Importing Data

In [1]:
```
import os
print("Current_Working_Directory:", os.getcwd())
```

```
Current_Working_Directory: C:\Users\viraj\Code_a\Machine_learning\Automate Data Clean
ing with Python
```

In [2]:
```
directory_path = "C:/Users/viraj/Code_a/Machine_learning/Automate Data Cleaning with P
contents = os.listdir(directory_path)
print("Contents")
for i in contents:
    print(i)
```

```
Contents
.ipynb_checkpoints
Automate.ipynb
car.c45-names
car.data
car.names
data.csv
data.xlsx
data1.csv
```

In [3]:
```
from ucimlrepo import fetch_ucirepo, list_available_datasets
# check which datasets can be imported
list_available_datasets()
```

```
-------------------------------------
The following datasets are available:
-------------------------------------
Dataset Name
ID
------------
--
Abalone
1
Adult
2
Annealing
3
Audiology (Standardized)
8
Auto MPG
9
Automobile
10
Balance Scale
12
Balloons
13
Breast Cancer
14
Breast Cancer Wisconsin (Original)
15
Breast Cancer Wisconsin (Prognostic)
16
Breast Cancer Wisconsin (Diagnostic)
17
Pittsburgh Bridges
18
Car Evaluation
19
Census Income
20
Chess (King-Rook vs. King-Pawn)
22
Chess (King-Rook vs. King)
23
Connect-4
26
Credit Approval
27
Japanese Credit Screening
28
Computer Hardware
29
Contraceptive Method Choice
30
Covertype
31
Cylinder Bands
32
Dermatology
33
Echocardiogram
38
Ecoli
```

89
Soybean (Large)
90
Soybean (Small)
91
Challenger USA Space Shuttle O-Ring
92
Spambase
94
SPECT Heart
95
SPECTF Heart
96
Tic-Tac-Toe Endgame
101
Congressional Voting Records
105
Waveform Database Generator (Version 1)
107
Wine
109
Yeast
110
Zoo
111
US Census Data (1990)
116
Census-Income (KDD)
117
El Nino
122
Statlog (Australian Credit Approval)
143
Statlog (German Credit Data)
144
Statlog (Heart)
145
Statlog (Landsat Satellite)
146
Statlog (Image Segmentation)
147
Statlog (Shuttle)
148
Statlog (Vehicle Silhouettes)
149
Connectionist Bench (Sonar, Mines vs. Rocks)
151
Cloud
155
Poker Hand
158
MAGIC Gamma Telescope
159
Mammographic Mass
161
Forest Fires
162
Concrete Compressive Strength
165
Ozone Level Detection

172
Parkinsons
174
Blood Transfusion Service Center
176
Communities and Crime
183
Acute Inflammations
184
Wine Quality
186
Parkinsons Telemonitoring
189
Cardiotocography
193
Steel Plates Faults
198
Communities and Crime Unnormalized
211
Vertebral Column
212
Bank Marketing
222
ILPD (Indian Liver Patient Dataset)
225
Skin Segmentation
229
Individual Household Electric Power Consumption
235
Energy Efficiency
242
Fertility
244
ISTANBUL STOCK EXCHANGE
247
User Knowledge Modeling
257
EEG Eye State
264
Banknote Authentication
267
Gas Sensor Array Drift at Different Concentrations
270
Bike Sharing
275
Thoracic Surgery Data
277
Airfoil Self-Noise
291
Wholesale customers
292
Combined Cycle Power Plant
294
Diabetes 130-US Hospitals for Years 1999-2008
296
Tennis Major Tournament Match Statistics
300
Dow Jones Index
312
Student Performance

320
Phishing Websites
327
Diabetic Retinopathy Debrecen
329
Online News Popularity
332
Chronic Kidney Disease
336
Mice Protein Expression
342
Default of Credit Card Clients
350
Online Retail
352
Occupancy Detection
357
Air Quality
360
Polish Companies Bankruptcy
365
Dota2 Games Results
367
Facebook Metrics
368
HTRU2
372
Drug Consumption (Quantified)
373
Appliances Energy Prediction
374
Website Phishing
379
YouTube Spam Collection
380
Beijing PM2.5
381
Cervical Cancer (Risk Factors)
383
Stock Portfolio Performance
390
Sales Transactions Weekly
396
Daily Demand Forecasting Orders
409
Autistic Spectrum Disorder Screening Data for Children
419
Autism Screening Adult
426
Absenteeism at work
445
Breast Cancer Coimbra
451
Drug Reviews (Druglib.com)
461
Drug Reviews (Drugs.com)
462
Superconductivty Data
464
Student Academics Performance

467
Online Shoppers Purchasing Intention Dataset
468
Electrical Grid Stability Simulated Data
471
Real Estate Valuation
477
Travel Reviews
484
Travel Review Ratings
485
Facebook Live Sellers in Thailand
488
Metro Interstate Traffic Volume
492
Hepatitis C Virus (HCV) for Egyptian patients
503
Heart Failure Clinical Records
519
Early Stage Diabetes Risk Prediction
529
Pedestrians in Traffic
536
Cervical Cancer Behavior Risk
537
Estimation of Obesity Levels Based On Eating Habits and Physical Condition
544
Rice (Cammeo and Osmancik)
545
Algerian Forest Fires
547
Gas Turbine CO and NOx Emission Data Set
551
Apartment for Rent Classified
555
Seoul Bike Sharing Demand
560
Iranian Churn
563
Bone marrow transplant: children
565
COVID-19 Surveillance
567
HCV data
571
Taiwanese Bankruptcy Prediction
572
Myocardial infarction complications
579
Student Performance on an Entrance Examination
582
Gender by Name
591
Productivity Prediction of Garment Employees
597
AI4I 2020 Predictive Maintenance Dataset
601
Dry Bean
602
In-Vehicle Coupon Recommendation

603
Predict Students' Dropout and Academic Success
697
Auction Verification
713
NATICUSdroid (Android Permissions)
722
Toxicity
728
DARWIN
732
Accelerometer Gyro Mobile Phone
755
Glioma Grading Clinical and Mutation Features
759
Multivariate Gait Data
760
Land Mines
763
Single Elder Home Monitoring: Gas and Position
799
Sepsis Survival Minimal Clinical Records
827
Secondary Mushroom
848
Power Consumption of Tetouan City
849
Raisin
850
Steel Industry Energy Consumption
851
Higher Education Students Performance Evaluation
856
Risk Factor Prediction of Chronic Kidney Disease
857
Maternal Health Risk
863
Room Occupancy Estimation
864
Cirrhosis Patient Survival Prediction
878
SUPPORT2
880
National Health and Nutrition Health Survey 2013-2014 (NHANES) Age Prediction Subset
887
AIDS Clinical Trials Group Study 175
890
CDC Diabetes Health Indicators
891
Recipe Reviews and User Feedback
911
Forty Soybean Cultivars from Subsequent Harvests
913
Differentiated Thyroid Cancer Recurrence
915
Infrared Thermography Temperature
925
National Poll on Healthy Aging (NPHA)
936
Regensburg Pediatric Appendicitis

```
938
RT-IoT2022
942
PhiUSIIL Phishing URL (Website)
967
```

