In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
# !pip install missingno
from datetime import date
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler, I
```

In [2]:
```
columns = ["age", "workclass", "fnlwgt", "education", "education-num", "marita
           "occupation", "relationship", "race", "sex", "capital-gain", "capita
           "hours-per-week", "native-country", "income"]
df_train = pd.read_csv("/Users/FolahanmiIlori/Downloads/adult/adult.data", name
df_train.head()
```

Out[2]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Fe

In [3]:
```
columns = ["age", "workclass", "fnlwgt", "education", "education-num", "marita
           "occupation", "relationship", "race", "sex", "capital-gain", "capita
           "hours-per-week", "native-country", "income"]
df_test = pd.read_csv("/Users/FolahanmiIlori/Downloads/adult/adult.test", names
df_test.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Fe

In [4]:
```
df_train.drop(columns=["fnlwgt"], inplace=True)
df_test.drop(columns=["fnlwgt"], inplace=True)
```

I was not sure what fnlwgt was initially but after looking it up it seems like a number used in calculating cencus data, but isn't very useful here.

In [5]:
```
df_train.shape
```

Out[5]:
```
(32561, 14)
```

In [6]:
```
df_test.shape
```

Out[6]:
```
(16281, 14)
```

In [7]:
```
df_train.isin(["?"]).sum()
```

Out[7]:
```
age                  0
workclass         1836
education            0
education-num        0
marital-status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     583
income               0
dtype: int64
```

Noticed some missing values had a question mark instead of Nan.

In [8]:
```
df_train.replace("?", np.nan, inplace=True)
df_test.replace("?", np.nan, inplace=True)
```

Replacing all ? with Nan

In [9]:
```
print("\nMissing Values in Training Set:")
print(df_train.isnull().sum())
```

```
Missing Values in Training Set:
age                  0
workclass         1836
education            0
education-num        0
marital-status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     583
income               0
dtype: int64
```

In [10]:
```
print("Training Data Info:")
df_train.info()
print("\nTest Data Info:")
df_test.info()
```

```
Training Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       30725 non-null  object
 2   education       32561 non-null  object
 3   education-num   32561 non-null  int64 
 4   marital-status  32561 non-null  object
 5   occupation      30718 non-null  object
 6   relationship    32561 non-null  object
 7   race            32561 non-null  object
 8   sex             32561 non-null  object
 9   capital-gain    32561 non-null  int64 
 10  capital-loss    32561 non-null  int64 
 11  hours-per-week  32561 non-null  int64 
 12  native-country  31978 non-null  object
 13  income          32561 non-null  object
dtypes: int64(5), object(9)
memory usage: 3.5+ MB

Test Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16281 entries, 0 to 16280
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             16281 non-null  int64 
 1   workclass       15318 non-null  object
 2   education       16281 non-null  object
 3   education-num   16281 non-null  int64 
 4   marital-status  16281 non-null  object
 5   occupation      15315 non-null  object
 6   relationship    16281 non-null  object
 7   race            16281 non-null  object
 8   sex             16281 non-null  object
 9   capital-gain    16281 non-null  int64 
 10  capital-loss    16281 non-null  int64 
 11  hours-per-week  16281 non-null  int64 
 12  native-country  16007 non-null  object
 13  income          16281 non-null  object
dtypes: int64(5), object(9)
memory usage: 1.7+ MB
```

In [11]:

```
sensitive_attributes = ['sex', 'race', 'age', 'marital-status']
key_features = ['education', 'workclass', 'occupation','hours-per-week','capit
target_variable = 'income'
```

In [12]:

```
print(df_train['income'].value_counts())
```

```
income
<=50K    24720
>50K      7841
Name: count, dtype: int64
```

Almost 3/4 of the individuals in the data earn less than or equal to 50k. This disproportion could impact the model, leading to bias in predicting a lower income more often.

In [13]:
```
numerical_features = ["age", "education-num", "capital-gain", "capital-loss", '
categorical_features = ["workclass", "education", "marital-status", "occupatio
                         "relationship", "race", "sex", "native-country"]
```

In [14]:
```
print("\nSummary Statistics for Numerical Features:")
df_train.describe()
```

```
Summary Statistics for Numerical Features:
```

Out[14]:

	age	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	10.080679	1077.648844	87.303830	40.437456
std	13.640433	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	12.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

There look to be some extreme outliers in capital gain, loss, and hours per week

In [15]:
```
print("\nSummary Statistics for Categorical Features:")
df_train.describe(include=['object'])
```

```
Summary Statistics for Categorical Features:
```

Out[15]:

	workclass	education	marital-status	occupation	relationship	race	sex	native-country	incom
count	30725	32561	32561	30718	32561	32561	32561	31978	3256
unique	8	16	7	14	6	5	2	41	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States	<=50I
freq	22696	10501	14976	4140	13193	27816	21790	29170	2472

Majority of individuals are White males from the United States, showing that the data doesnt have a large representation of foreign born individuals.

In [16]:
```
df_train["income"] = df_train["income"].apply(lambda x: 1 if ">50K" in x else (
df_test["income"] = df_test["income"].apply(lambda x: 1 if ">50K" in x else 0)
```

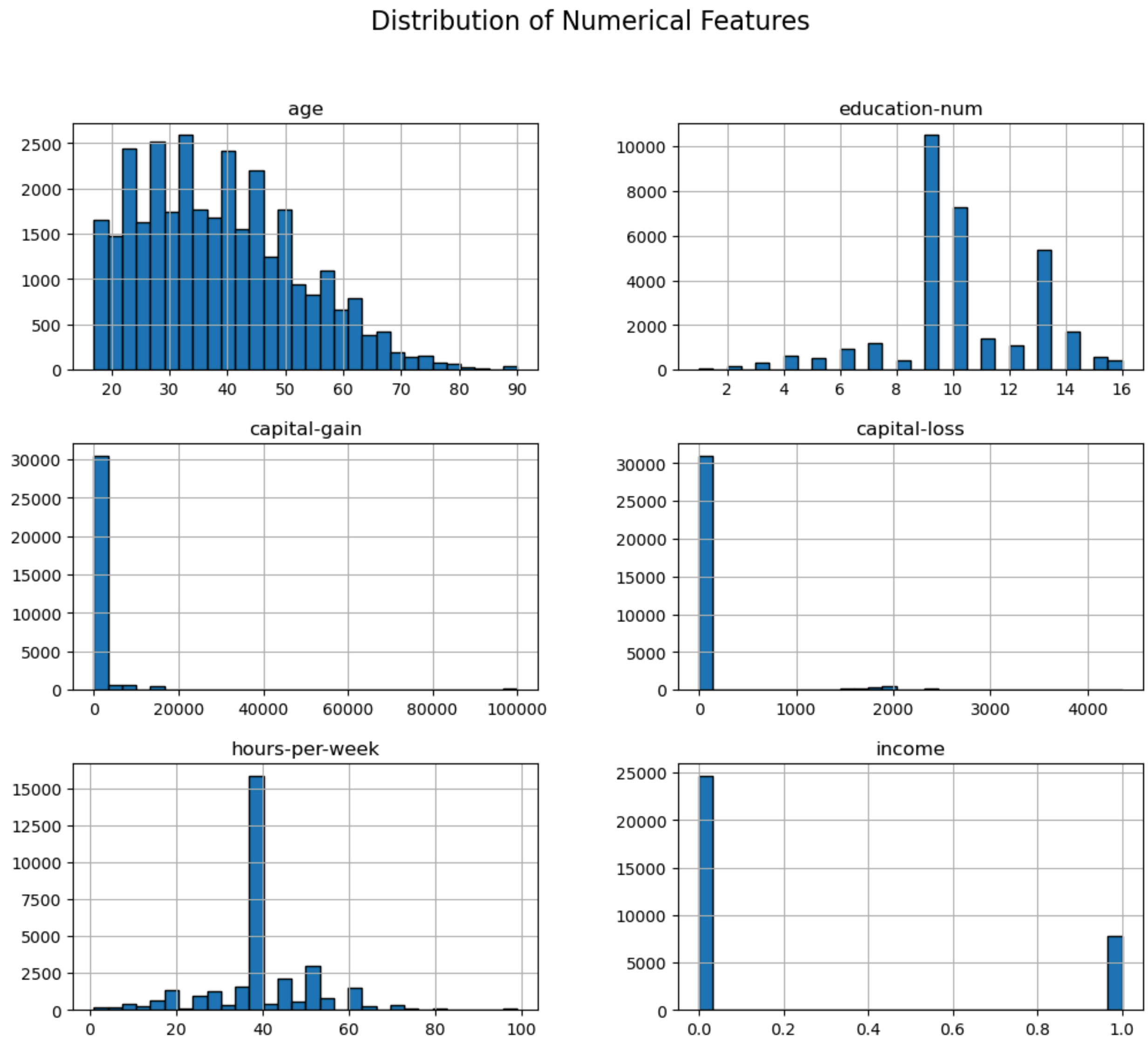
Converted income to binary (0 for <=50k, 1 for >50k)

Visualization of Numerical Features

In [17]:
```
plt.figure(figsize=(10, 5))
df_train.hist(figsize=(12, 10), bins=30, edgecolor='black')
```

```
plt.suptitle('Distribution of Numerical Features', fontsize=16)
plt.show()
```

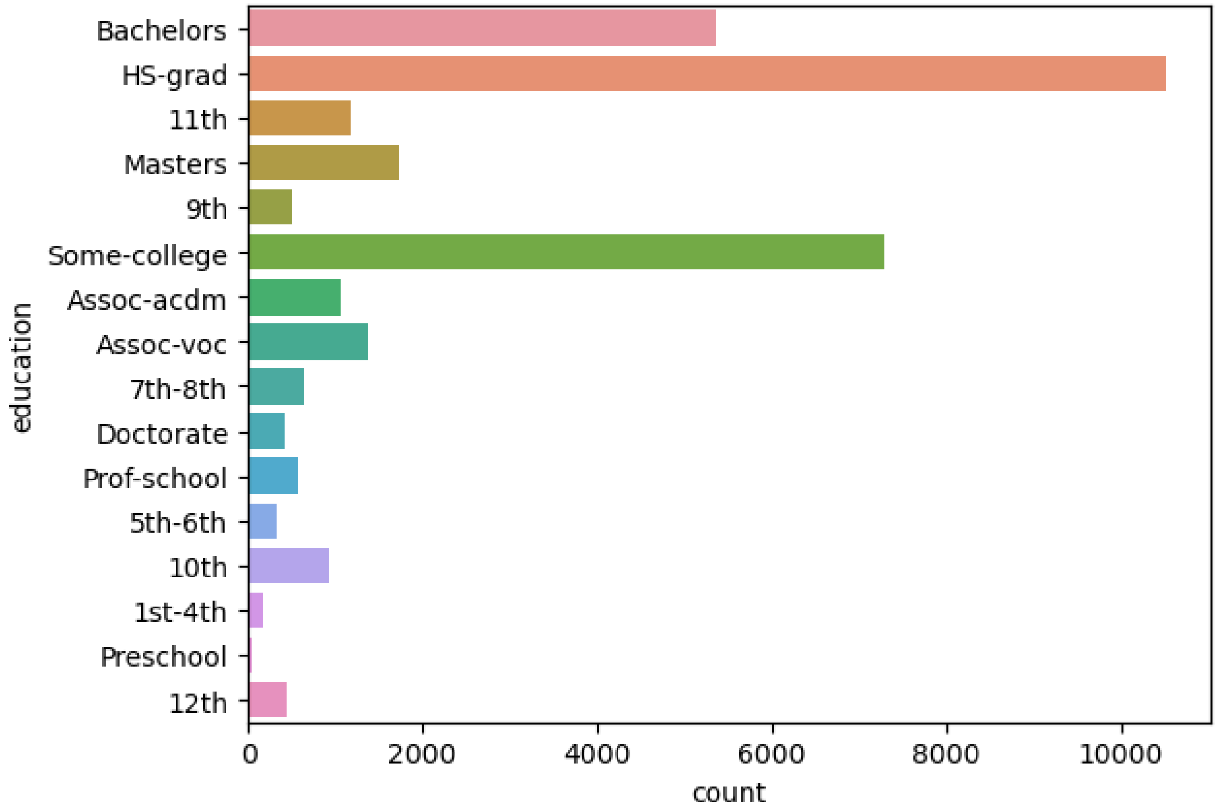
<Figure size 1000x500 with 0 Axes>



Visualization of Categorical Features

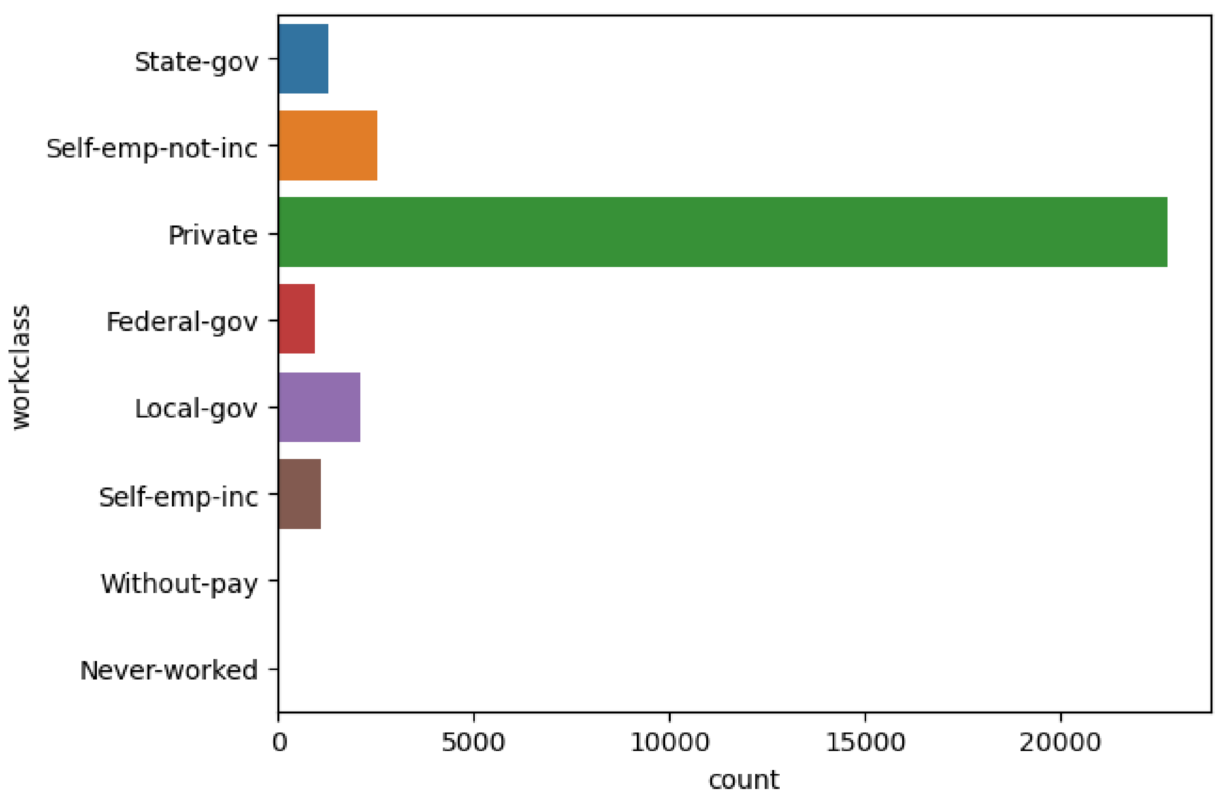
In [18]:
```
sns.countplot(y=df_train['education'])
```

