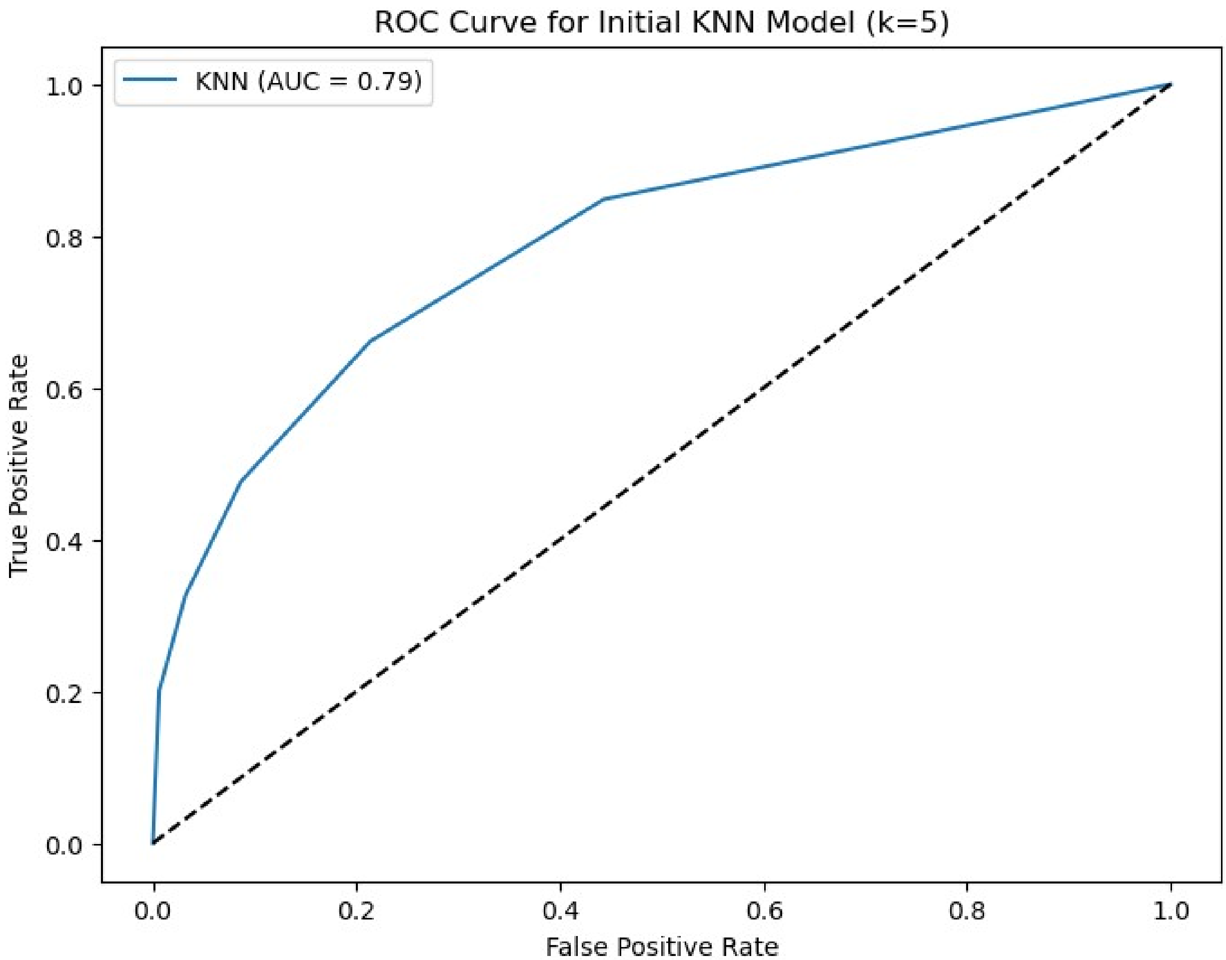
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize and train the KNN model with k=5
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict on the test set and evaluate
y_pred = knn.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[10198   968]
 [ 1825  1662]]
Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.91      0.88     11166
           1       0.63      0.48      0.54      3487

