UCI Adult Income Dataset - Data cleaning and preprocessing

In this notebook, we focus on **data preparation**, **cleaning**, and **preprocessing** for the **UCI Adult Income Dataset**, a popular dataset often used for classification tasks predicting whether an individual earns more or less than $50,000 annually based on demographic and work-related attributes.

Good data preprocessing is crucial for reliable and interpretable results in machine learning and analytics workflows. Here, we address common data issues such as **missing values, duplicates, and inconsistent categorical labels** while creating derived features to improve downstream analysis.

We start by importing essential Python libraries for data handling and manipulation.

- `pandas` for structured data operations.
- `numpy` for numerical operations.
- `os` for interacting with the operating system and directory structures.

```
# import libraries
import pandas as pd
import numpy as np
import os
```

Define and Create Directory Paths

To ensure reproducibility andorganized storage, we programmatically create directories for:

- **raw data**
- **processed data**
- **results**
- **documentation**

These directories will store intermediate and final outputs for reproducibility.

```
# Get working directory
current_dir= os.getcwd()
# Go one directory up to the root directory
project_root_dir= os.path.dirname(current_dir)
data_dir = os.path.join(project_root_dir,'data')
raw_dir = os.path.join(data_dir,'raw')
processed_dir = os.path.join(data_dir,'processed')
# define path to result folder
result_dir=os.path.join(project_root_dir,'result')
# define path to docs folder
docs_dir=os.path.join(project_root_dir,'docs')

# create directory if they do not exist
os.makedirs(raw_dir, exist_ok = True)
os.makedirs(processed_dir, exist_ok = True)
os.makedirs(result_dir, exist_ok = True)
os.makedirs(docs_dir, exist_ok = True)
```

Read in the data

We load the **Adult Income dataset** as a CSV file.

Key considerations here are:

- We treat `?` as missing values (`na_values = '?'`).
- We use `skipinitialspace = True` to remove extra spaces after delimeters which is common in text-based datasets.

After loading, we inspect the first few rows.

```
adult_data_filename = os.path.join(raw_dir,"adult.csv")
adult_df = pd.read_csv(adult_data_filename,header=None,na_values = '?', skipinitialspace = T
adult_df.head(10)
```

	0	1	2	3	4	5	6	7
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-famil
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-famil
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband

	0	1	2	3	4	5	6	7
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-famil
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband
8	31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-famil
9	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband

We also inspect the dataset's shape. We see that the data has *32,561* rows and *15* columns.

```
adult_df.shape
```

```
(32561, 15)
```

In addition, we check the data types using `.info`.

```
adult_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       32561 non-null  int64 
 1   1       30725 non-null  object
 2   2       32561 non-null  int64 
 3   3       32561 non-null  object
 4   4       32561 non-null  int64 
 5   5       32561 non-null  object
 6   6       30718 non-null  object
 7   7       32561 non-null  object
 8   8       32561 non-null  object
 9   9       32561 non-null  object
 10  10      32561 non-null  int64 
 11  11      32561 non-null  int64 
 12  12      32561 non-null  int64 
 13  13      31978 non-null  object
 14  14      32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Data Cleaning

1. Assign proper column names to the columns

One of the most stricking things from the above inspection is that the dataset lacks explicit column headers. We manually assign descriptive meaningful column names based on the description of the dataset. This is critical for readability and interpretability in the subsequent steps.