Out[18]: <Axes: xlabel='count', ylabel='education'>



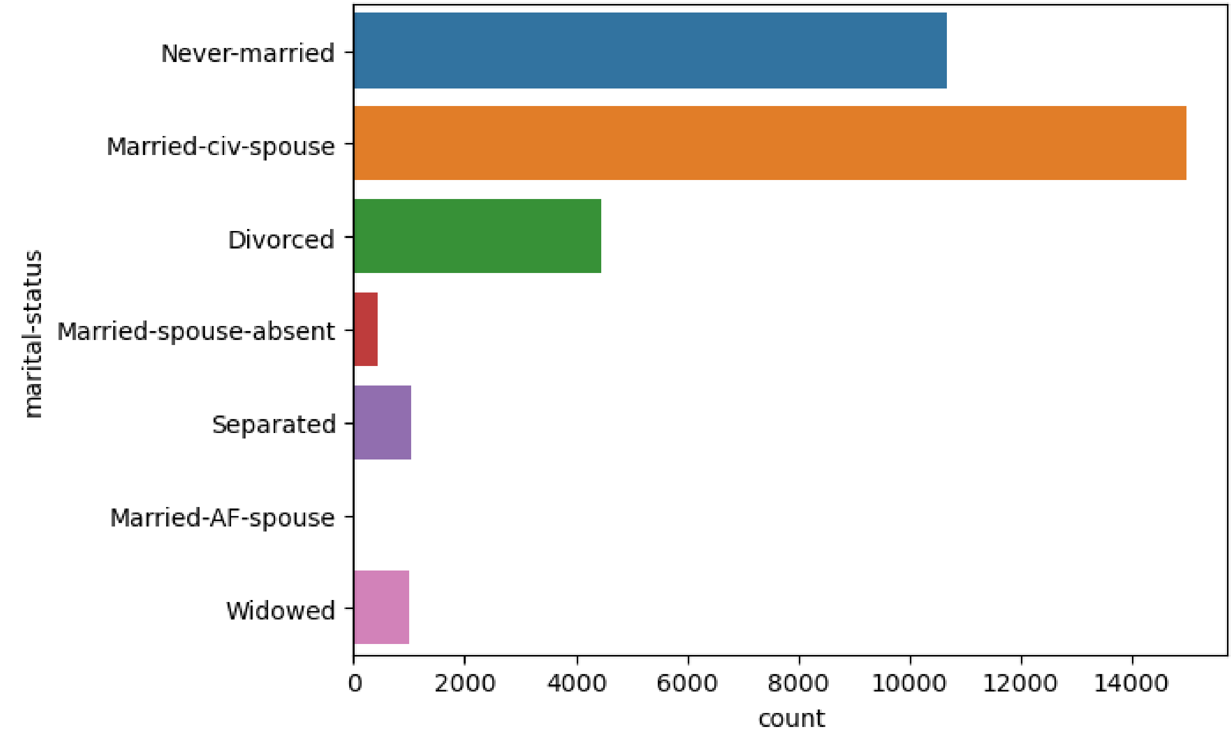
In [19]:
```
sns.countplot(y=df_train['workclass'])
```

Out[19]: <Axes: xlabel='count', ylabel='workclass'>



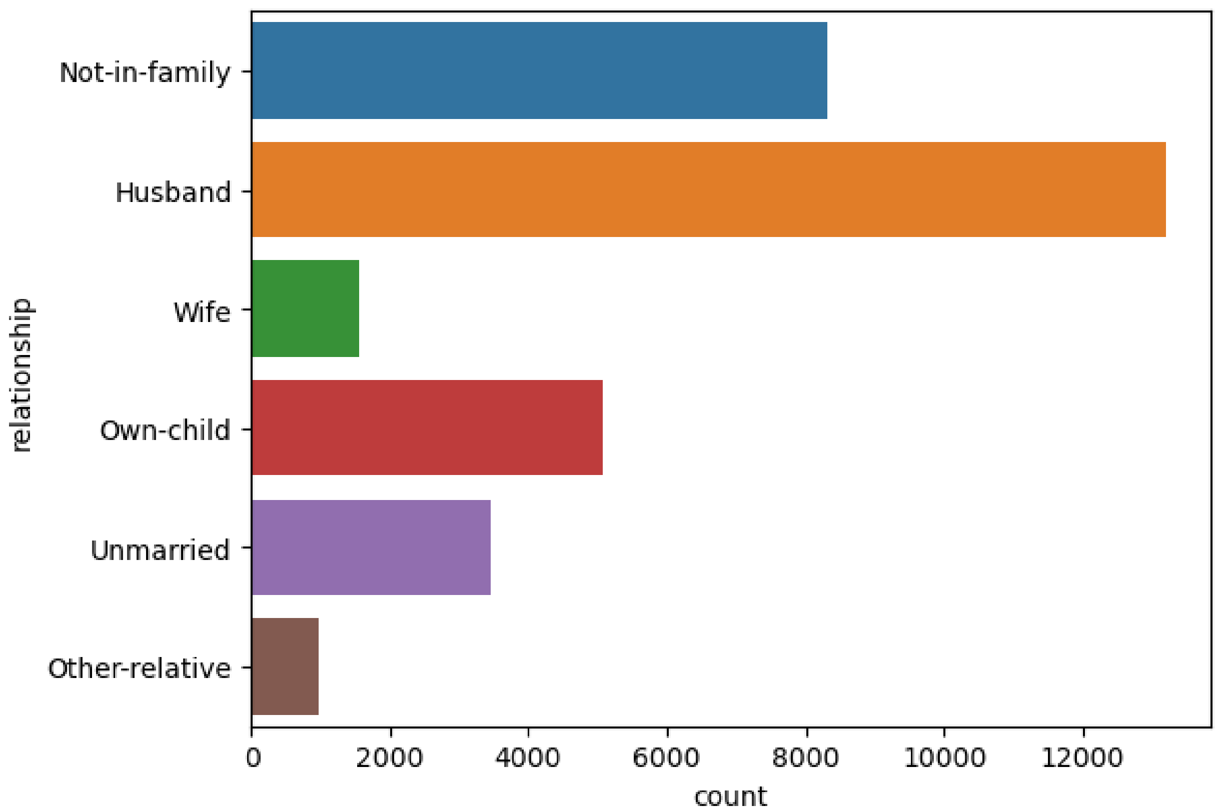
In [20]:
```
sns.countplot(y=df_train['marital-status'])
```

Out[20]: <Axes: xlabel='count', ylabel='marital-status'>



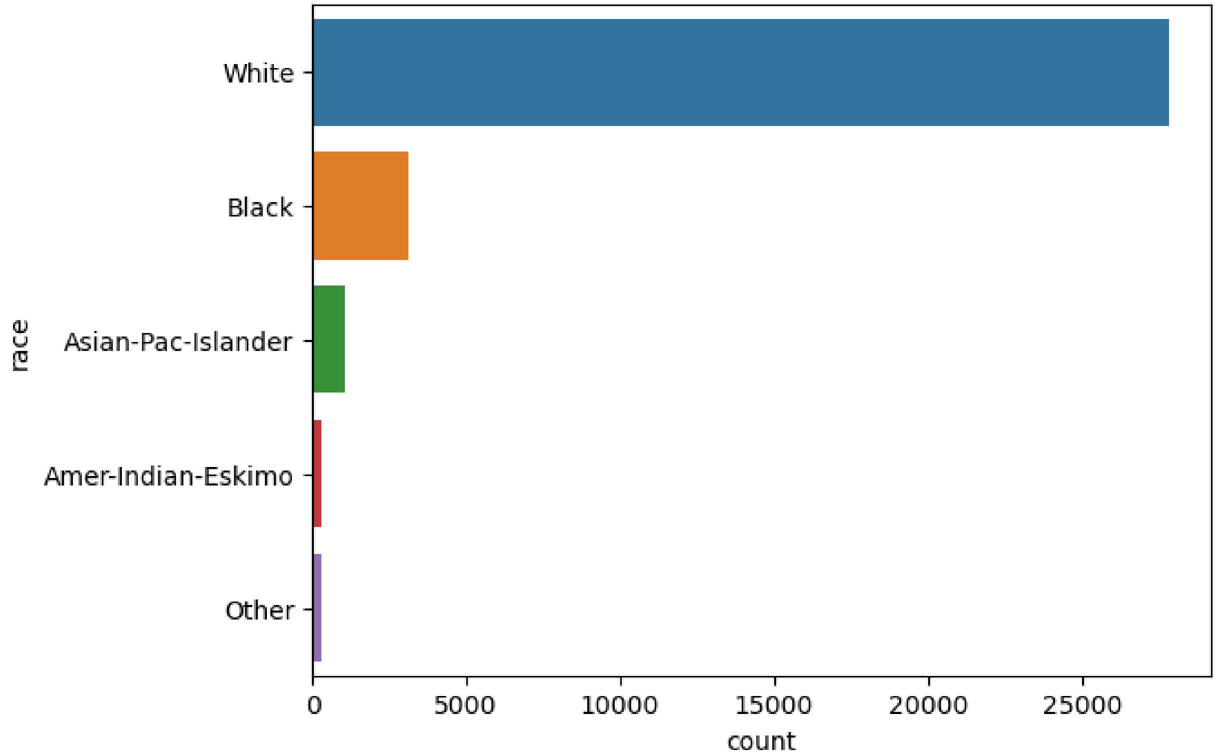
In [21]:
```
sns.countplot(y=df_train['relationship'])
```

Out[21]: <Axes: xlabel='count', ylabel='relationship'>



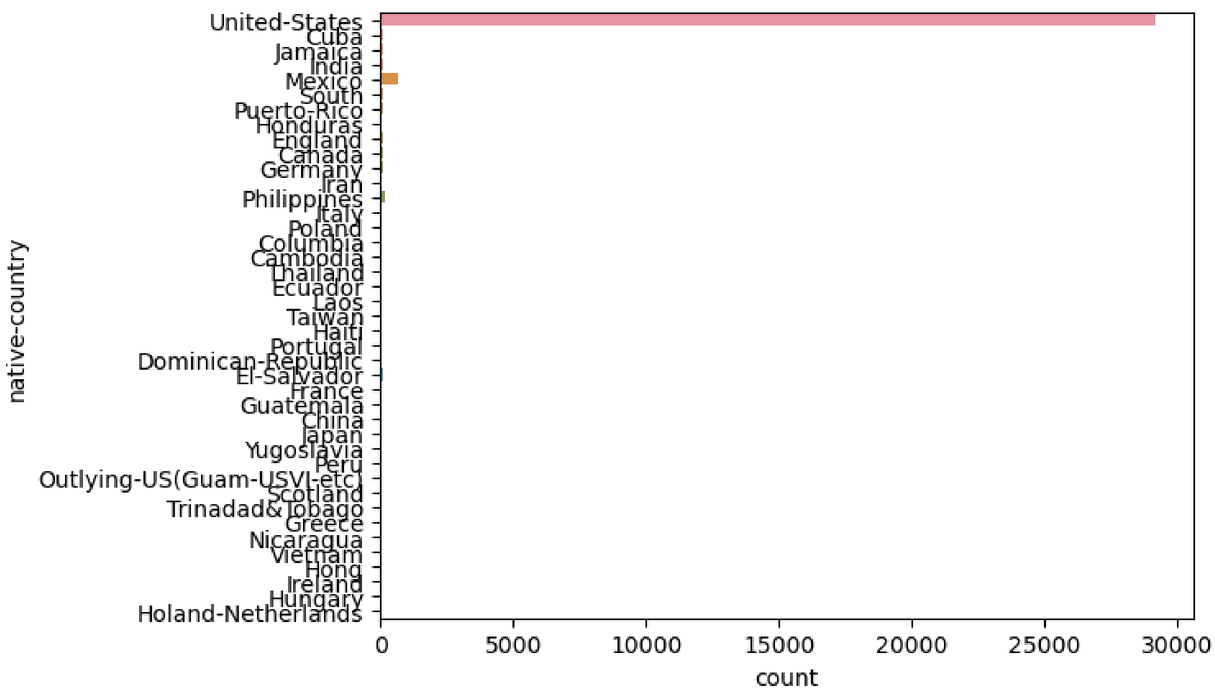
In [22]:
```
sns.countplot(y=df_train['race'])
```