    accuracy                           0.81     14653
   macro avg       0.74      0.69      0.71     14653
weighted avg       0.80      0.81      0.80     14653
```

This cell initializes a KNN classifier with k=5, trains it on the training data, and evaluates its performance on the test set. It generates a confusion matrix, classification report and the below cell gives the ROC curve, and calculates AUC.

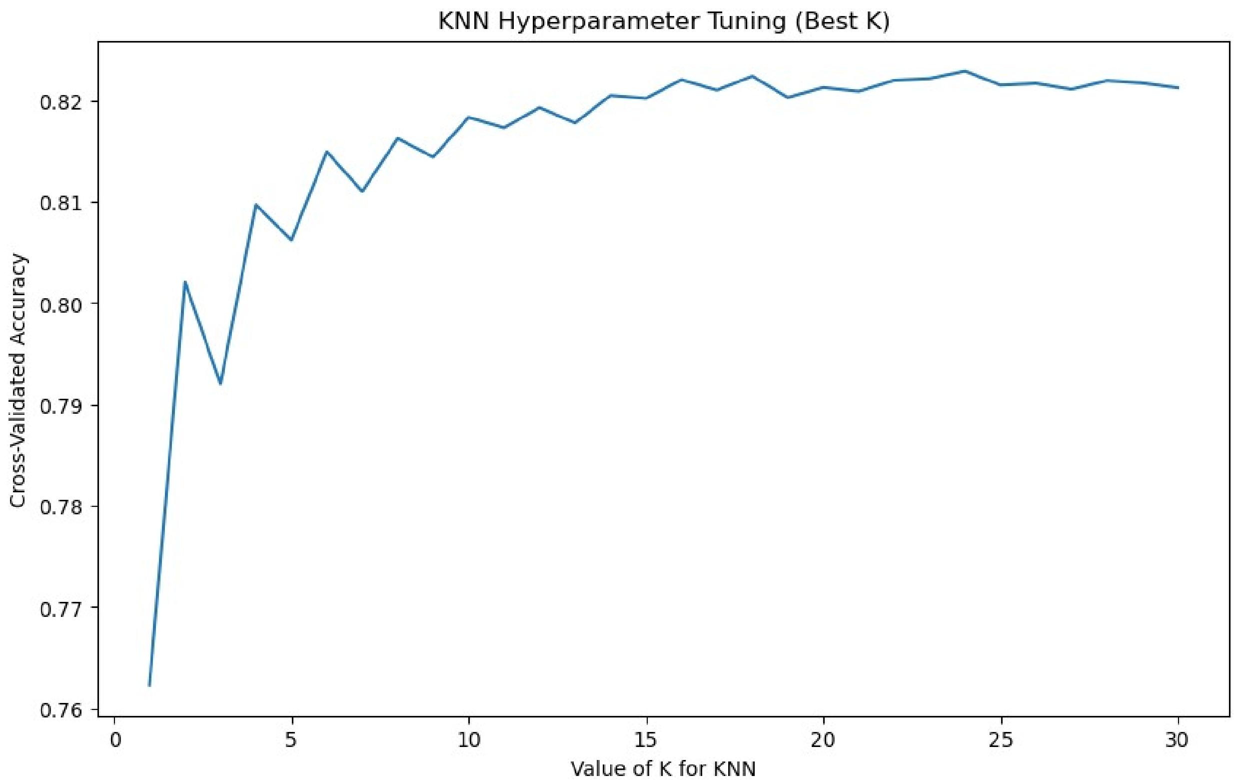
```
# Calculate ROC and AUC
y_pred_proba = knn.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"AUC Score: {roc_auc}")

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'KNN (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('ROC Curve for Initial KNN Model (k=5)')
plt.legend()
plt.show()

