1. Background and Aim

1.1 Problem Description

Factors that determine high income are of importance for individuals making life choices and policy makers alike. The present assignment aims to analyse a large set of salary data that is paired with socio-economic attributes. The goal of the project will be firstly to deduct through data mining techniques, which factors are associated with an above average annual salary. Association rule mining in the form of the Apriori algorithm as well as the FB-growth algorithm will be applied to find sets of related factors. In a second phase, the project will identify groups that are likely to be high earners in the data via clustering, with a focus on an above average salary without higher education. Conclusively, a decision tree will be developed to visualise the splitting points in the data and to predict high earners.

1.2 Aim

Research Question 01: What are the factors that influence someone to become a high earner?

Research Question 02: Which non-university-educated groups are predicted to earn over 50k?

2. Data Description

A dataset extracted from the 1994 Census database by Barry Becker will be utilized. The dataset includes demographic and occupational information from 32,561 individuals including a variable of salary >50k or <50k. This dataset is accessible via the UCI Machine Learning Repository and can be found at this link (Kohavi, 1996). The variables include information such as age, education, marital status, occupation, race, sex, financial details, weekly working hours, native country, and salary.

3. Algorithms and Techniques

3.1 Research Question 01: Apriori Algorithm for Association Rule Mining

The Apriori algorithm, a fundamental component of association rule mining, relies upon the concept of frequency, systematically uncovering combinations of attributes that display frequent concurrences (Agrawal & Srikant, 1994). In this research, the Apriori algorithm will be applied to identify patterns of socio-economic attributes that commonly manifest among individuals with high earnings.

3.2 Research Question 01: FP Growth

Moving on from the complexity of Apriori, the optimised FP Growth algorithm brings about a new level of efficiency. It turns large sets of data into a condensed structure called the FP-Tree, avoiding exhaustive searches and speeding up the process of finding patterns (Wang, He, & Han, 2021). The objective is to apply this algorithm, to quickly and accurately identify the socio-economic configurations that are indicative of high-income brackets.

3.3 Research Question 02: K-means clustering Algorithm.

The K-means clustering algorithm, a powerful unsupervised learning technique, will be employed to address the research question of identifying non-university-educated groups that are predicted to earn over $50,000 annually. By grouping individuals based on similar attributes such as age, occupation, marital status, and more, K-means clustering will reveal distinct clusters within this subset.

3.4 Research Question 02: Decision Tree

Decision Trees is an intuitive and interpretable machine learning algorithm that functions by dividing the dataset into smaller subsets based on critical attributes, eventually leading to a decision outcome. The main advantage of the algorithm is that it provides a clear vision of the factors and decision paths responsible for specific outcomes (Charbuty & Abdulazeez, 2021). In the research, a Decision Tree will be employed to analyze the influence of socio-economic attributes on the earning potential (>50k) of individuals who have not attended university.

4. Evaluation Measures

4.1 Measures for Apriori & FP-Growth Algorithms

- **Support**: Indicates the popularity of an itemset. For this question, it could show how common certain combinations of attributes are among those earning more than $50K without a university degree.
- **Confidence**: Indicates the likelihood of the consequence occurring given the forerunner. An example could be if a rule suggests that people in a particular occupation and age range often make more than $50K, the confidence indicates the probability of this being true.
- **Lift**: Gives the likelihood of the consequent (earning > $50K) given the antecedent, compared to the probability of the consequent in general. A lift value greater than 1 suggests a stronger association.

4.2 Measures for K-means Clustering Algorithm

To evaluate the generated clusters and the performance of the K-means algorithm, we will look at the number of clusters and the spread of the data-points within clusters (inertia). The goal will be to follow an elbow approach to balance the trade-offs of number of clusters and inertia (Data Camp, 2020).

4.3 Measures for Decision Tree Algorithm

- **Precision**: Precision measures the accuracy of positive predictions - the number of actual correct positive predictions.
- **Recall**: Recall measures the proportion of actual positives the model correctly identifies - essential were missing a true positive can have vital implications.
- **Confusion Matrix**: The Matrix provides a clear representation of a model's predictions, including true and false positives and negatives, offering a direct snapshot of its performance.

References

Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994.