Out[22]: <Axes: xlabel='count', ylabel='race'>



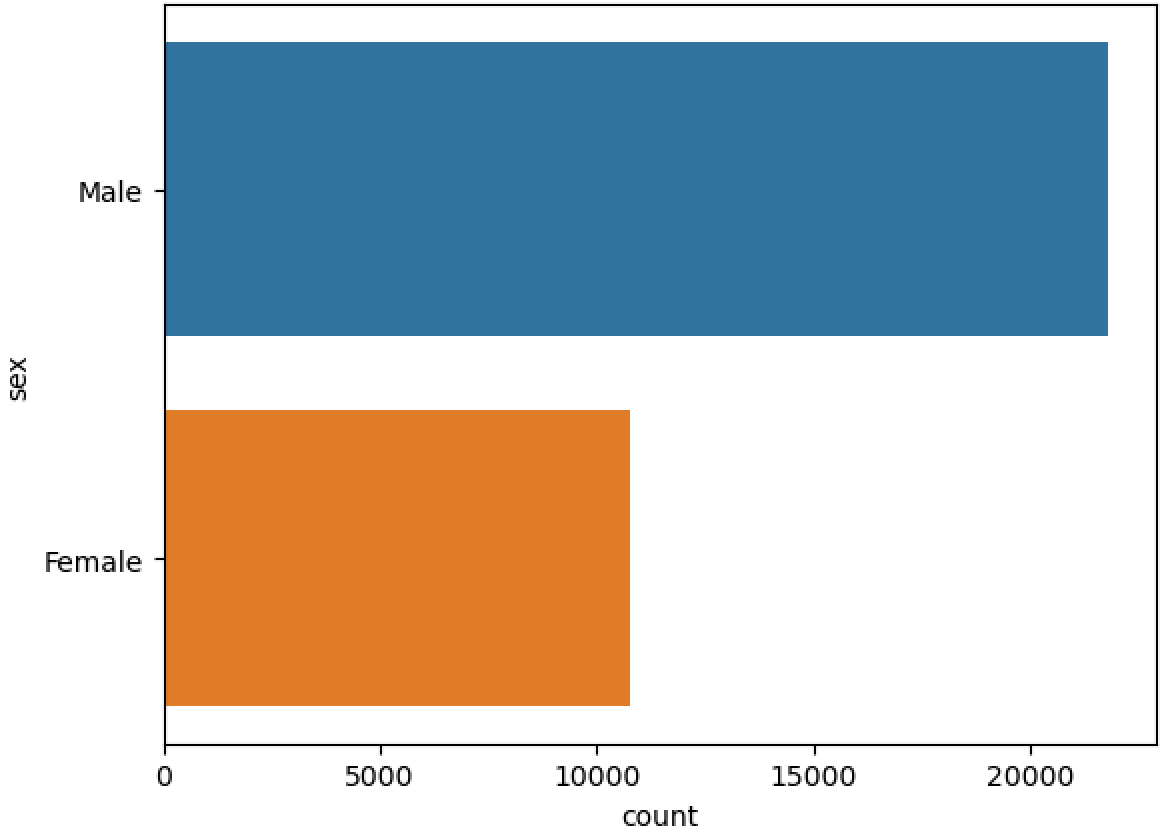
In [23]:
```
sns.countplot(y=df_train['native-country'])
```

Out[23]: <Axes: xlabel='count', ylabel='native-country'>



In [24]:
```
sns.countplot(y=df_train['sex'])
```

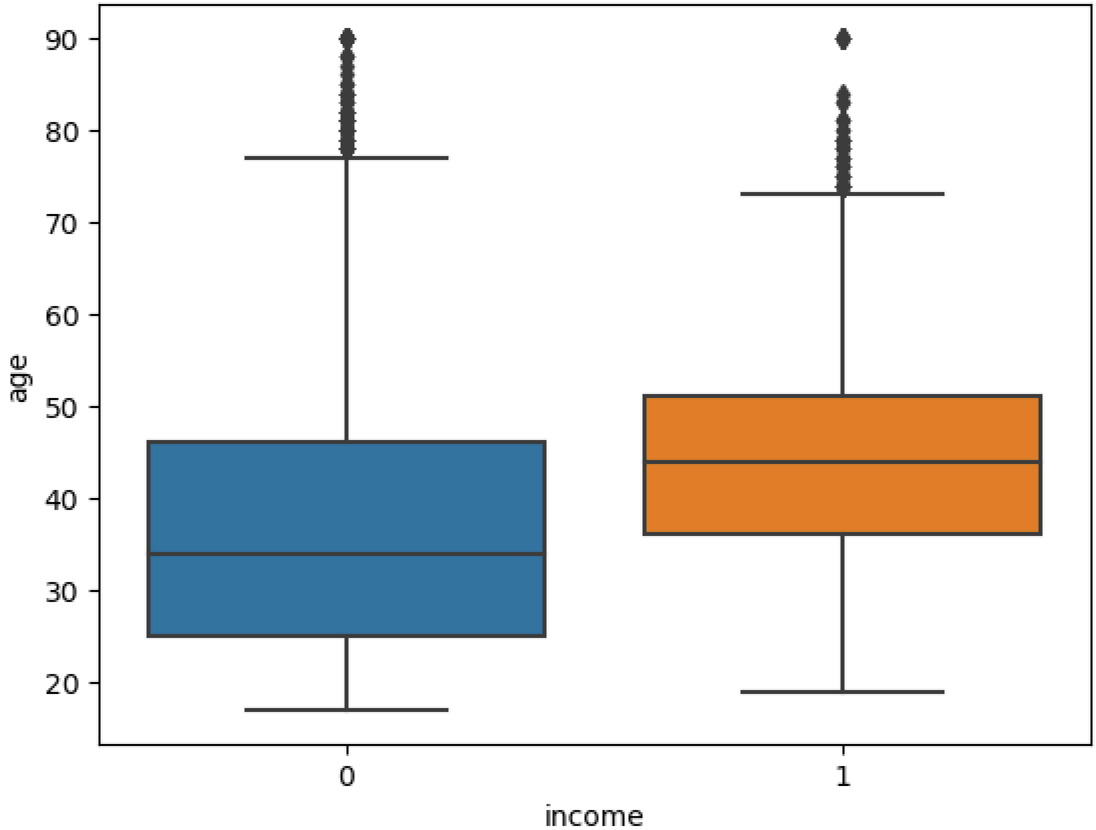
Out[24]: <Axes: xlabel='count', ylabel='sex'>



Relationship between Sensitive Atrributes and Target Variable

In [25]:
```
sns.boxplot(x=df_train['income'], y=df_train['age'])
```

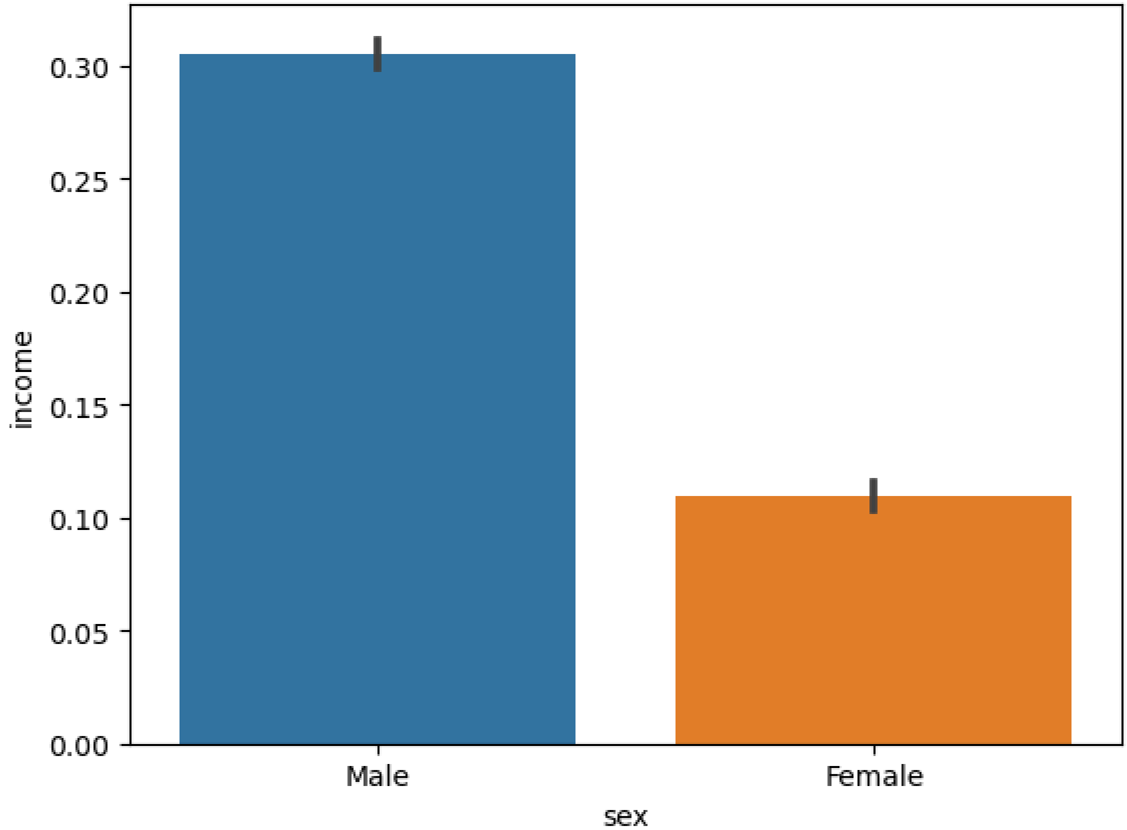
Out[25]:
```
<Axes: xlabel='income', ylabel='age'>
```



Older individuals tend to have higher income levels than younger ones, which can be seen here.

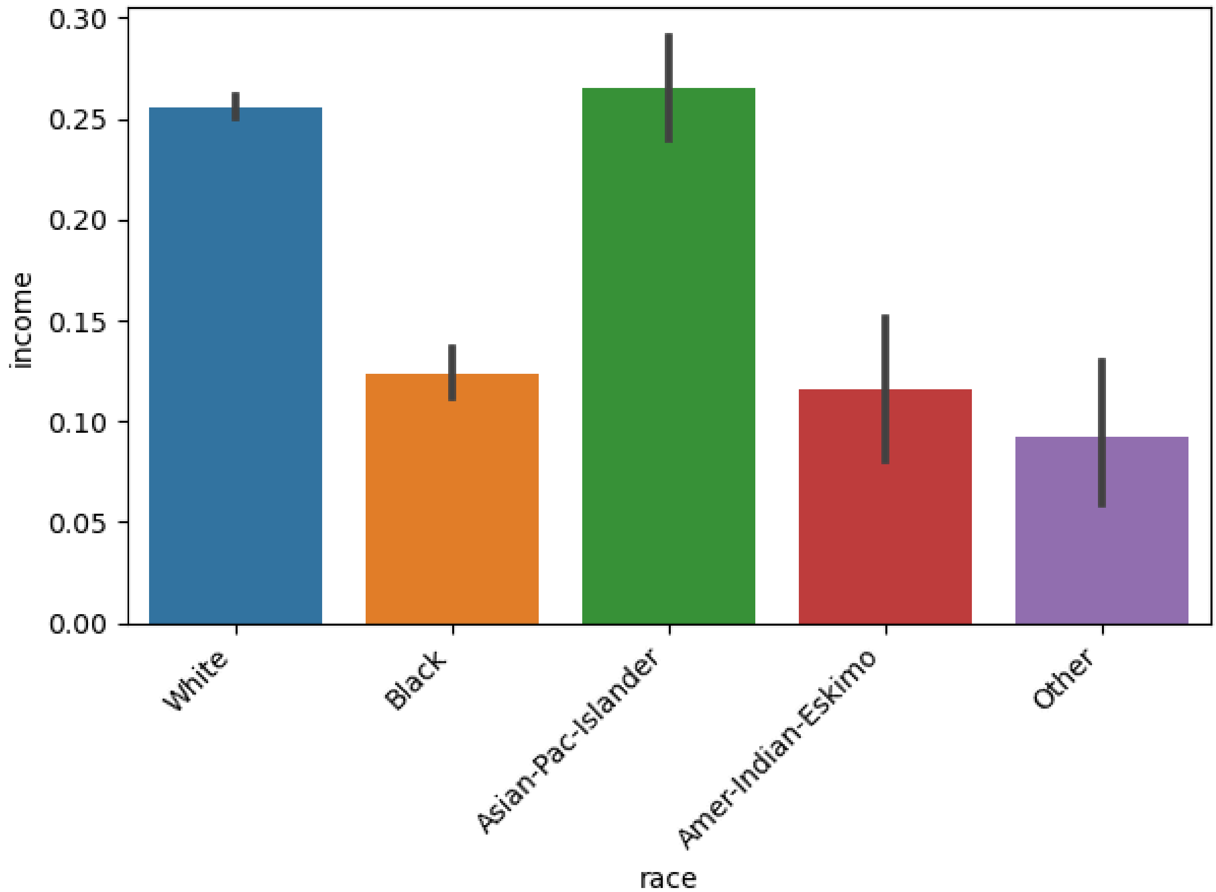
In [26]:
```
sns.barplot(x=df_train['sex'], y=df_train['income'])
```

