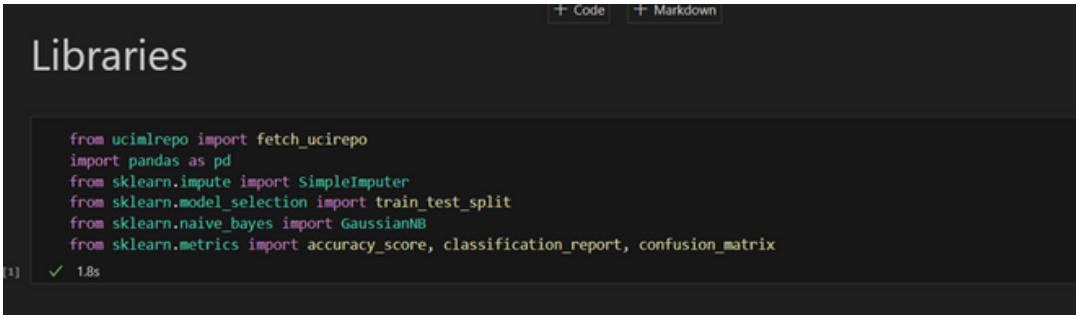
Assignment Report

PREPARED BY :

Ahmed Ehab

1) Libraries



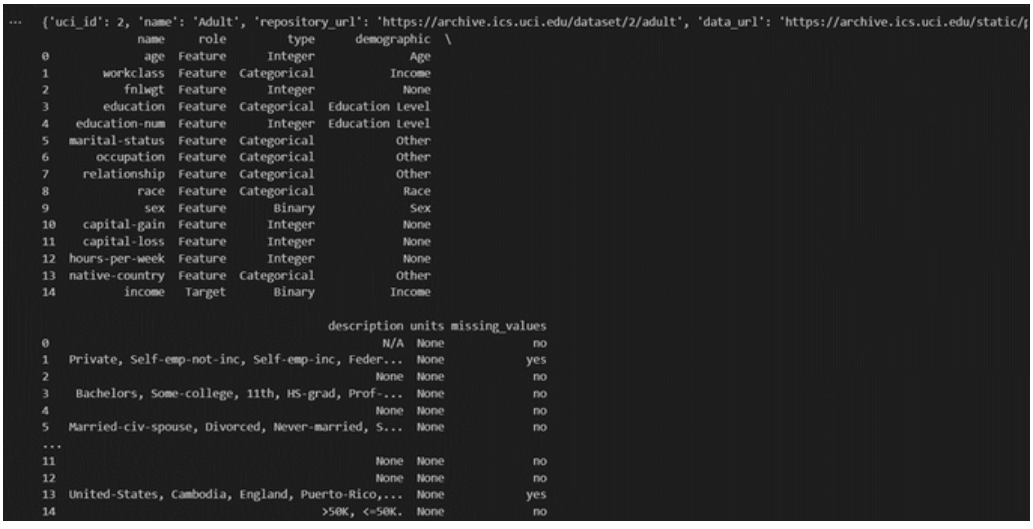
This is libraries I used:

- **Ucimlrepo: to import data**
- **Pandas: to deal with data**
- **Sklearn: to choose classifier, train, test model**

2) Data

This is how I imported data and split data into two data frame one for features and the other one for target
And there was a small problem in column (income) it has 4 classes (<=50k, >50k, <=50k., >50k.) so I replaced (.) with () to fix it and to have 2 classes only

And printed some info:



Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(train_data, test_data, test_size=0.2, random_state=42)
```

✓ 0.0s Python

Then I split data into train and test using (train_test_split)

3) Data cleaning

Check missing values in train and test data

```
missing_values_train = X_train.isnull().values.sum()
print("Total number of missing values for training data: ",missing_values_train)

#percentage of null values
missing_percentage_train = (X_train.isnull().sum() / len(X_train)) * 100
print("Percentage of missing values for training data:")
print(missing_percentage_train)

#highest column percentage of null values
highest_missing_percentage_train = missing_percentage_train.idxmax()
print(f"The column with the highest missing values percentage for training data is: {highest_missing_percentage_train}")

print("="*60)

missing_values_test = X_test.isnull().values.sum()
print("Total number of missing values for testing data: ",missing_values_test)

#percentage of null values
missing_percentage_test = (X_test.isnull().sum() / len(X_test)) * 100
print("Percentage of missing values for testing data:")
print(missing_percentage_test)

#highest column percentage of null values
highest_missing_percentage_test = missing_percentage_test.idxmax()
print(f"The column with the highest missing values percentage for testing data is: {highest_missing_percentage_test}")
```

[4] ✓ 0.0s Python

First we need to check if there is missing values in data, so we will print sum of null values, percentage of null value and highest column of null values for training data and test data

output:

```
Total number of missing values for training data:  1758
Percentage of missing values for training data:
age               0.000000
workclass         1.975789
fnlwgt            0.000000
education         0.000000
education-num     0.000000
marital-status    0.000000
occupation        1.980908
relationship      0.000000
race              0.000000
sex               0.000000
capital-gain      0.000000
capital-loss      0.000000
hours-per-week    0.000000
native-country    0.542574
dtype: float64
The column with the highest missing values percentage for training data is: occupation
============================================================
Total number of missing values for testing data:  445
Percentage of missing values for testing data:
age               0.000000
workclass         1.955164
fnlwgt            0.000000
education         0.000000
...
hours-per-week    0.000000
native-country    0.634661
dtype: float64
The column with the highest missing values percentage for testing data is: occupation
```

Handling missing data in categorical and numerical features