Charbuty, B., & Abdulazeez, A. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. *Journal of Applied Science and Technology Trends*, *2*(01), 20 - 28. https://doi.org/10.38094/jastt20165

Data Camp (2020) . Python Tutorial : Evaluating a clustering https://www.youtube.com/watch?v=pKBUmK4XT2I&ab_channel=DataCamp

Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and techniques*. Elsevier Science & Technology.

Kohavi, Ron. (1996). Census Income. UCI Machine Learning Repository. https://doi.org/10.24432/C5GP7S.

Shawkat, M., Badawi, M., El-ghamrawy, S., Arnous, R., & El-desoky, A. (2021). An optimized fp-growth algorithm for discovery of association rules. *The Journal of Supercomputing: An International Journal of High-Performance Computer Design, Analysis, and Use*, *78*(4), 5479–5506. https://doi.org/10.1007/s11227-021-04066-y

Appendix: Data Investigation Report

Introduction

In today's dynamic job market, accurate salary prediction is essential for job seekers, employers, and policymakers. This data investigation aims to delve into a comprehensive dataset that encompasses various socio-economic attributes such as age, workclass, education, marital status, occupation, race, sex, financial attributes, hours worked per week, native country, and salary. The objective is to gain a deep understanding of the data and leverage machine learning techniques to predict whether an individual's salary is likely to be greater than $50,000, focusing particularly on non-university-educated groups.

Data Exploration

The first phase of our investigation involved a thorough exploration of the dataset. We examined the range and distribution of each attribute, identified missing values, and assessed data types to understand the dataset's structure and quality. This exploration laid the foundation for subsequent analysis and modelling.

The number of the data samples: 32,561

The types of attributes :

Column	Attribute name	Column	Attribute name
age	Ratio-scaled Attribute	relationship	Nominal Attribute
workclass	Nominal Attribute	race	Nominal Attribute
fnlwgt	Continuous Attribute	sex	Nominal Attribute
education	Ordinal Attribute	hours-per-week	Continuous Attribute
education-num	Continuous Attribute	native-country	Nominal Attribute
marital-status	Nominal Attribute	salary	Ordinal Attribute
occupation	Nominal Attribute		

Importing Libraries & Loading the CSV File:

```
In [1]: import pandas as pd
        import numpy as np
```

```
In [2]: dataset = pd.read_csv(r"C:\Users\Nuwan-New\Desktop\test1.csv",header=0)
```

This code segment imports the pandas library, reads data from a CSV file located at the specified path, and stores it in a pandas DataFrame named "dataset". The data is assumed to have a header row containing column names

Preview the data:

```
In [3]: print(dataset.head())

   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

       marital-status         occupation   relationship   race     sex  \
0       Never-married       Adm-clerical  Not-in-family  White    Male
1  Married-civ-spouse    Exec-managerial        Husband  White    Male
2            Divorced  Handlers-cleaners  Not-in-family  White    Male
3  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
4  Married-civ-spouse     Prof-specialty           Wife  Black  Female

   capital-gain  capital-loss  hours-per-week native-country salary
0          2174             0              40  United-States  <=50K
1             0             0              13  United-States  <=50K
2             0             0              40  United-States  <=50K
3             0             0              40  United-States  <=50K
4             0             0              40           Cuba  <=50K
```

By executing print(dataset.head()), will see the first few rows of the "dataset" DataFrame displayed in your console or output window.

Describe the numerical data: five-number summary for the numerical attribute.

```
In [4]: print(dataset.describe())
```

```
                age        fnlwgt  education-num  capital-gain  capital-loss
count  32561.000000  3.256100e+04   32561.000000  32561.000000  32561.000000
mean      38.581647  1.897784e+05      10.080679   1077.648844     87.303830
std       13.640433  1.055500e+05       2.572720   7385.292085    402.960219
min       17.000000  1.228500e+04       1.000000      0.000000      0.000000
25%       28.000000  1.178270e+05       9.000000      0.000000      0.000000
50%       37.000000  1.783560e+05      10.000000      0.000000      0.000000
75%       48.000000  2.370510e+05      12.000000      0.000000      0.000000
max       90.000000  1.484705e+06      16.000000  99999.000000   4356.000000

       hours-per-week
count    32561.000000
mean        40.437456
std         12.347429
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
```

The above command displaying a tabular output of the summary statistics for each numerical column in the "dataset" DataFrame. This provides insights into the distribution and characteristics of the numerical data in the dataset.

