In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import shap
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import zero_one_loss, log_loss
```

In [2]:

```
X, y = shap.datasets.adult() # Numerical version of data
X_display, y_display = shap.datasets.adult(display=True) # Human - readable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [4]:

```
clf = GradientBoostingClassifier(n_estimators=100 , random_state=10)
clf.fit(X_train.values, y_train)
```

Out[4]:

```
▾      GradientBoostingClassifier
GradientBoostingClassifier(random_state=10)
```

In [16]:

```
# Make predictions on train and test sets
y_train_pred = clf.predict(X_train.values)
y_test_pred = clf.predict(X_test.values)

# Calculate zero-one classification error
train_error_zero_one = zero_one_loss(y_train, y_train_pred)
test_error_zero_one = zero_one_loss(y_test, y_test_pred)

# Calculate log-loss
train_log_loss = log_loss(y_train, y_train_pred)
test_log_loss = log_loss(y_test, y_test_pred)
```

Q4 - A)

In [17]:

```
print("Train set error with zero-one: "+str(train_error_zero_one))
print("Test set error with zero-one: "+str(test_error_zero_one))
print("Train set error with log-loss: "+ str(train_log_loss))
print("Test set error with log-loss: "+ str(test_log_loss))
```

```
Train set error with zero-one: 0.13148802211302213
Test set error with zero-one: 0.1337325349301397
Train set error with log-loss: 4.739308693862341
Test set error with log-loss: 4.820209135869959
```

Q4 - B

In [33]:

```
def perm_imp(X_test, y_test,row, loss_type):
    act_error = 0
    y_pred = clf.predict(X_test.values)

    if loss_type == "zero_one":
        original_error = zero_one_loss(y_test, y_pred)
    elif loss_type == "log_loss":
        original_error = log_loss(y_test, y_pred)

    permutation_errors = np.zeros(1)
    X_test_perm = X_test.copy()

    column_values = X_test_perm.iloc[:, row]
    column_values = column_values.to_numpy()
    np.random.shuffle(column_values)

    X_test_perm.iloc[:, row] = column_values

    y_pred_perm = clf.predict(X_test_perm.values)

    if loss_type == "zero_one":
        permutation_errors[0] = zero_one_loss(y_test, y_pred_perm)
    else:
        permutation_errors[0] = log_loss(y_test, y_pred_perm)

    importance = permutation_errors - original_error

    return np.mean(importance), np.std(importance)
```

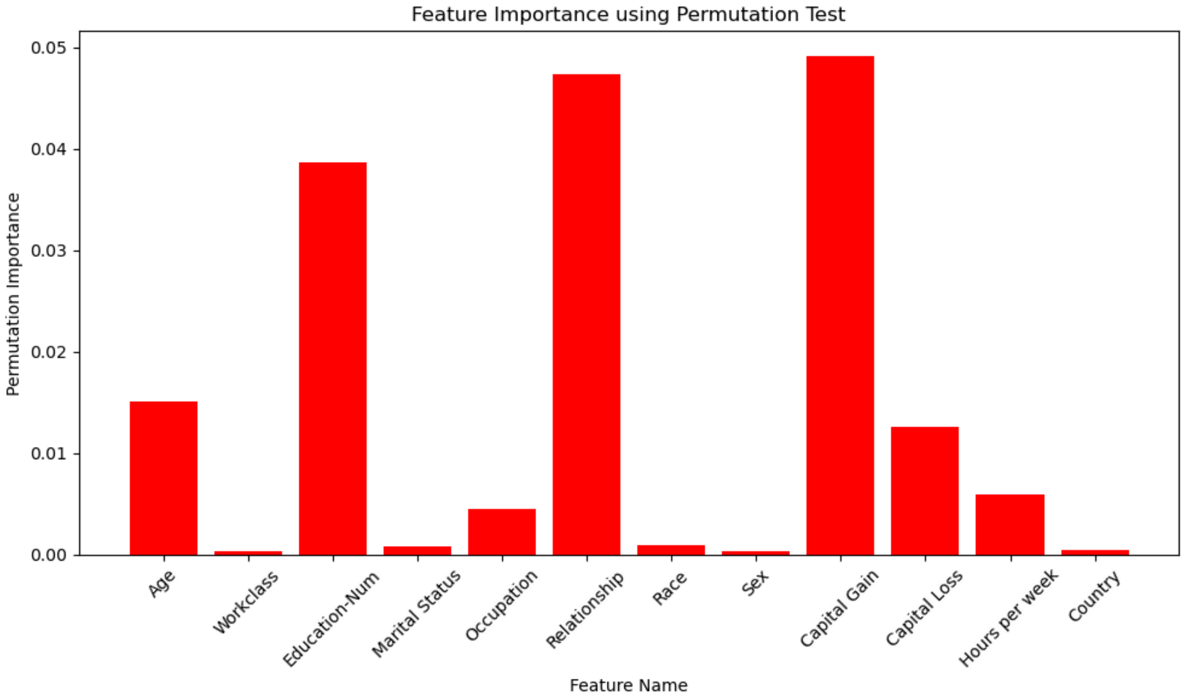
In [34]:

```
feature_imp = np.zeros(X.shape[1])
for i in range(X.shape[1]):
    feature_imp[i], _ = perm_imp(X_test, y_test, i,'zero_one')
```

