Assignment 5: Adult Income Dataset (OpenML)

Student Name: Bekzhanov Namazbek Alibekuly

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

%matplotlib inline
```

Q1: Load the Adult (version 2) dataset from OpenML.

After successfully loading the dataset:

1. Display its shape (number of rows and columns).
2. List all feature (attribute) names.

```
# Load the dataset from OpenML
adult_data = fetch_openml(name='adult', version=2, as_frame=True)

# Get the dataframe
df = adult_data.frame

# 1. Display the shape (rows, columns)
print("Shape of the dataset:", df.shape)

# 2. List all feature names
print("\nFeature names:")
print(df.columns.tolist())
```

```
Shape of the dataset: (48842, 15)

Feature names:
['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'ca
```

It is obvious that the target of this dataset is `class`

Q2: Using the dataset loaded in Question 1, display summarized information of the data by applying an appropriate method that shows:

- The non-null count for each feature.
- The data type of each feature.

```
# We use info() to see non-null counts and data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   workclass       46043 non-null  category
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  category
 4   education-num   48842 non-null  int64
 5   marital-status  48842 non-null  category
 6   occupation      46033 non-null  category
 7   relationship    48842 non-null  category
 8   race            48842 non-null  category
 9   sex             48842 non-null  category
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  47985 non-null  category
 14  class           48842 non-null  category
dtypes: category(9), int64(6)
memory usage: 2.7 MB
```

Q3: Generate a descriptive statistics summary of the dataset. Ensure that you:

- Display the summary for quantitative (numerical) columns.

- Display the summary for qualitative (categorical) columns.

Hint: Before applying the method to qualitative columns, check their data types to avoid errors.

```
# Summary for numerical columns
print("Numerical Columns Summary:")
display(df.describe())
```

```
Numerical Columns Summary:
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

```
# Summary for categorical columns
# We include 'object' and 'category' types
print("Categorical Columns Summary:")
display(df.describe(include=['object', 'category']))
```

```
Categorical Columns Summary:
```

	workclass	education	marital-status	occupation	relationship	race	sex	native-country	class
count	46043	48842	48842	46033	48842	48842	48842	47985	48842
unique	8	16	7	14	6	5	2	41	2
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States	<=50K
freq	33906	15784	22379	6172	19716	41762	32650	43832	37155

Q4: Show all categorical columns along with their unique values.

```
# Select only categorical columns
categorical_cols = df.select_dtypes(include=['object', 'category']).columns

# Loop through each categorical column and print unique values
for col in categorical_cols:
    print(f"Unique values in '{col}':")
    print(df[col].unique())
    print("-" * 30)
```

```
Unique values in 'workclass':
['Private', 'Local-gov', NaN, 'Self-emp-not-inc', 'Federal-gov', 'State-gov', 'Self-emp-inc', 'Without-pay', 'Never-worked']
Categories (8, object): ['Federal-gov', 'Local-gov', 'Never-worked', 'Private', 'Self-emp-inc',
                         'Self-emp-not-inc', 'State-gov', 'Without-pay']
------------------------------
Unique values in 'education':
['11th', 'HS-grad', 'Assoc-acdm', 'Some-college', '10th', ..., 'Assoc-voc', '9th', '12th', '1st-4th', 'Preschool']
Length: 16
Categories (16, object): ['10th', '11th', '12th', '1st-4th', ..., 'Masters', 'Preschool',
                          'Prof-school', 'Some-college']
------------------------------
Unique values in 'marital-status':
['Never-married', 'Married-civ-spouse', 'Widowed', 'Divorced', 'Separated', 'Married-spouse-absent', 'Married-AF-spouse']
Categories (7, object): ['Divorced', 'Married-AF-spouse', 'Married-civ-spouse',
                         'Married-spouse-absent', 'Never-married', 'Separated', 'Widowed']
------------------------------
Unique values in 'occupation':
['Machine-op-inspct', 'Farming-fishing', 'Protective-serv', NaN, 'Other-service', ..., 'Sales', 'Priv-house-serv', 'Transport-m
Length: 15
Categories (14, object): ['Adm-clerical', 'Armed-Forces', 'Craft-repair', 'Exec-managerial', ...,
                          'Protective-serv', 'Sales', 'Tech-support', 'Transport-moving']
------------------------------
Unique values in 'relationship':
['Own-child', 'Husband', 'Not-in-family', 'Unmarried', 'Wife', 'Other-relative']
Categories (6, object): ['Husband', 'Not-in-family', 'Other-relative', 'Own-child', 'Unmarried',
                         'Wife']
------------------------------
Unique values in 'race':
['Black', 'White', 'Asian-Pac-Islander', 'Other', 'Amer-Indian-Eskimo']
Categories (5, object): ['Amer-Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other', 'White']
------------------------------
Unique values in 'sex':
['Male', 'Female']
Categories (2, object): ['Female', 'Male']
------------------------------
Unique values in 'native-country':
['United-States', NaN, 'Peru', 'Guatemala', 'Mexico', ..., 'Greece', 'Trinadad&Tobago', 'Outlying-US(Guam-USVI-etc)', 'France',
Length: 42
Categories (41, object): ['Cambodia', 'Canada', 'China', 'Columbia', ..., 'Trinadad&Tobago',
                          'United-States', 'Vietnam', 'Yugoslavia']
------------------------------
```

```
Unique values in 'class':
['<=50K', '>50K']
Categories (2, object): ['<=50K', '>50K']
------------------------------
```