Display the summary of the attributes in the dataset.

```
In [5]: print(dataset.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  salary          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
None
```

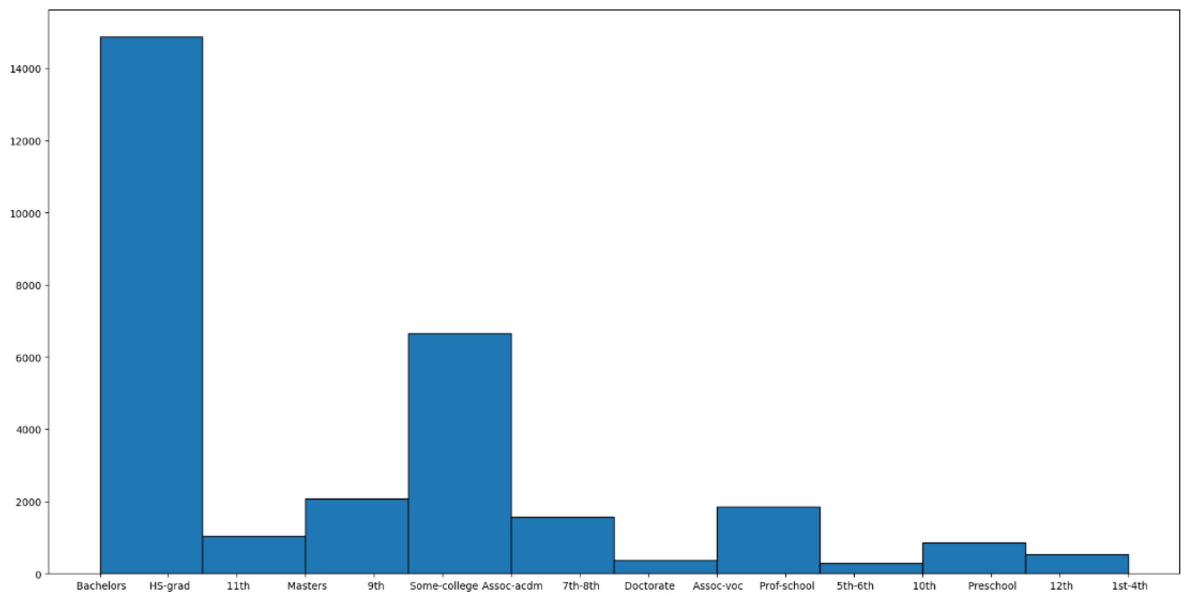
This command can use to display the summary of the DataFrame's structure, including the data types and the number of non-null values in each column. This information provides insights into the dataset's completeness and helps identify potential issues with missing or incorrect data.

Data Visualisation

Create a histogram for the "education" column.

The code segment provided the uses of Matplotlib to create a histogram for the "education" column of the dataset. The histogram is a graphical representation of the distribution of values in the column. The x-axis represents the different education levels, and the y-axis represents the frequency of each education level in the dataset.