In [4]:
```python
adult = fetch_ucirepo(id=2)
```

In [5]:
```python
import pandas as pd
import numpy as np
```

In [6]:
```python
adult
```

```
Out[6]: {'data': {'ids': None,
  'features':        age         workclass  fnlwgt  education  education-num  \
  0       39         State-gov   77516  Bachelors             13
  1       50  Self-emp-not-inc   83311  Bachelors             13
  2       38           Private  215646    HS-grad              9
  3       53           Private  234721       11th              7
  4       28           Private  338409  Bachelors             13
  ...    ...               ...     ...        ...            ...
  48837   39           Private  215419  Bachelors             13
  48838   64               NaN  321403    HS-grad              9
  48839   38           Private  374983  Bachelors             13
  48840   44           Private   83891  Bachelors             13
  48841   35      Self-emp-inc  182148  Bachelors             13

             marital-status         occupation    relationship  \
  0           Never-married       Adm-clerical   Not-in-family
  1      Married-civ-spouse    Exec-managerial         Husband
  2                Divorced  Handlers-cleaners   Not-in-family
  3      Married-civ-spouse  Handlers-cleaners         Husband
  4      Married-civ-spouse     Prof-specialty            Wife
  ...                   ...                ...             ...
  48837            Divorced     Prof-specialty   Not-in-family
  48838             Widowed                NaN  Other-relative
  48839  Married-civ-spouse     Prof-specialty         Husband
  48840            Divorced       Adm-clerical       Own-child
  48841  Married-civ-spouse    Exec-managerial         Husband

                       race     sex  capital-gain  capital-loss  hours-per-week  \
  0                   White    Male          2174             0              40
  1                   White    Male             0             0              13
  2                   White    Male             0             0              40
  3                   Black    Male             0             0              40
  4                   Black  Female             0             0              40
  ...                   ...     ...           ...           ...             ...
  48837               White  Female             0             0              36
  48838               Black    Male             0             0              40
  48839               White    Male             0             0              50
  48840  Asian-Pac-Islander    Male          5455             0              40
  48841               White    Male             0             0              60

        native-country
  0      United-States
  1      United-States
  2      United-States
  3      United-States
  4               Cuba
  ...              ...
  48837  United-States
  48838  United-States
  48839  United-States
  48840  United-States
  48841  United-States

  [48842 rows x 14 columns],
  'targets':        income
  0       <=50K
  1       <=50K
  2       <=50K
  3       <=50K
  4       <=50K
```

```
...        ...
48837  <=50K.
48838  <=50K.
48839  <=50K.
48840  <=50K.
48841   >50K.

[48842 rows x 1 columns],
'original':        age         workclass  fnlwgt  education  education-num  \
0       39         State-gov   77516  Bachelors             13
1       50  Self-emp-not-inc   83311  Bachelors             13
2       38           Private  215646    HS-grad              9
3       53           Private  234721       11th              7
4       28           Private  338409  Bachelors             13
...    ...               ...     ...        ...            ...
48837   39           Private  215419  Bachelors             13
48838   64               NaN  321403    HS-grad              9
48839   38           Private  374983  Bachelors             13
48840   44           Private   83891  Bachelors             13
48841   35      Self-emp-inc  182148  Bachelors             13

           marital-status         occupation    relationship  \
0           Never-married       Adm-clerical   Not-in-family
1      Married-civ-spouse    Exec-managerial         Husband
2                Divorced  Handlers-cleaners   Not-in-family
3      Married-civ-spouse  Handlers-cleaners         Husband
4      Married-civ-spouse     Prof-specialty            Wife
...                   ...                ...             ...
48837            Divorced     Prof-specialty   Not-in-family
48838             Widowed                NaN  Other-relative
48839  Married-civ-spouse     Prof-specialty         Husband
48840            Divorced       Adm-clerical       Own-child
48841  Married-civ-spouse    Exec-managerial         Husband

                     race     sex  capital-gain  capital-loss  hours-per-week  \
0                   White    Male          2174             0              40
1                   White    Male             0             0              13
2                   White    Male             0             0              40
3                   Black    Male             0             0              40
4                   Black  Female             0             0              40
...                   ...     ...           ...           ...             ...
48837               White  Female             0             0              36
48838               Black    Male             0             0              40
48839               White    Male             0             0              50
48840  Asian-Pac-Islander    Male          5455             0              40
48841               White    Male             0             0              60

      native-country  income
0      United-States   <=50K
1      United-States   <=50K
2      United-States   <=50K
3      United-States   <=50K
4               Cuba   <=50K
...              ...     ...
48837  United-States  <=50K.
48838  United-States  <=50K.
48839  United-States  <=50K.
48840  United-States  <=50K.
48841  United-States   >50K.
```

```
  [48842 rows x 15 columns],
  'headers': 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')},
 'metadata': {'uci_id': 2,
  'name': 'Adult',
  'repository_url': 'https://archive.ics.uci.edu/dataset/2/adult',
  'data_url': 'https://archive.ics.uci.edu/static/public/2/data.csv',
  'abstract': 'Predict whether income exceeds $50K/yr based on census data. Also know
n as "Census Income" dataset. ',
  'area': 'Social Science',
  'tasks': ['Classification'],
  'characteristics': ['Multivariate'],
  'num_instances': 48842,
  'num_features': 14,
  'feature_types': ['Categorical', 'Integer'],
  'demographics': ['Age', 'Income', 'Education Level', 'Other', 'Race', 'Sex'],
  'target_col': ['income'],
  'index_col': None,
  'has_missing_values': 'yes',
  'missing_values_symbol': 'NaN',
  'year_of_dataset_creation': 1996,
  'last_updated': 'Mon Aug 07 2023',
  'dataset_doi': '10.24432/C5XW20',
  'creators': ['Barry Becker', 'Ronny Kohavi'],
  'intro_paper': None,
  'additional_info': {'summary': 'Extraction was done by Barry Becker from the 1994 C
ensus database.  A set of reasonably clean records was extracted using the following
conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))\r\n\r\nPrediction tas
k is to determine whether a person makes over 50K a year.\r\n',
   'purpose': None,
   'funded_by': None,
   'instances_represent': None,
   'recommended_data_splits': None,
   'sensitive_data': None,
   'preprocessing_description': None,
   'variable_info': 'Listing of attributes:\r\n\r\n>50K, <=50K.\r\n\r\nage: continuou
s.\r\nworkclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, Sta
te-gov, Without-pay, Never-worked.\r\nfnlwgt: continuous.\r\neducation: Bachelors, So
me-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Ma
sters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.\r\neducation-num: continuous.\r
\nmarital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Ma
rried-spouse-absent, Married-AF-spouse.\r\noccupation: Tech-support, Craft-repair, Ot
her-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-in
spct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-se
rv, Armed-Forces.\r\nrelationship: Wife, Own-child, Husband, Not-in-family, Other-rel
ative, Unmarried.\r\nrace: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Blac
k.\r\nsex: Female, Male.\r\ncapital-gain: continuous.\r\ncapital-loss: continuous.\r
\nhours-per-week: continuous.\r\nnative-country: United-States, Cambodia, England, Pu
erto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South,
China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Po
rtugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia,
Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&
Tobago, Peru, Hong, Holand-Netherlands.',
   'citation': None}},
 'variables':               name     role         type      demographic  \
 0              age  Feature      Integer              Age   
 1        workclass  Feature  Categorical           Income   
```

```
2           fnlwgt  Feature      Integer                None
3        education  Feature  Categorical  Education Level
4    education-num  Feature      Integer  Education Level
5   marital-status  Feature  Categorical            Other
6       occupation  Feature  Categorical            Other
7     relationship  Feature  Categorical            Other
8             race  Feature  Categorical             Race
9              sex  Feature       Binary              Sex
10    capital-gain  Feature      Integer             None
11    capital-loss  Feature      Integer             None
12  hours-per-week  Feature      Integer             None
13  native-country  Feature  Categorical            Other
14          income   Target       Binary           Income

                                          description units missing_values
0                                                 N/A  None             no
1   Private, Self-emp-not-inc, Self-emp-inc, Feder...  None            yes
2                                                None  None             no
3    Bachelors, Some-college, 11th, HS-grad, Prof-...  None             no
4                                                None  None             no
5   Married-civ-spouse, Divorced, Never-married, S...  None             no
6   Tech-support, Craft-repair, Other-service, Sal...  None            yes
7   Wife, Own-child, Husband, Not-in-family, Other...  None             no
8   White, Asian-Pac-Islander, Amer-Indian-Eskimo,...  None             no
9                                       Female, Male.  None             no
10                                               None  None             no
11                                               None  None             no
12                                               None  None             no
13  United-States, Cambodia, England, Puerto-Rico,...  None            yes
14                                          >50K, <=50K.  None             no  }
```

