Faculty of Computer and Data Sciences, Alexandria University.
Course Title: Pattern Recognition 02-24-03203
Department: Intelligent Systems (2nd year)

Adult Dataset - UCI Machine Learning Repository Assignment

- مروان طارق امبابي رزق ابراهيم 22011615

Importing libraries and dataset

We start by loading a dataset from the UCI Machine Learning Repository using the *ucimlrepo* library. We print its metadata and variable information, extract features and target variable, and combine them into a Pandas DataFrame called data. This DataFrame is now ready for further analysis or machine learning modeling.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	<=50K.
48838	64	NaN	321403	HS-grad	9	Widowed	NaN	Other-relative	Black	Male	0	0	40	United-States	<=50K.
48839	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	<=50K.
48840	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	<=50K.
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	>50K.

48842 rows × 15 columns

Pre-processing

```
workclass
Private             33906
Self-emp-not-inc     3862
Local-gov            3136
State-gov            1981
?                    1836
Self-emp-inc         1695
Federal-gov          1432
Without-pay            21
Never-worked           10
```

```
occupation
Prof-specialty       6172
Craft-repair         6112
Exec-managerial      6086
Adm-clerical         5611
Sales                5504
Other-service        4923
Machine-op-inspct    3022
Transport-moving     2355
Handlers-cleaners    2072
?                    1843
Farming-fishing      1490
Tech-support         1446
Protective-serv       983
Priv-house-serv       242
Armed-Forces           15
```

```
native-country
United-States        43832
Mexico                 951
?                      583
Philippines            295
Germany                206
Puerto-Rico            184
Canada                 182
El-Salvador            155
India                  151
Cuba                   138
England                127
China                  122
South                  115
```

Here we can see that in this dataset we have both null (NaN) values and missing values denoted by “?”, we deal with both of these issues separately by deleting the rows containing them

Null Values

```
[49]: print(data.isnull().sum())

age                 0
workclass         963
fnlwgt              0
education           0
education-num       0
marital-status      0
occupation        966
relationship        0
race                0
sex                 0
capital-gain        0
capital-loss        0
hours-per-week      0
native-country    274
income              0
dtype: int64
```

Null Values after deletion

```
[51]: print(data.isnull().sum())

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

"?" values

```
[52]: data.isin(['?']).any()

[52]: age                False
      workclass           True
      fnlwgt             False
      education          False
      education-num      False
      marital-status     False
      occupation          True
      relationship       False
      race               False
      sex                False
      capital-gain       False
      capital-loss       False
      hours-per-week     False
      native-country      True
      income             False
      dtype: bool
```

After Deletion

```
[54]: data.isin(['?']).any()

[54]: age                False
      workclass          False
      fnlwgt             False
      education          False
      education-num      False
      marital-status     False
      occupation         False
      relationship       False
      race               False
      sex                False
      capital-gain       False
      capital-loss       False
      hours-per-week     False
      native-country     False
      income             False
      dtype: bool
```

Duplicate Values:

Here we print the number of duplicate rows in the DataFrame data before and after dropping duplicates.

```
Duplicate Values Before Deletion:
28
Duplicate Values After Deletion:
0
```

We remove duplicate rows from the DataFrame using the drop_duplicates() method and assigns the result back to data. Then we print the count of duplicate rows after removal.

Fixing mistyped (incorrect) values in the target column

First, we print the count of unique values in the 'income' column using `value_counts()` method, which counts occurrences of each unique value. Then, we replace any occurrences of '<=50K.' with '<=50K' and '>50K.' with '>50K' using the `replace()` method, making the values consistent. After replacement, we verify the changes.

Fix incorrect income values

```
[56]: print(data['income'].value_counts().to_string())
```

```
income
<=50K     22633
<=50K.    11355
>50K       7506
>50K.      3700
```

```
[57]: data['income'].replace('<=50K.', '<=50K', inplace=True)
      data['income'].replace('>50K.', '>50K', inplace=True)
```