Q5: Check for missing values in the dataset for each feature.

```
# Count missing values in each column
print("Missing values per feature:")
print(df.isnull().sum())
```

```
Missing values per feature:
age                  0
workclass         2799
fnlwgt               0
education            0
education-num        0
marital-status       0
occupation        2809
relationship         0
race                 0
sex                  0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     857
class                0
dtype: int64
```

Q5.1: Perform the following data preprocessing steps on the dataset:

1. Calculate the ratio of total missing values to the overall size of the dataset.
2. If the ratio is less than 20%, drop the rows containing missing values.
3. If the ratio is 20% or higher, replace missing values using:
   - The median for numerical features.
   - The mode for categorical features.

```
# 1. Calculate the ratio of missing values
total_cells = df.size
total_missing = df.isnull().sum().sum()
missing_ratio = (total_missing / total_cells) * 100

print(f"Total missing values: {total_missing}")
print(f"Total cells in dataset: {total_cells}")
print(f"Missing value ratio: {missing_ratio:.2f}%")
```

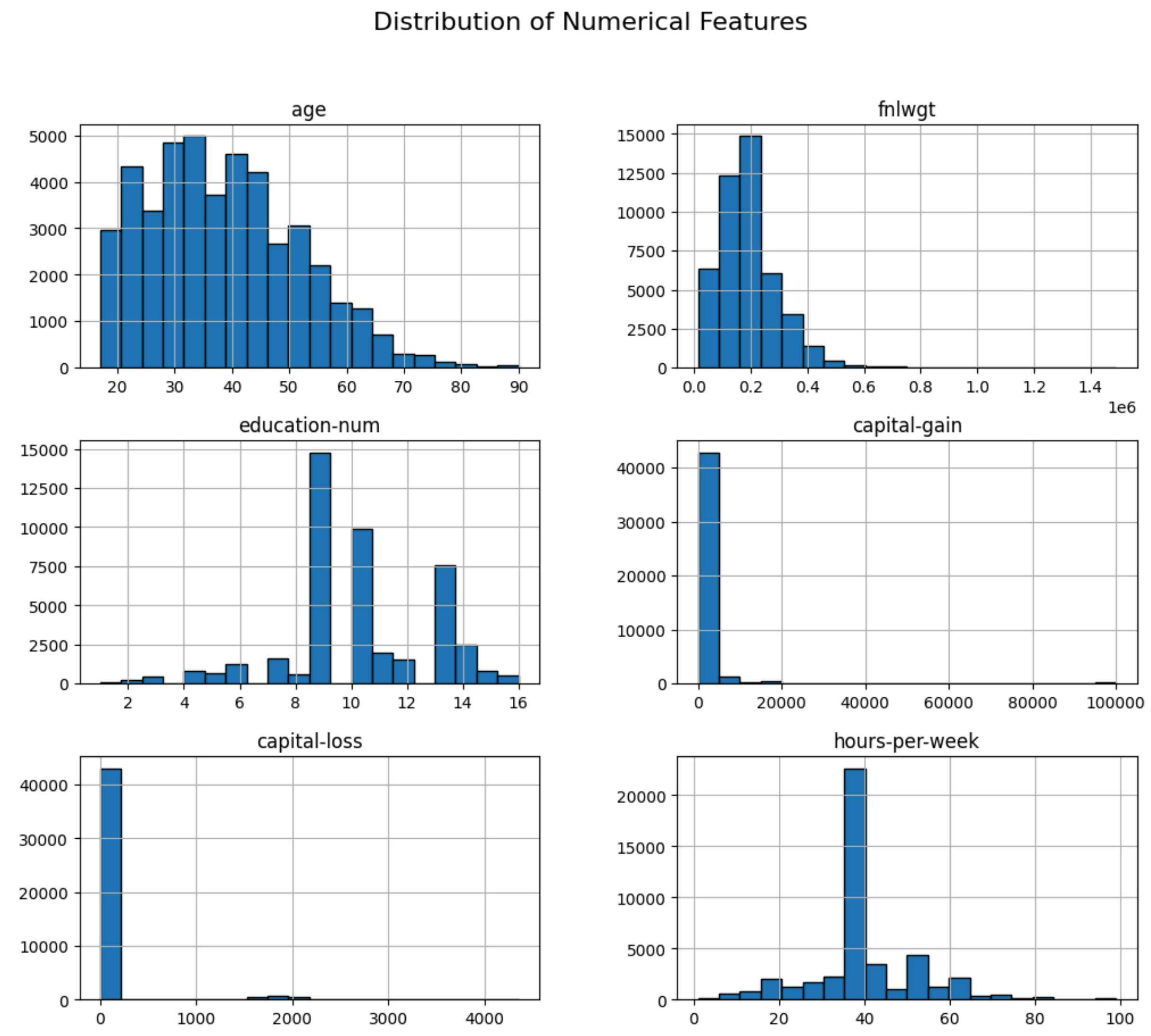
```
Total missing values: 6465
Total cells in dataset: 732630
Missing value ratio: 0.88%
```

```
# 2 & 3. Handle missing values based on the ratio
if missing_ratio < 20:
    print("Ratio is less than 20%. Dropping rows with missing values...")
    df = df.dropna()
else:
    print("Ratio is 20% or higher. Imputing missing values...")
    # Fill numerical columns with median
    num_cols = df.select_dtypes(include=['number']).columns
    for col in num_cols:
        median_val = df[col].median()
        df[col].fillna(median_val, inplace=True)

    # Fill categorical columns with mode (most frequent value)
    cat_cols = df.select_dtypes(include=['object', 'category']).columns
    for col in cat_cols:
        mode_val = df[col].mode()[0]
        df[col].fillna(mode_val, inplace=True)

# Check if any missing values remain
print("Missing values after processing:")
print(df.isnull().sum().sum())
```

```
Ratio is less than 20%. Dropping rows with missing values...
Missing values after processing:
0
```

Q6: Create a grid of histogram plots to display the distribution of all numerical columns in the dataset.

