Statistical Learning Final assignment-Final version

March 7, 2025

0.1 Introduction

In this project I am going to explore the predictability of annual income using demographic, educational, and occupational attributes. The goal is to classify individuals into two income groups — those earning 50,000 per year and those earning > 50,000 per year — using data from the 1994 U.S. Census Bureau.

By testing multiple statistical learning techniques, I aim to assess which methods best capture the underlying patterns that predict income level. Model performance will be evaluated based on accuracy, F1-score, and AUC to account for class imbalance.

1 Step 1: Importing necessary libraries

I'll start by importing the necessary libraries and loading the dataset.

[1]:
```python
# Standard imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Model imports
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Evaluation imports
from sklearn.metrics import accuracy_score, confusion_matrix,ConfusionMatrixDisplay, classification_report, roc_curve, auc

# Load dataset
df = pd.read_csv('adult.csv')

# Check the first few rows
df.head()
```

```
[1]:    age  workclass  fnlwgt     education  educational-num      marital-status  \
     0   25    Private  226802          11th                7       Never-married
     1   38    Private   89814       HS-grad                9  Married-civ-spouse
     2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
     3   44    Private  160323  Some-college               10  Married-civ-spouse
     4   18          ?  103497  Some-college               10       Never-married

               occupation relationship   race  gender  capital-gain  capital-loss  \
     0  Machine-op-inspct    Own-child  Black    Male             0             0
     1    Farming-fishing      Husband  White    Male             0             0
     2    Protective-serv      Husband  White    Male             0             0
     3  Machine-op-inspct      Husband  Black    Male          7688             0
     4                  ?    Own-child  White  Female             0             0

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

1.1 Step 2: Data Exploration and Preprocessing

I'll perform exploratory data analysis (EDA) and handle missing values, outliers, and categorical encoding.

1.1.1 2.1 Handle Missing Values

In our dataset, **3 categorical columns contain missing values**, represented by `?`:

- **Workclass**
- **Occupation**
- **Native-country**

These missing values are likely **MAR (Missing at Random)**, meaning their absence may depend on other observed variables rather than the missing values themselves.

For example: - People who didn't report their **workclass** may also be unemployed. - People who didn't report their **occupation** may have missing workclass as well. - People who didn't report their **native country** may belong to certain demographic groups.

1.1.2 My Approach

To handle these missing values, I chose to **replace them with the most frequent category** (mode).
This is a **simple and effective approach** when the missing data is relatively small, and it ensures we do not lose rows unnecessarily.

[2]:
```python
# Check missing values
print(df.isin(["?"]).sum())

# Check overall percentage of missing values
print("\nPercentage of missing values:\n", df.isin(["?"]).sum() / len(df) * 100)
```

```
age                   0
workclass          2799
fnlwgt                0
education             0
educational-num       0
marital-status        0
occupation         2809
relationship          0
race                  0
gender                0
capital-gain          0
capital-loss          0
hours-per-week        0
native-country      857
income                0
dtype: int64

Percentage of missing values:
 age                0.000000
workclass          5.730724
fnlwgt             0.000000
education          0.000000
educational-num    0.000000
marital-status     0.000000
occupation         5.751198
relationship       0.000000
race               0.000000
gender             0.000000
capital-gain       0.000000
capital-loss       0.000000
hours-per-week     0.000000
native-country     1.754637
income             0.000000
dtype: float64
```

[3]:
```python
# Replace '?' with NaN first
df.replace("?", pd.NA, inplace=True)

# Fill missing categorical values with the most frequent value (mode)
df.fillna(df.mode().iloc[0], inplace=True)
```

```python
# Verify missing values are handled
print(df.isnull().sum())
```

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

[4]:
```python
# Select numerical columns
num_columns = ["age", "fnlwgt", "educational-num", "capital-gain",␣
↪"capital-loss", "hours-per-week"]

# Display summary statistics
df[num_columns].describe()
```

[4]:
```
                age        fnlwgt  educational-num  capital-gain  \
count  48842.000000  4.884200e+04     48842.000000  48842.000000
mean      38.643585  1.896641e+05        10.078089   1079.067626
std       13.710510  1.056040e+05         2.570973   7452.019058
min       17.000000  1.228500e+04         1.000000      0.000000
25%       28.000000  1.175505e+05         9.000000      0.000000
50%       37.000000  1.781445e+05        10.000000      0.000000
75%       48.000000  2.376420e+05        12.000000      0.000000
max       90.000000  1.490400e+06        16.000000  99999.000000

       capital-loss  hours-per-week
count  48842.000000    48842.000000
mean      87.502314       40.422382
std      403.004552       12.391444
min        0.000000        1.000000
25%        0.000000       40.000000
50%        0.000000       40.000000
75%        0.000000       45.000000
max     4356.000000       99.000000
```

1.1.3 Remove Unnecessary Columns

- fnlwgt is a census weighting factor, not useful for prediction, so I removed it.

[5]:
```
df.drop(columns=["fnlwgt"], inplace=True)
```

[6]:
```
df.head()
```

[6]:
```
   age  workclass     education  educational-num      marital-status  \
0   25    Private          11th                7       Never-married
1   38    Private       HS-grad                9  Married-civ-spouse
2   28  Local-gov    Assoc-acdm               12  Married-civ-spouse
3   44    Private  Some-college               10  Married-civ-spouse
4   18    Private  Some-college               10       Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4     Prof-specialty    Own-child  White  Female             0             0

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

Here by observation we can see that **education** and **educational-num** give the same information. Hence we can drop one of the feature. We prefer to keep the numerial features so I will drop the education column

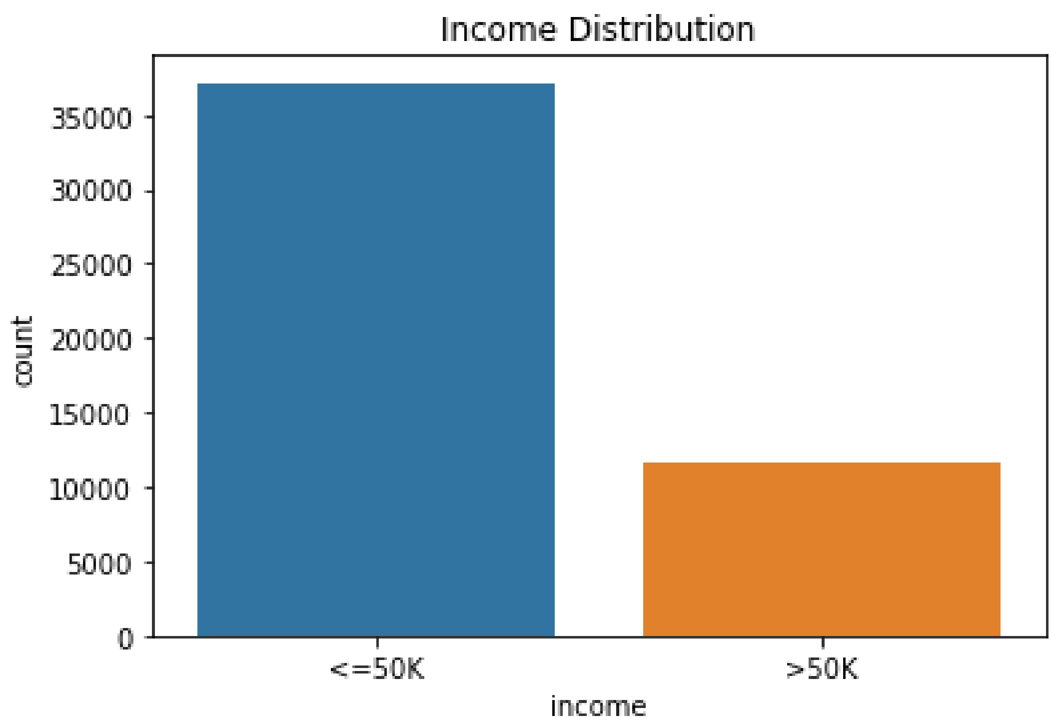
[7]:
```
df.drop('education', axis=1, inplace=True)

df.head()
```

[7]:
```
   age  workclass  educational-num      marital-status         occupation  \
0   25    Private                7       Never-married  Machine-op-inspct
1   38    Private                9  Married-civ-spouse    Farming-fishing
2   28  Local-gov               12  Married-civ-spouse    Protective-serv
3   44    Private               10  Married-civ-spouse  Machine-op-inspct
4   18    Private               10       Never-married     Prof-specialty

  relationship   race  gender  capital-gain  capital-loss  hours-per-week  \
0    Own-child  Black    Male             0             0              40
1      Husband  White    Male             0             0              50
2      Husband  White    Male             0             0              40
```

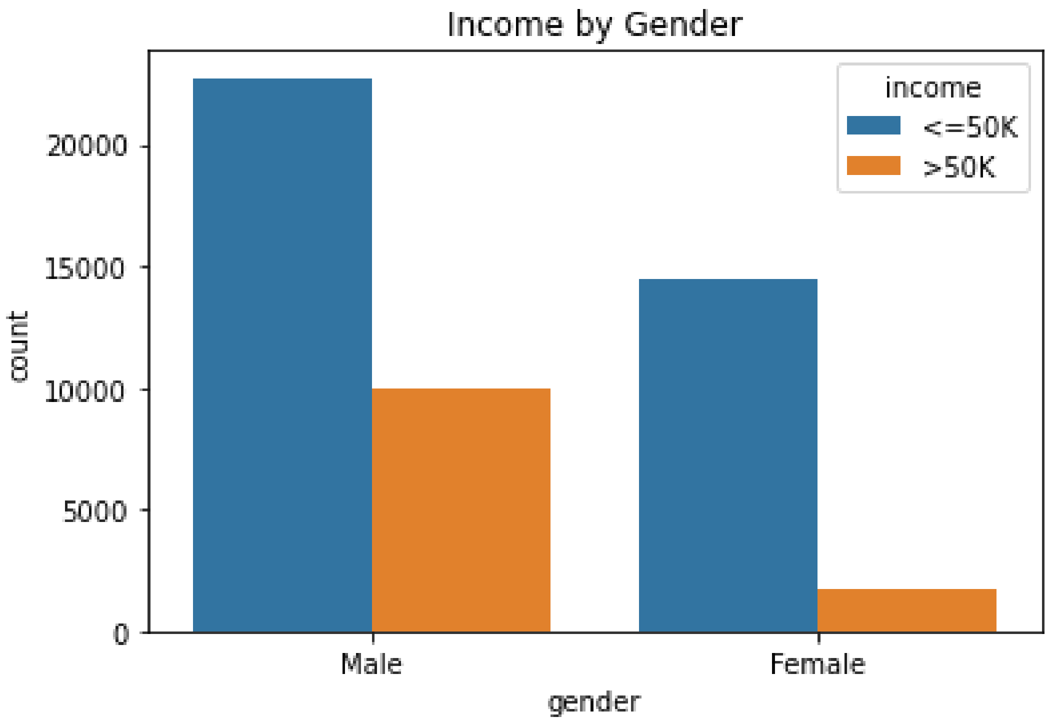
```
3      Husband  Black    Male          7688            0              40
4    Own-child  White  Female             0            0              30

  native-country income
0  United-States  <=50K
1  United-States  <=50K
2  United-States   >50K
3  United-States   >50K
4  United-States  <=50K
```

Now I am going to add some visualization to understand the data better.

[8]:
```
# Income distribution
sns.countplot(x='income', data=df)
plt.title('Income Distribution')
plt.show()
```



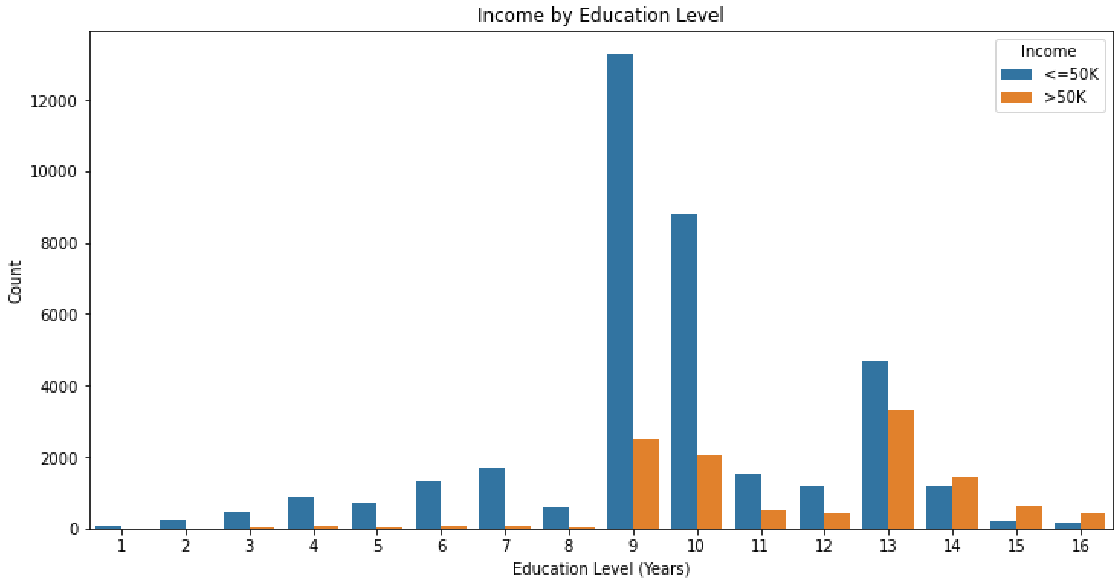
[9]:
```
# Income by Gender
sns.countplot(x='gender', hue='income', data=df)
plt.title('Income by Gender')
plt.show()
```



1.1.4 Observations

- The income distribution is highly imbalanced, with the majority earning 50K. This class imbalance may influence the performance of classification models, especially in terms of precision and recall.
- Men are more likely to belong to the > 50K income group compared to women, highlighting a potential gender disparity.