```
[58]: print(data['income'].value_counts().to_string())
```

```
income
<=50K    33988
>50K     11206
```

Dataset after preprocessing:

[59]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48836	33	Private	245211	Bachelors	13	Never-married	Prof-specialty	Own-child	White	Male	0	0	40	United-States	<=50K
48837	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	<=50K
48839	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	<=50K
48840	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	<=50K
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	>50K

45194 rows × 15 columns

Encoding Categorical Values:

[61]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	income	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	...	native-country_Portugal	native-country_Puerto-Rico	n country_Sc
0	39	77516	13	2174	0	40	<=50K	False	False	False	...	False	False	
1	50	83311	13	0	0	13	<=50K	False	False	False	...	False	False	
2	38	215646	9	0	0	40	<=50K	False	True	False	...	False	False	
3	53	234721	7	0	0	40	<=50K	False	True	False	...	False	False	
4	28	338409	13	0	0	40	<=50K	False	True	False	...	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
48836	33	245211	13	0	0	40	<=50K	False	True	False	...	False	False	
48837	39	215419	13	0	0	36	<=50K	False	True	False	...	False	False	
48839	38	374983	13	0	0	50	<=50K	False	True	False	...	False	False	
48840	44	83891	13	5455	0	40	<=50K	False	True	False	...	False	False	
48841	35	182148	13	0	0	60	>50K	False	False	True	...	False	False	

45194 rows × 97 columns

Here, we prepare categorical columns in the DataFrame `data` for machine learning by converting them into dummy variables through one-hot encoding. Initially, we assign the name of the target column, 'income', to `target_column`. Then, we generate a list of categorical columns (`categorical_columns`) excluding the target column and with data type 'object'. We apply one-hot encoding to these categorical columns, creating dummy variables while dropping the first level to prevent multicollinearity.

Splitting:

Divide Dataset into Features and Target

```
[62]: X=data.drop(columns=[target_column])
      y=data[target_column]

      ## Split Dataset using train_test_split

[64]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

[65]: print("X_train shape:", X_train.shape)
      print("y_train shape:", y_train.shape)

      X_train shape: (31635, 96)
      y_train shape: (31635,)
```

We first divide the dataset into features and target vector (income) and then using sklearn's train_test_split we split the dataset's rows in a 70/30 ratio into a training set and a testing set.

Using Gaussian Naive-Bayes Classifier to train dataset and predict target

```
[66]: from sklearn.naive_bayes import GaussianNB

      # Classifier
      classifier = GaussianNB()
      classifier.fit(X_train, y_train)

      # Prediction
      y_pred = classifier.predict(X_test)
      y_test_array = y_test.values

[67]: print(len(y_test_array))
      print(len(y_pred))

      13559
      13559
```

This code uses the Gaussian Naive Bayes classifier from sklearn. It initializes the classifier, fits it to the training data, predicts labels for the test data and converts the test labels to a NumPy array.

Calculating Evaluation Metrics from Confusion Matrix

```
[68]: def calculate_metrics(y_true, y_pred):
          TP = np.sum((y_true == '>50K') & (y_pred == '>50K'))
          TN = np.sum((y_true == '<=50K') & (y_pred == '<=50K'))
          FP = np.sum((y_true == '<=50K') & (y_pred == '>50K'))
          FN = np.sum((y_true == '>50K') & (y_pred == '<=50K'))

          accuracy = (TP + TN) / (TP + TN + FP + FN)
          sensitivity = TP / (TP + FN)
          specificity = TN / (TN + FP)

          return accuracy, sensitivity, specificity
```

Accuracy, Sensitivity and Specificity

```
[69]: print(f"Accuracy: {calculate_metrics(y_test, y_pred)[0]}\nSensitivity: {calculate_metrics(y_test, y_pred)[1]}\nSpecificity: {calculate_metrics(y_test, y_pr

      Accuracy: 0.7868574378641493
      Sensitivity: 0.307989307989308
      Specificity: 0.945054945054945
```

Using this function we can calculate the confusion matrix and its resulting metrics (Accuracy, Sensitivity, Specificity).

Calculating the posterior probabilities of making over 50k a year

```
[70]: probabilities = classifier.predict_proba(X_test)