In [7]:

```python
# Extracting the relevant data
features_data = adult['data']['features']
#[['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status','occup

targets_data = adult['data']['targets']

# Merging features and targets data
data = pd.concat([features_data, targets_data], axis=1)
```

In [8]:

```python
data.to_csv('data1.csv', index=False) #for an example lets first store the data in a c
```

Data Format

When automating a data pipeline, it's crucial to efficiently handle various data formats such as JSON, CSV, or XML. Identifying the format of incoming data becomes a crucial step in the process. One way to streamline this is by leveraging Python's `os` library to automatically detect the format using file extensions.

By doing this, we can create a function that scans the file path, recognizes the extension, and then directs the data to the appropriate parser for further processing. This automation eliminates the need for manual intervention in determining the format and selecting the corresponding parser. As a result, the data pipeline becomes more robust and efficient, allowing for smoother data cleaning and transformation tasks.

```python
In [9]: def read_data(file_path):
    _,file_ext = os.path.splitext(file_path)
    if file_ext == '.csv':
        return pd.read_csv(file_path)
    elif file_ext == '.json':
        return pd.read_json(file_path)
    elif file_ext in ['.xls', '.xlsx']:
        return pd.read_excel(file_path)
    elif file_ext == '.data':
        # Assuming comma-separated values with no header
        return pd.read_csv(file_path, header=None)
    else:
        raise ValueError("Unknown file format")
```

```python
In [10]: file_path=("C:/Users/viraj/Code_a/Machine_learning/Automate Data Cleaning with Python/
```

```python
In [11]: data = read_data(file_path)
```

Handle Duplicates

When we automate the handling of duplicates in a data pipeline, we're essentially ensuring that our data is clean and consistent throughout its journey. Imagine we're organizing a stack of papers - if we have duplicates, it's like having multiple copies of the same document cluttering your workspace. By automating the removal of these duplicates, we're effectively tidying up our data, making sure each piece of information is unique and valuable. This not only saves our time and effort but also helps in avoiding mistakes that could arise from analyzing duplicate data.