AUC Score: 0.7894146735031439
```



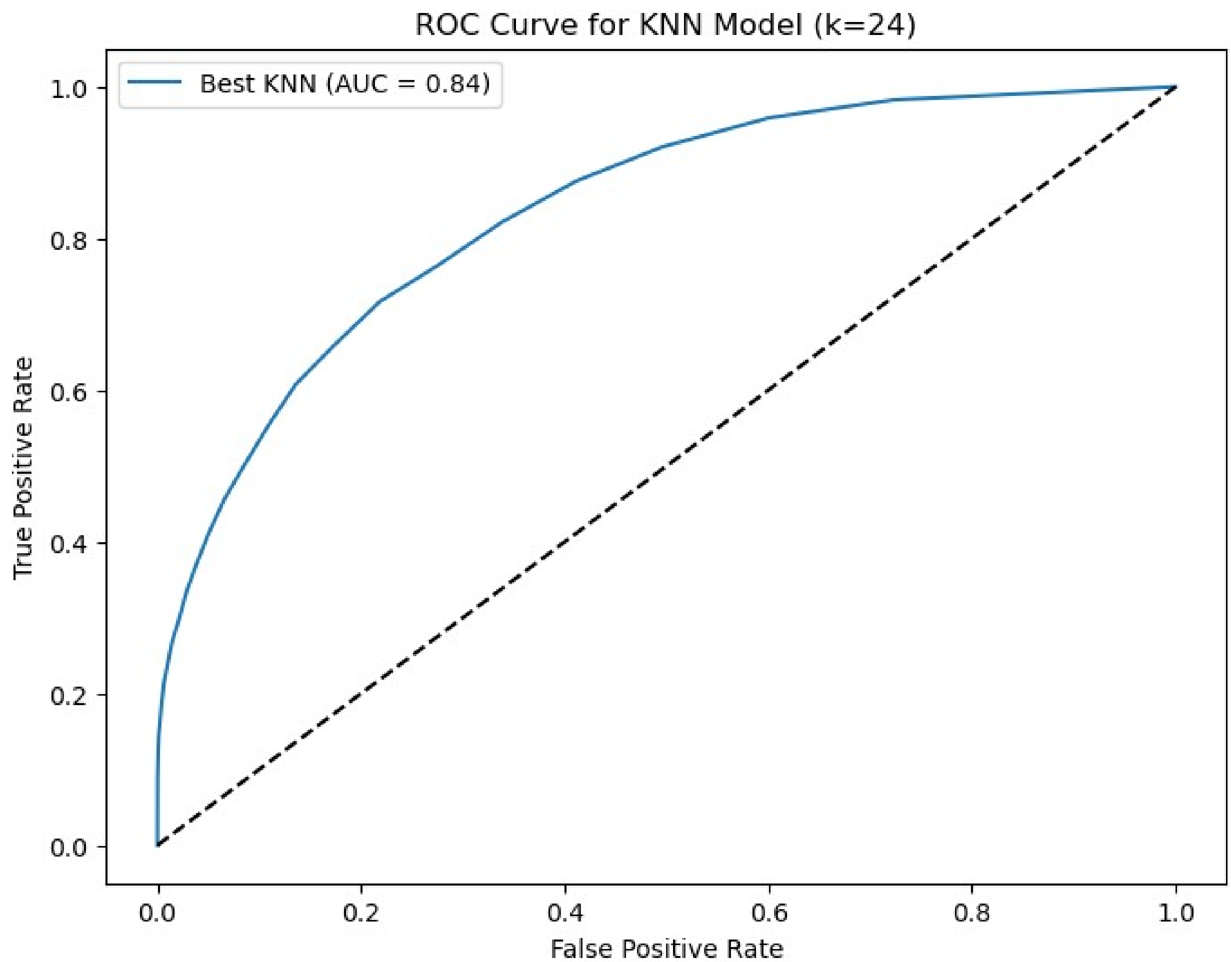

```
from sklearn.model_selection import cross_val_score
import numpy as np

# Perform cross-validation to find the best k
k_range = range(1, 31)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10,
scoring='accuracy')
    k_scores.append(scores.mean())

# Plot the cross-validation results
```

```
plt.figure(figsize=(10, 6))
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.title('KNN Hyperparameter Tuning (Best K)')
plt.show()

# Best K value
best_k = k_range[np.argmax(k_scores)]
print(f"Best K: {best_k}")
```



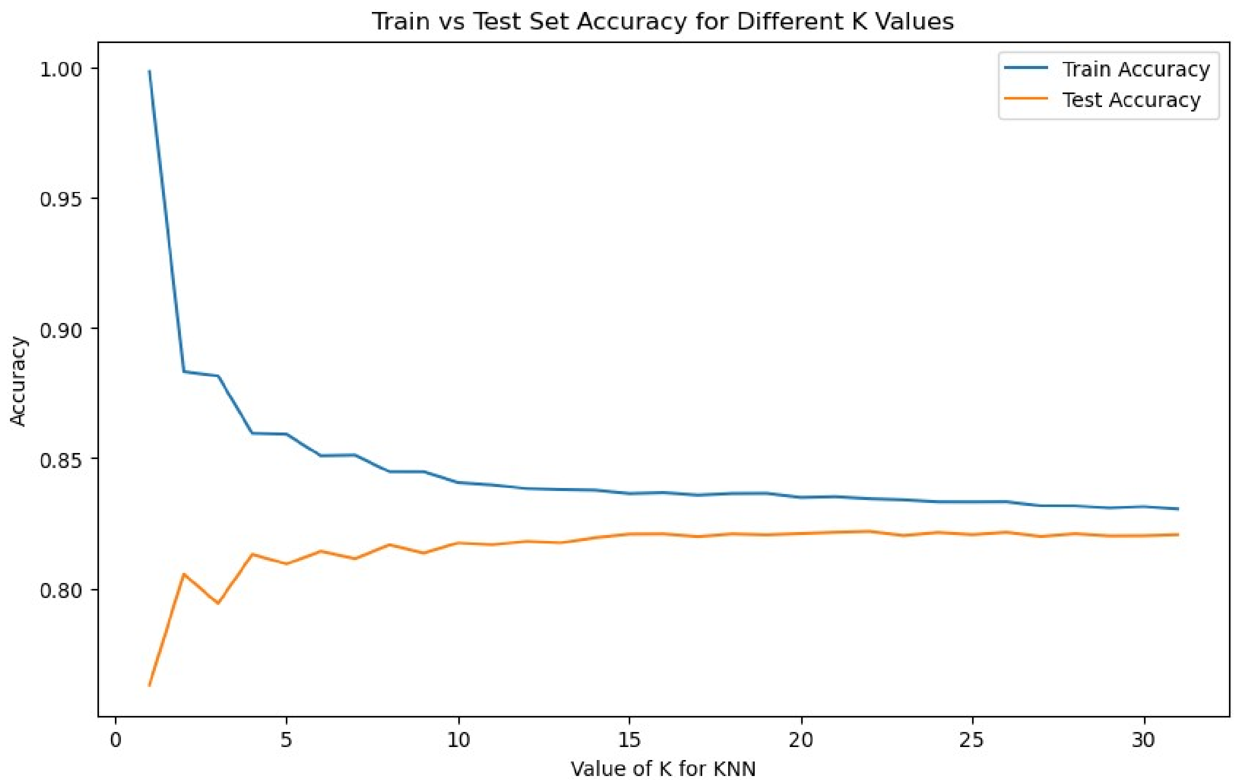