[71]: prob_over_50k = probabilities[:, 1]
      prob_over_50k

[71]: array([0.0143046 , 0.03193122, 0.00278249, ..., 0.02371454, 0.01353673,
             0.01025875])
```

We can find the posterior probabilities of the class being “>50k” by using the predict_proba() function from the classifier we trained previously.

Calculating MAP (Maximum A Posteriori)

[73]:
```python
# Find Prob for both classes in the dataset
predicted_classes = classifier.predict(X_test)

# Calculate the maximum posterior probability (MAP) for each instance
max_posterior_probs = probabilities.max(axis=1)

map_estimates = [prob if pred_class == 1 else 1 - prob for prob, pred_class in zip(max_posterior_probs, predicted_classes)]

print(f"Maximum A Posteriori (MAP) estimates:", map_estimates)
```

```
Maximum A Posteriori (MAP) estimates: [0.014304598210601038, 0.03193121839745905, 0.0027824853265639993, 0.008048190957849921, 2.2737367544323206e-13, 0.00
747449267305722, 0.008315234479099654, 0.0, 0.0033652891023130405, 0.0024827772687644467, 0.012663167069375425, 0.014323203349148828, 0.03629173133796715,
0.0029275801175145943, 0.0005623658372135854, 0.01616092535001412, 0.0005702934610837262, 0.01906047337876915, 1.7053025658242404e-13, 0.001906863782068279
9, 0.0020051855051174128, 0.018579393327482152, 0.014851421275694832, 0.009244691378330727, 0.01779946579270264, 7.584580141717367e-08, 0.00081574619553581
58, 0.008797245247889385, 0.003119794312856139, 0.009790785118442225, 0.01668846621341924, 0.1984044913915317, 0.009365459788997277, 0.0072610742526253436,
0.002285169320526581, 0.013626451917499338, 0.02696138358759681, 0.012238233407886434, 0.0019405876497728958, 0.030744185664165702, 0.0016662469496157017,
0.016021758899686045, 0.0014879744413383023, 0.009977800504598178, 0.012873612223553321, 0.006153013129452378, 0.010342489505890007, 0.008055724296464617,
0.018140628504526912, 0.004711063154741679, 0.0017243635137841329, 0.02462968176029956, 0.014017513199844012, 0.0046982563900198615, 0.0009331234112275322,
0.010107391117418096, 0.0037653464854324747, 0.016980601958695885, 0.01892783916778573, 0.010126380227233889, 0.01699541085200751, 0.006635896590653778, 0.
006551888829099406, 0.030567751666109033, 2.6945197648564623e-05, 0.012150517597101507, 0.00016166182202459467, 0.0, 0.010052945713537431, 0.02032639195550
9115, 0.0005032403338689484, 0.0018375958497156564, 0.01156389403339253, 0.01286521927980322, 0.006903033745596754, 0.0034400531029622483, 0.01447330219886
3669, 0.025362584687833367, 0.014092121662783064, 0.01130333565354058, 0.0007356493695193089, 0.0248406925992829, 0.01984699098986309, 0.007745178197661517
5, 0.005876656691185866, 0.0009905970085812843, 0.012354075169809686, 0.025474636903256576, 0.0194177068216822, 0.0015545500019609193, 0.006438393840162204
, 0.015335329941758347, 0.006962913945428251, 0.008521020230002896, 0.00286180510060563, 0.003100417218687701, 0.0023323103715560123, 0.010699075836778094,
0.024156897606910088, 0.012856379627430692, 9.345556507245334e-05, 0.0017156330129335373, 0.011377684639825358, 0.009070625759627249, 0.001473927072965786,
0.05447154603656079, 2.2159111200847903e-05, 0.011124673090237236, 0.0037536386046337356, 0.007202409999839565, 0.011750994772268597, 0.05471570111587731,
0.00164360837490507, 3.659389669496971e-05, 0.018105212353075495, 0.002597202506611662, 0.008641550684276633, 0.007565815615891647, 0.0006147699437318854,
3.538920694545755e-05, 0.006623273987770406, 0.0019903537514114333, 0.0033691060432672337, 0.01351888750752639, 0.032539361699626346, 0.0016962283198349581
```

This code calculates the Maximum A Posteriori (MAP) estimates for each instance in the dataset. It first predicts the classes for the test data and then calculates the maximum posterior probability (MAP) for each instance. Using these probabilities and predicted classes, it computes the MAP estimates and prints them.