So, automating this process streamlines our workflow, allowing us to focus on extracting meaningful insights from our data without worrying about redundant information. It's like having a helpful assistant who takes care of the tedious task of decluttering our data, leaving us with a clean and organized dataset to work with.

```python
In [12]: def handle_duplicates(df):
    # Detect duplicates
    duplicates = df.duplicated(keep='first')

    # If duplicates exist, remove them and return cleaned DataFrame
    if duplicates.any():
        print("Duplicate rows:")
        print(df[duplicates])
        df_cleaned = df.drop_duplicates(keep='first').copy()  # Keep only the first oc
        num_duplicates = duplicates.sum()
        print(f"\nRemoved {num_duplicates} duplicate rows.")
        return df_cleaned
    else:
        print("No duplicates found.")
        return df
```

```python
In [13]: cleaned_data = handle_duplicates(data)
```

```
Duplicate rows:
       age         workclass  fnlwgt     education  education-num  \
4881    25           Private  308144     Bachelors             13
5104    90           Private   52386  Some-college             10
9171    21           Private  250051  Some-college             10
11631   20           Private  107658  Some-college             10
13084   25           Private  195994       1st-4th              2
15059   21           Private  243368     Preschool              1
17040   46           Private  173243       HS-grad              9
18555   30           Private  144593       HS-grad              9
18698   19           Private   97261       HS-grad              9
21318   19           Private  138153  Some-college             10
21490   19           Private  146679  Some-college             10
21875   49           Private   31267       7th-8th              4
22300   25           Private  195994       1st-4th              2
22367   44           Private  367749     Bachelors             13
22494   49  Self-emp-not-inc   43479  Some-college             10
25872   23           Private  240137       5th-6th              3
26313   28           Private  274679       Masters             14
28230   27           Private  255582       HS-grad              9
28522   42           Private  204235  Some-college             10
28846   39           Private   30916       HS-grad              9
29157   38           Private  207202       HS-grad              9
30845   46           Private  133616  Some-college             10
31993   19           Private  251579  Some-college             10
32404   35           Private  379959       HS-grad              9
33425   24           Private  194630     Bachelors             13
43750   37           Private   52870     Bachelors             13
43773   29           Private   36440     Bachelors             13
46409   30           Private  180317     Assoc-voc             11
48521   18      Self-emp-inc  378036          12th              8

           marital-status         occupation   relationship  \
4881        Never-married       Craft-repair  Not-in-family
5104        Never-married      Other-service  Not-in-family
9171        Never-married     Prof-specialty      Own-child
11631       Never-married       Tech-support  Not-in-family
13084       Never-married    Priv-house-serv  Not-in-family
15059       Never-married    Farming-fishing  Not-in-family
17040  Married-civ-spouse       Craft-repair        Husband
18555       Never-married      Other-service  Not-in-family
18698       Never-married    Farming-fishing  Not-in-family
21318       Never-married       Adm-clerical      Own-child
21490       Never-married    Exec-managerial      Own-child
21875  Married-civ-spouse       Craft-repair        Husband
22300       Never-married    Priv-house-serv  Not-in-family
22367       Never-married     Prof-specialty  Not-in-family
22494  Married-civ-spouse       Craft-repair        Husband
25872       Never-married  Handlers-cleaners  Not-in-family
26313       Never-married     Prof-specialty  Not-in-family
28230       Never-married  Machine-op-inspct  Not-in-family
28522  Married-civ-spouse     Prof-specialty        Husband
28846  Married-civ-spouse       Craft-repair        Husband
29157  Married-civ-spouse  Machine-op-inspct        Husband
30845            Divorced       Adm-clerical      Unmarried
31993       Never-married      Other-service      Own-child
32404            Divorced      Other-service  Not-in-family
33425       Never-married     Prof-specialty  Not-in-family
43750  Married-civ-spouse    Exec-managerial        Husband
43773       Never-married       Adm-clerical  Not-in-family
```

```
46409           Divorced  Machine-op-inspct  Not-in-family
48521      Never-married    Farming-fishing      Own-child

                     race     sex  capital-gain  capital-loss  hours-per-week  \
4881                White    Male             0             0              40
5104   Asian-Pac-Islander    Male             0             0              35
9171                White  Female             0             0              10
11631               White  Female             0             0              10
13084               White  Female             0             0              40
15059               White    Male             0             0              50
17040               White    Male             0             0              40
18555               Black    Male             0             0              40
18698               White    Male             0             0              40
21318               White  Female             0             0              10
21490               Black    Male             0             0              30
21875               White    Male             0             0              40
22300               White  Female             0             0              40
22367               White  Female             0             0              45
22494               White    Male             0             0              40
25872               White    Male             0             0              55
26313               White    Male             0             0              50
28230               White  Female             0             0              40
28522               White    Male             0             0              40
28846               White    Male             0             0              40
29157               White    Male             0             0              48
30845               White  Female             0             0              40
31993               White    Male             0             0              14
32404               White  Female             0             0              40
33425               White    Male             0             0              35
43750               White    Male             0             0              40
43773               White  Female             0             0              40
46409               White    Male             0             0              40
48521               White    Male             0             0              10

      native-country  income
4881          Mexico   <=50K
5104   United-States   <=50K
9171   United-States   <=50K
11631  United-States   <=50K
13084      Guatemala   <=50K
15059         Mexico   <=50K
17040  United-States   <=50K
18555              ?   <=50K
18698  United-States   <=50K
21318  United-States   <=50K
21490  United-States   <=50K
21875  United-States   <=50K
22300      Guatemala   <=50K
22367         Mexico   <=50K
22494  United-States   <=50K
25872         Mexico   <=50K
26313  United-States   <=50K
28230  United-States   <=50K
28522  United-States    >50K
28846  United-States   <=50K
29157  United-States    >50K
30845  United-States   <=50K
31993  United-States   <=50K
32404  United-States   <=50K
33425  United-States  <=50K.
```

```
43750  United-States  <=50K.
43773  United-States  <=50K.
46409  United-States  <=50K.
48521  United-States  <=50K.

Removed 29 duplicate rows.
```

In [14]:

```
cleaned_data
```

Out[14]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	F
...	...	...	...	...	...	...	...	...	...	
48837	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	F
48838	64	NaN	321403	HS-grad	9	Widowed	NaN	Other-relative	Black	
48839	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	
48840	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	

48813 rows × 15 columns

Missing Values

When dealing with missing data in our pipeline, we have a few options to consider. We can either remove observations containing missing values or fill in these gaps using techniques like forward fill, backward fill, or substituting with the mean or median of the column. Pandas provides convenient methods like `.fillna()` and `.dropna()` to handle these scenarios effectively.

The decision on how to handle missing values depends on a couple of factors. Firstly, it depends on the type of values that are missing and secondly, on the proportion of missing values relative to the total number of records we have. Dealing with missing values is a critical task in data processing and can significantly impact the integrity of our analyses.

In our pipeline, we follow a structured approach. First, we check the total number of rows with null values. If this accounts for only 5% or less of the total records, we opt to simply remove these affected rows. However, if more rows have missing values, we assess each column individually. For columns with missing values, we either impute the median of the value or generate a warning for further investigation.

This process involves a hybrid human validation approach. We recognize that dealing with missing values requires careful consideration and cannot be overlooked in ensuring the accuracy and reliability of our data analysis.