Out[26]:
```
<Axes: xlabel='sex', ylabel='income'>
```



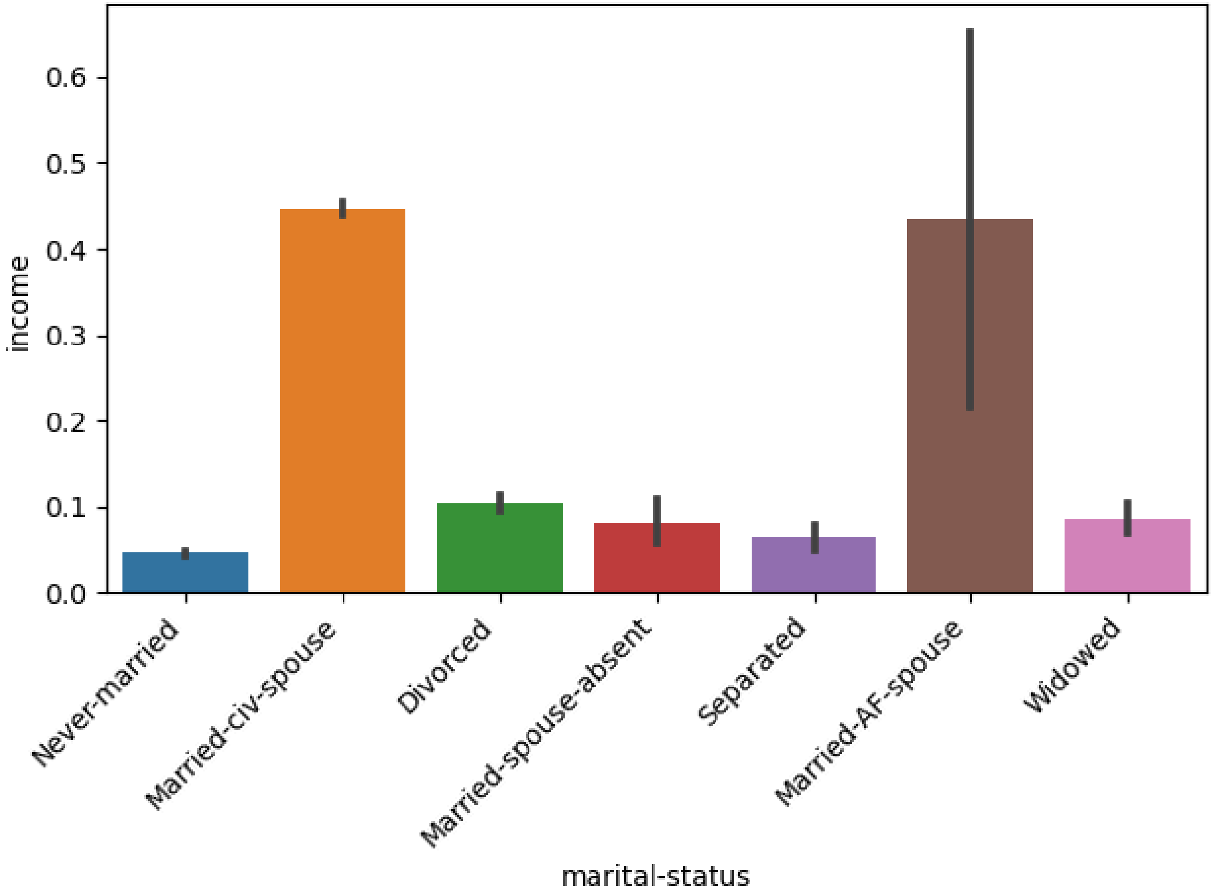
Men are significantly more likely to be in the >50k income category, the average income for men is also higher, which reflects real world data.

In [27]:
```
ax = sns.barplot(x=df_train['race'], y=df_train['income'])
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
plt.tight_layout()
```



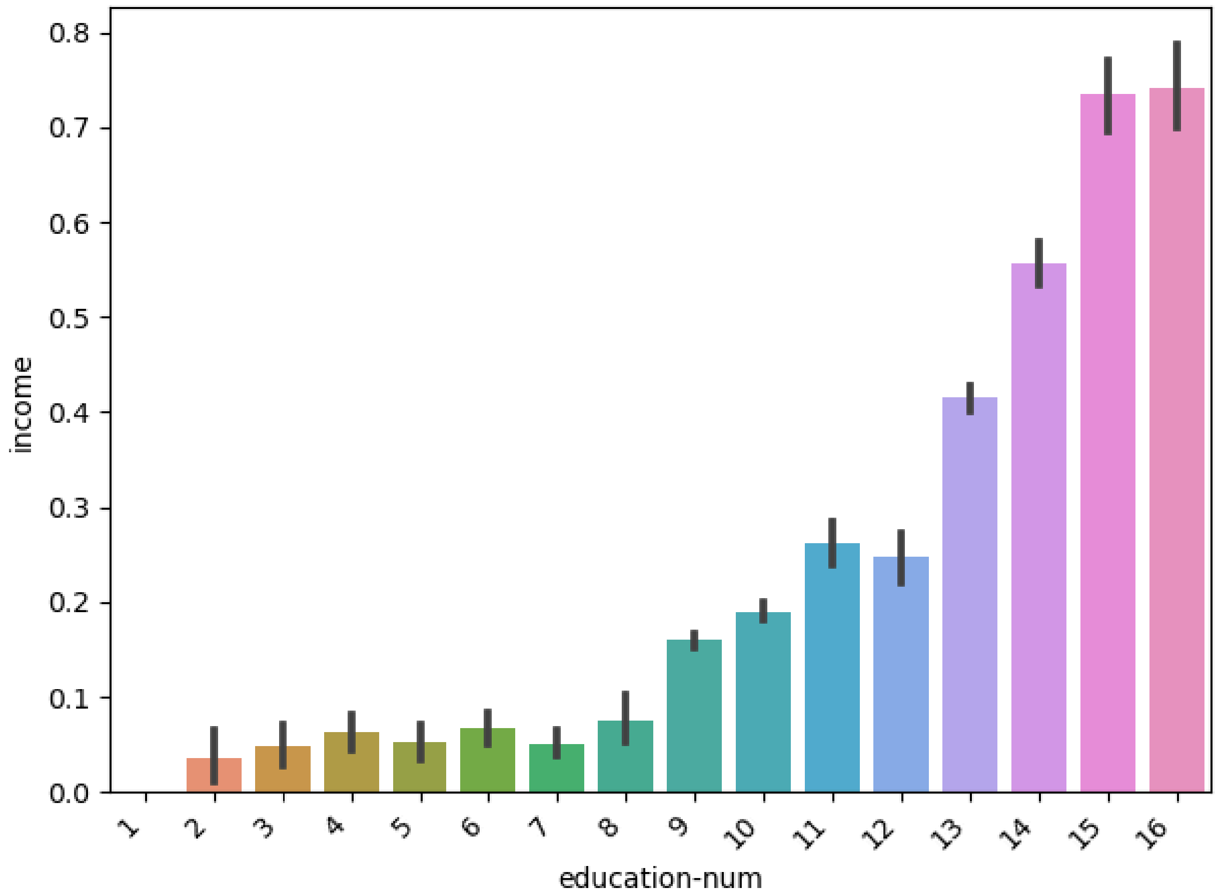
White individuals dominate the data set, but Asian-Pacific-Islanders are similar to Whites in terms of ratio between individuals over and below 50k. The other racial groups appear to have lower probablities of earning greater than 50k but further analysis is needed.

In [28]:
```
ax = sns.barplot(x=df_train['marital-status'], y=df_train['income'])

ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
plt.tight_layout()
```



Marital status is strongly correlated wih income, individuals who are maried tend to have a higher probability of earning >50k. While those with out a current partner are ore likely to be in the <= 50k group.

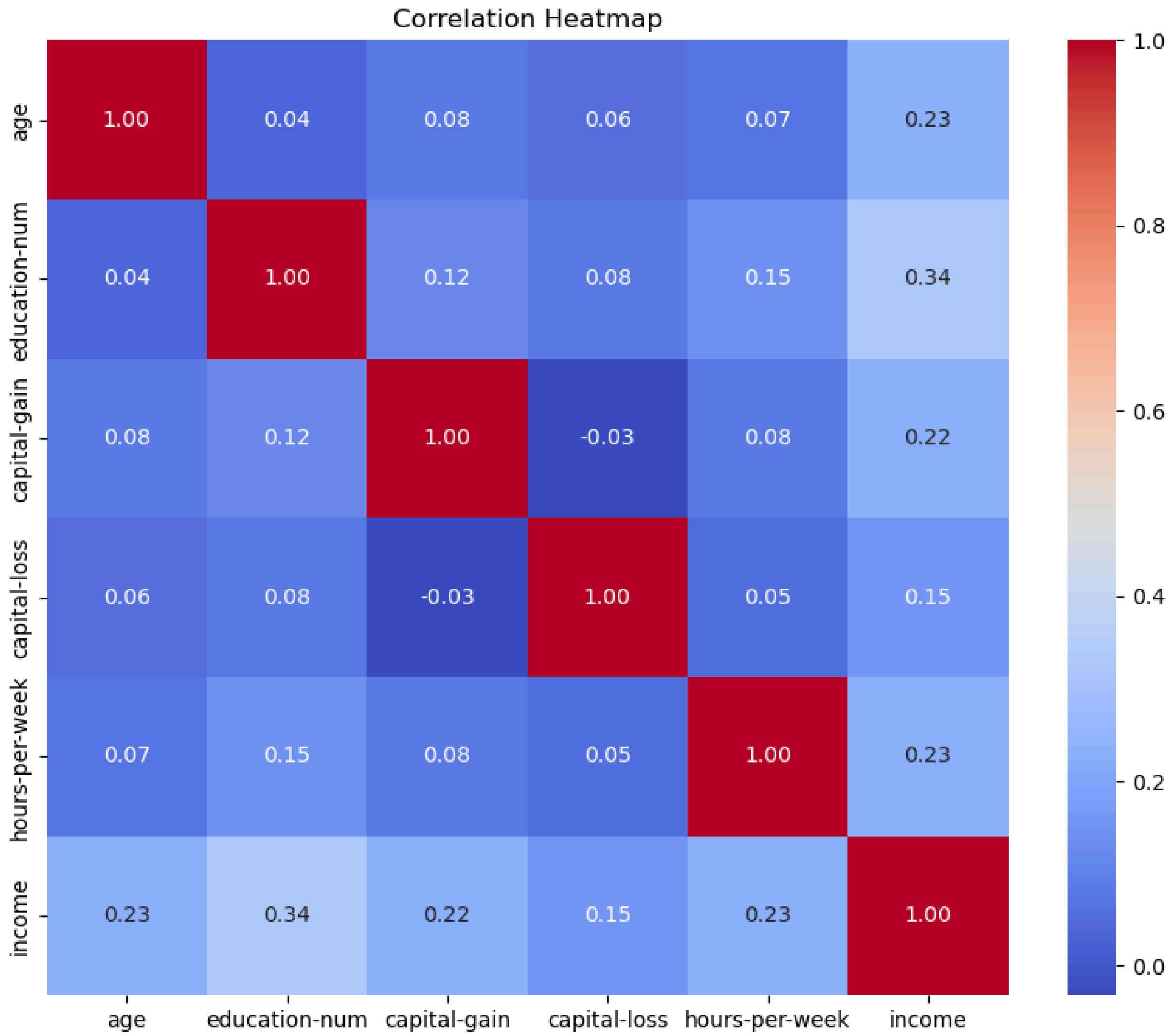
In [29]:
```
ax = sns.barplot(x=df_train['education-num'], y=df_train['income'])
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
plt.tight_layout()
```



As a expected there seems to be a correlation with higher levels of education leading to a higher pronbablity of earning >50k.

Correlation Heatmap

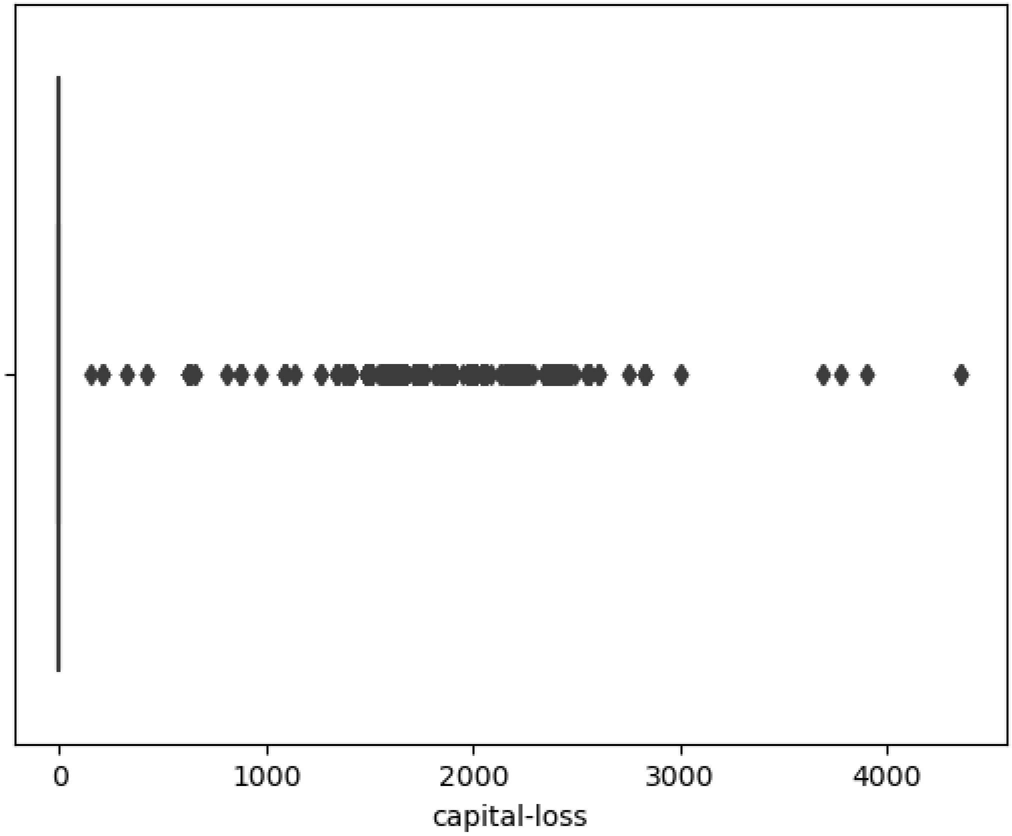
In [30]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(df_train[numerical_features + ["income"]].corr(), annot=True, cmap=
plt.title("Correlation Heatmap")
plt.show()
```



Boxplots for Outlier Detection

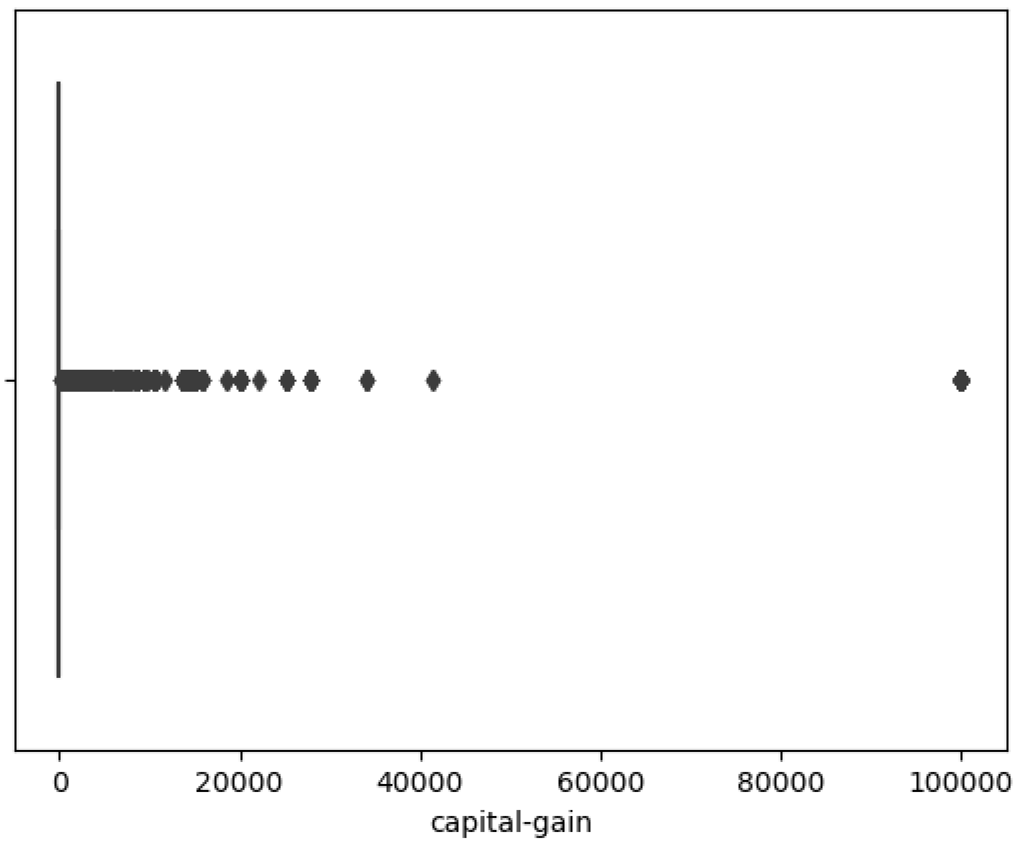
In [31]:
```
sns.boxplot(x=df_train['capital-loss'])
```

Out[31]: <Axes: xlabel='capital-loss'>



In [32]:
```
sns.boxplot(x=df_train['capital-gain'])
```

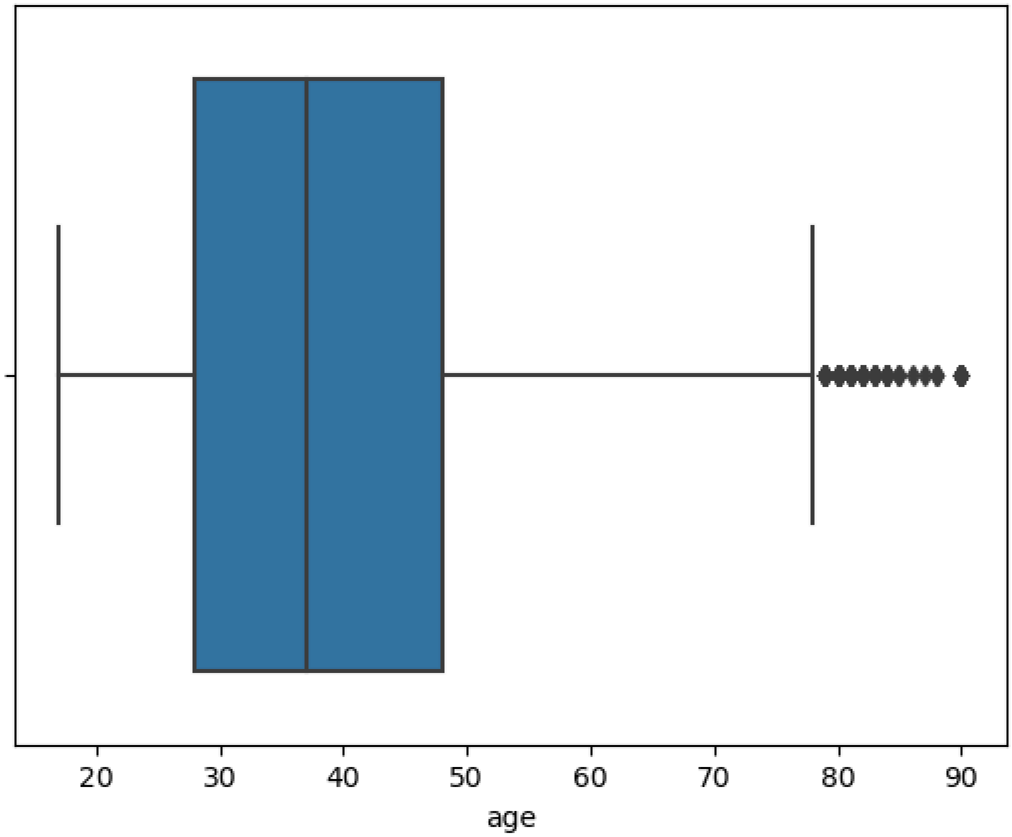
Out[32]: <Axes: xlabel='capital-gain'>



Extreme values were found in both capital loss and capital gains, with the capital gains having a few very high values, that may skew the data and have to be dealt with later.

In [33]:
```
sns.boxplot(x=df_train['age'])
```

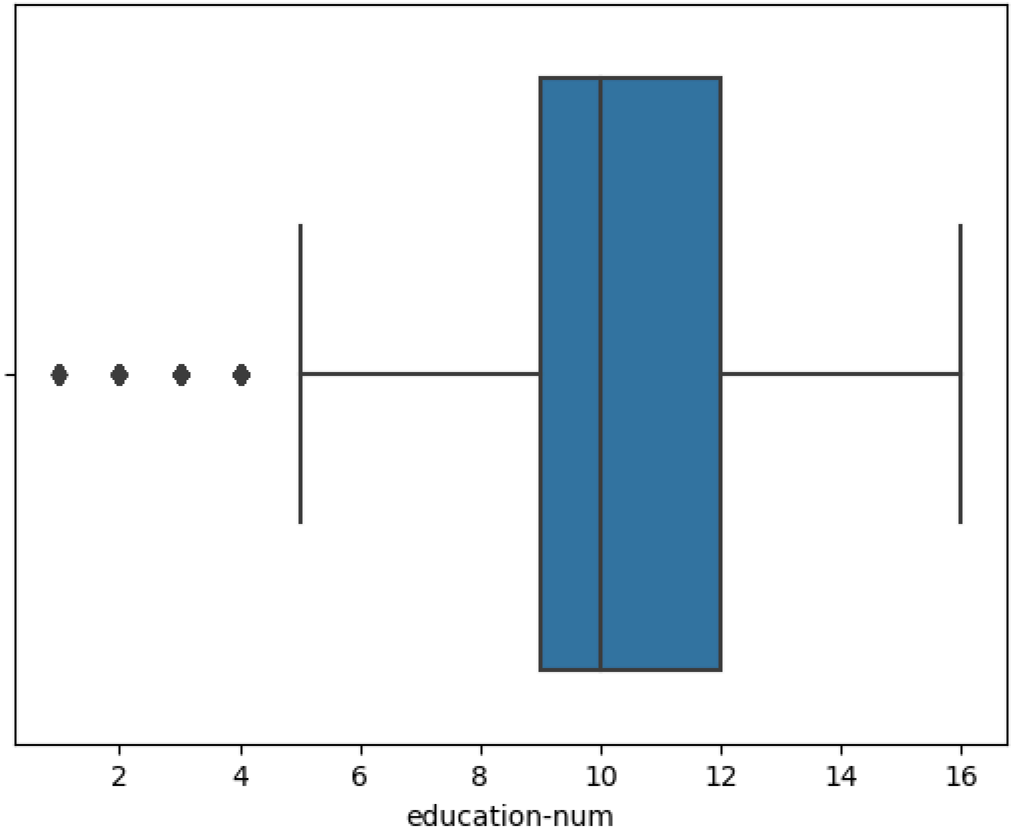
Out[33]: <Axes: xlabel='age'>



A few values were at extremes, with some very young workers and old individuals but nothing that was too unreasonable.

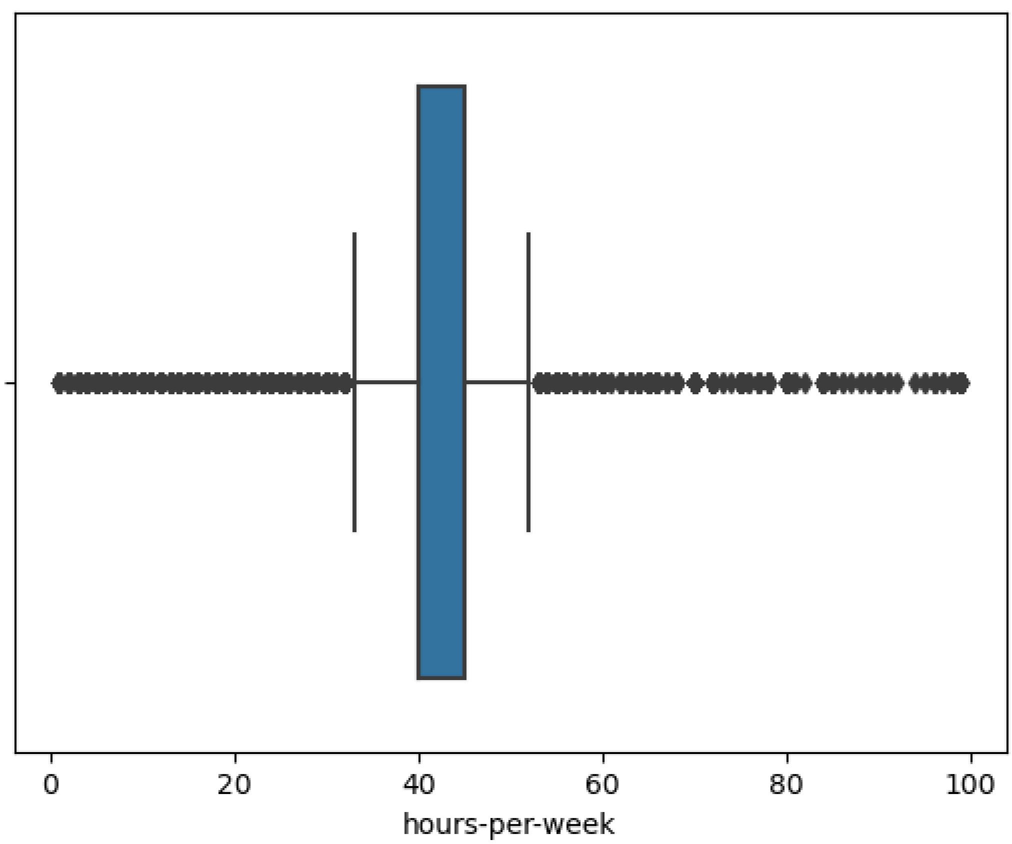
In [34]:
```
sns.boxplot(x=df_train['education-num'])
```

Out[34]: <Axes: xlabel='education-num'>



In [35]:
```
sns.boxplot(x=df_train['hours-per-week'])
```

Out[35]: <Axes: xlabel='hours-per-week'>



Some Individuals are working upwards of 80-99 hours per week which may be a little unrealistic.

In [36]:
```
categorical_features = ["workclass", "occupation", "native-country"]

for feature in categorical_features:
    mode_value = df_train[feature].mode()[0]
    df_train[feature].fillna(mode_value, inplace=True)
    df_test[feature].fillna(mode_value, inplace=True)
```

Filling Missing Values with the mode

In [37]:
```
df_train.drop(columns=["education"], inplace=True)
df_test.drop(columns=["education"], inplace=True)
```

Dropping education because education-num and education seem redundant.

In [38]:
```
for feature in ["capital-gain", "capital-loss", "hours-per-week"]:
    Q1 = df_train[feature].quantile(0.25)
    Q3 = df_train[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_train[feature] = np.clip(df_train[feature], lower_bound, upper_bound)
    df_test[feature] = np.clip(df_test[feature], lower_bound, upper_bound)
```

Handling outliers using clipping

In [39]:
```
df = df_train.copy()

label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

scaler = StandardScaler()
numerical_features = ["age", "education-num", "capital-gain", "capital-loss", '
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

Encode categorical features and scale numerical features

In [40]:
```
df_train.drop(columns=["native-country"], inplace=True)
df_test.drop(columns=["native-country"], inplace=True)
```

Dropping native country because it is heavily skewed towards United States and did not seem like a strong predictor.