```
numImpute = SimpleImputer(strategy='mean')
catImpute = SimpleImputer(strategy='most_frequent')
numerical_columns = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = [col for col in X_train.columns if pd.api.types.is_categorical_dtype(X_train[col]) or X_train[col].dtype == 'object']
X_train[numerical_columns] = numImpute.fit_transform(X_train[numerical_columns])
X_train[categorical_cols] = catImpute.fit_transform(X_train[categorical_cols])

missing_values_train = X_train.isnull().values.sum()
print("Total number of missing values for training data: ",missing_values_train)

#percentage of null values
missing_percentage_train = (X_train.isnull().sum() / len(X_train)) * 100
print("Percentage of missing values for training data:")
print(missing_percentage_train)

print("="*60)

numImpute = SimpleImputer(strategy='mean')
catImpute = SimpleImputer(strategy='most_frequent')
numerical_columns = X_test.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = [col for col in X_test.columns if pd.api.types.is_categorical_dtype(X_test[col]) or X_test[col].dtype == 'object']
X_test[numerical_columns] = numImpute.fit_transform(X_test[numerical_columns])
X_test[categorical_cols] = catImpute.fit_transform(X_test[categorical_cols])

missing_values_test = X_test.isnull().values.sum()
print("Total number of missing values for testing data: ",missing_values_test)

#percentage of null values
missing_percentage_test = (X_test.isnull().sum() / len(X_test)) * 100
print("Percentage of missing values for testing data:")
print(missing_percentage_test)
```

We need a way to handle this missing data if it was numerical or categorical, so we will use (simpleimputer) select mean strategy for numerical data and most frequent for categorical data, then we will select numerical columns and categorical columns from train and test data sets and fit them with our strategy using (simpleimputer), then check for missing values again to make sure there is no missing values any more

output:

```
Total number of missing values for training data:  0
Percentage of missing values for training data:
age               0.0
workclass         0.0
fnlwgt            0.0
education         0.0
education-num     0.0
marital-status    0.0
occupation        0.0
relationship      0.0
race              0.0
sex               0.0
capital-gain      0.0
capital-loss      0.0
hours-per-week    0.0
native-country    0.0
dtype: float64
============================================================
Total number of missing values for testing data:  0
Percentage of missing values for testing data:
age               0.0
workclass         0.0
fnlwgt            0.0
education         0.0
education-num     0.0
...
capital-loss      0.0
hours-per-week    0.0
native-country    0.0
dtype: float64
```

4) Encoding

Encode categorical variables

```
X_train = pd.get_dummies(X_train, columns=categorical_cols)
X_test = pd.get_dummies(X_test, columns=categorical_cols)

#ensure the train and test datasets have the same columns after encoding
missing_cols_test = set(X_train.columns) - set(X_test.columns)
for col in missing_cols_test:
    X_test[col] = 0

X_test = X_test[X_train.columns]
```

✓ 0.0s

we will make simple one-hot encoding to make sure that categorical columns in training and testing data set in same format

5) Training and Testing

Train the Naive Bayes classifier and make predictions on the test set

```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

y_pred = nb_classifier.predict(X_test)
```

[7] ✓ 0.1s Python

we will choose model (GaussianNB), then we will fit our training data usimg it (Training)
After tarin model, we will predict (Testing)

6) Score

Calculate statistics and score

```
#calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

#generate and print the classification report
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)

#calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

#check the shape of the confusion matrix
print("Confusion Matrix Shape:", conf_matrix.shape)

#extract values from the confusion matrix
if conf_matrix.shape == (2, 2):
    tn, fp, fn, tp = conf_matrix.ravel()
    #compute Sensitivity and Specificity
    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)
    print(f"Sensitivity: {sensitivity}, Specificity: {specificity}")
else:
    print("Error: Confusion matrix is not of shape (2, 2), check your data and predictions.")

#compute the posterior probability of making over 50K a year
posterior_probabilities = nb_classifier.predict_proba(X_test)[:, 1]
print("Posterior probabilities of making over 50K a year:")
print(posterior_probabilities)
```

[8] ✓ 0.4s Python

First, we will calculate accuracy of model using (accuracy_score), then we will print small report of classification using (classification_report), then we will make confusion matrix using (confusion_matrix) and make small function to make sure matrix is in right shape (2x2), then we will calculate (sensitivity, speciticity and posterior probability)

output:

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.81      0.95      0.88      7414
        >50K       0.67      0.31      0.42      2355

    accuracy                           0.80      9769
   macro avg       0.74      0.63      0.65      9769
weighted avg       0.78      0.80      0.77      9769

Confusion Matrix Shape: (2, 2)
Sensitivity: 0.30530785562632695, Specificity: 0.9511734556244942
Posterior probabilities of making over 50K a year:
[0.0003394  0.0003338  0.00294792 ... 0.01748743 0.02171417 0.01308272]
```