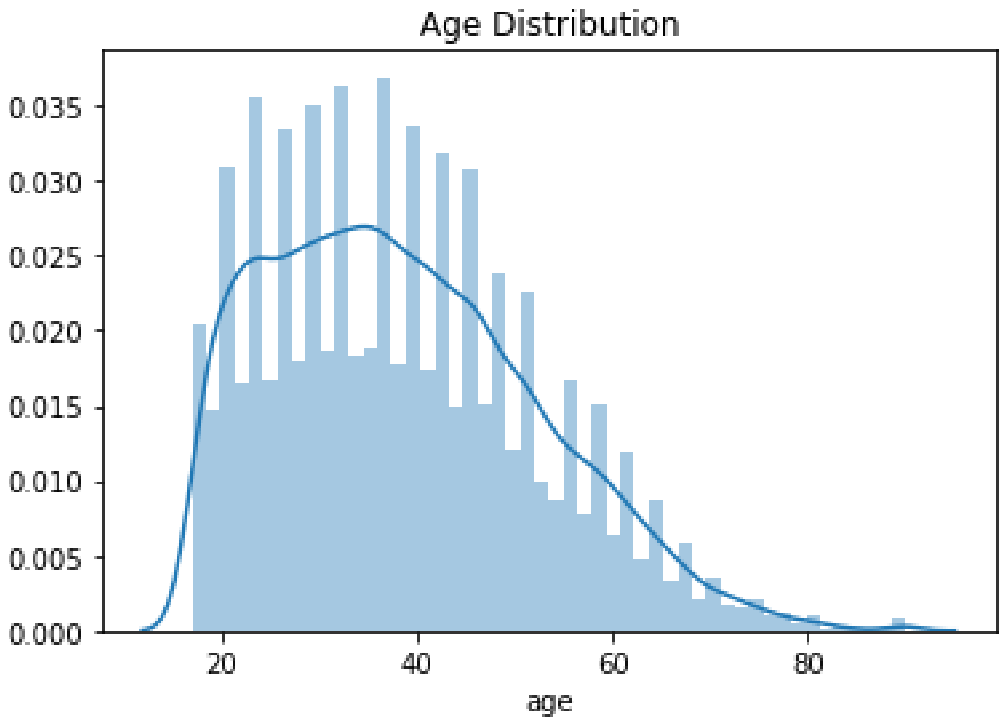
[10]:
```python
# Income by Education
plt.figure(figsize=(12, 6))
sns.countplot(x='educational-num', hue='income', data=df)
plt.title('Income by Education Level')
plt.xlabel('Education Level (Years)')
plt.ylabel('Count')
plt.legend(title='Income')
plt.show()
```



1.1.5 Observations

- Higher levels of education (such as Bachelor's, Master's, and Doctorate degrees) are **positively correlated with higher income**.
- Individuals with **lower educational attainment**, such as **some high school or high school graduates**, are predominantly in the 50K income group.
- This suggests **education is a strong predictor** of income level.

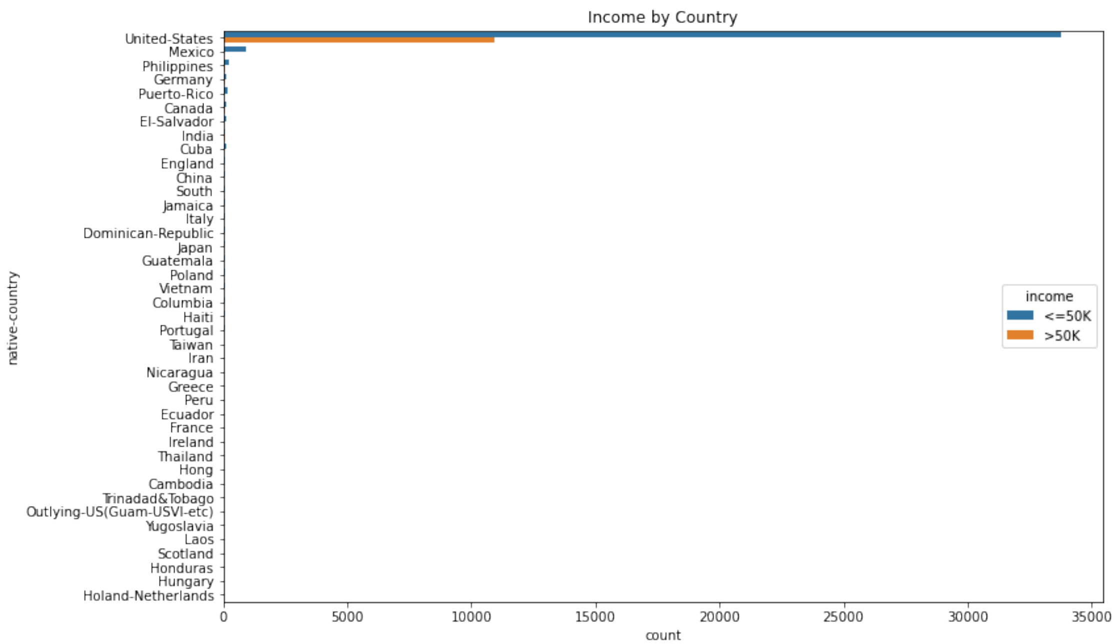
[11]:
```
# Age distribution
sns.distplot(df['age'], kde=True)
plt.title('Age Distribution')
plt.show()
```



1.1.6 Observations

- The age distribution is **slightly right-skewed**, with a higher concentration of younger individuals in the dataset.
- We also observe **some individuals with ages above 80**, which seems **unusual for a working population**. These data points may need further investigation to determine whether they are **valid records or potential data entry errors**.

[12]:
```
# Plot Income by Native Country
plt.figure(figsize=(12, 8))
sns.countplot(y='native-country', hue='income', data=df,␣
↪order=df['native-country'].value_counts().index)
plt.title('Income by Country')
plt.show()
```



1.1.7 Observations

- The majority of individuals are from the **United States**, with smaller representations from **Mexico, the Philippines, and Germany**.
- Given the **large number of low-frequency countries**, it may be beneficial to **group all countries except the United States and Mexico into a single "Other" category**. This simplifies the analysis while preserving the most relevant geographic distinctions.

[13]:
```
# Group countries with low counts into "Other"
threshold = 300
country_counts = df['native-country'].value_counts()
df['native-country-grouped'] = df['native-country'].apply(
    lambda x: x if country_counts[x] >= threshold else 'Other'
)
```

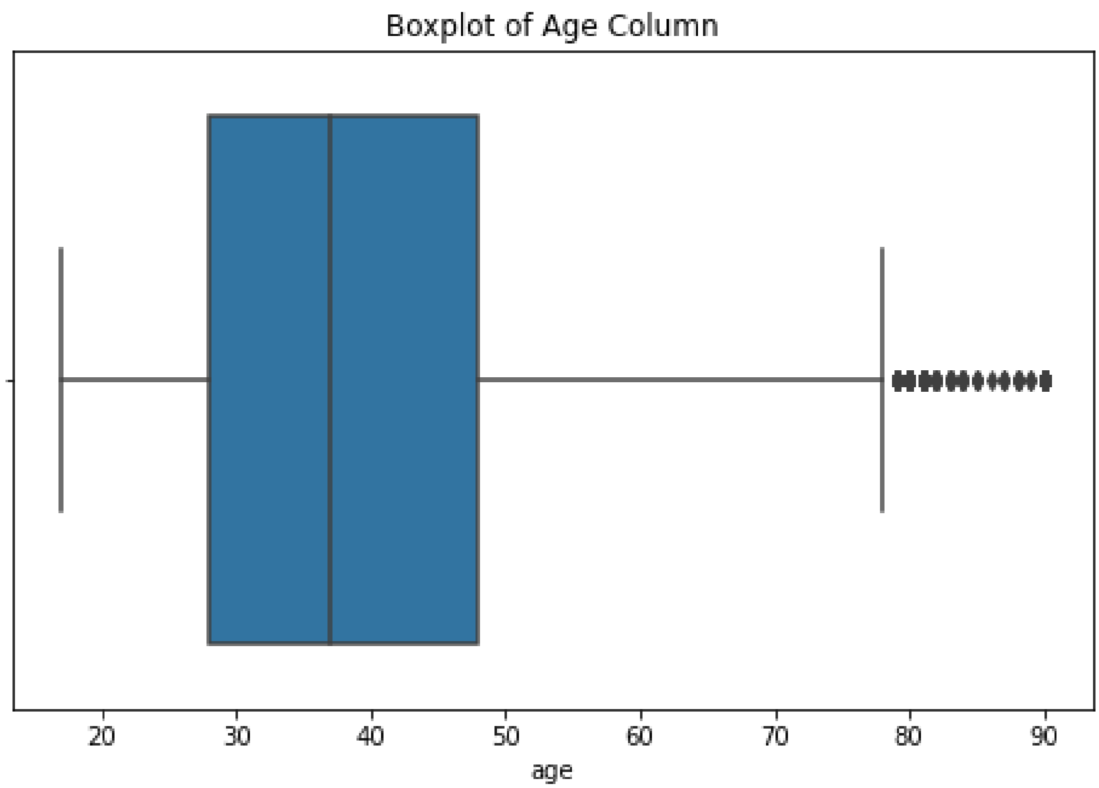
[14]:
```
df['native-country-grouped'].value_counts()
```

[14]:
```
United-States    44689
Other             3202
Mexico             951
Name: native-country-grouped, dtype: int64
```

1.1.8 Detecting Outliers in Age

To better understand the distribution of the `age` column and detect potential outliers, I used a boxplot.

[15]:
```
# Set figure size
plt.figure(figsize=(8, 5))

# Create a boxplot for 'age'
sns.boxplot(x=df["age"])

plt.title("Boxplot of Age Column")

plt.show()
```



We can see that we have some outliers and I decided to remove them.

[16]:
```
# Compute Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df["age"].quantile(0.25)
Q3 = df["age"].quantile(0.75)

# Compute IQR
```

```
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

print(f"Lower Bound: {lower_bound}, Upper Bound: {upper_bound}")

# Identify outliers
outliers = df[(df["age"] < lower_bound) | (df["age"] > upper_bound)]
print("Number of Outliers in Age Column:", outliers.shape[0])
```

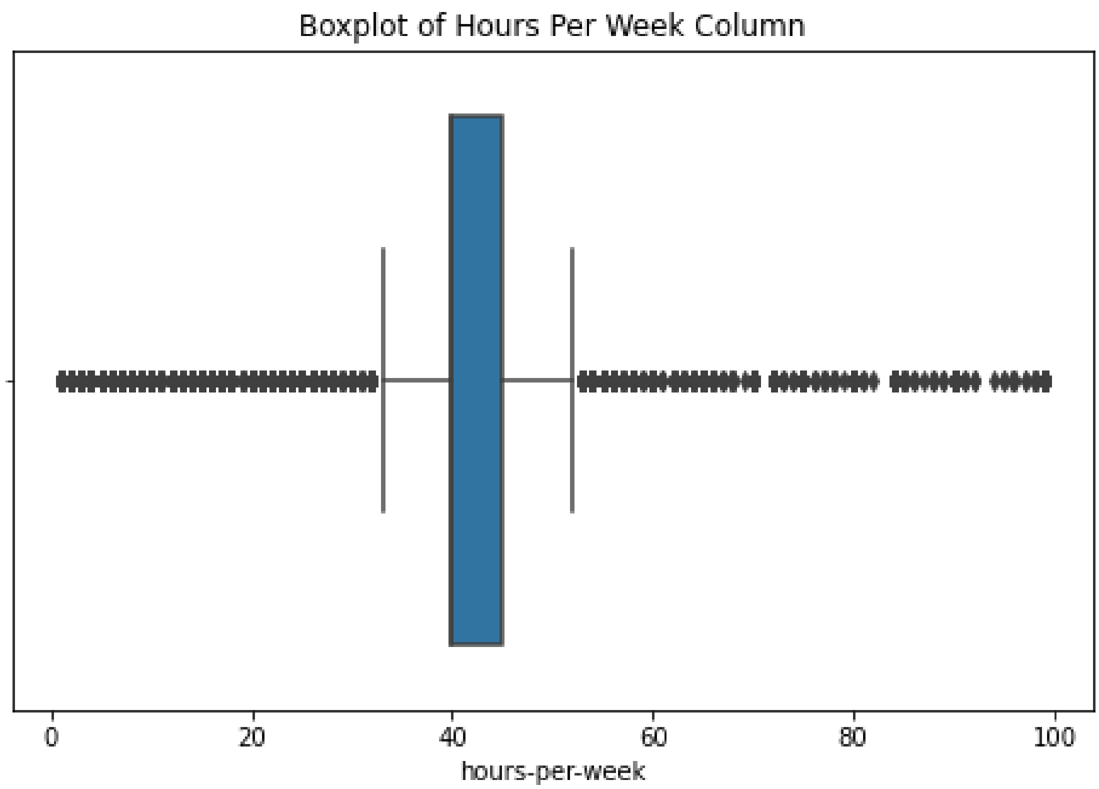
```
Lower Bound: -2.0, Upper Bound: 78.0
Number of Outliers in Age Column: 216
```