```
adult_df.columns=["age", "workclass", "fnlwgt", "education", " education_num", "martial_statu
adult_df
```

We inspect again to see whether they are properly assigned.

2. Understanding the dataset

Before proceeding with the cleaning, we would like to understanding the variables deeply. This would help guide the cleaning process. The subsequent tables detail the types, meaning and values or ranges of the variables in the dataset.

Table 1: Summary table of the variables in the dataset

Variable	Type	Description	Values / Range (excluding nan)
age	Numeric	Age in years	17 – 90
fnlwgt	Numeric	Final sampling weight	~12,285 – 1,484,705
education_num	Numeric	Education level in years	1 – 16
capital_gain	Numeric	Capital gain amounts (Profit from selling assets above purchase price within the survey year (in USD))	0 – 99,999
capital_loss	Numeric	Capital loss amounts (Loss from selling assets below purchase price within the survey year (in USD))	0 – 4,356
hours_per_week	Numeric	Weekly work hours	1 – 99
workclass	Categorical	Type of employment	8 categories
education	Categorical	Highest level of education achieved	16 categories
marital_status	Categorical	Marital status	7 categories
occupation	Categorical	Type of job	14 categories
relationship	Categorical	Relationship within household	6 categories
race	Categorical	Ethnic/racial group	5 categories
sex	Categorical	Gender	2 categories

Variable	Type	Description	Values / Range (excluding nan)
native_country	Categorical	Country of origin	41 categories
income	Categorical	Income category (target variable)	2 categories: <=50K, >50K

Table 2: Categorical Variables Table | Variable | Unique Value | Description | |:————————-|:——————|:————————————————————————-| | workclass | Private | Works for a private, for-profit company | | | Self-emp-not-inc | Self-employed without incorporated business status | | | Self-emp-inc | Self-employed with an incorporated business | | | Federal-gov | Employed by the federal government | | | State-gov | Employed by a state government | | | Local-gov | Employed by a local government | | | Without-pay | Works without receiving pay (e.g. unpaid family worker) | | | Never-worked | Has never worked in their lifetime | | education | Bachelors | Bachelor's degree | | | Some-college | Some college courses completed, no degree | | | 11th | 11th grade completed | | | HS-grad | High school graduate | | | Prof-school | Professional school (e.g. law, medicine) | | | Assoc-acdm | Associate degree (academic) | | | Assoc-voc | Associate degree (vocational) | | | 9th | 9th grade completed | | | 7th-8th | 7th or 8th grade completed | | | 12th | 12th grade, no diploma | | | Masters | Master's degree | | | 1st-4th | 1st to 4th grade completed | | | 10th | 10th grade completed | | | Doctorate | Doctoral degree | | | 5th-6th | 5th or 6th grade completed | | | Preschool | Preschool education | | marital-status | Married-civ-spouse | Married, living with spouse | | | Divorced | Divorced legally | | | Never-married | Never married | | | Separated | Separated legally but not divorced | | | Widowed | Spouse deceased | | | Married-spouse-absent| Married, spouse not present (e.g. estrangement) | | | Married-AF-spouse | Married to a spouse who is a member of the Armed Forces | | occupation | Tech-support | Technical support jobs | | | Craft-repair | Skilled manual trade and repair jobs | | | Other-service | Services not classified elsewhere | | | Sales | Sales-related jobs | | | Exec-managerial | Executive and managerial roles | | | Prof-specialty | Professional specialty occupations (e.g. scientist, lawyer) | | | Handlers-cleaners | Manual labor jobs involving cleaning, handling objects | | | Machine-op-inspct | Machine operators, inspectors | | | Adm-clerical | Administrative and clerical jobs | | | Farming-fishing | Agriculture, farming, fishing occupations | | | Transport-moving | Transport and moving equipment operators | | | Priv-house-serv | Private household service jobs | | | Protective-serv | Protective service jobs (e.g. security, law enforcement) | | | Armed-Forces | Military service | | relationship | Wife | Female spouse | | | Own-child | Biological or adopted child | | | Husband | Male spouse | | | Not-in-family | Not part of a family unit (e.g. living alone) | | | Other-relative | Other relative in household | | | Unmarried | Single person, not married | | race | White | White | | | Asian-Pac-Islander | Asian or Pacific Islander | | | Amer-Indian-Eskimo | American Indian or Eskimo | | | Other | Other race not listed | | | Black | Black | | sex | Female | Female | | | Male | Male | | native-country | United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-

Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad-Tobago, Peru, Hong, Holland-Netherlands | | | income | <=50K | Income less than or equal to USD 50,000 | | | >50K | Income greater than USD 50,000 |

```
np.unique(adult_df.workclass.to_list())
```

```
array(['Federal-gov', 'Local-gov', 'Never-worked', 'Private',
       'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-pay',
       'nan'], dtype='<U32')
```

```
np.unique(adult_df.fnlwgt.to_list())
```

```
array([  12285,   13769,   14878, ..., 1366120, 1455435, 1484705])
```

```
np.unique(adult_df.education.to_list())
```

```
array(['10th', '11th', '12th', '1st-4th', '5th-6th', '7th-8th', '9th',
       'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-grad',
       'Masters', 'Preschool', 'Prof-school', 'Some-college'],
      dtype='<U12')
```

2.Deal with missing values