In [15]:
```
cleaned_data.isnull().sum()
```

Out[15]:
```
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
```

In [16]:
```
cleaned_data.describe()
```

Out[16]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	48813.000000	4.881300e+04	48813.000000	48813.000000	48813.000000	48813.000000
mean	38.647348	1.896679e+05	10.078688	1079.708705	87.554299	40.425051
std	13.709005	1.056062e+05	2.570257	7454.185982	403.118605	12.390954
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175550e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781400e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376200e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

In [17]:
```
def count_unique_elements(df):
    unique_counts = {}
    for column in df.columns:
        unique_counts[column] = df[column].value_counts()
    return unique_counts
```

In [18]:
```
unique_counts = count_unique_elements(cleaned_data)
# Print the count of each unique element for each column
for column, counts in unique_counts.items():
    print(f"Column '{column}':")
    print(counts)
    print()
```

```
Column 'age':
age
36    1348
35    1336
33    1335
23    1328
31    1325
      ...
88       6
85       5
87       3
89       2
86       1
Name: count, Length: 74, dtype: int64

Column 'workclass':
workclass
Private             33879
Self-emp-not-inc     3861
Local-gov            3136
State-gov            1981
?                    1836
Self-emp-inc         1694
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: count, dtype: int64

Column 'fnlwgt':
fnlwgt
203488    21
190290    19
120277    19
126569    18
125892    18
          ..
119913     1
78170      1
279721     1
390867     1
350977     1
Name: count, Length: 28523, dtype: int64

Column 'education':
education
HS-grad         15777
Some-college    10869
Bachelors        8020
Masters          2656
Assoc-voc        2060
11th             1812
Assoc-acdm       1601
10th             1389
7th-8th           954
Prof-school       834
9th               756
12th              656
Doctorate         594
5th-6th           508
1st-4th           245
```

```
Preschool          82
Name: count, dtype: int64

Column 'education-num':
education-num
9     15777
10    10869
13     8020
14     2656
11     2060
7      1812
12     1601
6      1389
4       954
15      834
5       756
8       656
16      594
3       508
2       245
1        82
Name: count, dtype: int64

Column 'marital-status':
marital-status
Married-civ-spouse       22372
Never-married            16098
Divorced                  6630
Separated                 1530
Widowed                   1518
Married-spouse-absent      628
Married-AF-spouse           37
Name: count, dtype: int64

Column 'occupation':
occupation
Prof-specialty       6167
Craft-repair         6107
Exec-managerial      6084
Adm-clerical         5608
Sales                5504
Other-service        4919
Machine-op-inspct    3019
Transport-moving     2355
Handlers-cleaners    2071
?                    1843
Farming-fishing      1487
Tech-support         1445
Protective-serv       983
Priv-house-serv       240
Armed-Forces           15
Name: count, dtype: int64

Column 'relationship':
relationship
Husband           19709
Not-in-family     12567
Own-child          7576
Unmarried          5124
Wife               2331
```

```
Other-relative     1506
Name: count, dtype: int64

Column 'race':
race
White                 41736
Black                  4683
Asian-Pac-Islander     1518
Amer-Indian-Eskimo      470
Other                   406
Name: count, dtype: int64

Column 'sex':
sex
Male      32631
Female    16182
Name: count, dtype: int64

Column 'capital-gain':
capital-gain
0        44778
15024      513
7688       410
7298       364
99999      244
         ...
22040        1
2387         1
1639         1
1111         1
6612         1
Name: count, Length: 123, dtype: int64

Column 'capital-loss':
capital-loss
0       46531
1902      304
1977      253
1887      233
2415       72
        ...
1539        1
1870        1
1911        1
2465        1
1421        1
Name: count, Length: 99, dtype: int64

Column 'hours-per-week':
hours-per-week
40    22787
50     4244
45     2716
60     2177
35     1935
      ...
79        1
94        1
82        1
87        1
```

```
69        1
Name: count, Length: 96, dtype: int64

Column 'native-country':
native-country
United-States                  43810
Mexico                           947
?                                582
Philippines                      295
Germany                          206
Puerto-Rico                      184
Canada                           182
El-Salvador                      155
India                            151
Cuba                             138
England                          127
China                            122
South                            115
Jamaica                          106
Italy                            105
Dominican-Republic               103
Japan                             92
Poland                            87
Guatemala                         86
Vietnam                           86
Columbia                          85
Haiti                             75
Portugal                          67
Taiwan                            65
Iran                              59
Greece                            49
Nicaragua                         49
Peru                              46
Ecuador                           45
France                            38
Ireland                           37
Hong                              30
Thailand                          30
Cambodia                          28
Trinadad&Tobago                   27
Laos                              23
Yugoslavia                        23
Outlying-US(Guam-USVI-etc)        23
Scotland                          21
Honduras                          20
Hungary                           19
Holand-Netherlands                 1
Name: count, dtype: int64

Column 'income':
income
<=50K     24698
<=50K.    12430
>50K       7839
>50K.      3846
Name: count, dtype: int64
```

In [19]:

```python
cleaned_data[cleaned_data == '?'] = np.nan
```

```
cleaned_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 48813 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             48813 non-null  int64 
 1   workclass       46014 non-null  object
 2   fnlwgt          48813 non-null  int64 
 3   education       48813 non-null  object
 4   education-num   48813 non-null  int64 
 5   marital-status  48813 non-null  object
 6   occupation      46004 non-null  object
 7   relationship    48813 non-null  object
 8   race            48813 non-null  object
 9   sex             48813 non-null  object
 10  capital-gain    48813 non-null  int64 
 11  capital-loss    48813 non-null  int64 
 12  hours-per-week  48813 non-null  int64 
 13  native-country  47957 non-null  object
 14  income          48813 non-null  object
dtypes: int64(6), object(9)
memory usage: 6.0+ MB
```

In [20]:
```
# count the Uniques elements
def count_unique_elements(df):
    unique_counts = {}
    for column in df.columns:
        unique_counts[column] = df[column].nunique()
    return unique_counts

unique_counts = count_unique_elements(cleaned_data)

#print unique elements for each cloumn
for column, count in unique_counts.items():
    print(f"Column '{column}': {count} unique elements")
```

```
Column 'age': 74 unique elements
Column 'workclass': 8 unique elements
Column 'fnlwgt': 28523 unique elements
Column 'education': 16 unique elements
Column 'education-num': 16 unique elements
Column 'marital-status': 7 unique elements
Column 'occupation': 14 unique elements
Column 'relationship': 6 unique elements
Column 'race': 5 unique elements
Column 'sex': 2 unique elements
Column 'capital-gain': 123 unique elements
Column 'capital-loss': 99 unique elements
Column 'hours-per-week': 96 unique elements
Column 'native-country': 41 unique elements
Column 'income': 4 unique elements
```

In [21]:
```
def detect_missing_values(df):
    # Calculate total number of rows
    total_rows = len(df)

    # Calculate number and proportion of missing values for each column
    missing_values_info = {}
    for col in df.columns:
```

```
        missing_count = df[col].isnull().sum()
        missing_proportion = missing_count / total_rows
        missing_values_info[col] = {'count': missing_count, 'proportion': missing_prop

    # Print missing values info for each column
    for col, info in missing_values_info.items():
        print(f"Column '{col}': {info['count']} missing values, {info['proportion']:.2
```