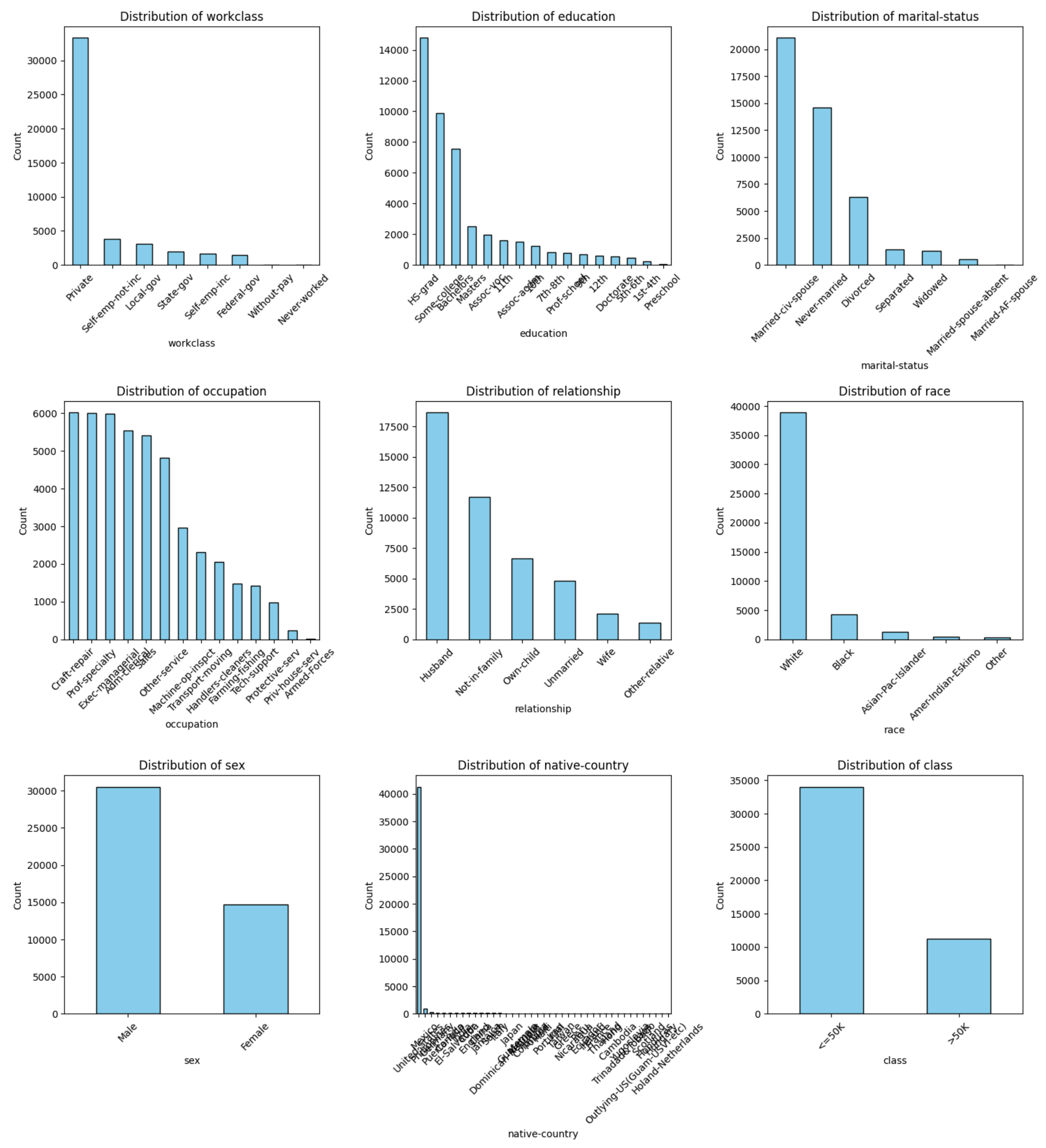
```
# Plot histograms for all numerical columns
df.hist(figsize=(12, 10), bins=20, edgecolor='black')
plt.suptitle("Distribution of Numerical Features", fontsize=16)
plt.show()
```



Q7: For all categorical features in the dataset:

1. Calculate the frequency of each category using the `value_counts()` method.
2. Plot the resulting Series as bar charts arranged in a grid layout to visualize the distributions.

```
# Get categorical columns excluding the target 'class' for now if we want, but question says "all"
# We will plot all categorical columns
cat_cols = df.select_dtypes(include=['object', 'category']).columns

# Set up the figure size and grid
num_plots = len(cat_cols)
cols_grid = 3  # Number of columns in the grid
rows_grid = (num_plots // cols_grid) + 1

plt.figure(figsize=(15, rows_grid * 5))

for i, col in enumerate(cat_cols):
    plt.subplot(rows_grid, cols_grid, i + 1)
    # Calculate frequency
    counts = df[col].value_counts()
    # Plot bar chart
    counts.plot(kind='bar', color='skyblue', edgecolor='black')
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=45)

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



Q8: Assign all feature columns of the dataset to variable X and the target column to variable y.

```
# 'class' is the target variable
X = df.drop('class', axis=1)
y = df['class']
```

```
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
Shape of X: (45222, 14)
Shape of y: (45222,)
```

Q9: Apply one-hot encoding to categorical features using pandas `get_dummies()`.

```
# Convert categorical variables into dummy/indicator variables
# drop_first=True helps to avoid multicollinearity (dummy variable trap)
X = pd.get_dummies(X, drop_first=True)

print("New shape of X after One-Hot Encoding:", X.shape)
print("First few rows of X:")
display(X.head())
```

```
New shape of X after One-Hot Encoding: (45222, 97)
First few rows of X:
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc	..
0	25	226802	7	0	0	40	False	False	True	False	.
1	38	89814	9	0	0	50	False	False	True	False	.
2	28	336951	12	0	0	40	True	False	False	False	.
3	44	160323	10	7688	0	40	False	False	True	False	.
5	34	198693	6	0	0	30	False	False	True	False	.

5 rows × 97 columns

Q10: Split the dataset into train (80%) and test (20%) sets using stratified sampling on the target feature, and print their shapes.

```
# Split data: 80% training, 20% testing
# stratify=y ensures the class distribution is the same in both sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

```
Training set shape: (36177, 97)
Testing set shape: (9045, 97)
```

Q11: Train a logistic regression classifier on the training set.

```
# Initialize Logistic Regression model
# max_iter is increased to ensure the model converges
log_reg = LogisticRegression(max_iter=10000, random_state=42)

# Train the model
log_reg.fit(X_train, y_train)

print("Logistic Regression model trained successfully.")
```

```
Logistic Regression model trained successfully.
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (st
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Q12: Report the accuracy score of the trained Logistic Regression classifier on the test set.

```
# Predict on the test set
y_pred_log = log_reg.predict(X_test)

# Calculate accuracy
accuracy_log = accuracy_score(y_test, y_pred_log)

print(f"Logistic Regression Accuracy: {accuracy_log:.4f}")
```

```
Logistic Regression Accuracy: 0.8429
```

Q13: Train a K-Nearest Neighbors (KNN) classifier with k=5.

```
# Initialize KNN model with k=9
knn = KNeighborsClassifier(n_neighbors=9)

# Train the model
knn.fit(X_train, y_train)

print("KNN model trained successfully.")
```

```
KNN model trained successfully.
```

Q14: Report the accuracy score of the trained K-Nearest Neighbors (KNN) classifier on the test set.

```
# Predict on the test set
y_pred_knn = knn.predict(X_test)

# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)

print(f"KNN Accuracy: {accuracy_knn:.4f}")
```

```
KNN Accuracy: 0.7842
```

Q15: Write two sentences explaining the difference between Logistic Regression and K-Nearest Neighbors (KNN).

Answer:

Logistic Regression is a model that uses a math formula to separate classes. It learns the relationship between inputs and output. K-Nearest Neighbors (KNN) is a model that does not learn a formula. Instead, it looks at the closest data points to make a prediction based on similarity.

In our case, Logistic Regression predicts better results because it has a score of 0.84. KNN has a score of 0.78.

This means Logistic Regression is more accurate for our problem. It is the better choice.