[17]:
```
# Remove rows with extreme outliers
df = df[(df["age"] >= lower_bound) & (df["age"] <= upper_bound)]
print("Shape after removing outliers:", df.shape)
```

```
Shape after removing outliers: (48626, 14)
```

1.1.9 Outlier Analysis: Hours Per Week

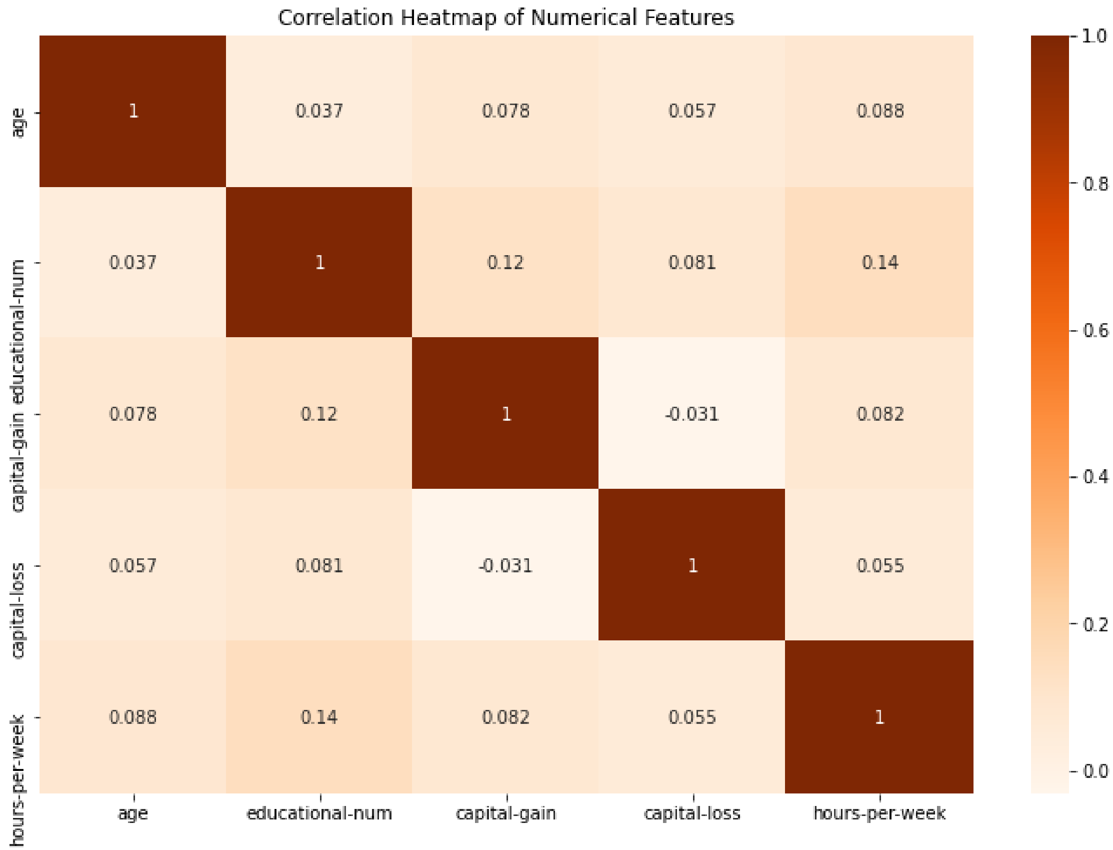
[18]:
```
# Set figure size
plt.figure(figsize=(8, 5))

# Create a boxplot for 'hours-per-week'
sns.boxplot(x=df["hours-per-week"])

plt.title("Boxplot of Hours Per Week Column")

plt.show()
```



I decided to keep hours per week unchanged as:

- Many professionals such as business owners, doctors, and executives often work well above 52.5 hours per week. Removing these values could eliminate valid records.
- Lower Work Hours Are Also Realistic: Many individuals work part-time jobs, especially younger or semi-retired workers. Removing low-hour workers would distort the dataset's real-world representation.

[19]:
```
# Controll the correlations with displaying the correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap="Oranges")

plt.title("Correlation Heatmap of Numerical Features")

plt.show()
```



1.1.10 Correlation Heatmap Analysis

- Low Correlation Overall: Most numerical features have correlation coefficients close to 0, meaning they do not strongly influence each other. This is a good indication that multicollinearity is not a concern, so all numerical features can be retained.
- Capital Gain & Capital Loss: These two features show a very slight negative correlation (-0.07). This is expected since individuals tend to either have capital gains or capital losses, but rarely both.

[20]:
```
# Number of unique values in each feature
for col in df.columns:
    print(col, len(df[col].unique()))
    if len(df[col].unique()) < 10:    # If there are few unique values, print␣
 ↪them to see what they are
        print(df[col].unique())
```

```
age 62
workclass 8
['Private' 'Local-gov' 'Self-emp-not-inc' 'Federal-gov' 'State-gov'
 'Self-emp-inc' 'Without-pay' 'Never-worked']
```

```
educational-num 16
marital-status 7
['Never-married' 'Married-civ-spouse' 'Widowed' 'Divorced' 'Separated'
 'Married-spouse-absent' 'Married-AF-spouse']
occupation 14
relationship 6
['Own-child' 'Husband' 'Not-in-family' 'Unmarried' 'Wife' 'Other-relative']
race 5
['Black' 'White' 'Asian-Pac-Islander' 'Other' 'Amer-Indian-Eskimo']
gender 2
['Male' 'Female']
capital-gain 123
capital-loss 99
hours-per-week 96
native-country 41
income 2
['<=50K' '>50K']
native-country-grouped 3
['United-States' 'Other' 'Mexico']
```

[1]:
```
# Let's drop the native-country column as we created a new column␣
↪"native-country-grouped"
df.drop(columns=['native-country'], inplace=True)
```

```
      ␣
↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
↪last)

        <ipython-input-1-9b916ab74f7b> in <module>
    ----> 1 df.drop(columns=['native-country'], inplace=True)


        NameError: name 'df' is not defined
```

[22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48626 entries, 0 to 48841
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   age                     48626 non-null  int64
 1   workclass               48626 non-null  object
 2   educational-num         48626 non-null  int64
```

```
 3   marital-status          48626 non-null  object
 4   occupation              48626 non-null  object
 5   relationship            48626 non-null  object
 6   race                    48626 non-null  object
 7   gender                  48626 non-null  object
 8   capital-gain            48626 non-null  int64
 9   capital-loss            48626 non-null  int64
 10  hours-per-week          48626 non-null  int64
 11  income                  48626 non-null  object
 12  native-country-grouped  48626 non-null  object
dtypes: int64(5), object(8)
memory usage: 5.2+ MB
```

[23]:

```
# Label encode binary columns
df['income'] = df['income'].map({'<=50K': 0, '>50K': 1})
df['gender'] = df['gender'].map({'Male': 0, 'Female': 1})

# One-hot encode nominal categorical columns
categorical_columns = ['workclass', 'marital-status', 'occupation',
                       'relationship', 'race', 'native-country-grouped']

df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

df.head()
```

[23]:

```
   age  educational-num  gender  capital-gain  capital-loss  hours-per-week  \
0   25                7       0             0             0              40
1   38                9       0             0             0              50
2   28               12       0             0             0              40
3   44               10       0          7688             0              40
4   18               10       1             0             0              30

   income  workclass_Local-gov  workclass_Never-worked  workclass_Private  \
0       0                    0                       0                  1
1       0                    0                       0                  1
2       1                    1                       0                  0
3       1                    0                       0                  1
4       0                    0                       0                  1

   …  relationship_Other-relative  relationship_Own-child  \
0  …                            0                       1
1  …                            0                       0
2  …                            0                       0
3  …                            0                       0
4  …                            0                       1

   relationship_Unmarried  relationship_Wife  race_Asian-Pac-Islander  \
```

```
0                        0                   0                         0
1                        0                   0                         0
2                        0                   0                         0
3                        0                   0                         0
4                        0                   0                         0

   race_Black  race_Other  race_White  native-country-grouped_Other  \
0           1           0           0                             0
1           0           0           1                             0
2           0           0           1                             0
3           1           0           0                             0
4           0           0           1                             0

   native-country-grouped_United-States
0                                     1
1                                     1
2                                     1
3                                     1
4                                     1

[5 rows x 44 columns]
```

[24]:
```python
# Verify new dataframe structure
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48626 entries, 0 to 48841
Data columns (total 44 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   age                                   48626 non-null  int64
 1   educational-num                       48626 non-null  int64
 2   gender                                48626 non-null  int64
 3   capital-gain                          48626 non-null  int64
 4   capital-loss                          48626 non-null  int64
 5   hours-per-week                        48626 non-null  int64
 6   income                                48626 non-null  int64
 7   workclass_Local-gov                   48626 non-null  uint8
 8   workclass_Never-worked                48626 non-null  uint8
 9   workclass_Private                     48626 non-null  uint8
 10  workclass_Self-emp-inc                48626 non-null  uint8
 11  workclass_Self-emp-not-inc            48626 non-null  uint8
 12  workclass_State-gov                   48626 non-null  uint8
 13  workclass_Without-pay                 48626 non-null  uint8
 14  marital-status_Married-AF-spouse      48626 non-null  uint8
 15  marital-status_Married-civ-spouse     48626 non-null  uint8
 16  marital-status_Married-spouse-absent  48626 non-null  uint8
```

```
 17  marital-status_Never-married          48626 non-null  uint8
 18  marital-status_Separated              48626 non-null  uint8
 19  marital-status_Widowed                48626 non-null  uint8
 20  occupation_Armed-Forces               48626 non-null  uint8
 21  occupation_Craft-repair               48626 non-null  uint8
 22  occupation_Exec-managerial            48626 non-null  uint8
 23  occupation_Farming-fishing            48626 non-null  uint8
 24  occupation_Handlers-cleaners          48626 non-null  uint8
 25  occupation_Machine-op-inspct          48626 non-null  uint8
 26  occupation_Other-service              48626 non-null  uint8
 27  occupation_Priv-house-serv            48626 non-null  uint8
 28  occupation_Prof-specialty             48626 non-null  uint8
 29  occupation_Protective-serv            48626 non-null  uint8
 30  occupation_Sales                      48626 non-null  uint8
 31  occupation_Tech-support               48626 non-null  uint8
 32  occupation_Transport-moving           48626 non-null  uint8
 33  relationship_Not-in-family            48626 non-null  uint8
 34  relationship_Other-relative           48626 non-null  uint8
 35  relationship_Own-child                48626 non-null  uint8
 36  relationship_Unmarried                48626 non-null  uint8
 37  relationship_Wife                     48626 non-null  uint8
 38  race_Asian-Pac-Islander               48626 non-null  uint8
 39  race_Black                            48626 non-null  uint8
 40  race_Other                            48626 non-null  uint8
 41  race_White                            48626 non-null  uint8
 42  native-country-grouped_Other          48626 non-null  uint8
 43  native-country-grouped_United-States  48626 non-null  uint8
dtypes: int64(7), uint8(37)
memory usage: 4.7 MB
None
```

The columns **workclass_Never-worked** and **workclass_Without-pay**, indicating little to no earned income, may have limited predictive power for income classification. I will assess their relevance by calculating their frequency.

[2]:
```python
print(df[['workclass_Never-worked', 'workclass_Without-pay']].sum())
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-2-cf0bbf01ce1f> in <module>
   ----> 1 print(df[['workclass_Never-worked', 'workclass_Without-pay']].sum())
```

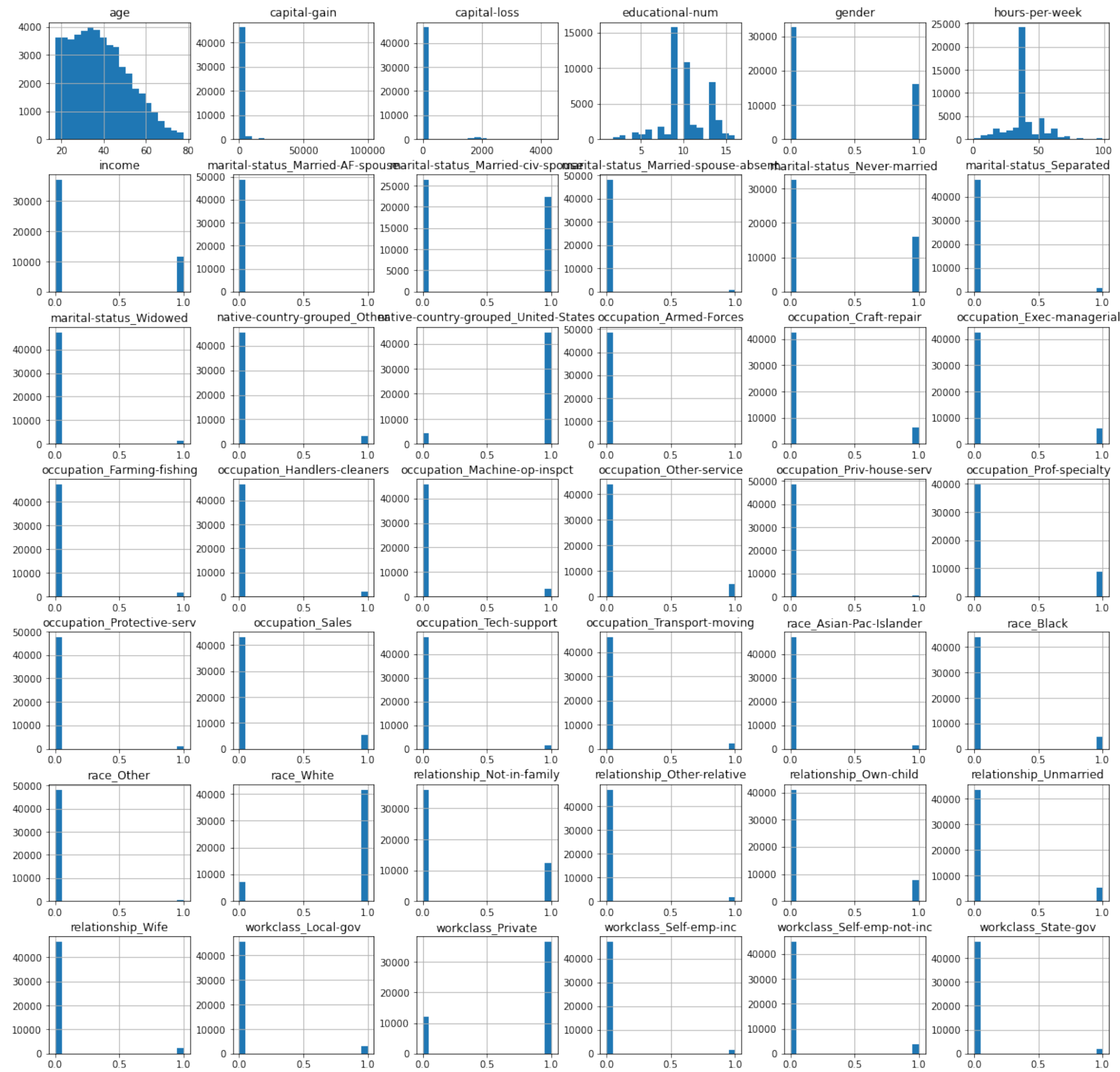
```
NameError: name 'df' is not defined
```

Based on their low frequency and indication of little to no earned income, the columns workclass_Never-worked and workclass_Without-pay are deemed unhelpful for predicting income levels. Therefore, I will remove them from the dataset.

[3]:
```
df.drop(columns=['workclass_Never-worked', 'workclass_Without-pay'],␣
↪inplace=True)
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-3-de41cfdc16bd> in <module>
----> 1 df.drop(columns=['workclass_Never-worked', 'workclass_Without-pay'],␣
↪inplace=True)