```
adult_df.isnull().sum()
```

```
age                  0
workclass         1836
fnlwgt               0
education            0
 education_num       0
martial_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
```

Using .isnull().sum(), we identify columns with missing values. They are:

workclass with 1,836 missing values occupation with 1,843 missing values native_country with 583 missing values We address these by:

Imputing categorical missing values with Unknown for the columns workclass and occupation Imputing categorical missing values with Other for the column native_country This has been done to preserve data consistency while acknowledging uncertainity.

```
adult_df['workclass']=adult_df['workclass'].fillna('Unknown')
adult_df['native_country']=adult_df['native_country'].fillna('Other')
adult_df['occupation']=adult_df['occupation'].fillna('Unknown')
```

We inspect one more time to ensure we don't have any missing values.

```
adult_df.isnull().sum()
```

```
age               0
workclass         0
fnlwgt            0
education         0
 education_num    0
martial_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
```

3. Removing Duplicates

Duplicates can distort statistical summaries and model performance. Using `.duplicated().sum()`, we count duplicate records.

```
adult_df.duplicated().sum()
```

```
24
```

We then inspect the duplicated records.

```
adult_df[adult_df.duplicated(keep=False)]
```

	age	workclass	fnlwgt	education	education_num	martial_status	occupation
2303	90	Private	52386	Some-college	10	Never-married	Other-service
3917	19	Private	251579	Some-college	10	Never-married	Other-service
4325	25	Private	308144	Bachelors	13	Never-married	Craft-repair
4767	21	Private	250051	Some-college	10	Never-married	Prof-specialty
4881	25	Private	308144	Bachelors	13	Never-married	Craft-repair
4940	38	Private	207202	HS-grad	9	Married-civ-spouse	Machine-op-in
5104	90	Private	52386	Some-college	10	Never-married	Other-service
5579	27	Private	255582	HS-grad	9	Never-married	Machine-op-in
5805	20	Private	107658	Some-college	10	Never-married	Tech-support
5842	25	Private	195994	1st-4th	2	Never-married	Priv-house-ser
6990	19	Private	138153	Some-college	10	Never-married	Adm-clerical
7053	49	Self-emp-not-inc	43479	Some-college	10	Married-civ-spouse	Craft-repair
7920	49	Private	31267	7th-8th	4	Married-civ-spouse	Craft-repair
8080	21	Private	243368	Preschool	1	Never-married	Farming-fishin
8679	28	Private	274679	Masters	14	Never-married	Prof-specialty
9171	21	Private	250051	Some-college	10	Never-married	Prof-specialty
10367	42	Private	204235	Some-college	10	Married-civ-spouse	Prof-specialty
11631	20	Private	107658	Some-college	10	Never-married	Tech-support
11965	46	Private	133616	Some-college	10	Divorced	Adm-clerical
13084	25	Private	195994	1st-4th	2	Never-married	Priv-house-ser
15059	21	Private	243368	Preschool	1	Never-married	Farming-fishin
15189	19	Private	146679	Some-college	10	Never-married	Exec-manager
16297	46	Private	173243	HS-grad	9	Married-civ-spouse	Craft-repair
16846	35	Private	379959	HS-grad	9	Divorced	Other-service
16975	30	Private	144593	HS-grad	9	Never-married	Other-service
17040	46	Private	173243	HS-grad	9	Married-civ-spouse	Craft-repair

	age	workclass	fnlwgt	education	education_num	martial_status	occupation
17673	19	Private	97261	HS-grad	9	Never-married	Farming-fishin
17916	44	Private	367749	Bachelors	13	Never-married	Prof-specialty
18555	30	Private	144593	HS-grad	9	Never-married	Other-service
18698	19	Private	97261	HS-grad	9	Never-married	Farming-fishin
21103	23	Private	240137	5th-6th	3	Never-married	Handlers-clean
21318	19	Private	138153	Some-college	10	Never-married	Adm-clerical
21490	19	Private	146679	Some-college	10	Never-married	Exec-manager
21875	49	Private	31267	7th-8th	4	Married-civ-spouse	Craft-repair
22300	25	Private	195994	1st-4th	2	Never-married	Priv-house-ser
22367	44	Private	367749	Bachelors	13	Never-married	Prof-specialty
22494	49	Self-emp-not-inc	43479	Some-college	10	Married-civ-spouse	Craft-repair
25624	39	Private	30916	HS-grad	9	Married-civ-spouse	Craft-repair
25872	23	Private	240137	5th-6th	3	Never-married	Handlers-clean
26313	28	Private	274679	Masters	14	Never-married	Prof-specialty
28230	27	Private	255582	HS-grad	9	Never-married	Machine-op-in
28522	42	Private	204235	Some-college	10	Married-civ-spouse	Prof-specialty
28846	39	Private	30916	HS-grad	9	Married-civ-spouse	Craft-repair
29157	38	Private	207202	HS-grad	9	Married-civ-spouse	Machine-op-in
30845	46	Private	133616	Some-college	10	Divorced	Adm-clerical
31993	19	Private	251579	Some-college	10	Never-married	Other-service
32404	35	Private	379959	HS-grad	9	Divorced	Other-service

Finally, we remove them with `.drop_duplicates()`.