```
Best K: 24

# Train and evaluate KNN model with the best k value found from cross-
validation
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(X_train, y_train)

# Predict and evaluate with the best k
y_pred_best = best_knn.predict(X_test)
print("Confusion Matrix with Best K:\n", confusion_matrix(y_test,
y_pred_best))
print("Classification Report with Best K:\n",
```

```
classification_report(y_test, y_pred_best))


Confusion Matrix with Best K:
 [[10602   564]
 [ 2052  1435]]
Classification Report with Best K:
               precision    recall  f1-score   support

           0       0.84      0.95      0.89     11166
           1       0.72      0.41      0.52      3487

    accuracy                           0.82     14653
   macro avg       0.78      0.68      0.71     14653
weighted avg       0.81      0.82      0.80     14653


# Calculate ROC and AUC with best k
y_pred_best_proba = best_knn.predict_proba(X_test)[:, 1]
fpr_best, tpr_best, thresholds_best = roc_curve(y_test,
y_pred_best_proba)
roc_auc_best = roc_auc_score(y_test, y_pred_best_proba)
print(f"AUC Score with Best K: {roc_auc_best}")

# Plot ROC curve for best k
plt.figure(figsize=(8, 6))
plt.plot(fpr_best, tpr_best, label=f'Best KNN (AUC =
{roc_auc_best:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for KNN Model (k={best_k})')
plt.legend()
plt.show()

AUC Score with Best K: 0.8359600519233668
```




```python
# Building a plot to compare train and test set accuracy for different
values of k (from 1 to 31)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Define the range for k values
k_range = range(1, 32)
train_accuracy = []
test_accuracy = []

# Loop over different values of k to compute train and test accuracies
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Calculate accuracy on the training set
    y_train_pred = knn.predict(X_train)
    train_accuracy.append(accuracy_score(y_train, y_train_pred))
```

```
    # Calculate accuracy on the test set
    y_test_pred = knn.predict(X_test)
    test_accuracy.append(accuracy_score(y_test, y_test_pred))

# Plotting train and test accuracy
plt.figure(figsize=(10, 6))
plt.plot(k_range, train_accuracy, label='Train Accuracy')
plt.plot(k_range, test_accuracy, label='Test Accuracy')
plt.xlabel('Value of K for KNN')
plt.ylabel('Accuracy')
plt.title('Train vs Test Set Accuracy for Different K Values')
plt.legend()
plt.show()
```




```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
import numpy as np

# Assuming 'X_train', 'y_train', 'X_test', and 'y_test' are already
defined with only numerical features

# Train the KNN model with the best k found from cross-validation
best_knn = KNeighborsClassifier(n_neighbors=24)  # best 'k' value
```

```
best_knn.fit(X_train, y_train)

# Calculate permutation importance on the test set
perm_importance = permutation_importance(best_knn, X_test, y_test,
n_repeats=10, random_state=42)

# Get feature names and importances
feature_names = X.columns
importances = perm_importance.importances_mean

# Sort features by importance
indices = np.argsort(importances)[::-1]
sorted_features = feature_names[indices]
sorted_importances = importances[indices]

# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(sorted_features, sorted_importances, color='skyblue')
plt.xlabel("Mean Decrease in Accuracy")
plt.title("Feature Importance for KNN (using Permutation Importance)")
plt.gca().invert_yaxis()  # Invert y-axis for descending order
plt.show()
```