NameError: name 'df' is not defined
```

[27]:
```
# Now that categorical features are encoded, we can review the distributions of␣
↪all features using histograms.
# This helps understand the numerical representation the model will work with.

# Set figure size to make it clearer
df.hist(figsize=(20, 20), bins=20)

# Show plot
plt.show()
```



[28]:
```
print(df.groupby('income').mean())
```

```
             age  educational-num    gender  capital-gain  capital-loss  \
income
0      36.647294         9.603375  0.388431    146.530546     53.921438
1      44.145678        11.600996  0.151515   4028.612413    193.407589

       hours-per-week  workclass_Local-gov  workclass_Private  \
income
0           38.901344             0.059442           0.782405
1           45.482702             0.079406           0.654906

       workclass_Self-emp-inc  workclass_Self-emp-not-inc  …  \
income                                                     …
```

```
0                      0.020283                      0.074371  …
1                      0.079749                      0.092197  …

        relationship_Other-relative  relationship_Own-child  \
income
0                          0.039214                0.201801
1                          0.004464                0.009443

        relationship_Unmarried  relationship_Wife  race_Asian-Pac-Islander  \
income
0                     0.129810           0.033453                 0.029802
1                     0.026526           0.093570                 0.035110

        race_Black  race_Other  race_White  native-country-grouped_Other  \
income
0         0.111123    0.009628    0.838278                      0.066582
1         0.048502    0.004292    0.907374                      0.061980

        native-country-grouped_United-States
income
0                                   0.908997
1                                   0.933986

[2 rows x 41 columns]
```

1.1.11 Income Group Summary

- Higher income linked to older age and more education.
- Capital gains much higher for high earners.
- Married people earn more; unmarried/children common in lower income.
- Higher income group has more White individuals and more from the US.
- Education, age, and capital gains are key predictors.

[29]:
```
# Define features (X) and target (y)
X = df.drop(columns=['income'])
y = df['income']

# First split: train + temp (where temp will be split into validation & test)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4,␣
↪random_state=42)

# Second split: split temp into validation and test (each 20% of the whole data)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
↪random_state=42)

# Final shape check
print(f"Train shape: {X_train.shape}")
```

```
print(f"Validation shape: {X_val.shape}")
print(f"Test shape: {X_test.shape}")
```

```
Train shape: (29175, 41)
Validation shape: (9725, 41)
Test shape: (9726, 41)
```

[30]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

[31]:

```
# reindex y_val
y_val = y_val.reset_index(drop=True)
print(y_val)
```

```
0       1
1       0
2       0
3       0
4       0
       ..
9720    1
9721    0
9722    0
9723    0
9724    0
Name: income, Length: 9725, dtype: int64
```

[32]:

```
# Combine train and validation sets
X_train_val = np.concatenate([X_train, X_val], axis=0)
y_train_val = np.concatenate([y_train, y_val], axis=0)
```

2 Linear Model

Although Linear Regression is **not the standard choice** for classification problems, I chose to test it for several important reasons:

- **Initial Exploration & Baseline:**
  Linear Regression offers a **simple and interpretable starting point**. Even though it is typically used for regression (predicting continuous values), the idea was to **see how well it performs when predicting a binary target** (income >50K or 50K) by rounding the predicted values to 0 or 1.
- **Evaluating Class Separation:**
  Linear Regression can still **indicate how separable the two income classes are** by observing how far the predicted values are from the 0.5 threshold. If Linear Regression

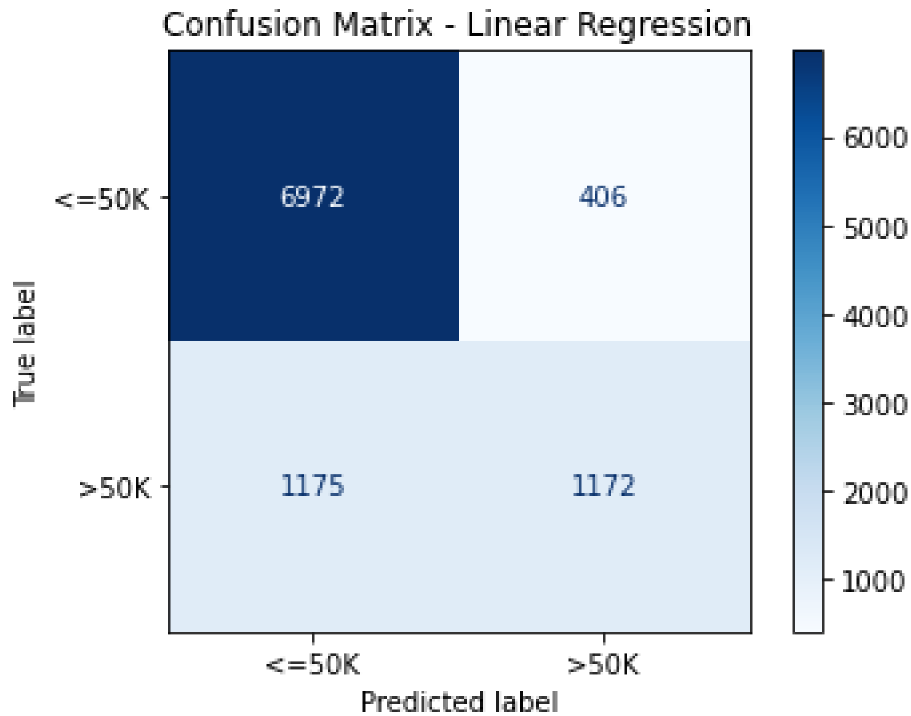
performs reasonably well, it suggests **some linear separability exists in the data**, which can inform future modeling choices.

[33]:
```
# Initialize and fit linear regression model on the train set
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on validation set
y_pred_val_linear = model.predict(X_val)

# Since this is classification (binary 0/1), we round predictions
y_pred_val_linear_class = (y_pred_val_linear > 0.5).astype(int)

# Evaluate with classification report
print("Linear Regression Validation Classification Report:")
print(classification_report(y_val, y_pred_val_linear_class))
```

```
Linear Regression Validation Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      7378
           1       0.74      0.50      0.60      2347

    accuracy                           0.84      9725
   macro avg       0.80      0.72      0.75      9725
weighted avg       0.83      0.84      0.83      9725
```

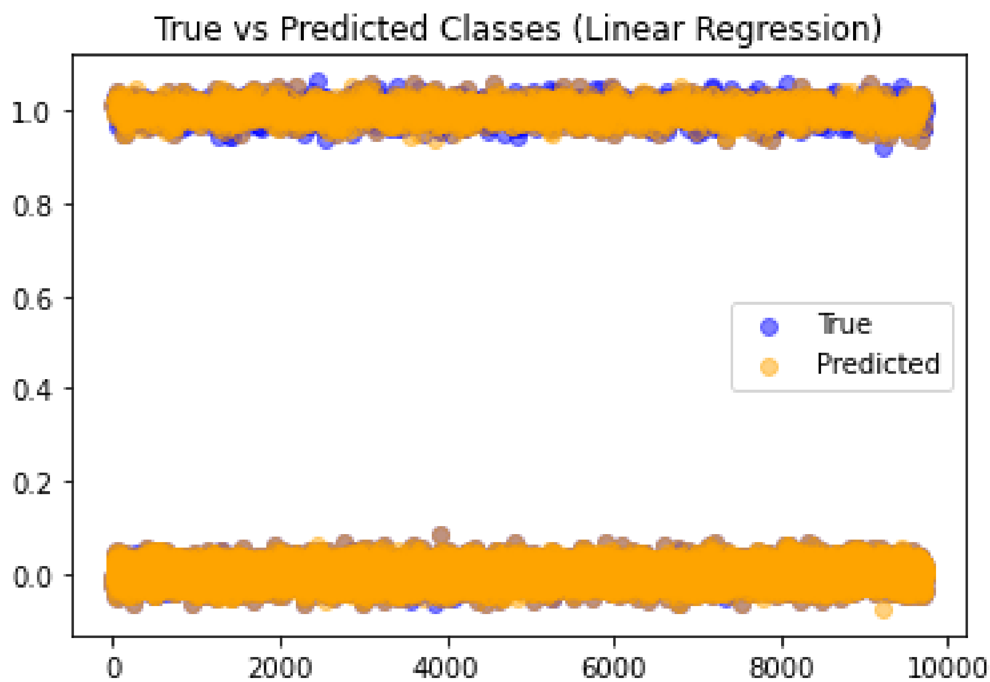
[34]:
```
# Plot confusion matrix
conf_matrix = confusion_matrix(y_val, y_pred_val_linear_class)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["<=50K", ">50K"])

plt.figure(figsize=(6,6))
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix - Linear Regression")
plt.show()
```

```
<Figure size 432x432 with 0 Axes>
```

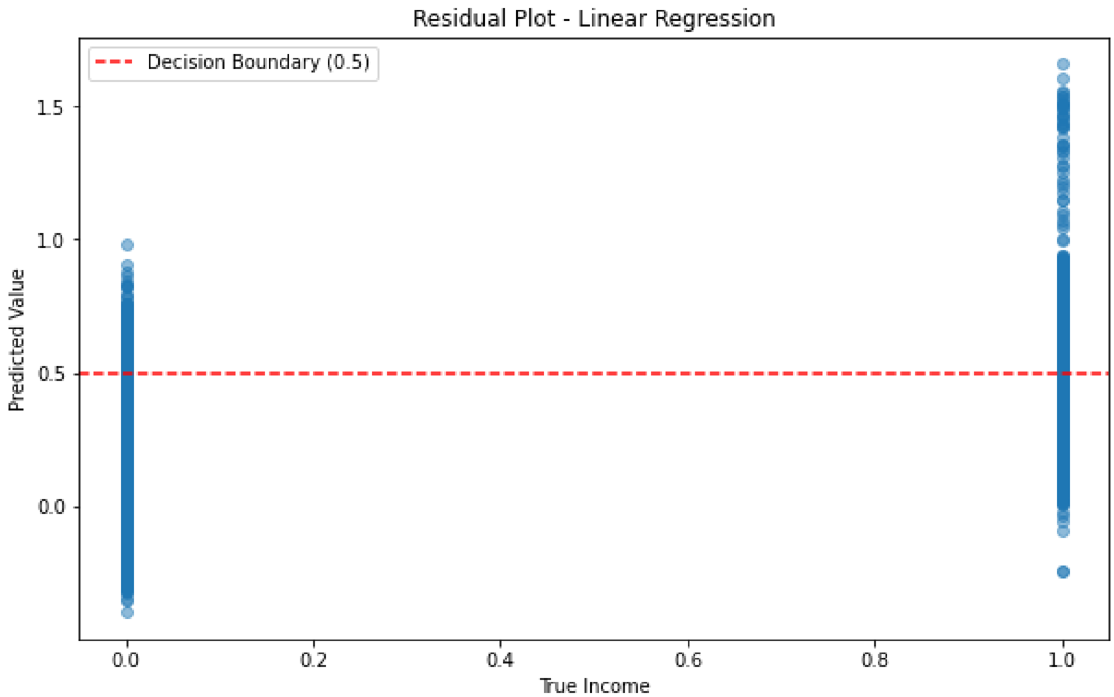


[35]:

```
# Plot True vs Predicted Classes

jitter = np.random.normal(0, 0.02, size=len(y_val))  # small vertical shift
plt.scatter(range(len(y_val)), y_val + jitter, label='True', alpha=0.5,␣
 ↪color='blue')
plt.scatter(range(len(y_val)), y_pred_val_linear_class + jitter,␣
 ↪label='Predicted', alpha=0.5, color='orange')

plt.legend()
plt.title('True vs Predicted Classes (Linear Regression)')
plt.show()
```

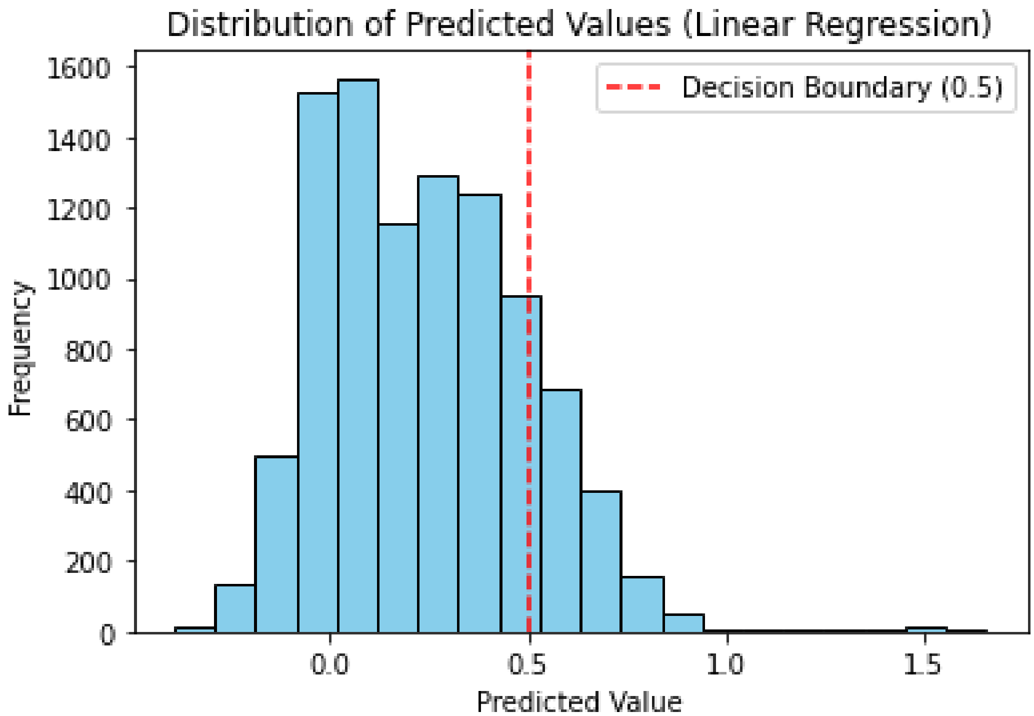


[36]:
```
# Residual Plot - Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_val, y_pred_val_linear, alpha=0.5)
plt.axhline(0.5, color='red', linestyle='--', label='Decision Boundary (0.5)')
plt.xlabel('True Income')
plt.ylabel('Predicted Value')
plt.legend()
plt.title('Residual Plot - Linear Regression')
plt.show()
```

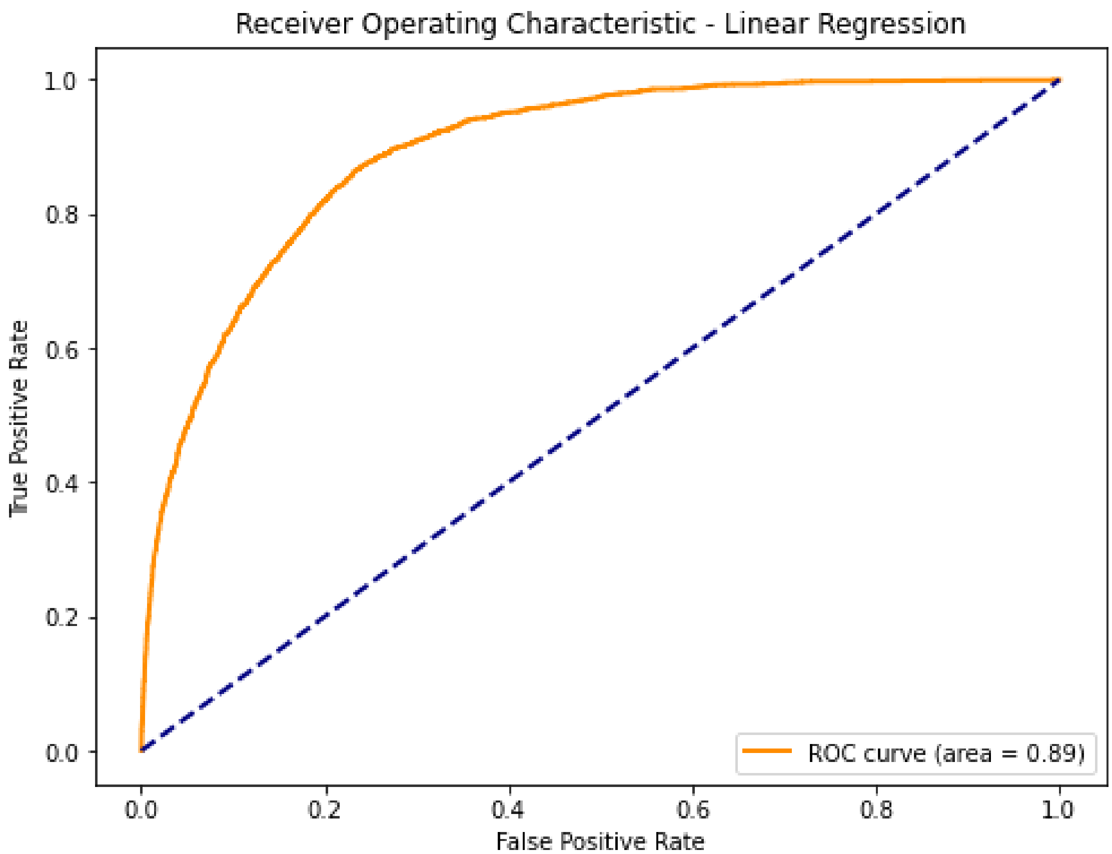


[37]:

```python
# Distribution of Predicted Values
plt.hist(y_pred_val_linear, bins=20, color='skyblue', edgecolor='black')
plt.axvline(0.5, color='red', linestyle='--', label='Decision Boundary (0.5)')
plt.xlabel('Predicted Value')
plt.ylabel('Frequency')
plt.legend()
plt.title('Distribution of Predicted Values (Linear Regression)')
plt.show()
```



[38]:
```
# ROC Curve and AUC Calculation
fpr, tpr, _ = roc_curve(y_val, y_pred_val_linear)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
 ↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Linear Regression')
plt.legend(loc='lower right')
plt.show()
```



2.1 Linear Regression Model Evaluation

2.1.1 True vs Predicted Classes

- This scatter plot compares actual labels with predicted classes.
- Linear Regression struggles to cleanly separate the categories, especially for intermediate cases.

2.1.2 Residual Plot

- Shows actual labels against predicted values.
- Many predictions fall between 0 and 1, indicating the model's lack of clear separation.
- This confirms Linear Regression is not ideal for classification tasks.

2.1.3 Distribution of Predicted Values

- Most predictions are near 0 or 1, but some are in between.
- This uncertainty highlights the model's limitations in handling categorical labels.

2.1.4 ROC Curve

- The ROC AUC is **0.89**, showing the model captures some patterns.
- However, this does not mean Linear Regression is the best choice — classification models would handle this task better.

2.1.5 Conclusion

- **Linear Regression is not designed for classification.**
- A better approach would be to use models specifically designed for classification, such as **Logistic Regression** or **Decision Trees**.

3 Logistic Regression

3.0.1 Why Logistic Regression Was Selected

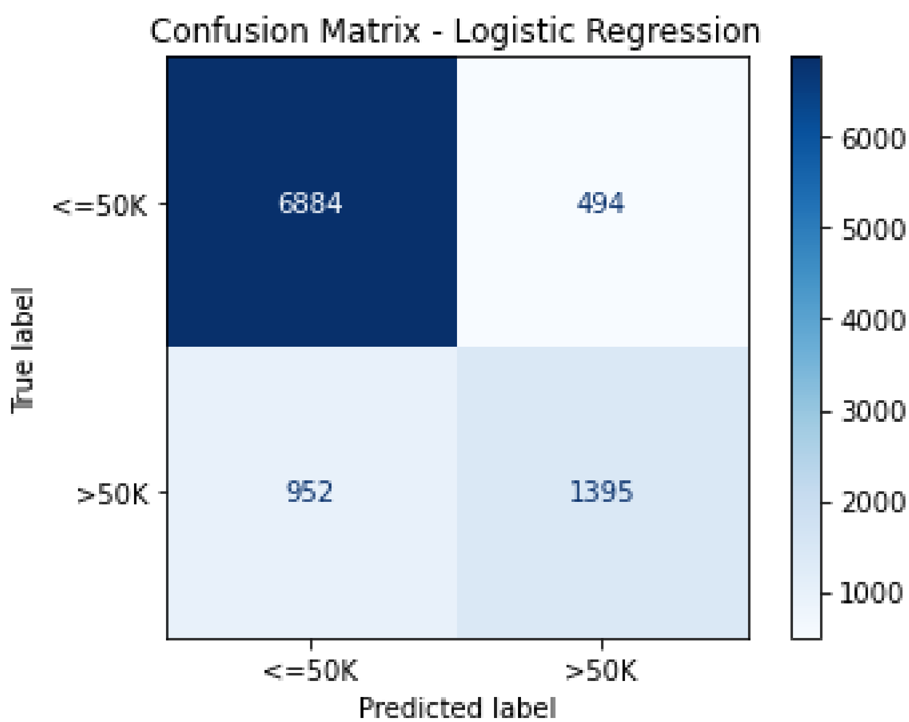
- Specifically designed for **binary classification** tasks.
- Directly outputs **probabilities**, making it easier to apply thresholds.
- Provides a **simple and interpretable model**.
- Serves as a **strong baseline** for comparison with more complex models.

[39]:
```
# Fit Logistic Regression Model
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train)

# Predict probabilities and classes
y_pred_val_log_proba = log_model.predict_proba(X_val)[:, 1]  # Probability of␣
↪class 1
y_pred_val_log_class = log_model.predict(X_val)  # Predicted class (0 or 1)

# Classification Report
print("Logistic Regression Classification Report:")
print(classification_report(y_val, y_pred_val_log_class))
```

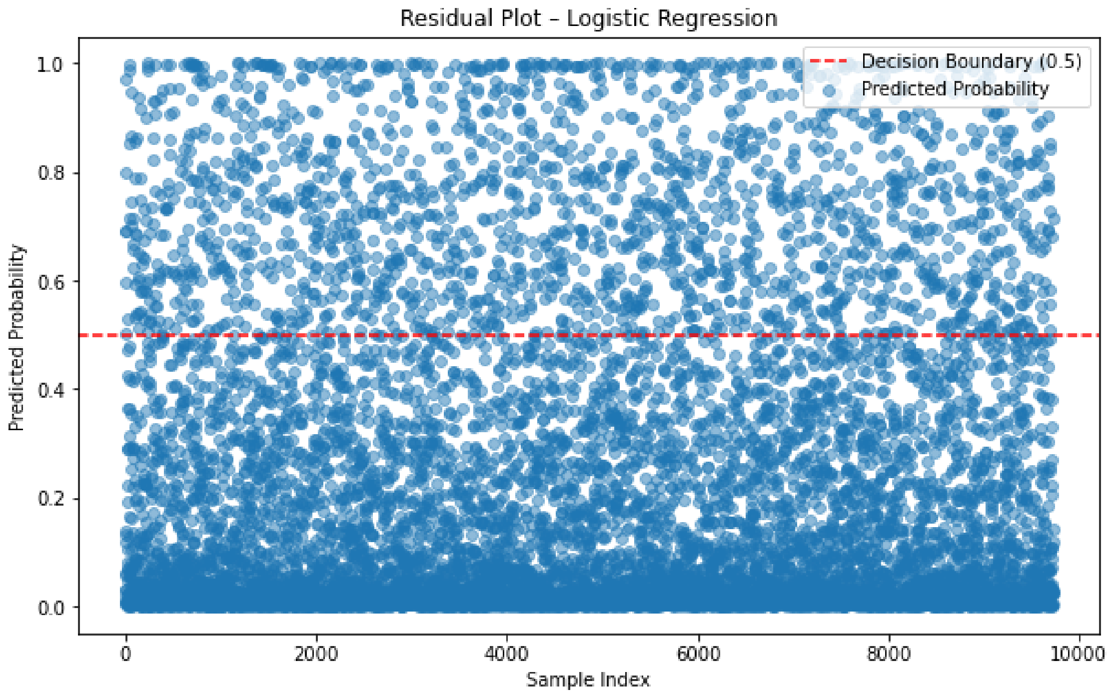
```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      7378
           1       0.74      0.59      0.66      2347

    accuracy                           0.85      9725
   macro avg       0.81      0.76      0.78      9725
weighted avg       0.84      0.85      0.85      9725
```

[40]:
```
# Confusion Matrix
conf_matrix = confusion_matrix(y_val, y_pred_val_log_class)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["<=50K", ">50K"])
plt.figure(figsize=(6,6))
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```

```
<Figure size 432x432 with 0 Axes>
```