4 b)

In [83]:

```
# Visualize feature importances
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_imp, color='r')
plt.xlabel("Feature Name")
plt.ylabel("Permutation Importance")
plt.title("Feature Importance using Permutation Test")
plt.xticks(range(X.shape[1]),[i for i in X.columns], rotation=45)
plt.tight_layout()
plt.show()
```



In [53]:

```
def permutation_importance(X_test, y_test, row, n_permutations=1, loss_type=
    original_error = 0
    y_pred = clf.predict(X_test.values)

    if loss_type == "log_loss":
        original_error = log_loss(y_test, y_pred)
    elif loss_type == "zero_one":
        original_error = zero_one_loss(y_test, y_pred)

    permutation_errors = np.zeros(n_permutations)

    for i in range(n_permutations):
        X_test_perm = X_test.copy()
        column_values = X_test_perm.iloc[:, row].to_numpy()
        np.random.shuffle(column_values)
        X_test_perm.iloc[:, row] = column_values
        y_pred_perm = clf.predict(X_test_perm.values)

        if loss_type == "zero_one":
            permutation_errors[i] = zero_one_loss(y_test, y_pred_perm)
        else:
            permutation_errors[i] = log_loss(y_test, y_pred_perm)

    importance = permutation_errors - original_error
    return np.mean(importance), np.std(importance)
```

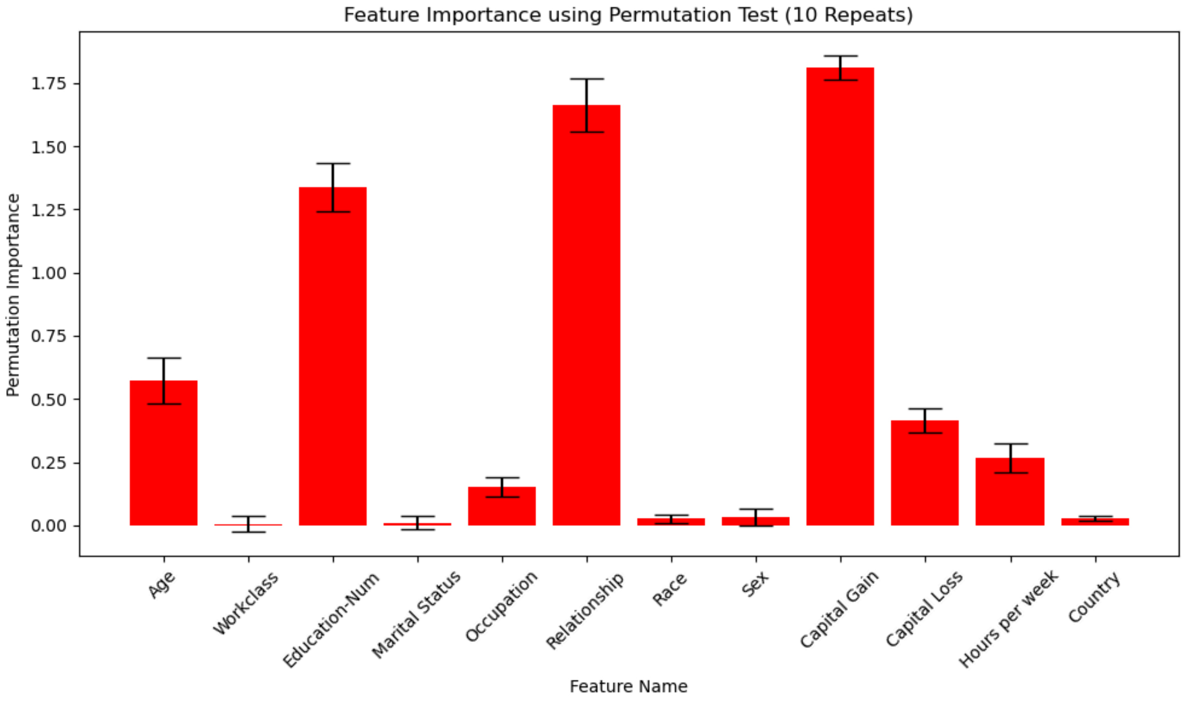
In [58]:

```
feature_importances2 = np.zeros(X.shape[1])
feature_importances_std2 = np.zeros(X.shape[1])
for i in range(X.shape[1]):
    feature_importances2[i], feature_importances_std2[i] = permutation_impo
```