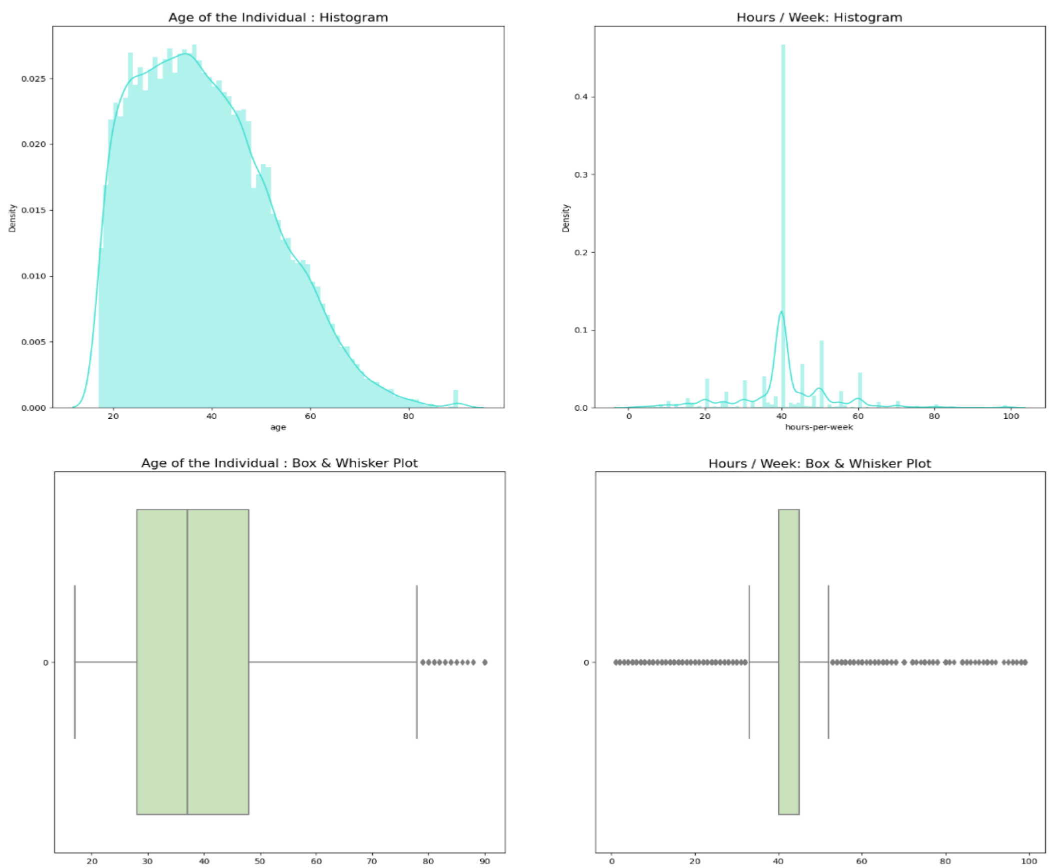
```
In [20]: import matplotlib.pyplot as plt

plt.subplots(figsize=(20,10))

# Histogram
plt.hist(dataset['education'],edgecolor='black')
plt.show()
```



Create a histograms and box plots for the age and hours-per-week.

The code segment provided the uses of Seaborn library to create a grid of subplots containing histograms and box plots for the "age" and "hours-per-week" columns of your dataset. In the diagram below shows a grid of four subplots. The top-left and top-right subplots are histograms of "age" and "hours-per-week" respectively. The bottom-left and bottom-right subplots are box plots of the same columns. These visualizations help you understand the distribution and variability of the data in these columns.

```
In [28]: import seaborn as sns

         plt.subplots(figsize=(20,20))

         plt.subplot(2,2,1)
         plt.title('Age of the Individual : Histogram', fontdict={'fontsize':15})
         sns.distplot(dataset.age, color='#40E0D0', bins=73)

         plt.subplot(2,2,2)
         plt.title('Hours / Week: Histogram', fontdict={'fontsize':15})
         sns.distplot(dataset['hours-per-week'], color='#40E0D0', bins=98)

         plt.subplot(2,2,3)
         plt.title('Age of the Individual : Box & Whisker Plot', fontdict={'fontsize':15})
         sns.boxplot(dataset['age'], orient='h',color="#c7e9b4")

         plt.subplot(2,2,4)
         plt.title('Hours / Week: Box & Whisker Plot', fontdict={'fontsize':15})
         sns.boxplot(dataset['hours-per-week'], orient='h', color="#c7e9b4")

         plt.show()
```



Create a box and whisker plots for the fnlwgt, capital-gain and capital-loss.

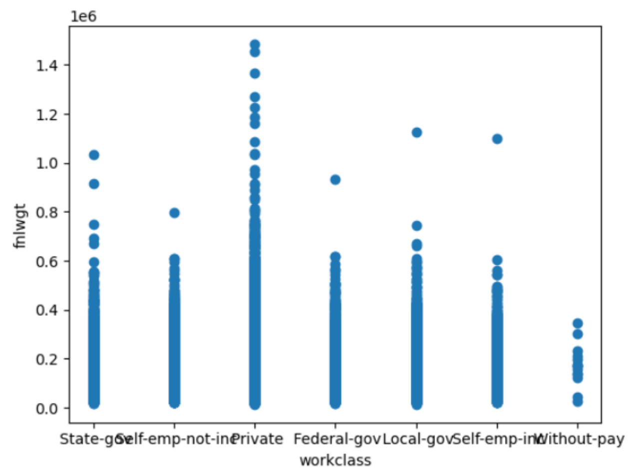
Creating the creating a series of separate box and whisker plots using Seaborn for the "fnlwgt," "capital-gain," and "capital-loss" columns of the dataset.

```
In [22]: plt.subplots(figsize=(10,5))
         plt.title('fnlwgt: Box & Whisker Plot', fontdict={'fontsize':15})
         sns.boxplot(dataset['fnlwgt'], orient='h', color="#c7e9b4")

         plt.subplots(figsize=(10,5))
         plt.title('capital-gain: Box & Whisker Plot', fontdict={'fontsize':15})
         sns.boxplot(dataset['capital-gain'], orient='h', color="#c7e9b4")

         plt.subplots(figsize=(10,5))
         plt.title('capital-loss: Box & Whisker Plot', fontdict={'fontsize':15})
         sns.boxplot(dataset['capital-loss'], orient='h', color="#c7e9b4")

         plt.show()
```

Create a Scatter plot between workclass and fnlwgt.

The below scatter plot shows the relationship between the "workclass" and "fnlwgt" columns. The plt.scatter() function is used to create the scatter plot, and you've specified the x-axis as "workclass" and the y-axis as "fnlwgt."

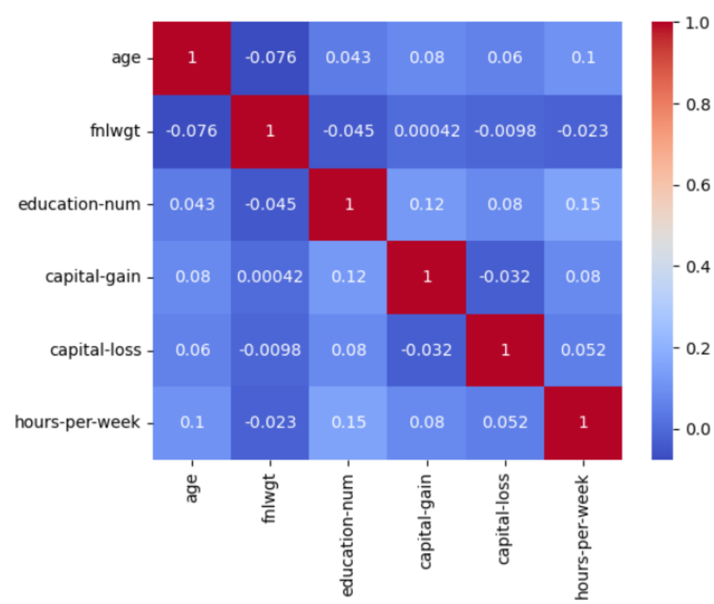
In [23]:
```
# Scatter plot for two variables
plt.scatter(dataset['workclass'], dataset['fnlwgt'])
plt.xlabel('workclass')
plt.ylabel('fnlwgt')
plt.show()
```



Create a Correlation Matrix Heatmap

The following command can use to create a correlation matrix heatmap.

```
# Correlation matrix
correlation_matrix = dataset.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

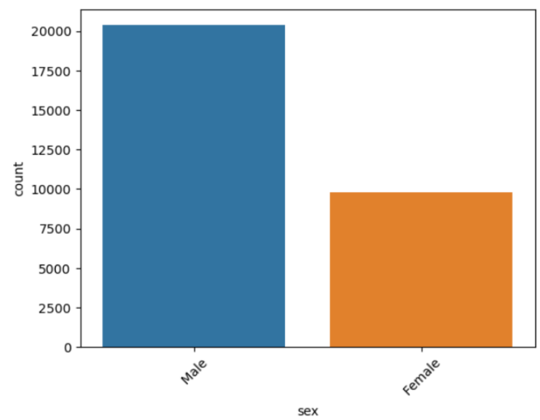


Both scatter plot and Correlation Matrix Heatmap provides insights into the relationships and correlations within the data. The scatter plot helps visualize the relationship between two variables, while the correlation matrix heatmap provides a more systematic view of how variables are correlated with each other.

Create a count plot.

Using the below command can create count plot using Seaborn. The sns.countplot() function is used to create the plot. It specified the data parameter as dataset and the x parameter as 'sex', which means that to count the occurrences of each gender in the dataset. The plt.xticks(rotation=45) line rotates the x-axis labels for better readability.

```
In [24]: # Count plot
         sns.countplot(data=dataset, x='sex')
         plt.xticks(rotation=45)
         plt.show()
```



Create Cross-Tabulation:

In the below code segement using the pd.crosstab() function from pandas to create a cross-tabulation (also known as a contingency table) between the "occupation" and "native-country" columns. A cross-tabulation displays the frequency distribution of two categorical variables in a tabular format. The function calculates the counts of occurrences where each combination of occupation and native country appears together.

```
# Cross-tabulation
pd.crosstab(dataset['occupation'], dataset['native-country'])
```

native-country	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	Ecuador	El-Salvador	England	France	...	Portugal	Puerto-Rico	Scotland	South	Taiwan
occupation																
Adm-clerical	0	12	2	8	12	4	1	4	7	1	...	4	17	1	4	3
Armed-Forces	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
Craft-repair	6	15	3	9	7	6	4	15	8	0	...	12	14	0	8	2
Exec-managerial	1	13	10	4	16	2	1	2	20	7	...	2	10	3	13	11
Farming-fishing	1	2	0	0	2	0	0	2	1	1	...	1	5	0	0	0
Handlers-cleaners	0	2	0	3	3	5	1	7	2	0	...	5	4	0	2	0
Machine-op-inspct	4	5	8	10	6	22	8	6	3	0	...	6	16	2	2	0
Other-service	1	12	16	8	13	12	3	40	7	2	...	2	16	2	11	1
Priv-house-serv	0	0	0	1	2	1	1	6	3	2	...	0	0	0	0	0
Prof-specialty	3	24	22	5	11	3	2	7	21	8	...	0	10	2	11	21
Protective-serv	0	2	0	0	2	1	0	0	3	1	...	1	3	1	0	0
Sales	2	9	5	3	10	8	3	7	7	2	...	1	9	0	19	3
Tech-support	0	3	2	3	0	0	1	0	4	3	...	0	0	0	0	1
Transport-moving	0	8	0	2	8	3	2	4	0	0	...	0	5	0	1	0

14 rows × 41 columns

Both count plot and cross-tabulation insights into the distribution and relationships within the data. The count plot shows the distribution of gender, while the cross-tabulation displays how different occupations are distributed across various native countries.

Data Pre-Processing

Identifying whether any duplicates are available in the dataset.

```
In [6]: dataset.duplicated().any()
Out[6]: True
```

The code dataset.duplicated().any() is used to check whether there are any duplicated rows in the DataFrame "dataset". When you execute dataset.duplicated().any(), it will return either True or False, indicating whether there are any duplicated rows in the "dataset" DataFrame. If the result is True, it means that there are duplicated rows; if the result is False, it means that there are no duplicated rows. In this result set duplicate values are available since the answer is true.

Assign the duplicate values to new dataset.

The code segment creates a new DataFrame "sal_dup" that contains rows from the original "dataset" DataFrame which are identified as duplicates based on the parameter keep='last'. This means that if a row is duplicated, the last occurrence of that duplicated row will be included in the "sal_dup" DataFrame.

Removing duplicates from original dataset.

```
In [7]: sal_dup=dataset[dataset.duplicated(keep='last')]

In [8]: sal_dup
Out[8]:
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
2303	90	Private	52386	Some-college	10	Never-married	Other-service	Not-in-family	Asian-Pac-Islander	Male	0	0	35	United-States	<=50K
3917	19	Private	251579	Some-college	10	Never-married	Other-service	Own-child	White	Male	0	0	14	United-States	<=50K
4325	25	Private	308144	Bachelors	13	Never-married	Craft-repair	Not-in-family	White	Male	0	0	40	Mexico	<=50K
4767	21	Private	250051	Some-college	10	Never-married	Prof-specialty	Own-child	White	Female	0	0	10	United-States	<=50K
4940	38	Private	207202	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	48	United-States	>50K
5579	27	Private	255582	HS-grad	9	Never-married	Machine-op-inspct	Not-in-family	White	Female	0	0	40	United-States	<=50K
5805	20	Private	107658	Some-college	10	Never-married	Tech-support	Not-in-family	White	Female	0	0	10	United-States	<=50K
5842	25	Private	195994	1st-4th	2	Never-married	Priv-house-serv	Not-in-family	White	Female	0	0	40	Guatemala	<=50K
6990	19	Private	138153	Some-college	10	Never-married	Adm-clerical	Own-child	White	Female	0	0	10	United-States	<=50K
7053	49	Self-emp-not-inc	43479	Some-college	10	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
7920	49	Private	31267	7th-8th	4	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
8080	21	Private	243368	Preschool	1	Never-married	Farming-fishing	Not-in-family	White	Male	0	0	50	Mexico	<=50K

```
In [9]: dataset=dataset.drop_duplicates()
        dataset.shape

Out[9]: (32537, 15)
```

The above code segment is used to remove duplicated rows from the "dataset" DataFrame and then display the resulting shape of the DataFrame.

Confirming that duplicates remove from original dataset.

```
In [10]: dataset.duplicated().any()

Out[10]: False
```

The above code segment is used to check if there are any duplicated rows in a dataset. After removing the duplicate values from the original dataset can figure out that it results false. Consider that removing or handling duplicated(noisy) data can be an important step in data preprocessing, as duplicated data can skew analysis and lead to incorrect conclusions.

Checking the missing values fields in the dataset

Consider the attribute of the workclass. The below code segment is used to calculates the frequency of each unique value (how many times each of these values appears) in the "workclass" column of the dataset and then prints out the resulting value frequency counts.

```
In [11]: values=dataset.workclass.value_counts()
         print(values)

 Private             22673
 Self-emp-not-inc     2540
 Local-gov            2093
 ?                    1836
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
```

Based on the result set can identify the missing values in the workclass attribute which is indicates by “?”. So can determine that the workclass attribute contains the missing data.
Similarly, can check the other attributes and figure it out that occupation and native country also includes the “**?**” which indicates the noisy data.

Replacing and removing the noisy and missing data.

In [21]:
```
values=dataset.occupation.value_counts()
print(values)
```

```
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3770
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: occupation, dtype: int64
```

In [12]:
```
dataset['workclass'] = dataset['workclass'].replace(' ?', np.nan)
dataset['occupation'] = dataset['occupation'].replace(" ?", np.nan)
dataset['native-country'] = dataset['native-country'].replace(" ?", np.nan)
```

The above code segment used to replace specific values (" ?") in the "workclass," "occupation," and "native-country" columns of the dataset with NaN (Not a Number) values. This is often done as a part of data preprocessing to handle missing or noisy values.

Counting the no of NaN values available in each attribute.

```
In [13]: print(dataset.isna().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     582
salary               0
dtype: int64
```

The above code segment used to calculate and prints the number of missing values (NaN) in each column of the dataset. This information is important for identifying which columns have missing and noisy data.

Handling the Missing and Noisy values in the dataset

The below code segment is used to fill in missing values in the "workclass," "occupation," and "native-country" columns of the dataset.

```
In [14]: dataset['workclass'] = dataset['workclass'].fillna('other')
         ex1=dataset['occupation'].fillna(dataset['occupation'].mode()[0])
         ex2=dataset['native-country'].fillna(dataset['native-country'].mode()[0])
```

Line No 01: This line of code fills in missing values in the "workclass" column with the value "other." Since the workclass is categorical column, replaced the missing values with a specific category that represents "other" or "unknown."

Line No 02: This line of code fills in missing values in the "occupation" column using the mode of the "occupation" column. The. mode()[0] part returns the most frequent value in the "occupation" column. So, any missing values in the "occupation" column will be replaced with the most common occupation.

Line No 03: Similarly, this line of code fills in missing values in the "native-country" column using the mode of the "native-country" column.

Re-Evaluate the workclass field.