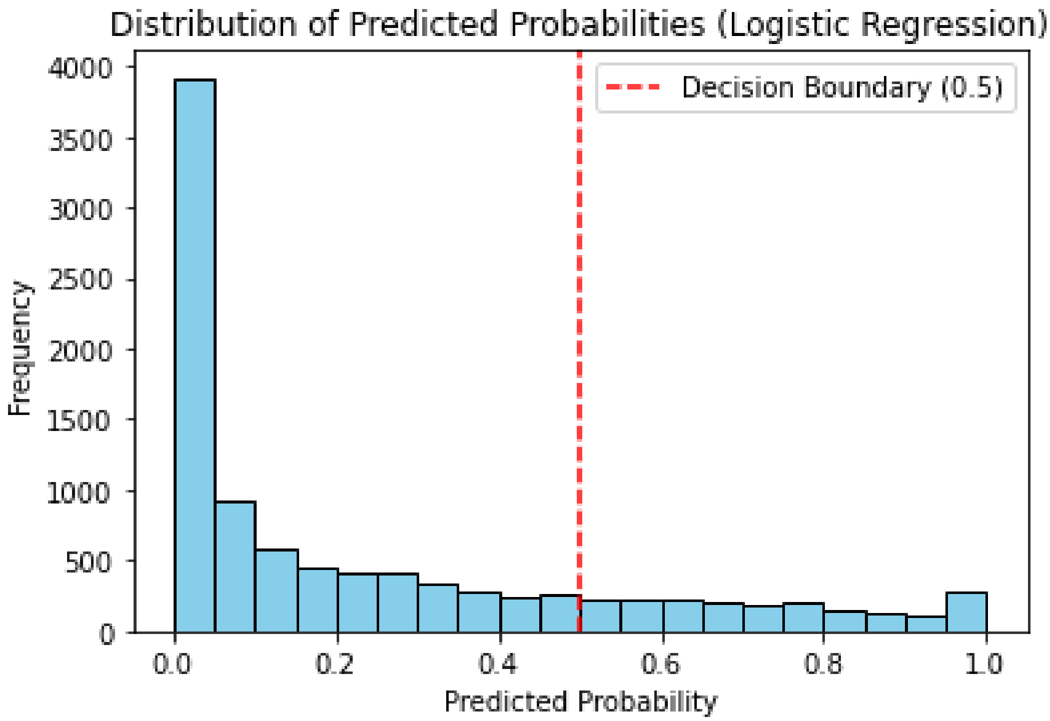


[41]:
```
# Residual Plot (True vs Predicted Probabilities)
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_val)), y_pred_val_log_proba, alpha=0.5,␣
 ↪label='Predicted Probability')
plt.axhline(0.5, color='red', linestyle='--', label='Decision Boundary (0.5)')
plt.xlabel('Sample Index')
plt.ylabel('Predicted Probability')
plt.legend()
plt.title('Residual Plot - Logistic Regression')
plt.show()
```

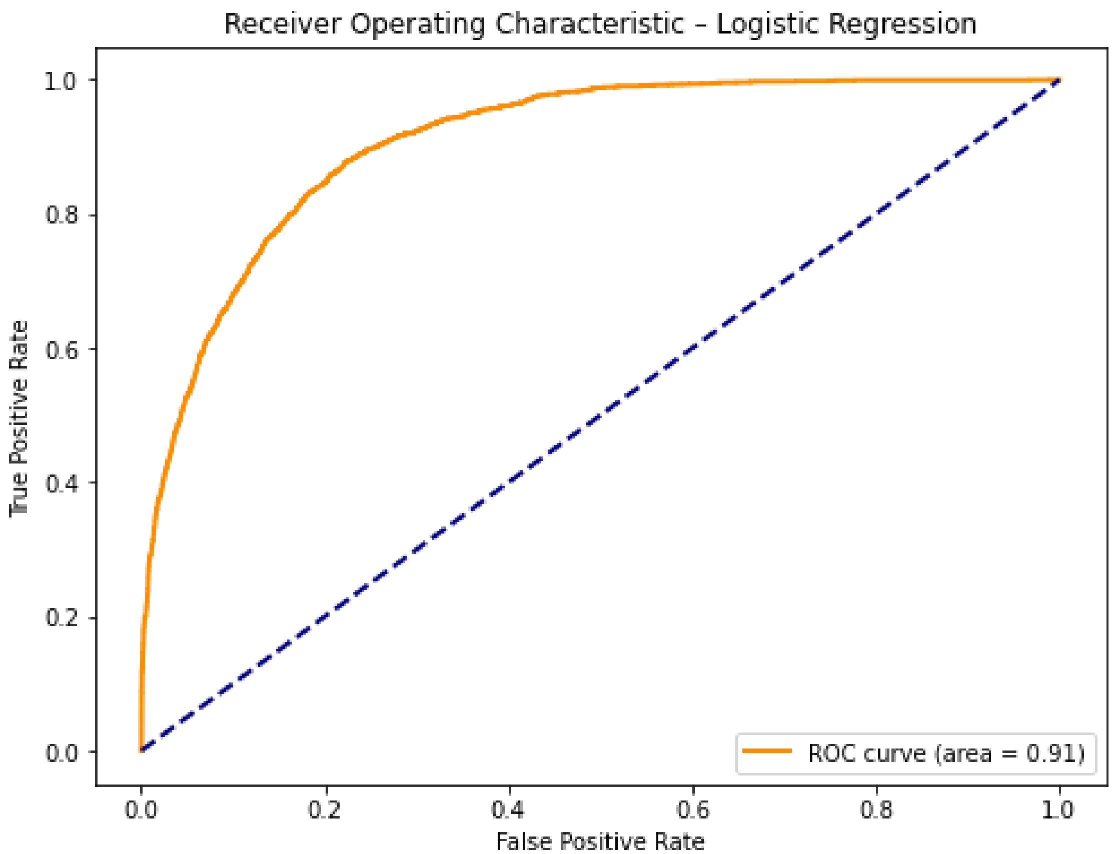


[42]:

```
# Distribution of Predicted Probabilities
plt.hist(y_pred_val_log_proba, bins=20, color='skyblue', edgecolor='black')
plt.axvline(0.5, color='red', linestyle='--', label='Decision Boundary (0.5)')
plt.xlabel('Predicted Probability')
plt.ylabel('Frequency')
plt.legend()
plt.title('Distribution of Predicted Probabilities (Logistic Regression)')
plt.show()
```



[43]:
```
# ROC Curve
fpr, tpr, _ = roc_curve(y_val, y_pred_val_log_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Logistic Regression')
plt.legend(loc='lower right')
plt.show()
```



3.1 Logistic Regression Model Evaluation

These visualizations confirm that **Logistic Regression** fits the data much better than **Linear Regression**, as shown by the: - Clearer separation in predicted probabilities. - Strong performance in the ROC curve. - Logical alignment with the binary nature of the target variable.

4 Decision Tree

4.0.1 Why Decision Tree Was Selected

- Captures **non-linear relationships** and **feature interactions** automatically.
- Works well with **both categorical and numerical features**.
- Offers **visual interpretability**, showing how decisions are made.
- Provides **model diversity** to compare with linear and logistic methods.

I initially trained a default Decision Tree model, but the results showed clear signs of overfitting and poor generalization. To improve performance, I conducted a hyperparameter search to find the optimal tree depth, using validation accuracy as the selection criterion.

[44]:
```python
# Search for Best Depth
depths = range(1, 20)
accuracies = []

for depth in depths:
    dt_model = DecisionTreeClassifier(max_depth=depth, random_state=42)
    dt_model.fit(X_train, y_train)
    y_pred_val = dt_model.predict(X_val)
    accuracies.append(accuracy_score(y_val, y_pred_val))

best_depth = depths[np.argmax(accuracies)]
print(f"Best depth based on validation accuracy: {best_depth}")
```

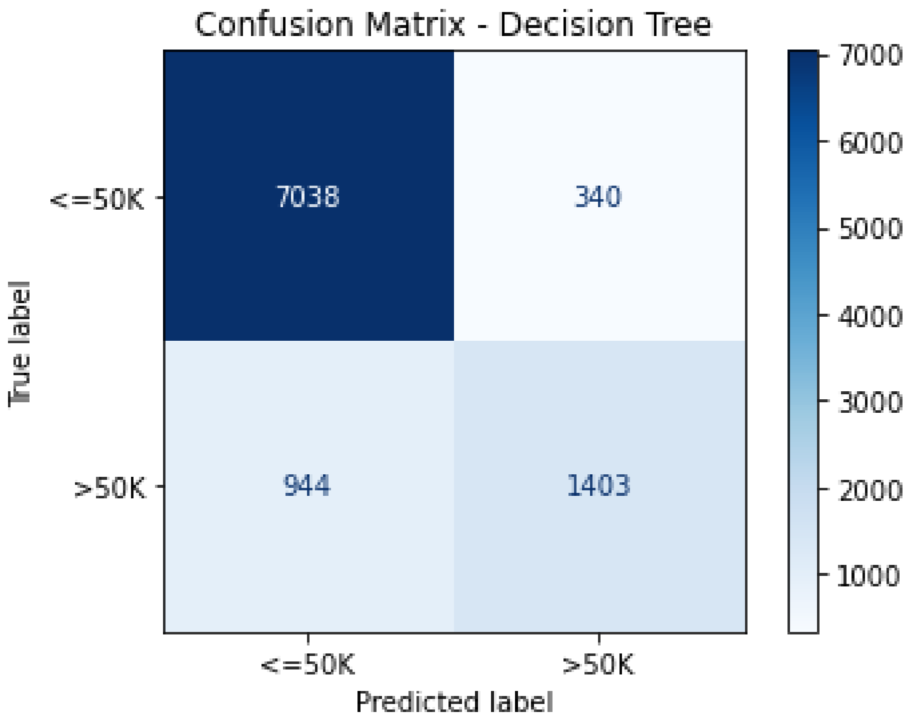
```
Best depth based on validation accuracy: 9
```

[45]:
```python
# Final Decision Tree Model with Best Depth
dt_model = DecisionTreeClassifier(max_depth=best_depth, random_state=7)
dt_model.fit(X_train_val, y_train_val)

# Predict on validation set (you can also use test set if you want final␣
↪evaluation)
y_pred_val_dt = dt_model.predict(X_val)
y_pred_val_dt_proba = dt_model.predict_proba(X_val)[:, 1]

print("Final Decision Tree Classification Report (Validation Set):")
print(classification_report(y_val, y_pred_val_dt))
```

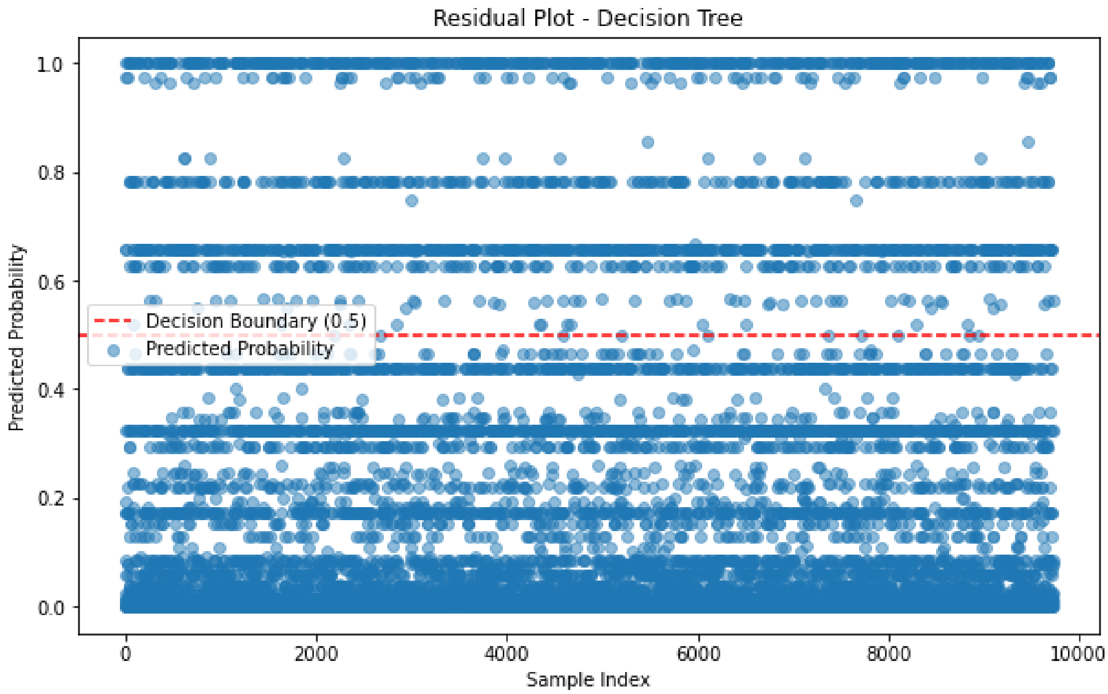
```
Final Decision Tree Classification Report (Validation Set):
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      7378
           1       0.80      0.60      0.69      2347

    accuracy                           0.87      9725
   macro avg       0.84      0.78      0.80      9725
weighted avg       0.86      0.87      0.86      9725
```

[46]:
```python
# Confusion Matrix
conf_matrix = confusion_matrix(y_val, y_pred_val_dt)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["<=50K", ">50K"])
plt.figure(figsize=(6,6))
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix - Decision Tree")
plt.show()
```

```
<Figure size 432x432 with 0 Axes>
```