4 c)

In [84]:

```
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_importances2, yerr=feature_importances_st
plt.xlabel("Feature Name")
plt.ylabel("Permutation Importance")
plt.title("Feature Importance using Permutation Test (10 Repeats)")
plt.xticks(range(X.shape[1]),[i for i in X.columns], rotation=45)  # Rotate
plt.tight_layout()
plt.show()
```



4 d)

In [70]:

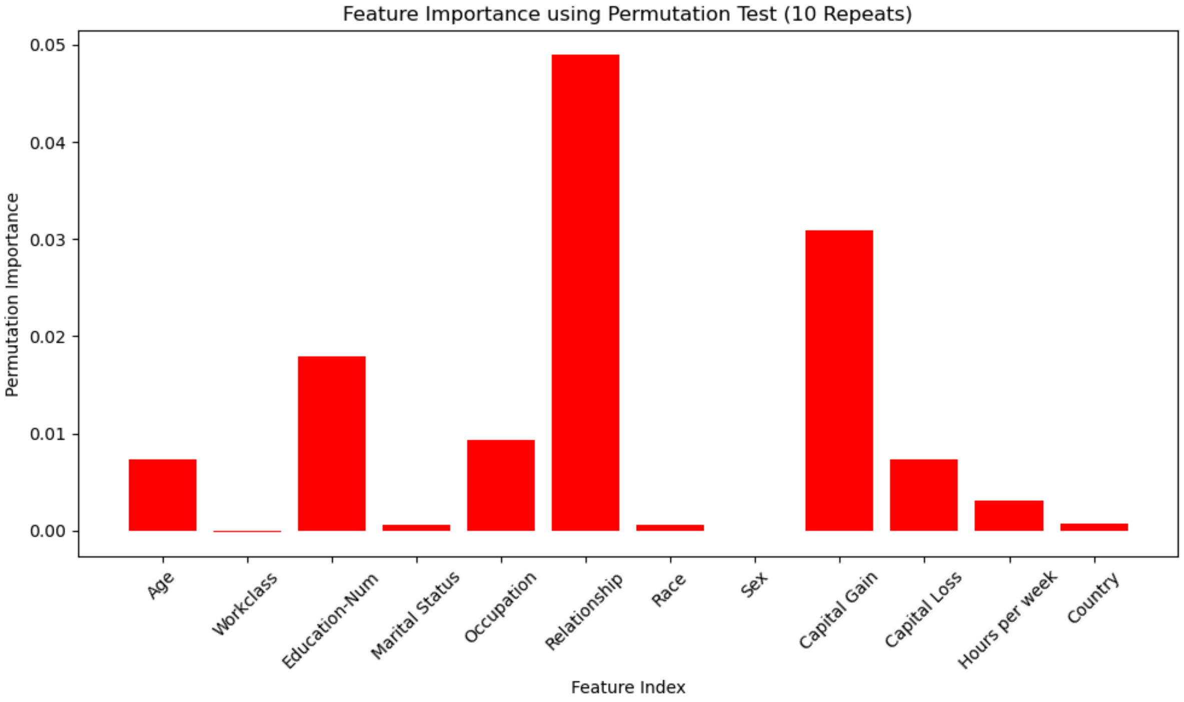
```
def permutation_importance(X_test, y_test, row, n_permutations=1, loss_type
    original_error = 0
    y_pred = clf.predict(X_test.values)

    if loss_type == "log_loss":
        original_error = log_loss(y_test, y_pred)
    elif loss_type == "zero_one":
        original_error = zero_one_loss(y_test, y_pred)

    permutation_errors = np.zeros(n_permutations)

    for i in range(n_permutations):
        X_test_perm = X_test.copy()
        column_values = X_test_perm.iloc[:, row]
        column_values = column_values.mean()
        X_test_perm.iloc[:, row] = column_values
        y_pred_perm = clf.predict(X_test_perm.values)

        if loss_type == "zero_one":
            permutation_errors[i] = zero_one_loss(y_test, y_pred_perm)
        else:
            permutation_errors[i] = log_loss(y_test, y_pred_perm)

    importance = permutation_errors - original_error
    return np.mean(importance), np.std(importance)
```