In [22]:
```
# Run the functions on cleaned_data
pt = detect_missing_values(cleaned_data)
```

```
Column 'age': 0 missing values, 0.00% proportion
Column 'workclass': 2799 missing values, 5.73% proportion
Column 'fnlwgt': 0 missing values, 0.00% proportion
Column 'education': 0 missing values, 0.00% proportion
Column 'education-num': 0 missing values, 0.00% proportion
Column 'marital-status': 0 missing values, 0.00% proportion
Column 'occupation': 2809 missing values, 5.75% proportion
Column 'relationship': 0 missing values, 0.00% proportion
Column 'race': 0 missing values, 0.00% proportion
Column 'sex': 0 missing values, 0.00% proportion
Column 'capital-gain': 0 missing values, 0.00% proportion
Column 'capital-loss': 0 missing values, 0.00% proportion
Column 'hours-per-week': 0 missing values, 0.00% proportion
Column 'native-country': 856 missing values, 1.75% proportion
Column 'income': 0 missing values, 0.00% proportion
```

In [23]:
```
# all three columns that have missing value have categorical data thus we can discard
```

In [24]:
```
cleaned_data.dropna(inplace=True)
```

When handling missing data, the choice of imputation method depends on the nature of the data:

1. For numerical data that is normally distributed or lacks significant outliers, mean imputation is suitable.

2. If the data contains outliers or is skewed, making the mean less representative, median imputation is preferred.

3. Categorical variables with few missing values can be imputed using mode imputation, where the mode (most frequent value) adequately represents the category distribution.

These guidelines help ensure that the imputed values accurately reflect the characteristics of the dataset, improving the reliability of subsequent analyses.

In [25]:
```
def dealing_missing_data(df):
    # handle the missing values
    values = 100 * (round(df.isnull().sum() / df.count(), 2))
    to_delete = []
    to_impute = []
    to_check = []
    for name, proportion in values.items():
        if int(proportion) == 0:
            continue
        elif int(proportion) <= 10:
```

```
            to_impute.append(name)
            df[name].fillna(df[name].median(), inplace=True)  # Impute with median inp
        else:
            to_check.append(name)
    print(f"\nThe missing values in {to_impute} have been replaced by the median.")
    print(f"The columns {to_check} should be further understood")
    return df
```

In [26]:
```
pt = dealing_missing_data(cleaned_data)
```

```
The missing values in [] have been replaced by the median.
The columns [] should be further understood
```

In [27]:
```
cleaned_data.head()
```

Out[27]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female

DETECTING DATA TYPES MISMATCHES

As a data professional, automating the detection of data type mismatches is paramount for ensuring the accuracy and reliability of our analyses. When working with large datasets in a fast-paced environment, manually checking each column for data type inconsistencies is time-consuming and prone to errors.

By automating this process within our data pipeline, we can efficiently identify mismatches between detected and expected data types. This not only saves valuable time but also helps us maintain data integrity by ensuring that each column contains the appropriate data type.

Detecting and addressing data type mismatches early in the pipeline is essential for preventing downstream issues. For example, using numeric operations on columns with string data types can lead to errors or unexpected results in our analyses. By automating this detection process, we can catch these issues early and take corrective actions to ensure the accuracy of our analyses.

Additionally, automating the handling of data type mismatches allows us to maintain consistency and reliability in our data processing workflows. Whether it involves converting data to the correct type, removing mismatched rows, or raising alerts for further investigation, automating these tasks ensures that our data remains clean and consistent throughout the pipeline.

In [28]:

```
expected_types = {'age': 'int64',
                  'workclass': 'str',
                  'fnlwgt': 'int64',
                  'education': 'str',
                  'education-num': 'int64',
                  'marital-status': 'str',
                  'occupation': 'str',
                  'relationship': 'str'        ,
                  'race':'str',
                  'sex':'str',
                  'capital-gain':'int64',
                  'capital-loss':'int64',
                  'hours-per-week':'int64',
                  'native-country':'str',
                  'income':'object'
                  }

def check_data_types(df, expected_types):
    """
    Check the data types of a DataFrame against expected types.

    Parameters:
    - df (pd.DataFrame): The DataFrame to check.
    - expected_types (dict): A dictionary mapping column names to expected data types

    Returns:
    - dict: A report of mismatches and suggested corrections.
    """
    for column, expected_type in expected_types.items():
        actual_type = df[column].dtype

        # Create a readable version of numpy dtype for reporting
        readable_type = np.dtype(actual_type).name
        if not np.issubdtype(actual_type, np.dtype(expected_type).type):
            message = f"Column '{column}' has type '{readable_type}' instead of '{expe
            suggestion = f"Convert '{column}' to '{expected_type}'."
            print(f"{message}", f"{suggestion}")

    print("No data types mismatch detected")
```

In [29]:

```
dt=check_data_types(cleaned_data,expected_types)
```

```
Column 'workclass' has type 'object' instead of 'str'. Convert 'workclass' to 'str'.
Column 'education' has type 'object' instead of 'str'. Convert 'education' to 'str'.
Column 'marital-status' has type 'object' instead of 'str'. Convert 'marital-status'
to 'str'.
Column 'occupation' has type 'object' instead of 'str'. Convert 'occupation' to 'st
r'.
Column 'relationship' has type 'object' instead of 'str'. Convert 'relationship' to
'str'.
Column 'race' has type 'object' instead of 'str'. Convert 'race' to 'str'.
Column 'sex' has type 'object' instead of 'str'. Convert 'sex' to 'str'.
Column 'native-country' has type 'object' instead of 'str'. Convert 'native-country'
to 'str'.
No data types mismatch detected
```

Detecting Numerical and Categorical Columns

As a data professional, detecting numerical and categorical columns is fundamental to understanding the makeup of our dataset. It's like examining the ingredients of a recipe before cooking - knowing what we're working with helps us plan our approach effectively.

Identifying numerical columns allows us to perform mathematical operations, statistical analyses, and build machine learning models. These columns often represent quantities, measurements, or continuous variables, providing valuable insights into trends and patterns within our data.

On the other hand, recognizing categorical columns informs us about the different categories or groups present in our dataset. These columns might include variables like gender, product types, or geographic regions. Understanding categorical data enables us to conduct segmentation, perform comparisons, and create meaningful visualizations.

By automating the detection of numerical and categorical columns in our data pipeline, we streamline the process of preparing and analyzing data. This automation not only saves time but also ensures consistency and accuracy in our workflows. Ultimately, it empowers us to extract meaningful insights and make informed decisions based on the nuances of our data.

In [30]:

```python
def detect_categorical_columns(df):
    # Select columns with categorical data types (objects, categoricals)
    categorical_columns = df.select_dtypes(include=['object', 'category']).columns.tol
    return categorical_columns

# Example usage:
categorical_columns = detect_categorical_columns(cleaned_data)
print("Categorical Columns:", categorical_columns)
```

```
Categorical Columns: ['workclass', 'education', 'marital-status', 'occupation', 'rela
tionship', 'race', 'sex', 'native-country', 'income']
```

In [31]:

```python
def detect_numerical_columns(df):
    # Select columns with numerical data types (integers, floats)
    numerical_columns = df.select_dtypes(include=['int', 'float']).columns.tolist()
    return numerical_columns

# Example usage:
```

```
numerical_columns = detect_numerical_columns(cleaned_data)
print("Numerical Columns:", numerical_columns)
```

```
Numerical Columns: ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss',
'hours-per-week']
```

Checking whether Data is Normalized

A QQ plot, or quantile-quantile plot, is a graphical tool used to assess whether a given dataset follows a particular probability distribution, such as the normal distribution. Here's a more conversational explanation:

Imagine you have a dataset, and you want to know if it looks like it comes from a specific type of distribution, like the normal distribution. A QQ plot helps you with that. It's like having a visual aid to see if your dataset matches up with what you'd expect from the distribution you're interested in.

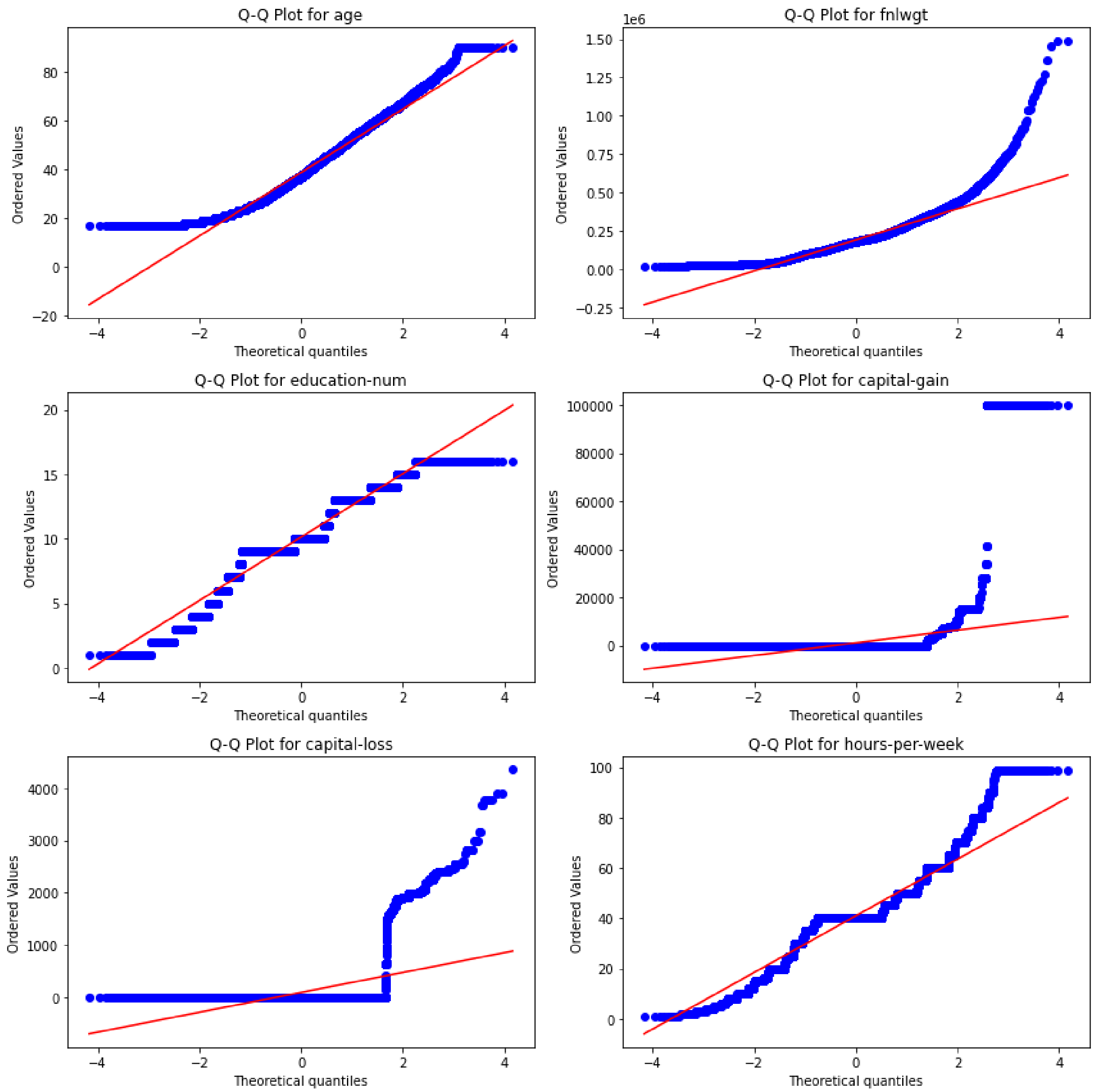
Here's how it works: On a QQ plot, you have two axes. One axis represents the quantiles of your dataset, which are essentially points that divide your data into equal-sized groups. The other axis represents the quantiles of the theoretical distribution you're comparing against, like the normal distribution.

If your dataset closely follows the theoretical distribution, the points on the QQ plot will form a straight line. But if there are deviations, the points will deviate from the straight line, indicating that your dataset differs from the expected distribution.