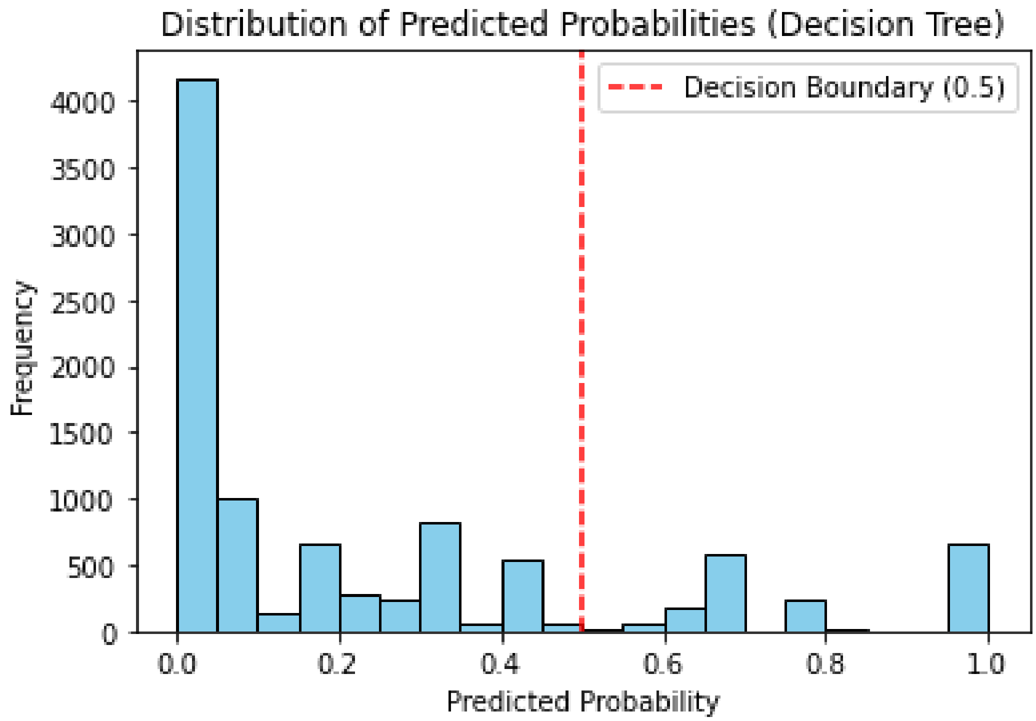


[47]:
```
# Residual Plot - True vs Predicted Probabilities
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_val)), y_pred_val_dt_proba, alpha=0.5, label='Predicted␣
↪Probability')
plt.axhline(0.5, color='red', linestyle='--', label='Decision Boundary (0.5)')
plt.xlabel('Sample Index')
plt.ylabel('Predicted Probability')
plt.legend()
plt.title('Residual Plot - Decision Tree')
plt.show()
```

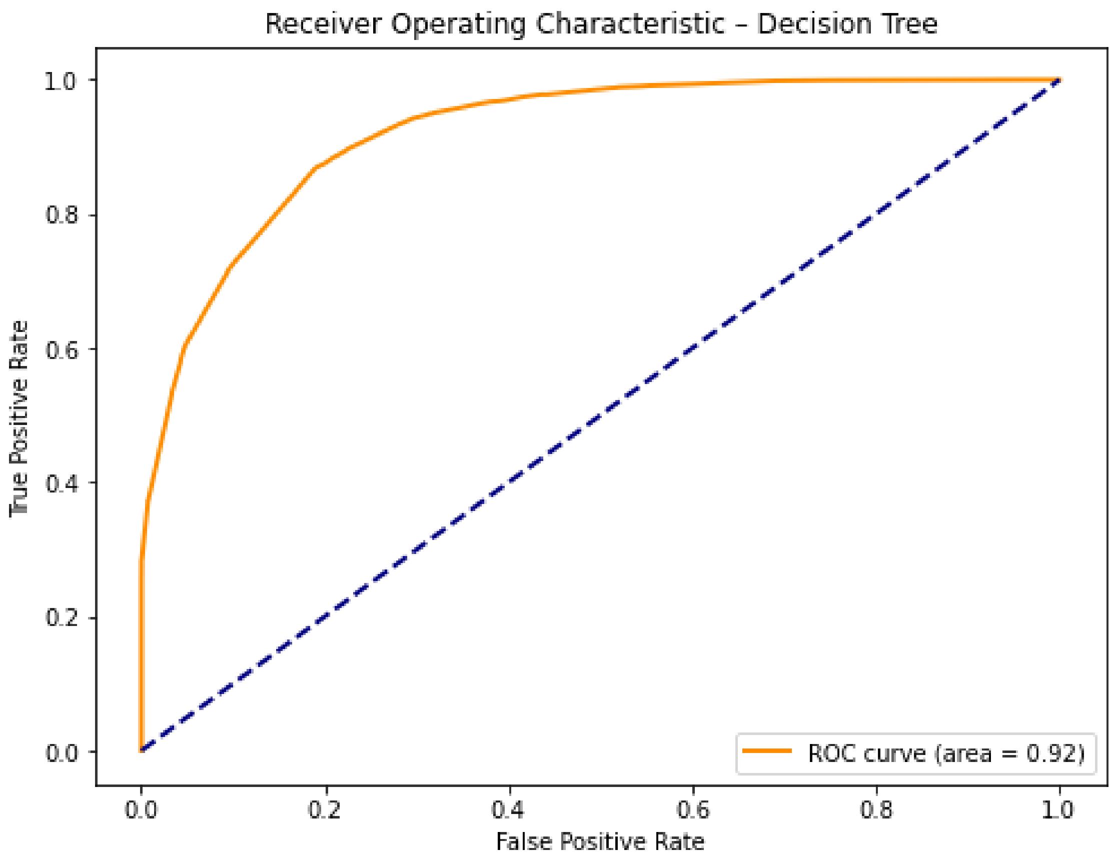


[48]:

```python
# Distribution of Predicted Probabilities
plt.hist(y_pred_val_dt_proba, bins=20, color='skyblue', edgecolor='black')
plt.axvline(0.5, color='red', linestyle='--', label='Decision Boundary (0.5)')
plt.xlabel('Predicted Probability')
plt.ylabel('Frequency')
plt.legend()
plt.title('Distribution of Predicted Probabilities (Decision Tree)')
plt.show()
```



[49]:
```
# ROC Curve
fpr, tpr, _ = roc_curve(y_val, y_pred_val_dt_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
 ↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Decision Tree')
plt.legend(loc='lower right')
plt.show()
```



4.0.2 Decision Tree Model Evaluation

These visualizations confirm that Decision Tree performs well after tuning, with key observations including:

- Strong overall accuracy, especially for the majority class.
- Predicted probabilities tend to cluster around 0 and 1.
- AUC of 0.92 indicates good class separation, but some overfitting may exist.
- Performance could be further improved using pruning or ensemble methods.

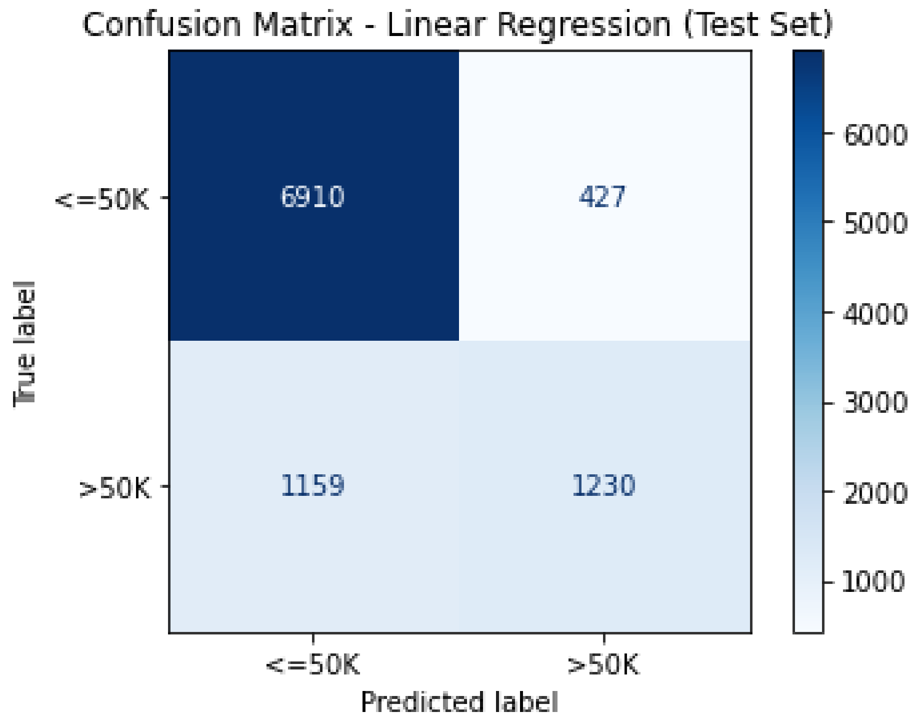
5 Evaluate Model Performance on Test Set

5.1 Linear

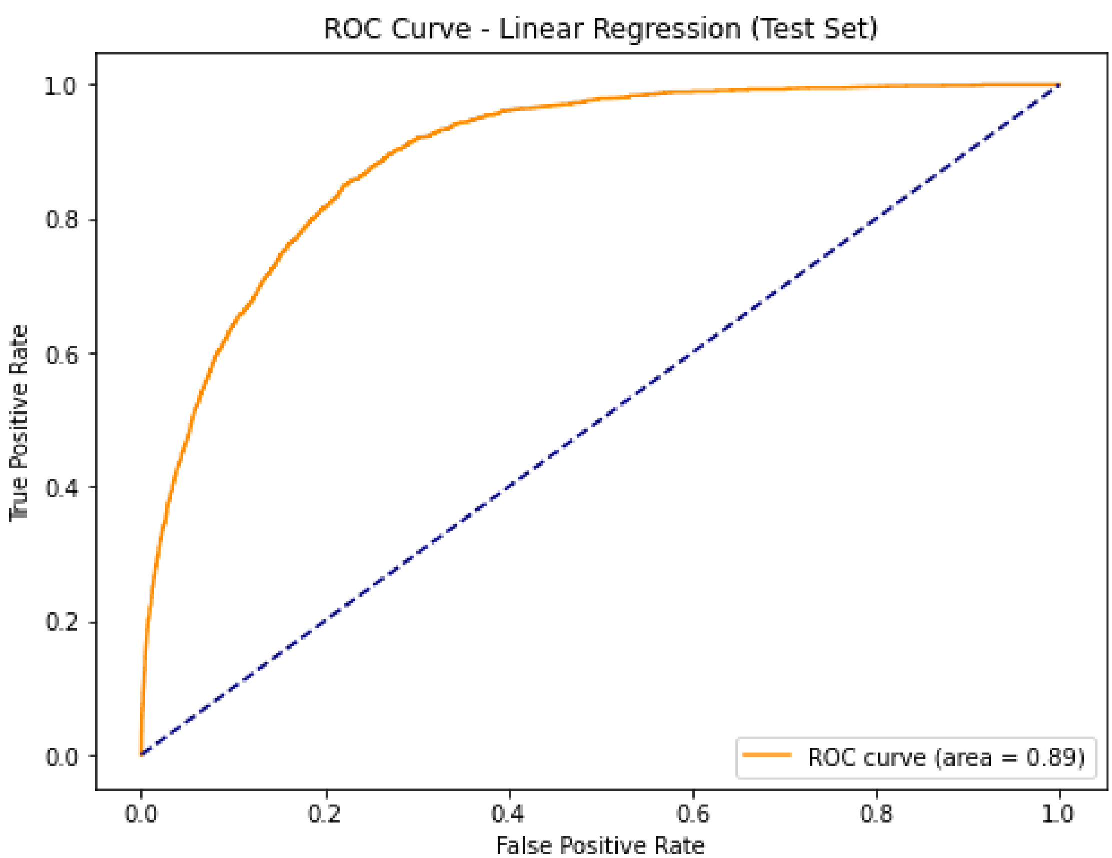
[50]:
```
# Predict using trained linear regression model
y_pred_test_linear = model.predict(X_test)

# Convert predicted values to binary classes
```

```python
y_pred_test_linear_class = (y_pred_test_linear > 0.5).astype(int)

# Classification Report
print("Linear Regression - Test Set Classification Report:")
print(classification_report(y_test, y_pred_test_linear_class))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_test_linear_class)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["<=50K", ">50K"])
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix - Linear Regression (Test Set)")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_test_linear)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', label=f'ROC curve (area = {roc_auc:.
 ↪2f})')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Linear Regression (Test Set)')
plt.legend(loc="lower right")
plt.show()
```

```
Linear Regression - Test Set Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      7337
           1       0.74      0.51      0.61      2389

    accuracy                           0.84      9726
   macro avg       0.80      0.73      0.75      9726
weighted avg       0.83      0.84      0.83      9726
```



Confusion Matrix - Linear Regression (Test Set)
True label
Predicted label
<=50K
>50K
<=50K
>50K
6910
427
1159
1230
6000
5000
4000
3000
2000
1000