In [41]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
```

```
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc,
    confusion_matrix, classification_report
)

X = df_train.drop(columns=["income"])
y = df_train["income"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42
)

numerical_features = ['age', 'education-num', 'capital-gain', 'capital-loss',
categorical_features = ['workclass', 'marital-status', 'occupation', 'relations

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)
```

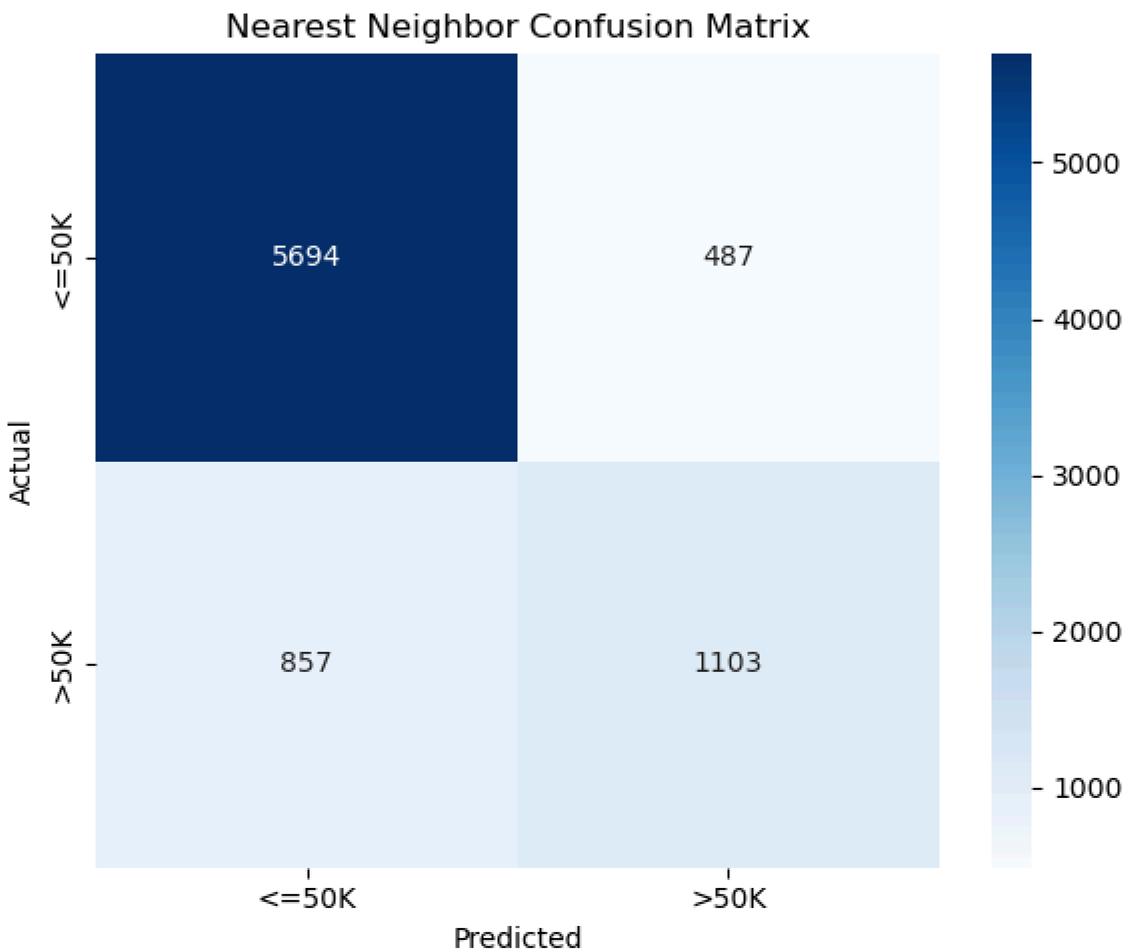
In [42]:

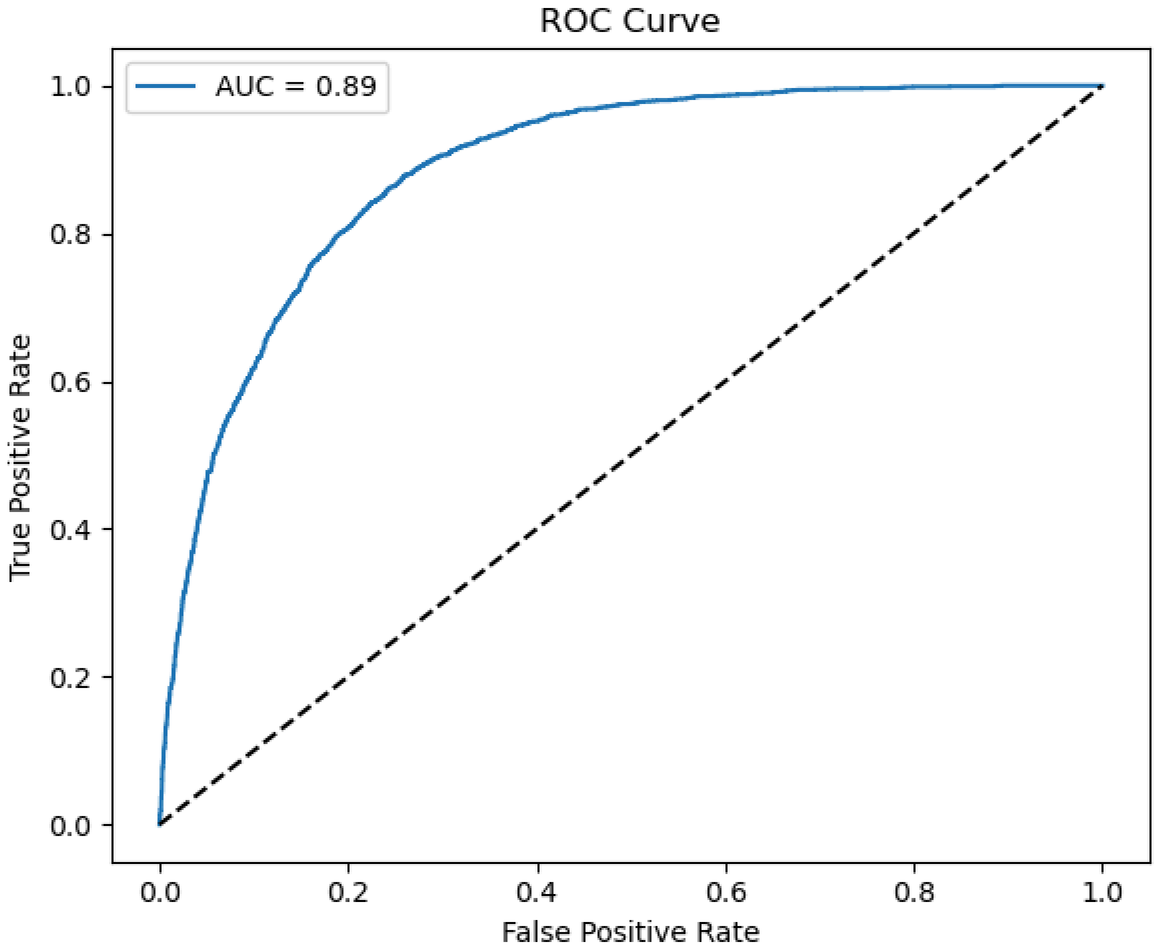
```
def evaluate_model(model, X_train, X_test, y_train, y_test, preprocessor):
    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    y_prob = pipeline.predict_proba(X_test)[:, 1]

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc_score = auc(fpr, tpr)

    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print(f"AUC: {auc_score:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    plt.figure(figsize=(6, 5))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='B
                xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
    plt.title("Nearest Neighbor Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, label=f'AUC = {auc_score:.2f}')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
```

```
    plt.title("ROC Curve")
    plt.legend()
    plt.tight_layout()
    plt.show()

    return pipeline
```

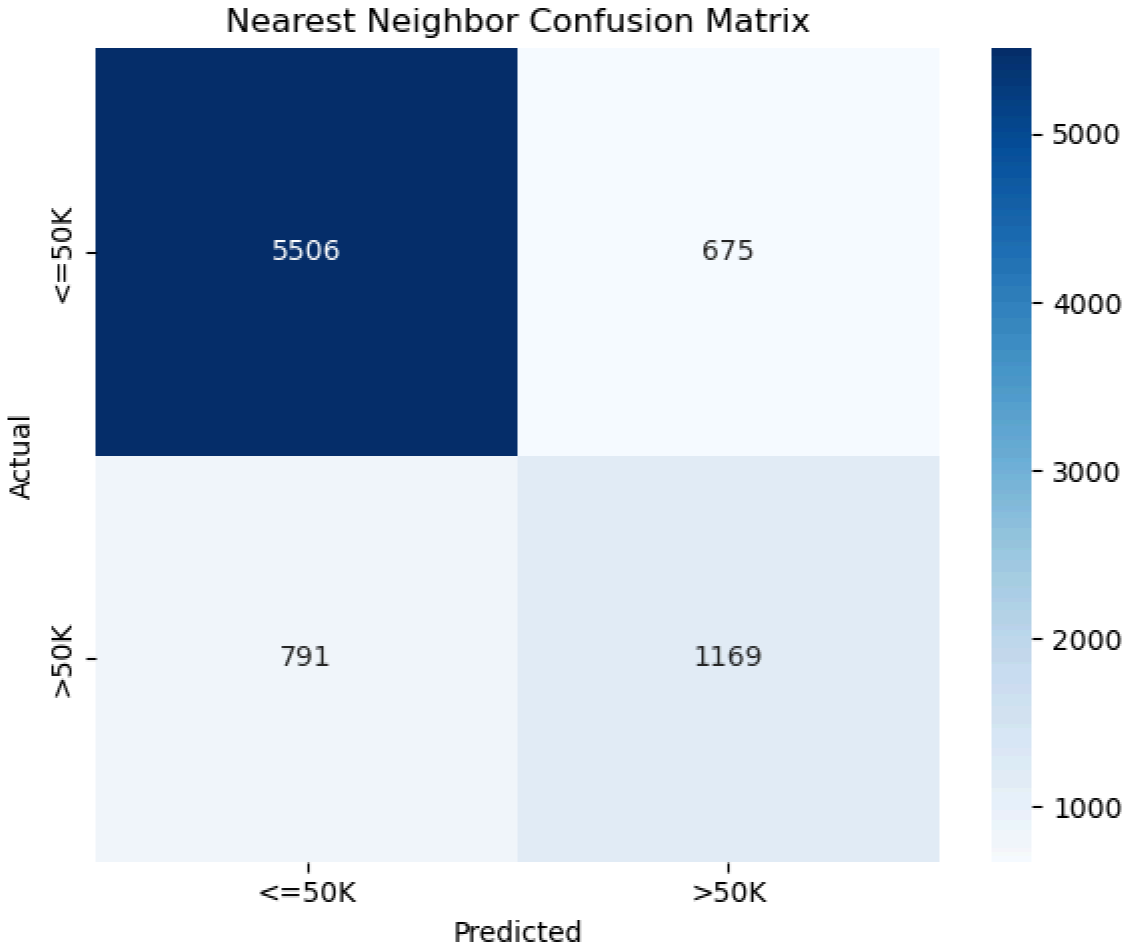
In [43]:
```
print("\nLogistic Regression")
log_reg_model = LogisticRegression(max_iter=1000, random_state=42)
log_reg_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', log_reg_model)
])
log_reg_pipeline.fit(X_train, y_train)
evaluate_model(log_reg_model, X_train, X_test, y_train, y_test, preprocessor)

print("\nK-Nearest Neighbors")
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_pipeline = evaluate_model(knn_model, X_train, X_test, y_train, y_test, prep

X_preprocessed = preprocessor.fit_transform(X)
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(X_preprocessed)

cluster_0_majority = np.bincount(y[kmeans_labels == 0]).argmax()
cluster_1_majority = np.bincount(y[kmeans_labels == 1]).argmax()
if cluster_0_majority != cluster_1_majority:
    kmeans_pred = np.where(kmeans_labels == 0, cluster_0_majority, cluster_1_ma
else:
    mapping1 = np.where(kmeans_labels == 0, 0, 1)
    mapping2 = np.where(kmeans_labels == 0, 1, 0)
    acc1 = accuracy_score(y, mapping1)
    acc2 = accuracy_score(y, mapping2)
    kmeans_pred = mapping1 if acc1 > acc2 else mapping2

acc = accuracy_score(y, kmeans_pred)
prec = precision_score(y, kmeans_pred)
rec = recall_score(y, kmeans_pred)
f1 = f1_score(y, kmeans_pred)
print("\nK-Means Clustering")
print(f"Accuracy: {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall: {rec:.4f}")
print(f"F1 Score: {f1:.4f}")
print("\nClassification Report:")
print(classification_report(y, kmeans_pred))
```

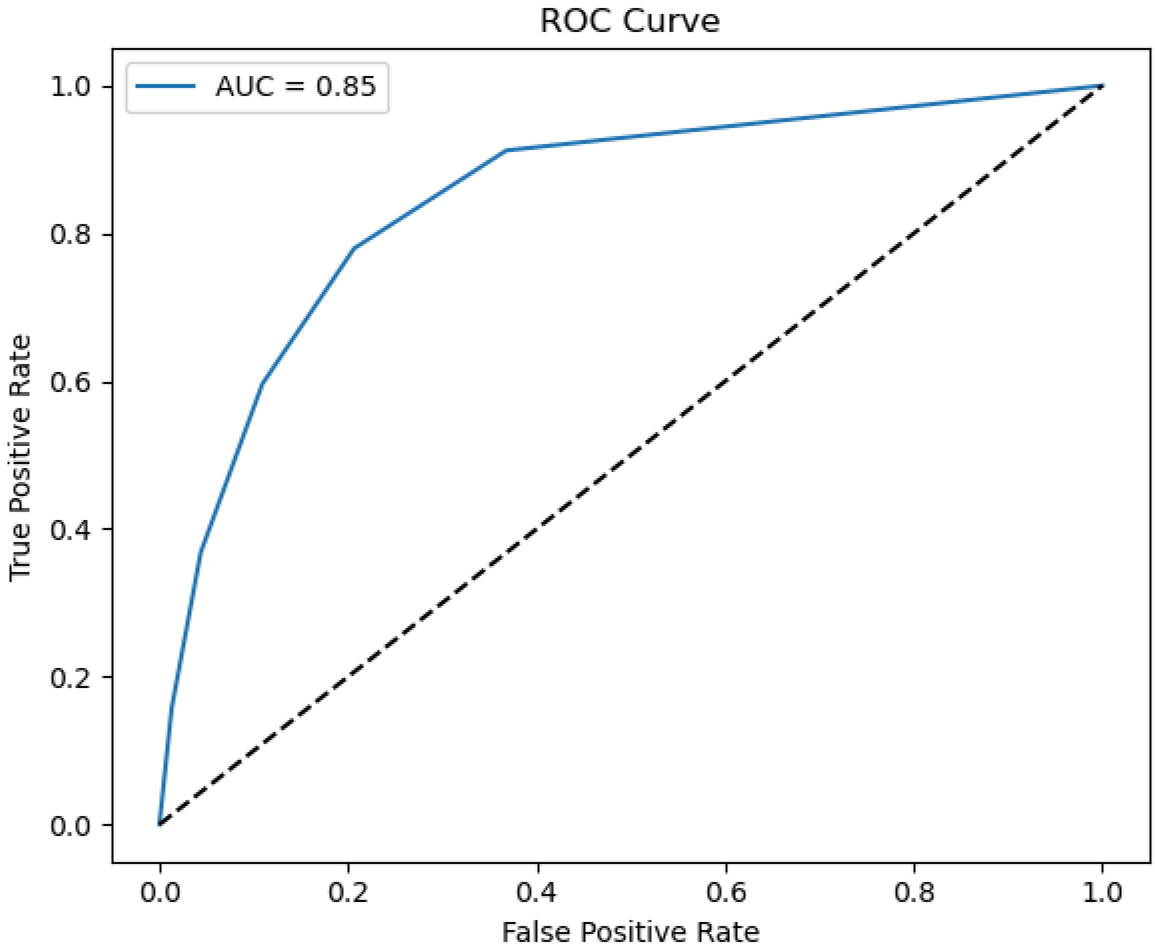
```
Logistic Regression
Accuracy: 0.8349
Precision: 0.6937
Recall: 0.5628
F1 Score: 0.6214
AUC: 0.8871

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.92      0.89      6181
           1       0.69      0.56      0.62      1960

    accuracy                           0.83      8141
   macro avg       0.78      0.74      0.76      8141
weighted avg       0.83      0.83      0.83      8141
```






```
K-Nearest Neighbors
Accuracy: 0.8199
Precision: 0.6339
Recall: 0.5964
F1 Score: 0.6146
AUC: 0.8485

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      6181
           1       0.63      0.60      0.61      1960

    accuracy                           0.82      8141
   macro avg       0.75      0.74      0.75      8141
weighted avg       0.82      0.82      0.82      8141
```


Nearest Neighbor Confusion Matrix
Actual
<=50K
>50K
5506
675
791
1169
Predicted
<=50K
>50K
5000
4000
3000
2000
1000




```
K-Means Clustering
Accuracy: 0.6779
Precision: 0.4199
Recall: 0.8850
F1 Score: 0.5695

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.61      0.74     24720
           1       0.42      0.88      0.57      7841

    accuracy                           0.68     32561
   macro avg       0.68      0.75      0.66     32561
weighted avg       0.82      0.68      0.70     32561
```

Part 3: Model Training and Evaluation

In this section, I developed and evaluated three different binary classification models to predict whether an individual earns more than $50K based on demographic and work-related features. The goal was not only to assess predictive performance but also to lay the groundwork for fairness analysis in later steps.

Preprocessing Pipeline

We used `ColumnTransformer` from `scikit-learn` to create a clean and scalable pipeline:

- Numerical Features (scaled using StandardScaler):
  - `age`, `education-num`, `capital-gain`, `capital-loss`, `hours-per-week`
- Categorical Features (one-hot encoded):
  - `workclass`, `marital-status`, `occupation`, `relationship`, `race`, `sex`

Using a preprocessing pipeline ensures that data transformations are applied consistently during both training and testing. This also makes the code modular and ready for integration into other frameworks.

Train-Test Split

We split the data into training and testing sets using:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=42)

Making a 25/75 split

Key Observations

- KNN slightly outperformed Logistic Regression in F1 and Recall.
- Both supervised models handled class imbalance relatively well.
- K-Means performed poorly, as expected, because it is not optimized for labeled classification tasks.

In [44]:
```
import pandas as pd

data = {
    "Model": ["Logistic Regression", "K-Nearest Neighbors", "K-Means Clusterin
    "Accuracy": [0.8349, 0.8199, 0.6779],
    "Precision": [0.6937, 0.6339, 0.4199],
    "Recall": [0.5628, 0.5964, 0.8850],
    "F1 Score": [0.6214, 0.6146, 0.5695],
    "AUC Score": [0.8871, 0.8485, None]  # AUC not applicable to clustering
}

df = pd.DataFrame(data)

styled_df = df.style.set_caption(" Model Performance Comparison")\
    .format(precision=4)\
    .background_gradient(cmap="YlGnBu", subset=["Accuracy", "Precision", "Reca
    .highlight_null(color="lightgray", subset=["AUC Score"])\
```

```
    .set_table_styles([{
        'selector': 'caption',
        'props': [('font-size', '16px'), ('font-weight', 'bold'), ('color', '#
    }])

styled_df
```

Out[44]:

Model Performance Comparison

	Model	Accuracy	Precision	Recall	F1 Score	AUC Score
0	Logistic Regression	0.8349	0.6937	0.5628	0.6214	0.8871
1	K-Nearest Neighbors	0.8199	0.6339	0.5964	0.6146	0.8485
2	K-Means Clustering	0.6779	0.4199	0.8850	0.5695	nan

In [48]:

```
!pip install fairlearn

from fairlearn.metrics import MetricFrame, demographic_parity_difference, equa

# Predict using trained logistic regression pipeline
y_pred = log_reg_pipeline.predict(X_test)
y_true = y_test.reset_index(drop=True)

# Extract sensitive features
sensitive_features = X_test[['sex', 'race']].reset_index(drop=True)

# Fairness metrics by sex
sex_metrics = MetricFrame(
    metrics={
        'Demographic Parity': selection_rate,
        'True Positive Rate': true_positive_rate,
        'False Positive Rate': false_positive_rate
    },
    y_true=y_true,
    y_pred=y_pred,
    sensitive_features=sensitive_features['sex']
)

# Fairness metrics by race
race_metrics = MetricFrame(
    metrics={
        'Demographic Parity': selection_rate,
        'True Positive Rate': true_positive_rate,
        'False Positive Rate': false_positive_rate
    },
    y_true=y_true,
    y_pred=y_pred,
    sensitive_features=sensitive_features['race']
)

# Print metric tables
print("=== Fairness Metrics by Sex ===")
print(sex_metrics.by_group)
print("\n=== Fairness Metrics by Race ===")
print(race_metrics.by_group)

# Disparity scores
print("\n=== Disparity Measures ===")
print(f"Demographic Parity Difference (Sex): {demographic_parity_difference(y_
```

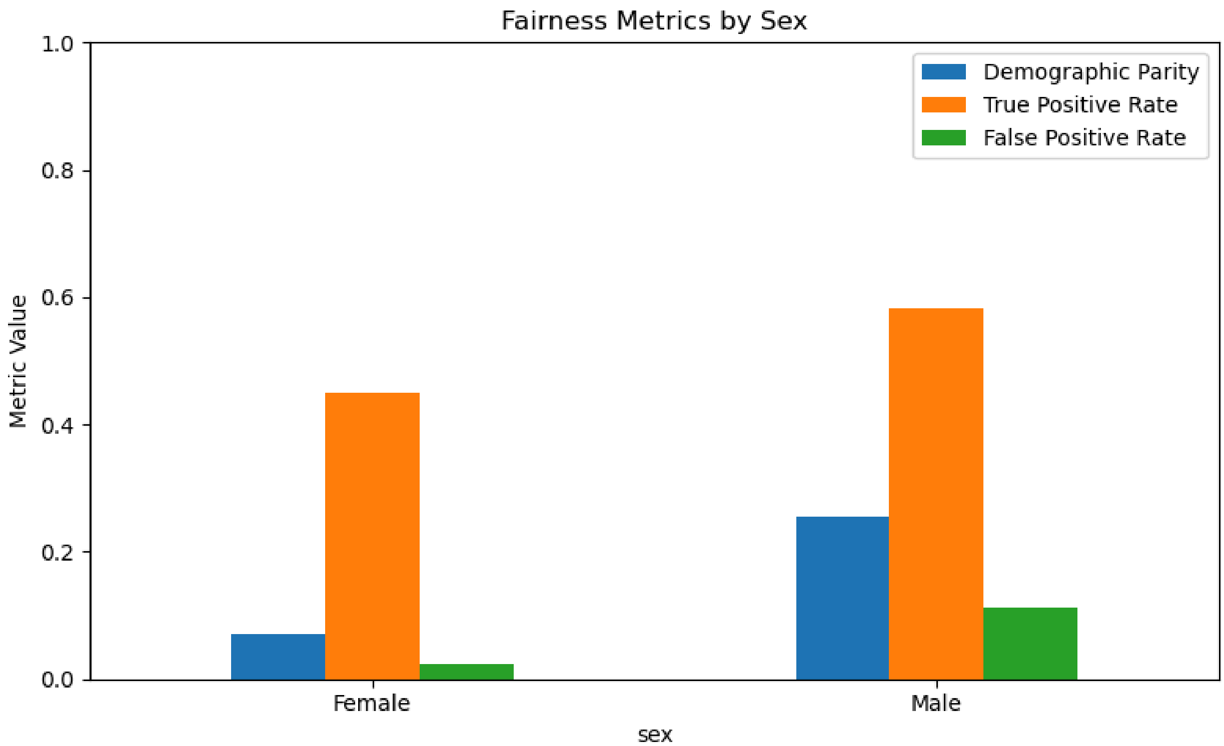
```
print(f"Equalized Odds Difference (Sex): {equalized_odds_difference(y_true, y_
print(f"Demographic Parity Difference (Race): {demographic_parity_difference(y_
print(f"Equalized Odds Difference (Race): {equalized_odds_difference(y_true, y_
```

```
Requirement already satisfied: fairlearn in /Users/FolahanmiIlori/anaconda3/li
b/python3.11/site-packages (0.12.0)
Requirement already satisfied: numpy>=1.24.4 in /Users/FolahanmiIlori/anaconda
3/lib/python3.11/site-packages (from fairlearn) (1.26.4)
Requirement already satisfied: pandas>=2.0.3 in /Users/FolahanmiIlori/anaconda
3/lib/python3.11/site-packages (from fairlearn) (2.0.3)
Requirement already satisfied: scikit-learn>=1.2.1 in /Users/FolahanmiIlori/an
aconda3/lib/python3.11/site-packages (from fairlearn) (1.3.0)
Requirement already satisfied: scipy>=1.9.3 in /Users/FolahanmiIlori/anaconda
3/lib/python3.11/site-packages (from fairlearn) (1.11.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/FolahanmiIlor
i/anaconda3/lib/python3.11/site-packages (from pandas>=2.0.3->fairlearn) (2.8.
2)
Requirement already satisfied: pytz>=2020.1 in /Users/FolahanmiIlori/anaconda
3/lib/python3.11/site-packages (from pandas>=2.0.3->fairlearn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /Users/FolahanmiIlori/anacond
a3/lib/python3.11/site-packages (from pandas>=2.0.3->fairlearn) (2023.3)
Requirement already satisfied: joblib>=1.1.1 in /Users/FolahanmiIlori/anaconda
3/lib/python3.11/site-packages (from scikit-learn>=1.2.1->fairlearn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/FolahanmiIlori/a
naconda3/lib/python3.11/site-packages (from scikit-learn>=1.2.1->fairlearn)
(2.2.0)
Requirement already satisfied: six>=1.5 in /Users/FolahanmiIlori/anaconda3/li
b/python3.11/site-packages (from python-dateutil>=2.8.2->pandas>=2.0.3->fairle
arn) (1.16.0)
=== Fairness Metrics by Sex ===
        Demographic Parity  True Positive Rate  False Positive Rate
sex
Female            0.071455            0.450847             0.024390
Male              0.255121            0.581982             0.112017

=== Fairness Metrics by Race ===
                    Demographic Parity  True Positive Rate  \
race
Amer-Indian-Eskimo            0.011364            0.100000
Asian-Pac-Islander            0.254980            0.594203
Black                         0.067754            0.339450
Other                         0.065574            0.571429
White                         0.210685            0.577337

                    False Positive Rate
race
Amer-Indian-Eskimo             0.000000
Asian-Pac-Islander             0.126374
Black                          0.024709
Other                          0.000000
White                          0.085731

=== Disparity Measures ===
Demographic Parity Difference (Sex): 0.184
Equalized Odds Difference (Sex): 0.131
Demographic Parity Difference (Race): 0.244
Equalized Odds Difference (Race): 0.494
```

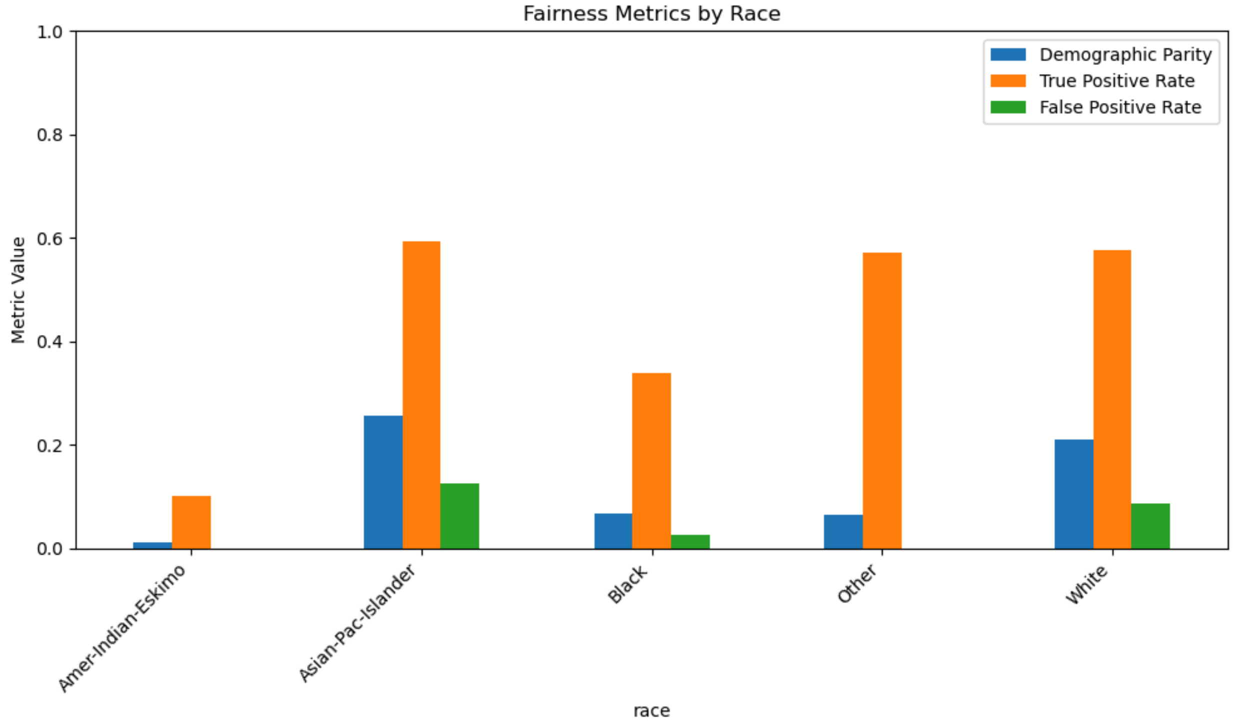
In [50]:

```
sex_metrics.by_group.plot(kind='bar', figsize=(8, 5))
plt.title("Fairness Metrics by Sex")
```

```
plt.ylabel("Metric Value")
plt.xticks(rotation=0)
plt.ylim(0, 1)
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```



In [52]:
```
race_metrics.by_group.plot(kind='bar', figsize=(10, 6))
plt.title("Fairness Metrics by Race")
plt.ylabel("Metric Value")
plt.xticks(rotation=45, ha='right')
plt.ylim(0, 1)
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```



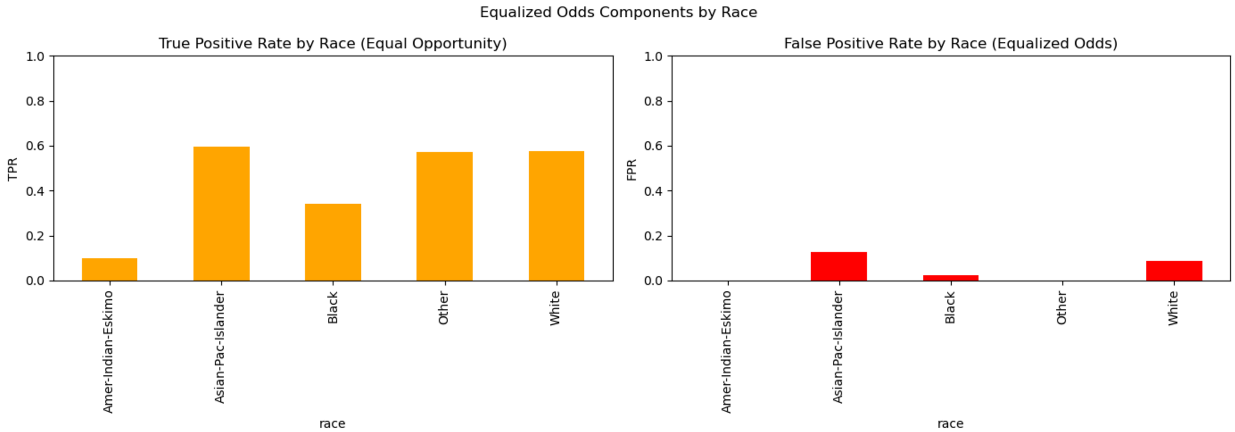
In [54]:
```
disparity_data = {
    "Attribute": ["Sex", "Race"],
    "Demographic Parity Diff": [
        demographic_parity_difference(y_true, y_pred, sensitive_features=sensi
        demographic_parity_difference(y_true, y_pred, sensitive_features=sensi
    ],
    "Equalized Odds Diff": [
        equalized_odds_difference(y_true, y_pred, sensitive_features=sensitive_
        equalized_odds_difference(y_true, y_pred, sensitive_features=sensitive_
    ]
}

disparity_df = pd.DataFrame(disparity_data)
(disparity_df.round(3))
```

Out[54]:

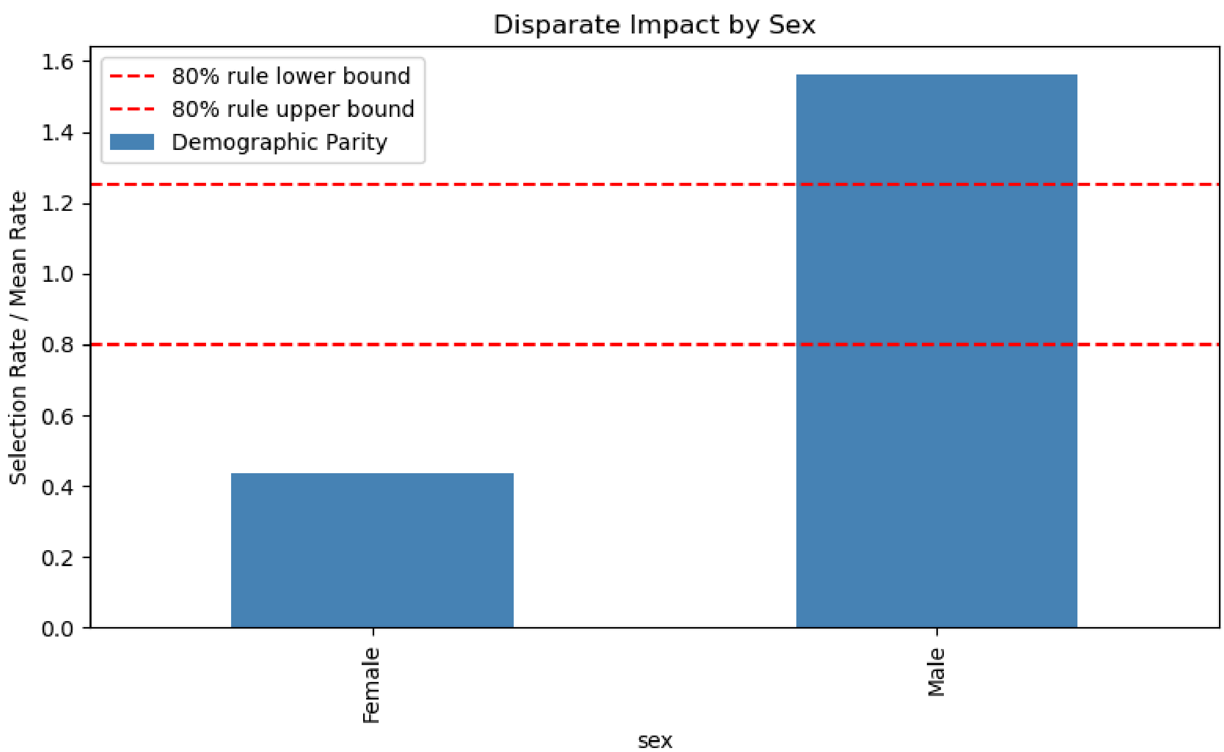
	Attribute	Demographic Parity Diff	Equalized Odds Diff
0	Sex	0.184	0.131
1	Race	0.244	0.494

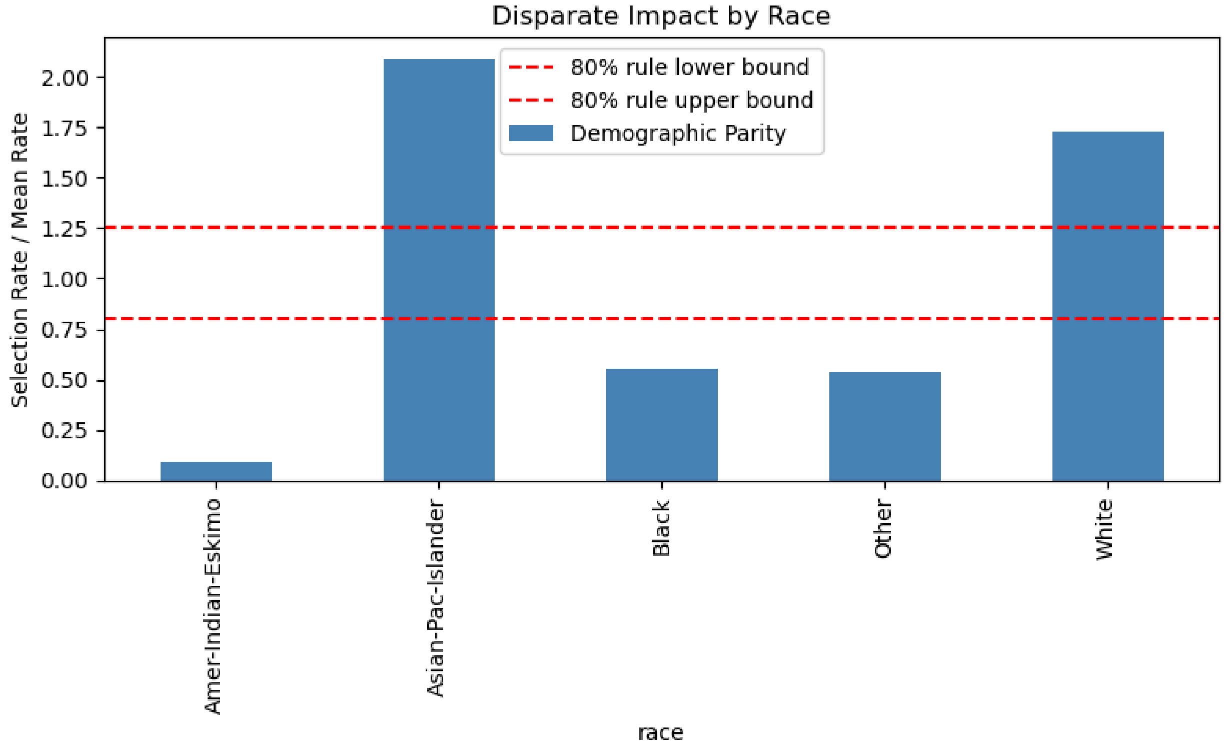
In [55]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# TPR by race
race_metrics.by_group["True Positive Rate"].plot(kind='bar', ax=axes[0], color=
axes[0].set_title("True Positive Rate by Race (Equal Opportunity)")
axes[0].set_ylabel("TPR")
axes[0].set_ylim(0, 1)

# FPR by race
race_metrics.by_group["False Positive Rate"].plot(kind='bar', ax=axes[1], colo
axes[1].set_title("False Positive Rate by Race (Equalized Odds)")
axes[1].set_ylabel("FPR")
axes[1].set_ylim(0, 1)
```

```
plt.suptitle("Equalized Odds Components by Race")
plt.tight_layout()
plt.show()
```



In [57]:
```
def plot_disparate_impact(df, group_name):
    group_rates = df.by_group["Demographic Parity"]
    average_rate = group_rates.mean()
    impact_ratios = group_rates / average_rate

    plt.figure(figsize=(8, 5))
    bars = impact_ratios.plot(kind="bar", color="steelblue")
    plt.axhline(0.8, color="red", linestyle="--", label="80% rule lower bound"
    plt.axhline(1.25, color="red", linestyle="--", label="80% rule upper bound'
    plt.title(f"Disparate Impact by {group_name}")
    plt.ylabel("Selection Rate / Mean Rate")
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_disparate_impact(sex_metrics, "Sex")
plot_disparate_impact(race_metrics, "Race")
```





Part 4: Fairness Assessment

To evaluate how the models performed across demographic groups, we applied fairness metrics using the `fairlearn` library. Specifically, we analyzed predictions made by the Logistic Regression model on the Adult Income dataset using **Demographic Parity**, **Equal Opportunity**, **Equalized Odds**, and **Disparate Impact**.

Group-Based Metric Frames

Using `MetricFrame`, we calculated fairness metrics by **sex** and **race**, including:

- **Selection Rate** (Demographic Parity)
- **True Positive Rate** (Equal Opportunity)
- **False Positive Rate** (used in Equalized Odds)

By sex, males had a much higher selection rate (0.255) compared to females (0.071), yielding a **Demographic Parity Difference** of **0.184**. This means men were over 3.5 times as likely to be predicted as earning >$50K.

True Positive Rate was also higher for males (0.582 vs. 0.451), indicating better recall for men. False Positive Rate was similarly skewed: males had an FPR of 0.112, compared to just 0.024 for females.

By race, we found additional disparities:

- Asians and Whites had the highest selection rates (0.255 and 0.211).
- Black individuals had a selection rate of only 0.068.

- The resulting **Demographic Parity Difference** for race was **0.244**.
- Equal Opportunity was also lower for Black individuals (TPR = 0.339), compared to Asians (0.594) and Whites (0.577).

False Positive Rates varied significantly, with Asians and Whites showing higher FPRs than other groups.

Summary of Disparity Measures

Attribute	Demographic Parity Diff	Equalized Odds Diff
Sex	0.184	0.131
Race	0.244	0.494

- **Demographic Parity Difference** measures the gap in selection rates across groups.
- **Equalized Odds Difference** captures divergence in both true and false positive rates.

Visual Summary

Visuals in the notebook (e.g., bar plots and grouped metric plots) help demonstrate these disparities:

- *Fairness Metrics by Sex*
- *Fairness Metrics by Race*
- *True Positive Rate by Race* (Equal Opportunity)
- *False Positive Rate by Race* (Equalized Odds)
- *Disparate Impact Ratios* by group (Sex and Race)

Groups like women and Black individuals fall outside the 80% rule range for selection rates (0.8 to 1.25), further illustrating systemic inequality in model outcomes.

Interpretation

Although the logistic regression model performed well on traditional metrics (e.g., accuracy and F1), fairness analysis uncovered significant disparities. These results highlight why fairness metrics must be part of model evaluation—without them, biases that disadvantage certain groups can go undetected.

In [ ]:

```
```

In [ ]:

```
```