```
adult_df=adult_df.drop_duplicates()
```

We can confirm that we have no duplicates left in the dataset at this juncture.

```
adult_df.duplicated().sum()
```

```
0
```

We also inspect the current shape of the dataset and see that we have *32,537* rows and *15* columns.

```
adult_df.shape
```

```
(32537, 15)
```

5. Standardize Categorical Variables

Remove any leading or trailing spaces and convert the strings to lowercase

To prepare categorical variables for consistent processing, we first of all remove extra spaces and convert them to lowercase. This step ensures categorical variables are clean and consistently organized.

```
adult_df.dtypes == object
```

```
age                False
workclass           True
fnlwgt             False
education           True
 education_num     False
martial_status      True
occupation          True
relationship        True
race                True
sex                 True
capital_gain       False
capital_loss       False
hours_per_week     False
native_country      True
income              True
dtype: bool
```

```
categorical_cols = adult_df.columns[adult_df.dtypes == object]
for col in categorical_cols:
    adult_df.loc[:, col] = adult_df[col].str.strip().str.lower()
adult_df
```

	age	workclass	fnlwgt	education	education_num	martial_status	occupation
0	39	state-gov	77516	bachelors	13	never-married	adm-clerical
1	50	self-emp-not-inc	83311	bachelors	13	married-civ-spouse	exec-managerial
2	38	private	215646	hs-grad	9	divorced	handlers-cleaner
3	53	private	234721	11th	7	married-civ-spouse	handlers-cleaner
4	28	private	338409	bachelors	13	married-civ-spouse	prof-specialty
...	...	...	...	...	...	...	...
32556	27	private	257302	assoc-acdm	12	married-civ-spouse	tech-support
32557	40	private	154374	hs-grad	9	married-civ-spouse	machine-op-insp

	age	workclass	fnlwgt	education	education_num	martial_status	occupation
32558	58	private	151910	hs-grad	9	widowed	adm-clerical
32559	22	private	201490	hs-grad	9	never-married	adm-clerical
32560	52	self-emp-inc	287927	hs-grad	9	married-civ-spouse	exec-managerial

```
adult_df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', ' education_num',
       'martial_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income'],
      dtype='object')
```

Re-code the `workclass` column

We re-code the `workclass` column to broader categories like `government`, `private`, `self-employed`, etc. Table 3 shows the new encoding:

Table 3: Re-encoding of the `workclass` column

Old categories	New Categories
`state-gov`	`government`
`local-gov`	`government`
`federal-gov`	`government`
`self-emp-not-inc`	`self-employed`
`self-emp-inc`	`self-employed`
`never-worked`	`unemployed`
`without-pay`	`voluntary`

```
adult_df['workclass'].unique()
```

```
array(['state-gov', 'self-emp-not-inc', 'private', 'federal-gov',
       'local-gov', nan, 'self-emp-inc', 'without-pay', 'never-worked'],
      dtype=object)
```

```
adult_df.loc[:,'workclass'] = adult_df['workclass'].replace({
    'state-gov': 'government' ,
    'loc-gov': 'government' ,
    'federal-gov': 'government' ,
```

```
    'self-emp-not-inc': 'self-employed' ,
    'self-emp-inc': 'self-employed' ,
    'never-worked': 'unemployed' ,
    'without-pay': 'voluntary' ,
})
```

```
adult_df['workclass'].unique()
```

```
array(['state-gov', 'self-emp-not-inc', 'private', 'federal-gov',
       'local-gov', nan, 'self-emp-inc', 'without-pay', 'never-worked'],
      dtype=object)
```

Re-code the `education` column

We create a new column `education_level` with broader education groups. The mapping from `education` to `education_level` is as follows:

Table 4: Mapping from `education` to `education_level`

Education	Education Level
bachelors	tertiary
masters	tertiary
doctorate	tertiary
prof-school	tertiary
some-college	some college
assoc-acdm	associate
assoc-voc	associate
hs-grad	secondary-school graduate
12th	secondary
11th	secondary
10th	secondary
9th	secondary
7th-8th	primary
5th-6th	primary
1st-4th	primary
preschool	preschool

```
adult_df['education'].unique()
```

```
array(['bachelors', 'hs-grad', '11th', 'masters', '9th', 'some-college',
       'assoc-acdm', 'assoc-voc', '7th-8th', 'doctorate', 'prof-school',
       '5th-6th', '10th', '1st-4th', 'preschool', '12th'], dtype=object)
```

```python
adult_df.loc[:,'education_level'] = adult_df['education'].map({
    'bachelors': 'tertiary',
    'masters': 'tertiary',
    'doctorate': 'tertiary',
    'prof-school': 'tertiary',
    'some-college': 'some college',
    'assoc-acdm': 'associate',
    'assoc-voc': 'associate',
    'hs-grad': 'high school graduate',
    '12th': 'secondary',
    '11th': 'secondary',
    '10th': 'secondary',
    '9th': 'secondary',
    '7th-8th': 'primary',
    '5th-6th': 'primary',
    '1st-4th': 'primary',
    'preschool': 'preschool'
})
```

```python
adult_df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', ' education_num',
       'martial_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income', 'education_level'],
      dtype='object')
```

```python
adult_df['education_level'].unique()
```

```
array(['tertiary', 'high school graduate', 'secondary', 'some college',
       'associate', 'primary', 'preschool'], dtype=object)
```

Re-code the `marital_status` column

The categories in`marital_status` are simplified into `single`, `married`, `divorced or separated` and `widowed`. See Table 5 for details.

Table 5: Re-encoding of the `marital_status` column

Old categories	New categories
never-married	single
married-civ-spouse	married
married-spouse-absent	divorced or separated
divorced	divorced or separated
separated	divorced or separated
married-af-spouse	married

```
adult_df['martial_status'].unique()
```

```
array(['never-married', 'married-civ-spouse', 'divorced',
       'married-spouse-absent', 'separated', 'married-af-spouse',
       'widowed'], dtype=object)
```

```
adult_df.loc[:,'martial_status'] = adult_df['martial_status'].replace({
    'never-married': 'single' ,
    'married-civ-spouse': 'married' ,
   'married-spouse-absent': 'divorced or separated' ,
   'divorced': 'divorced or separated' ,
   'separated': 'divorced or separated' ,
   'married-af-spouse': 'married'

})
```

```
adult_df['martial_status'].unique()
```

```
array(['single', 'married', 'divorced or separated', 'widowed'],
      dtype=object)
```

Re-code the `occupation` column

A new column, `occupation_grouped`, is created. This new column groups the occupations into the categories `white collar`, `blue collar`, `service`, `unknown` and `military`. The exact map ping is illustrated in Table 6.

Occupation	Occupation Grouped
adm-clerical	white collar
exec-managerial	white collar
handlers-cleaners	blue collar

Occupation	Occupation Grouped
prof-specialty	white collar
other-service	service
sales	white collar
craft-repair	blue collar
transport-moving	blue collar
farming-fishing	blue collar
machine-op-inspct	blue collar
tech-support	white collar
protective-serv	service
armed-forces	military
priv-house-serv	service
unknown	unknown

```
adult_df['occupation'].unique()
```

```
array(['adm-clerical', 'exec-managerial', 'handlers-cleaners',
       'prof-specialty', 'other-service', 'sales', 'craft-repair',
       'transport-moving', 'farming-fishing', 'machine-op-inspct',
       'tech-support', nan, 'protective-serv', 'armed-forces',
       'priv-house-serv'], dtype=object)
```

```
adult_df.loc[:,'occupation_grouped'] = adult_df['occupation'].map({
    'adm-clerical': 'white collar',
    'exec-managerial': 'white collar',
    'handlers-cleaners': 'blue collar',
    'prof-specialty': 'white collar',
    'other-service': 'service',
    'sales': 'white collar',
    'craft-repair': 'blue collar',
    'transport-moving': 'blue collar',
    'farming-fishing': 'blue collar',
    'machine-op-inspct': 'blue collar',
    'tech-support': 'white collar',
    'protective-serv': 'service',
    'armed-forces': 'military',
    'priv-house-serv': 'service',
    'unknown': 'unknown'
})
```

```
adult_df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', ' education_num',
       'martial_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income', 'occupation_grouped'],
      dtype='object')
```

```
adult_df['occupation_grouped'].unique()
```

```
array(['white collar', 'blue collar', 'service', nan, 'military'],
      dtype=object)
```

Re-code the `relationship` column

We normalize the `race` column to indicate roles within a family or individual status.

Table 7 shows the re-encoding:

Table 7: Re-encoding of the `race` column

Old relationship	New relationship
`wife`	`female spouse`
`own-child`	`child`
`not-in-family`	`single`
`other-relative`	`extended relative`
`unmarried`	`single`
`husband`	`male spouse`

```
adult_df['relationship'].unique()
```

```
array(['not-in-family', 'husband', 'wife', 'own-child', 'unmarried',
       'other-relative'], dtype=object)
```

```
adult_df.loc[:,'relationship'] = adult_df['relationship'].replace({
    'wife': 'female spouse' ,
    'own-child': 'child' ,
    'not-in-family': 'single' ,
    'other-relative': 'extended relative' ,
    'unmarried': 'single' ,
```

```
    'husband': 'male spouse'

})
```

```
adult_df['relationship'].unique()
```

```
array(['single', 'male spouse', 'female spouse', 'child',
       'extended relative'], dtype=object)
```

Re-code the race column

We standardize the `race` column to have clearer names. Table 8 shows the record values that were re-encoded:

Table 8: Re-encoding of the race column

Old categories	New categories
asian-pac-islander	asian or pacific islander
amer-indian-eskimo	american indian or eskimo

```
adult_df['race'].unique()
```

```
array(['white', 'black', 'asian-pac-islander', 'amer-indian-eskimo',
       'other'], dtype=object)
```

```
adult_df.loc[:,'race'] = adult_df['race'].replace({
    'white': 'white' ,
    'asian-pac-islander': 'asian or pacific islander' ,
    'amer-indian-eskimo': 'american indian or eskimo' ,
    'other-relative': 'extended relative' ,
    'black': 'black',
    'other': 'other'

})
```

```
adult_df['race'].unique()
```

```
array(['white', 'black', 'asian or pacific islander',
       'american indian or eskimo', 'other'], dtype=object)
```

Re-code the native_country column

We create a new colum native_region which maps native_country to geographical regions (e.g., north america, asia, etc.). The mapping is as follows:

Table 9: Mapping from native_country to native_region

native_country	native_region
united-states	north america
canada	north america
puerto-rico	north america
outlying-us(guam-usvi-etc)	north america
mexico	north america
cuba	central america
jamaica	central america
honduras	central america
dominican-republic	central america
el-salvador	central america
guatemala	central america
nicaragua	central america
trinadad&tobago	central america
haiti	central america
columbia	south america
ecuador	south america
peru	south america
south	south america
india	asia
china	asia
iran	asia
japan	asia
philippines	asia
cambodia	asia
thailand	asia
laos	asia
taiwan	asia
vietnam	asia
hong	asia
england	europe
germany	europe
france	europe
italy	europe
poland	europe
portugal	europe

native_country	native_region
yugoslavia	europe
scotland	europe
greece	europe
ireland	europe
hungary	europe
holand-netherlands	europe
other	other

```
adult_df['native_country'].unique()
```

```
array(['united-states', 'cuba', 'jamaica', 'india', nan, 'mexico',
       'south', 'puerto-rico', 'honduras', 'england', 'canada', 'germany',
       'iran', 'philippines', 'italy', 'poland', 'columbia', 'cambodia',
       'thailand', 'ecuador', 'laos', 'taiwan', 'haiti', 'portugal',
       'dominican-republic', 'el-salvador', 'france', 'guatemala',
       'china', 'japan', 'yugoslavia', 'peru',
       'outlying-us(guam-usvi-etc)', 'scotland', 'trinadad&tobago',
       'greece', 'nicaragua', 'vietnam', 'hong', 'ireland', 'hungary',
       'holand-netherlands'], dtype=object)
```

```
adult_df.loc[:,'native_region'] = adult_df['native_country'].map({
    'united-states': 'north america',
    'cambodia': 'asia',
    'england': 'europe',
    'puerto-rico': 'north america',
    'canada': 'north america',
    'germany': 'europe',
    'outlying-us(guam-usvi-etc)': 'north america',
    'india': 'asia',
    'japan': 'asia',
    'greece': 'europe',
    'south': 'south america',
    'china': 'asia',
    'cuba': 'central america',
    'iran': 'asia',
    'honduras': 'central america',
    'philippines': 'asia' ,
    'italy': 'europe' ,
    'poland': 'europe' ,
```

```
    'jamaica': 'central america' ,
    'vietnam': 'asia' ,
    'mexico': 'north america',
    'portugal': 'europe' ,
    'ireland': 'europe' ,
    'france': 'europe' ,
    'dominican-republic': 'central america' ,
    'laos': 'asia' ,
    'ecuador': 'south america',
    'taiwan': 'asia',
    'haiti': 'central america',
    'columbia': 'south america',
    'hungary': 'europe',
    'guatemala': 'central america',
    'nicaragua': 'central america',
    'scotland': 'europe',
    'thailand': 'asia',
    'yugoslavia': 'europe',
    'el-salvador': 'central america',
    'trinadad&tobago': 'central america',
    'peru': 'south america',
    'hong': 'asia',
    'other': 'other',
    'holand-netherlands': 'europe'

})
```

```
adult_df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', ' education_num',
       'martial_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income', 'education_level', 'occupation_grouped', 'native_region'],
      dtype='object')
```

```
adult_df['native_region'].unique()
```

```
array(['north america', 'central america', 'asia', nan, 'south america',
       'europe'], dtype=object)
```

```
adult_df
```

	age	workclass	fnlwgt	education	education_num	martial_status	occupation
0	39	state-gov	77516	bachelors	13	single	adm-clerical
1	50	self-emp-not-inc	83311	bachelors	13	married	exec-manage
2	38	private	215646	hs-grad	9	divorced or separated	handlers-clea
3	53	private	234721	11th	7	married	handlers-clea
4	28	private	338409	bachelors	13	married	prof-specialty
...	...	...	...	...	...	...	...
32556	27	private	257302	assoc-acdm	12	married	tech-support
32557	40	private	154374	hs-grad	9	married	machine-op-i
32558	58	private	151910	hs-grad	9	widowed	adm-clerical
32559	22	private	201490	hs-grad	9	single	adm-clerical
32560	52	self-emp-inc	287927	hs-grad	9	married	exec-manage

```
clean_filename = os.path.join(processed_dir, "6.csv")
adult_df.to_csv(clean_filename, index = False)
print(f"\nCleaned data saved to: {clean_filename}")
```

```
Cleaned data saved to: C:\Users\HP\Downloads\Adult_income\data\processed\6.csv
```

6. Create age groups based on the `age` column

Age is binned into groups such as `<18`, `18-25`, …, `76+` to facilitate easier demographic analysis.

```
adult_df['age'].unique()
```

```
array([39, 50, 38, 53, 28, 37, 49, 52, 31, 42, 30, 23, 32, 40, 34, 25, 43,
       54, 35, 59, 56, 19, 20, 45, 22, 48, 21, 24, 57, 44, 41, 29, 18, 47,
       46, 36, 79, 27, 67, 33, 76, 17, 55, 61, 70, 64, 71, 68, 66, 51, 58,
       26, 60, 90, 75, 65, 77, 62, 63, 80, 72, 74, 69, 73, 81, 78, 88, 82,
       83, 84, 85, 86, 87], dtype=int64)
```

```
adult_df['age'].unique().min()
```

```
17
```

```
adult_df['age'].unique().max()
```

```
90
```

```
bins = [0, 18, 25, 35, 45, 60, 75, 100]
labels = ['<18', '18-25', '26-35', '36-45', '46-60', '61-75', '76+']
adult_df['age_group'] = pd.cut(adult_df['age'], bins = bins, labels = labels, right=True, in
```

```
adult_df['age_group'].unique()
```

```
['36-45', '46-60', '26-35', '18-25', '<18', '76+', '61-75']
Categories (7, object): ['<18' < '18-25' < '26-35' < '36-45' < '46-60' < '61-75' < '76+']
```

7. Drop unnecessary columns

After recoding, some columns such as `education`, `native_country` and `occupation` become redundant. We drop them to avoid multicollinearity and simplify our dataset. We notably retain the `age` column in case there is need to model it as a continuous variable.

```
adult_df.drop(columns = ['education', 'native_country', 'occupation'], inplace=True)
```

```
adult_df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', ' education_num', 'martial_status',
       'relationship', 'race', 'sex', 'capital_gain', 'capital_loss',
       'hours_per_week', 'income', 'education_level', 'occupation_grouped',
       'native_region', 'age_group'],
      dtype='object')
```

```
adult_df.shape
```

```
(32561, 13)
```

We confirm that there are no null values

```
adult_df.isnull().sum()
```

```
age               0
workclass         0
fnlwgt            0
 education_num    0
martial_status    0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
income            0
age_group         0
dtype: int64
```

However, we note that there are new duplicated values given that we merged some categories in the re-encoding process. We inadvertently drop the duplicates.

```python
adult_df.duplicated().sum()
```

```
66
```

```python
adult_df[adult_df.duplicated(keep=False)]
```

	age	workclass	fnlwgt	education_num	martial_status	relationship	race	sex	ca
531	26	Private	108658	9	Never-married	Not-in-family	White	Male	0
594	23	Private	117789	13	Never-married	Own-child	White	Female	0
706	40	Private	229148	8	Married-civ-spouse	Husband	Black	Male	0
1194	19	Private	187161	10	Never-married	Own-child	White	Female	0
1457	22	Private	310152	10	Never-married	Not-in-family	White	Male	0
...	...	...	...	...	...	...	...	...	...
31459	18	Private	170544	7	Never-married	Own-child	White	Male	0
31760	25	Private	182866	9	Never-married	Own-child	White	Male	0
31993	19	Private	251579	10	Never-married	Own-child	White	Male	0
32404	35	Private	379959	9	Divorced	Not-in-family	White	Female	0
32555	22	Private	310152	10	Never-married	Not-in-family	White	Male	0

```python
adult_df = adult_df.drop_duplicates()
```

Save the Clean Dataset

Before saving the clean dataset, we re-inspect it to ensure no new issues have risen up due to re-encoding. We first of all inspect the shape of the dataset. We see that we have *32,537* rows and *16* columns. This means that there is a new column, `age_group`, added to the original dataset.

```
adult_df.duplicated().sum()
```

```
0
```

The final shape of the clean dataset is thus *32,513* rows and *16* columns.

```
adult_df.shape
```

```
(32495, 13)
```

Finally, we save the clean, processed dataset as a CSV file in our `processed` directory for future modelling and analysis.

```
# Save the file in the processed data folder
final_file = os.path.join(processed_dir, 'adult_cleaned.csv')
adult_df.to_csv(final_file, index=False)
```

```
pd.cut([1,2,3], bins=[0, 2, 4], right=True )
```

```
[(0, 2], (0, 2], (2, 4]]
Categories (2, interval[int64, right]): [(0, 2] < (2, 4]]
```

```
pd.cut([1,2,3], bins=[0, 2, 4], right=False )
```

```
[[0, 2), [2, 4), [2, 4)]
Categories (2, interval[int64, left]): [[0, 2) < [2, 4)]
```

```
pd.cut([0,1,2], bins=[0, 1, 2], right=True, include_lowest=False )
```

```
[NaN, (0.0, 1.0], (1.0, 2.0]]
Categories (2, interval[int64, right]): [(0, 1] < (1, 2]]
```