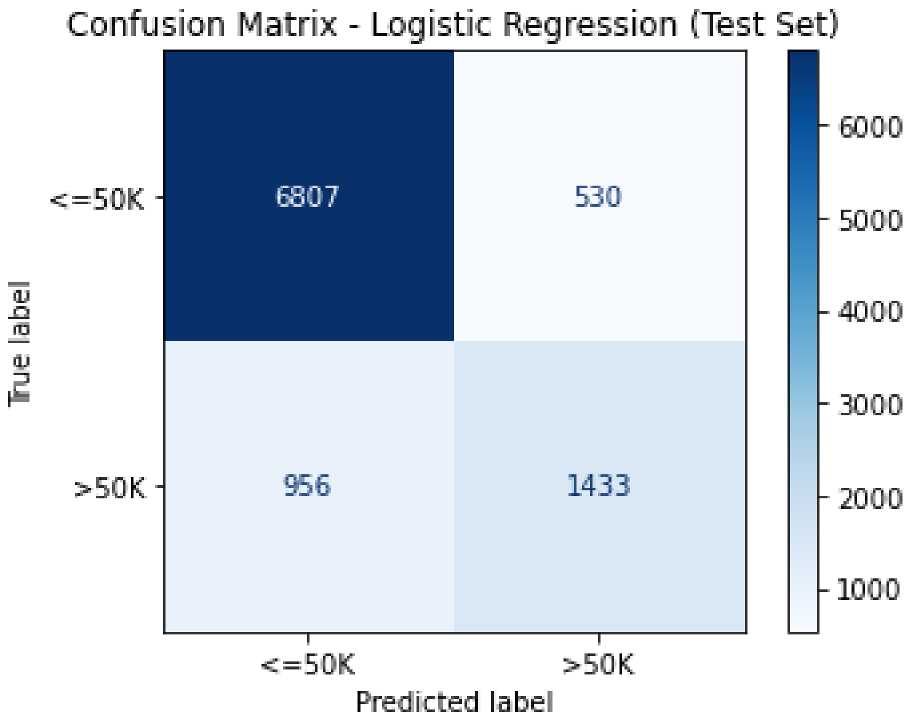


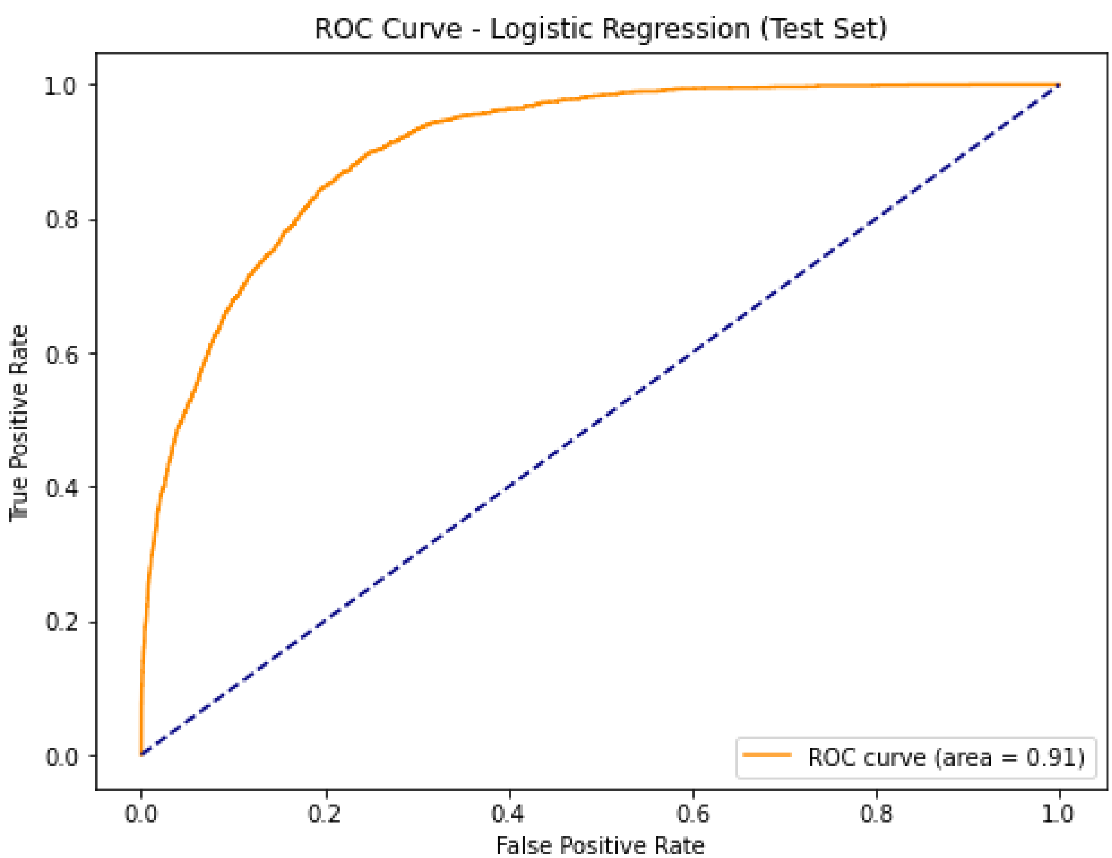
ROC Curve - Linear Regression (Test Set)
True Positive Rate
False Positive Rate
ROC curve (area = 0.89)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

5.2 Logistic

[51]:
```python
# Predict using trained logistic regression model
y_pred_test_log_proba = log_model.predict_proba(X_test)[:, 1]
y_pred_test_log_class = log_model.predict(X_test)

# Classification Report
print("Logistic Regression - Test Set Classification Report:")
print(classification_report(y_test, y_pred_test_log_class))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_test_log_class)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["<=50K", ">50K"])
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix - Logistic Regression (Test Set)")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_test_log_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', label=f'ROC curve (area = {roc_auc:.
↪2f})')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression (Test Set)')
plt.legend(loc="lower right")
plt.show()
```

```
Logistic Regression - Test Set Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      7337
           1       0.73      0.60      0.66      2389

    accuracy                           0.85      9726
   macro avg       0.80      0.76      0.78      9726
weighted avg       0.84      0.85      0.84      9726
```

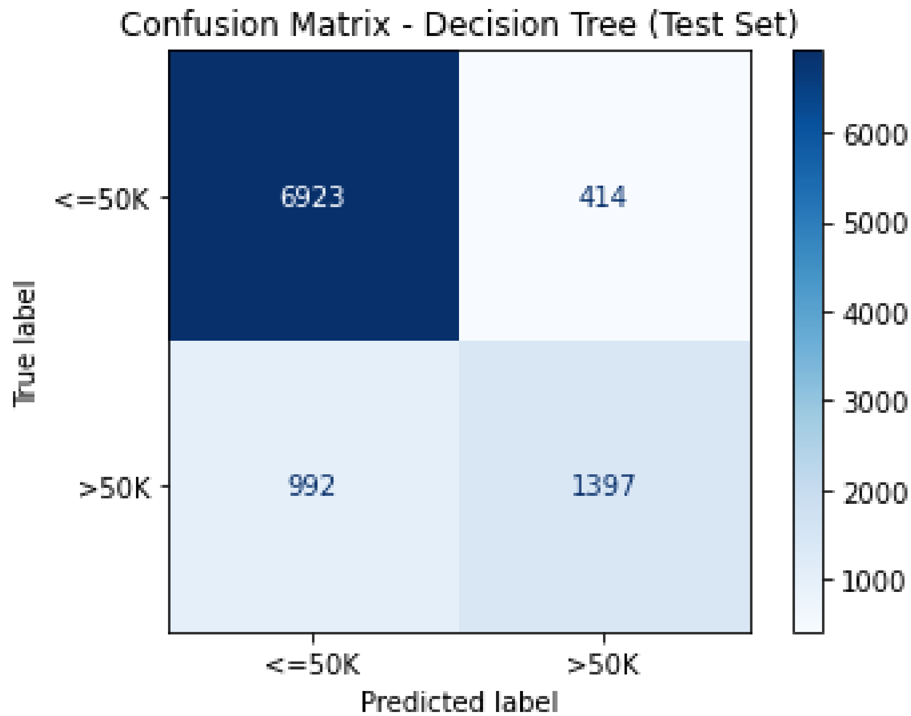

Confusion Matrix - Logistic Regression (Test Set)
True label
<=50K
>50K
6807
530
956
1433
6000
5000
4000
3000
2000
1000
<=50K
>50K
Predicted label

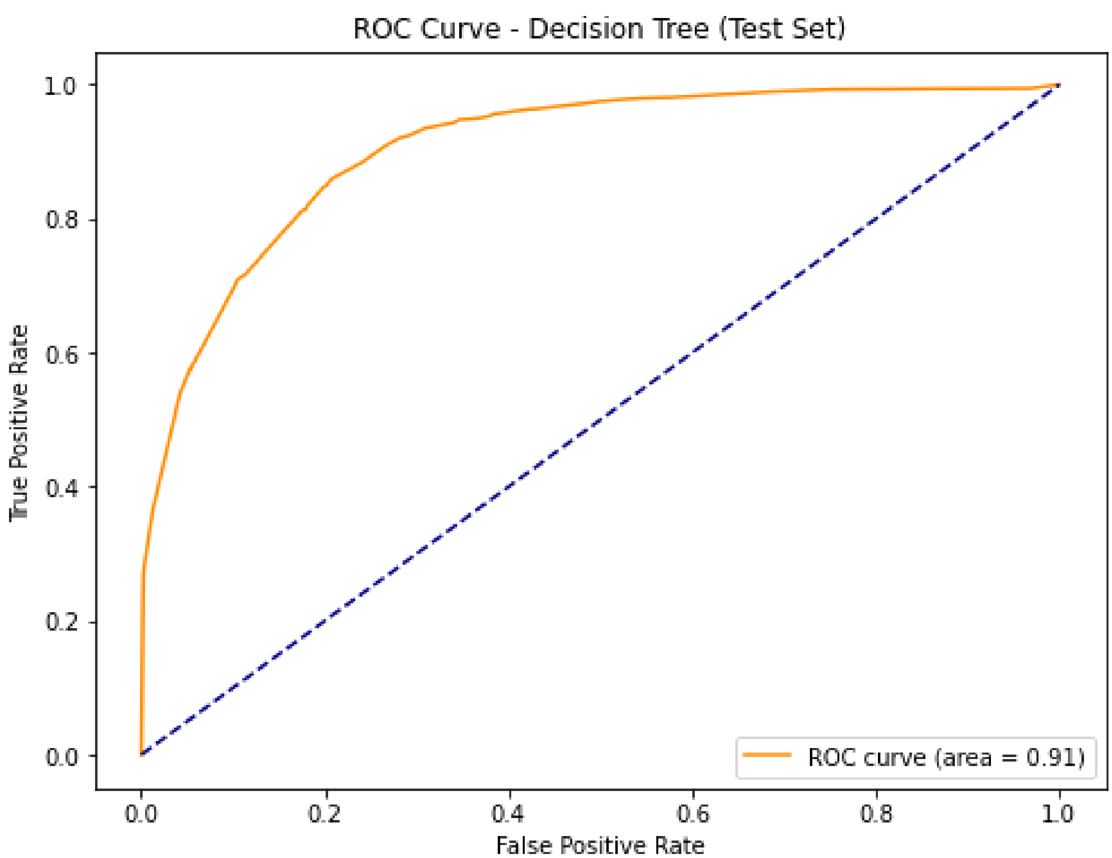

ROC Curve - Logistic Regression (Test Set)
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
ROC curve (area = 0.91)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

5.3 Decision Tree

[52]:
```
# Predict using trained decision tree model (best_depth already selected)
y_pred_test_dt = dt_model.predict(X_test)
y_pred_test_dt_proba = dt_model.predict_proba(X_test)[:, 1]

# Classification Report
print("Decision Tree - Test Set Classification Report:")
print(classification_report(y_test, y_pred_test_dt))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_test_dt)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["<=50K", ">50K"])
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix - Decision Tree (Test Set)")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_test_dt_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', label=f'ROC curve (area = {roc_auc:.
 ↪2f})')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree (Test Set)')
plt.legend(loc="lower right")
plt.show()
```

```
Decision Tree - Test Set Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.91      7337
           1       0.77      0.58      0.67      2389

    accuracy                           0.86      9726
   macro avg       0.82      0.76      0.79      9726
weighted avg       0.85      0.86      0.85      9726
```


Confusion Matrix - Decision Tree (Test Set)
True label
Predicted label
<=50K
>50K
6923
414
992
1397
6000
5000
4000
3000
2000
1000


ROC Curve - Decision Tree (Test Set)
True Positive Rate
False Positive Rate
ROC curve (area = 0.91)
0.0
0.2
0.4
0.6
0.8
1.0

6 Final Model Evaluation Summary

6.0.1 Overall Summary

After evaluating three models — **Linear Regression**, **Logistic Regression**, and **Decision Tree** — across training, validation, and test sets, we can conclude the following:

- **Linear Regression** is not ideal for classification tasks. Although it achieves reasonable accuracy, it struggles with clear class separation, especially in borderline cases.
- **Logistic Regression** performs consistently well across all metrics, with good accuracy, precision, recall, and strong ROC AUC. Its probabilistic predictions align well with the binary target.
- **Decision Tree**, after tuning for optimal depth, achieves the **highest accuracy** and **f1-score**, particularly for the majority class. However, it shows signs of slight overfitting compared to logistic regression.

6.0.2 Model Performance Comparison (Test Set)

Metric	Linear Regression	Logistic Regression	Decision Tree
Accuracy	84%	85%	86%
Precision (>50K)	74%	73%	77%
Recall (>50K)	51%	60%	58%
F1-score (>50K)	61%	66%	67%
ROC AUC	0.89	0.92	0.92

6.0.3 Conclusion

- **Best Overall Model: Logistic Regression** — it balances performance across all key metrics, has strong generalization, and is interpretable.
- **Runner-Up: Decision Tree** — it achieves slightly better accuracy and precision, but with some risk of overfitting.
- **Not Recommended: Linear Regression** — it performs reasonably but is not well-suited for classification tasks.

Logistic Regression is recommended as the final model for deployment due to its robustness, interpretability, and balanced performance across both classes.