Method 01:

The last step replaced the nan values of the workclass column with other. Using the below command can verify nan has replaced with other by displaying the frequency counts in the "workclass" column.

.

```
In [16]: values=dataset.workclass.value_counts()
         print(values)
```

```
 Private             22673
 Self-emp-not-inc     2540
 Local-gov            2093
other                 1836
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
```

Method 02:

The below command will provide an output showing the count of missing or noisy values in each column of the dataset. For the workclass field does not contains NaN values since the count is zero.

```
In [17]: print(dataset.isna().sum())
```

```
age                  0
workclass            0
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     582
salary               0
dtype: int64
```

Removing noisy data from the native-country and occupation fields.

```
In [18]: dataset=dataset.dropna(how='any')
```

```
In [19]: print(dataset.isna().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
salary            0
dtype: int64
```

The first command is used to remove rows with any missing values from the dataset. That means if any column in a row contains a missing value (NaN), the entire row will be dropped. With the second command again can check the count of missing or noisy values in each column of the dataset. But now can see that both native-country and occupation fields also the NaN values are removed.

Calculates the lower and upper bounds for outliers using the Interquartile Range (IQR) method.

The code segment calculates the lower and upper bounds for outliers using the Interquartile Range (IQR) method and then applies these bounds to the "fnlwgt" column of the dataset.

```
In [25]: q1 = dataset['fnlwgt'].quantile(0.25)
         q3 = dataset['fnlwgt'].quantile(0.75)
         iqr = q3 - q1
         lower_bound = q1 - 1.5 * iqr
         upper_bound = q3 + 1.5 * iqr
         dataset['fnlwgt'] = dataset['fnlwgt'].clip(lower_bound, upper_bound)
         print("",q1,"",q3,"",iqr,"",lower_bound,"",upper_bound)

 117627.5  237604.5  119977.0  -62338.0  417570.0
```

By applying the IQR method and clipping the "fnlwgt" column's values, required to handle potential outliers in a way that prevents them from significantly affecting statistical analyses or visualizations.

Select and display the specific columns from the dataset.

Can use the below code segment to select specific columns from the dataset DataFrame. It's creating a new DataFrame that includes only the columns listed.

```
In [26]: dataset=dataset.loc[:, ['workclass',"fnlwgt","education","education-num","occupation",
                        "capital-gain","capital-loss","hours-per-week","native-country","salary"]]
```

```
In [27]: print(dataset.head())
```

```
          workclass  fnlwgt  education  education-num         occupation  \
0         State-gov   77516  Bachelors             13       Adm-clerical
1  Self-emp-not-inc   83311  Bachelors             13    Exec-managerial
2           Private  215646    HS-grad              9  Handlers-cleaners
3           Private  234721       11th              7  Handlers-cleaners
4           Private  338409  Bachelors             13     Prof-specialty

   capital-gain  capital-loss  hours-per-week  native-country  salary
0          2174             0              40   United-States   <=50K
1             0             0              13   United-States   <=50K
2             0             0              40   United-States   <=50K
3             0             0              40   United-States   <=50K
4             0             0              40            Cuba   <=50K
```

Encode categorical variables in the dataset into numerical values.

With use of the below code segment can encode categorical variables in the dataset DataFrame into numerical values. This is a common step in preprocessing data for machine learning algorithms. The encoded values can be useful for training machine learning models, as they convert categorical data into a

```
In [28]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         dataset['workclass'] = le.fit_transform(dataset['workclass'])
         dataset['education'] = le.fit_transform(dataset['education'])
         dataset['occupation'] = le.fit_transform(dataset['occupation'])
         dataset['native-country'] = le.fit_transform(dataset['native-country'])
         dataset['salary'] = le.fit_transform(dataset['salary'])
```

```
In [29]: dataset.head()
```

Out[29]:

	workclass	fnlwgt	education	education-num	occupation	capital-gain	capital-loss	hours-per-week	native-country	salary
0	5	77516	9	13	0	2174	0	40	38	0
1	4	83311	9	13	3	0	0	13	38	0
2	2	215646	11	9	5	0	0	40	38	0
3	2	234721	1	7	5	0	0	40	38	0
4	2	338409	9	13	9	0	0	40	4	0

format that can be used in mathematical computations.