In [71]:

```
feature_importances2 = np.zeros(X.shape[1])
feature_importances_std2 = np.zeros(X.shape[1])
for i in range(X.shape[1]):
    feature_importances2[i], feature_importances_std2[i] = permutation_impo
```

In [75]:

```
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_importances2, capsize=10, color = 'r')
plt.xlabel("Feature Index")
plt.ylabel("Permutation Importance")
plt.title("Feature Importance using Permutation Test (10 Repeats)")
plt.xticks(range(X.shape[1]),[i for i in X.columns], rotation=45)  # Rotate
plt.tight_layout()
plt.show()
```

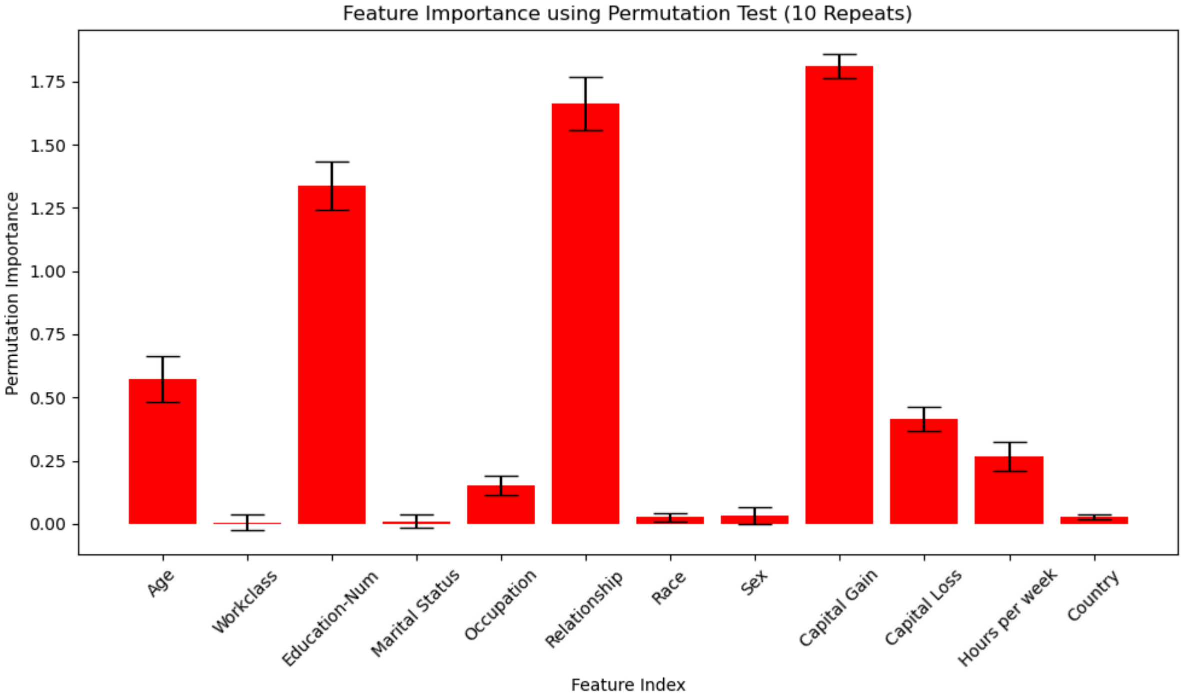


In [81]:

```
def permutation_importance(X_test, y_test, row, n_permutations=1):
    original_error = 0
    y_pred = clf.predict(X_test.values)

    original_error = log_loss(y_test, y_pred)


    permutation_errors = np.zeros(n_permutations)

    for i in range(n_permutations):
        X_test_perm = X_test.copy()
        column_values = X_test_perm.iloc[:, row].to_numpy()
        np.random.shuffle(column_values)
        X_test_perm.iloc[:, row] = column_values
        y_pred_perm = clf.predict(X_test_perm.values)
        permutation_errors[i] = log_loss(y_test, y_pred_perm)

    importance = permutation_errors - original_error
    return np.mean(importance), np.std(importance)
```

In [82]:

```
feature_importances2 = np.zeros(X.shape[1])
feature_importances_std2 = np.zeros(X.shape[1])
for i in range(X.shape[1]):
    feature_importances2[i], feature_importances_std2[i] = permutation_impo
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_importances2, yerr=feature_importances_s
plt.xlabel("Feature Index")
plt.ylabel("Permutation Importance")
plt.title("Feature Importance using Permutation Test (10 Repeats)")
plt.xticks(range(X.shape[1]),[i for i in X.columns], rotation=45)  # Rotate
plt.tight_layout()
plt.show()
```



In [ ]: