UCI Adult Income Dataset - Data Cleaning and Prepocessing

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)

# Define paths to the data folder
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 paths to results folder
results_dir = os.path.join(project_root_dir, 'results')

# Define paths to docs folder
docs_dir = os.path.join(project_root_dir, 'docs')

# Create directories if they do not exist
os.makedirs(raw_dir, exist_ok = True)
os.makedirs(processed_dir, exist_ok = True)
os.makedirs(results_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
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
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", "marital_sta
```

We inspect again to see whether they are properly assigned

```
adult_df.head(10)
```

	age	workclass	fnlwgt	education	education_num	marital_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-cleaners
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial
8	31	Private	45781	Masters	14	Never-married	Prof-specialty
9	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial

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
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.age.to_list())
np.unique(adult_df.workclass.to_list())
np.unique(adult_df.fnlwgt.to_list())
np.unique(adult_df.education.to_list)
np.unique(adult_df.education_num.to_list())
np.unique(adult_df.marital_status.to_list())
np.unique(adult_df.occupation.to_list())
np.unique(adult_df.relationship.to_list())
np.unique(adult_df.race.to_list())
np.unique(adult_df.sex.to_list())
np.unique(adult_df.capital_gain.to_list())
np.unique(adult_df.capital_loss.to_list())
np.unique(adult_df.hours_per_week.to_list())
np.unique(adult_df.native_country.to_list())
np.unique(adult_df.income.to_list())
```

```
array(['<=50K', '>50K'], dtype='<U5')
```

3. Deal with missing values

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

```
age                  0
workclass         1836
fnlwgt               0
education            0
education_num        0
marital_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.

```python
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
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
```

4. 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	marital_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

	age	workclass	fnlwgt	education	education_num	marital_status	occupation
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
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

	age	workclass	fnlwgt	education	education_num	marital_status	occupation
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
marital_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
```

```
adult_df.columns
```

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

```
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	marital_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
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

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

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

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

```python
adult_df.loc[:,'workclass'] = adult_df['workclass'].replace({
    'state-gov': 'government',
    'local-gov': 'government',
    'federal-gov': 'government',
    'self-emp-inc': 'self-employed',
    'self-emp-not-inc': 'self-employed',
    'never-worked': 'unemployed',
    'without-pay': 'voluntary'
})
```

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

```
array(['government', 'self-employed', 'private', 'unknown', 'voluntary',
       'unemployed'], dtype=object)
```

Re-code the `education` column

We create a new colum `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

```python
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',
```

```python
    'preschool': 'preschool'
})
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_26244\1219575138.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  adult_df.loc[:,'education_level'] = adult_df['education'].map({
```

```python
adult_df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_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`

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

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

```
adult_df.loc[:,'marital_status'] = adult_df['marital_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',
 'widowed': 'widowed',
})
```

```
adult_df['marital_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`
`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`

```python
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', 'unknown', 'protective-serv', 'armed-forces',
       'priv-house-serv'], dtype=object)
```

```python
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',
    'unknown': 'unknown',
    'protective-serv': 'service',
    'armed-forces': 'military',
    'priv-house-serv': 'service'
})
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_26244\4034686476.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  adult_df.loc[:,'occupation_grouped'] = adult_df['occupation'].map({
```

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

```
array(['white collar', 'blue collar', 'service', 'unknown', '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({
    'not-in-family': 'single',
    'husband': 'male spouse',
    'wife': 'female spouse',
    'own-child': 'child',
    'unmarried': 'single',
    'other-relative': 'extended relative'
})
```

```
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 more clear 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',
    'black': 'black',
    'asian-pac-islander': 'asian or pacific islander',
    'amer-indian-eskimo': 'american indian or eskimo',
    '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

native_country	native_region
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
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', 'other', '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',
    'cuba': 'central america',
    'jamaica': 'central america',
    'india': 'asia',
    'mexico': 'north america',
    'south': 'south america',
    'puerto-rico': 'north america',
    'honduras': 'central america',
    'england': 'europe',
    'canada': 'north america',
    'germany': 'europe',
    'iran': 'asia',
    'philippines': 'asia',
    'italy': 'europe',
    'poland': 'europe',
    'columbia': 'south america',
    'cambodia': 'asia',
    'thailand': 'asia',
    'ecuador': 'south america',
    'laos': 'asia',
    'taiwan': 'asia',
    'haiti': 'central america',
    'portugal': 'europe',
    'dominican-republic': 'central america',
    'el-salvador': 'central america',
    'france': 'europe',
    'guatemala': 'central america',
    'china': 'asia',
    'japan': 'asia',
    'yugoslavia': 'europe',
    'peru': 'south america',
    'outlying-us(guam-usvi-etc)': 'north america',
    'scotland': 'europe',
    'trinadad&tobago': 'central america',
    'greece': 'europe',
```

```
    'nicaragua': 'central america',
    'vietnam': 'asia',
    'hong': 'asia',
    'ireland': 'europe',
    'hungary': 'europe',
    'holand-netherlands': 'europe',
    'other': 'other'
})
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_26244\1986502432.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  adult_df.loc[:,'native_region'] = adult_df['native_country'].map({
```

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)
```

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

```
C:\Users\HP\AppData\Local\Temp\ipykernel_26244\1111254358.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  adult_df.loc[:,'age_group'] = pd.cut(adult_df['age'], bins = bins, labels = labels, right=
```

```
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)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_26244\95770546.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  adult_df.drop(columns=['education', 'native_country', 'occupation'], inplace=True)
```

```
adult_df.columns
```

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

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.shape
```

```
(32537, 16)
```

We confirm that there are no null values.

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

```
age                   0
workclass             0
fnlwgt                0
education_num         0
marital_status        0
relationship          0
race                  0
sex                   0
capital_gain          0
capital_loss          0
hours_per_week        0
income                0
education_level       0
occupation_grouped    0
native_region         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.

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

```
24
```

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

	age	workclass	fnlwgt	education_num	marital_status	relationship	race	sex
531	26	private	108658	9	single	single	white	male
594	23	private	117789	13	single	child	white	female
2896	46	private	271828	9	married	male spouse	white	male
3261	26	private	108658	9	single	single	white	male
3586	28	private	50814	9	single	single	white	female
3692	46	private	271828	9	married	male spouse	white	male
3960	43	private	174575	10	divorced or separated	single	white	male

	age	workclass	fnlwgt	education_num	marital_status	relationship	race	sex
4511	24	private	140001	13	single	single	white	male
5110	21	private	118693	10	single	child	white	male
5805	20	private	107658	10	single	single	white	female
6403	26	private	174921	13	single	single	white	female
6763	44	private	104196	14	married	male spouse	white	male
7713	28	private	50814	9	single	single	white	female
8342	33	private	198211	9	married	male spouse	white	male
8794	33	private	198211	9	married	male spouse	white	male
9680	29	private	115677	13	single	single	white	male
9980	29	private	115677	13	single	single	white	male
10302	25	private	182866	9	single	child	white	male
11331	23	private	117789	13	single	child	white	female
12180	26	private	174921	13	single	single	white	female
12199	27	private	183523	13	single	single	white	male
12233	22	government	262819	10	single	single	white	female
12596	28	private	205337	9	married	male spouse	white	male
13396	31	private	209538	6	married	male spouse	white	male
17202	25	private	178478	13	single	child	white	female
17630	33	private	136331	9	married	male spouse	white	male
18147	58	private	205410	9	married	male spouse	white	male
19098	42	private	177989	9	married	male spouse	white	male
20373	28	private	205337	9	married	male spouse	white	male
21264	38	private	108907	9	divorced or separated	single	white	male
21488	20	private	107658	10	single	single	white	female
22840	56	private	220187	10	married	male spouse	white	male
23520	22	government	262819	10	single	single	white	female
23674	21	private	118693	10	single	child	white	male
23785	24	private	140001	13	single	single	white	male
23851	25	private	367306	10	single	child	white	female
24400	44	private	104196	14	married	male spouse	white	male
24942	25	private	178478	13	single	child	white	female
25467	31	private	209538	6	married	male spouse	white	male
26004	56	private	220187	10	married	male spouse	white	male
26044	42	private	177989	9	married	male spouse	white	male
26441	58	private	205410	9	married	male spouse	white	male
26572	33	private	136331	9	married	male spouse	white	male
27921	43	private	174575	10	divorced or separated	single	white	male
28841	38	private	108907	9	divorced or separated	single	white	male
29225	27	private	183523	13	single	single	white	male
30132	25	private	367306	10	single	child	white	female

31760	25	private	182866	9	single	child	white	male

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

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

```
0
```

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

```
adult_df.shape
```

```
(32513, 16)
```

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)
```