In [32]:

```
import scipy.stats as stats
import matplotlib.pyplot as plt

def qq_plot_integer_columns(df):
    # Get numerical columns that are of integer data type
    int_columns = df.select_dtypes(include=['int']).columns.tolist()

    # Determine the number of rows and columns for subplots
    num_rows = (len(int_columns) + 1) // 2  # Add 1 to round up if odd
    num_cols = min(len(int_columns), 2)  # Limit to 2 columns for readability

    # Create subplots for QQ plots
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 12))

    # Flatten axes if there is only one row or one column
    if num_rows == 1:
        axes = axes.reshape(1, -1)
    if num_cols == 1:
        axes = axes.reshape(-1, 1)

    # Plot Q-Q plots for each numerical column
    for i, column in enumerate(int_columns):
        row_index = i // num_cols
        col_index = i % num_cols
        stats.probplot(df[column], dist="norm", plot=axes[row_index, col_index])
        axes[row_index, col_index].set_title(f'Q-Q Plot for {column}')

    plt.tight_layout()
```

```
    plt.show()

# Example usage:
qq_plot_integer_columns(cleaned_data)
```



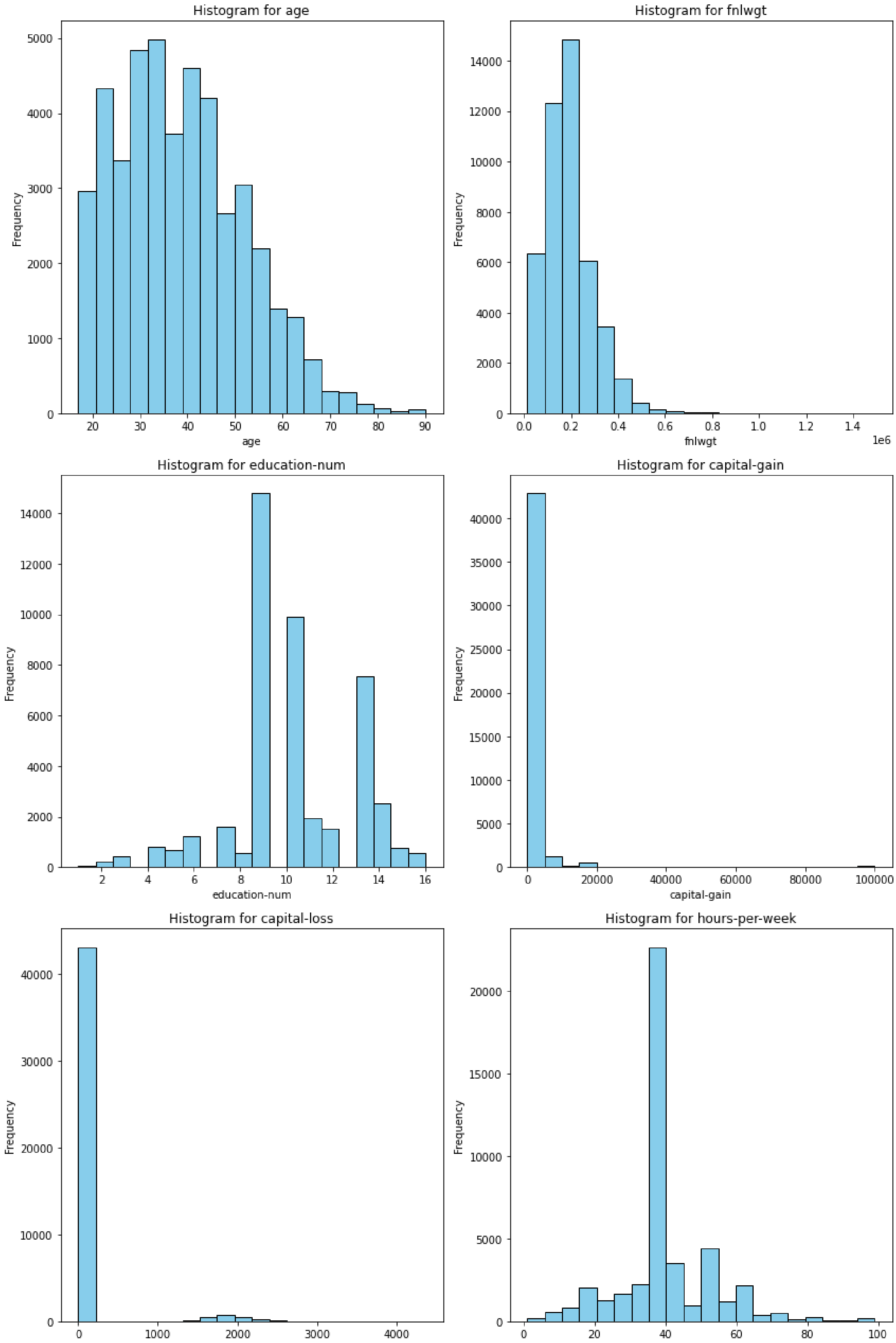
As a data professional, ensuring that data is normalized is crucial for maintaining consistency and reliability in our analyses. It's like ensuring that all the ingredients in a recipe are in the right proportions - without normalization, our analyses may be skewed or inaccurate.

Normalizing data involves scaling it to a common range or distribution, which makes comparisons and interpretations more meaningful. This is particularly important when working with features or variables that have different scales or units of measurement. For example, if one variable ranges from 0 to 100 and another from 0 to 100,000, without normalization, the larger variable would dominate the analysis, leading to biased results.

By checking whether data is normalized, we ensure that our analyses are fair and unbiased, allowing us to draw accurate conclusions and make informed decisions. Normalized data also tends to be more conducive to modeling and machine learning algorithms, as it helps mitigate issues such as overfitting and instability.

In [33]:
```
import matplotlib.pyplot as plt

def plot_histograms(df):
    int_columns = df.select_dtypes(include=['int']).columns.tolist()
    num_cols = len(int_columns)
    num_rows = (num_cols + 1) // 2  # Add 1 to round up if odd

    fig, axes = plt.subplots(num_rows, 2, figsize=(12, 6 * num_rows))

    # Flatten axes if there are multiple rows
    if num_rows > 1:
        axes = axes.flatten()

    for i, column in enumerate(int_columns):
        ax = axes[i]
        ax.hist(df[column], bins=20, color='skyblue', edgecolor='black')
        ax.set_title(f'Histogram for {column}')
        ax.set_xlabel(column)
        ax.set_ylabel('Frequency')

    # Hide unused subplots
    for j in range(num_cols, num_rows * 2):
        axes[j].axis('off')

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

In [34]:
```
plot_histograms(cleaned_data)
```


Histogram for age
Frequency
age
0
1000
2000
3000
4000
5000
20
30
40
50
60
70
80
90
Histogram for fnlwgt
Frequency
fnlwgt
1e6
0
2000
4000
6000
8000
10000
12000
14000
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Histogram for education-num
Frequency
education-num
0
2000
4000
6000
8000
10000
12000
14000
2
4
6
8
10
12
14
16
Histogram for capital-gain
Frequency
capital-gain
0
5000
10000
15000
20000
25000
30000
35000
40000
0
20000
40000
60000
80000
100000
Histogram for capital-loss
Frequency
capital-loss
0
10000
20000
30000
40000
0
1000
2000
3000
4000
Histogram for hours-per-week
Frequency
hours-per-week
0
5000
10000
15000
20000
0
20
40
60
80
100