Bilenkin540Weeks_3_&_4_Exercises

March 28, 2025

1. The Data Wrangling Workshop: Activity 3.01, page 155

1. Load the necessary libraries.

[300]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Read in the Boston Housing dataset (given as a .csv file) from the local directory.

[301]:
```
dataset_file_path = r"C:\Users\maxim\OneDrive\Desktop\BU\DSC 540\Boston_housing.
↪csv"
df = pd.read_csv(dataset_file_path)
```

3. Check the first 10 records. Find the total number of records.

[302]:
```
# Displaying the first 10 rows from dataset Boston_housing.csv
print(df.head(10))
print(f"Total number of records: {df.shape[0]}")
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31     0  0.538  6.575   65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421   78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185   61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998   45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147   54.2  6.0622    3  222     18.7
5  0.02985   0.0   2.18     0  0.458  6.430   58.7  6.0622    3  222     18.7
6  0.08829  12.5   7.87     0  0.524  6.012   66.6  5.5605    5  311     15.2
7  0.14455  12.5   7.87     0  0.524  6.172   96.1  5.9505    5  311     15.2
8  0.21124  12.5   7.87     0  0.524  5.631  100.0  6.0821    5  311     15.2
9  0.17004  12.5   7.87     0  0.524  6.004   85.9  6.5921    5  311     15.2

        B  LSTAT  PRICE
0  396.90   4.98   24.0
1  396.90   9.14   21.6
2  392.83   4.03   34.7
3  394.63   2.94   33.4
4  396.90   5.33   36.2
5  394.12   5.21   28.7
6  395.60  12.43   22.9
Total number of records: 506
```

```
7  396.90  19.15   27.1
8  386.63  29.93   16.5
9  386.71  17.10   18.9
Total number of records: 506
```

4. Create a smaller DataFrame with columns that do not include CHAS, NOX, B, and LSTAT:

Chas: Charlse River Dummy variable

Nox: Nitric Oxide concentration

B: Proportion of the population that is African American

LSTAT: Percentage of lower-income population

[303]:
```
# Dropping the columns with names CHAS, NOX, B,and LSTAT
df_new = df.drop(columns=['CHAS', 'NOX', 'B', 'LSTAT'])
```

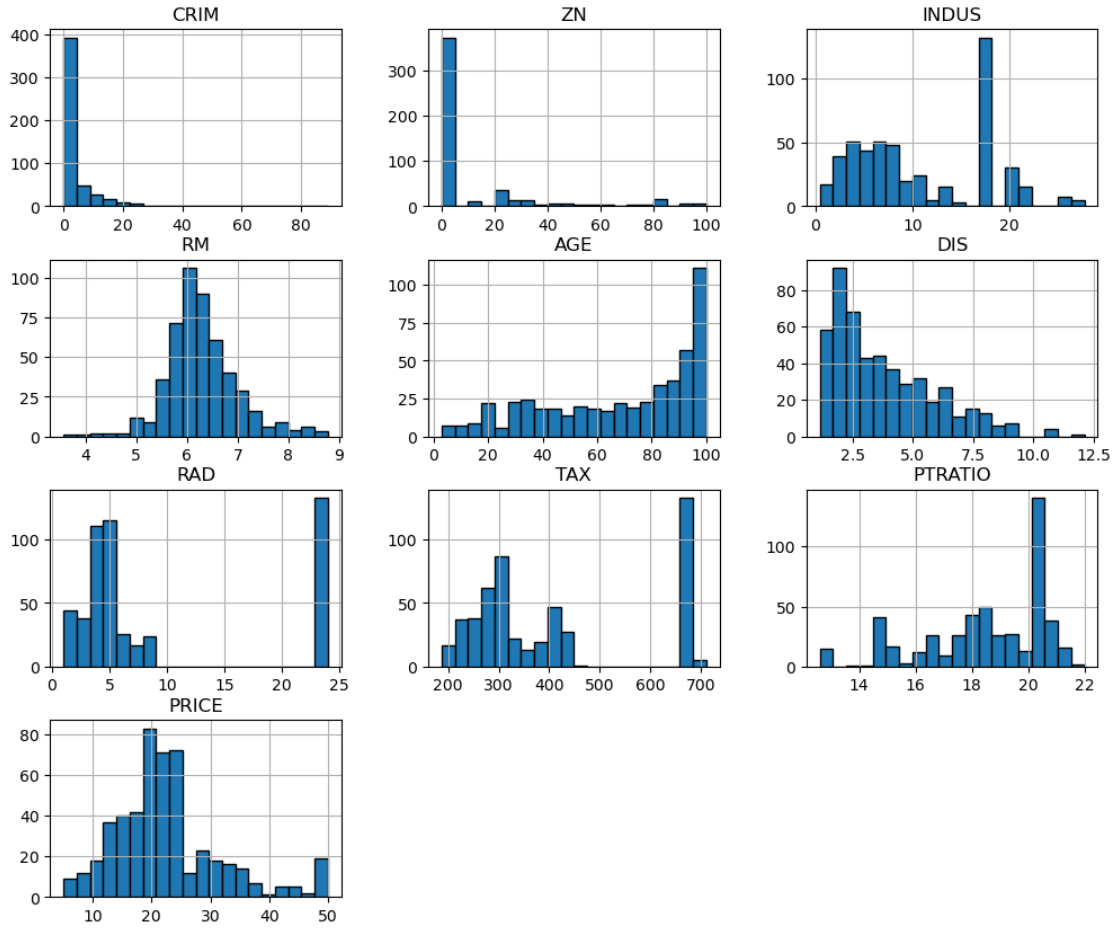
5. Check the last seven records of the new DataFrame you just created.

[304]:
```
# Displaying the last 7 records of the new DataFrame using tail() method
print(df_new.tail(7))
```

```
        CRIM   ZN  INDUS     RM   AGE     DIS  RAD  TAX  PTRATIO  PRICE
499  0.17783  0.0   9.69  5.569  73.5  2.3999    6  391     19.2   17.5
500  0.22438  0.0   9.69  6.027  79.7  2.4982    6  391     19.2   16.8
501  0.06263  0.0  11.93  6.593  69.1  2.4786    1  273     21.0   22.4
502  0.04527  0.0  11.93  6.120  76.7  2.2875    1  273     21.0   20.6
503  0.06076  0.0  11.93  6.976  91.0  2.1675    1  273     21.0   23.9
504  0.10959  0.0  11.93  6.794  89.3  2.3889    1  273     21.0   22.0
505  0.04741  0.0  11.93  6.030  80.8  2.5050    1  273     21.0   11.9
```

6. Plot the histograms of all the variables (columns) in the new DataFrame.

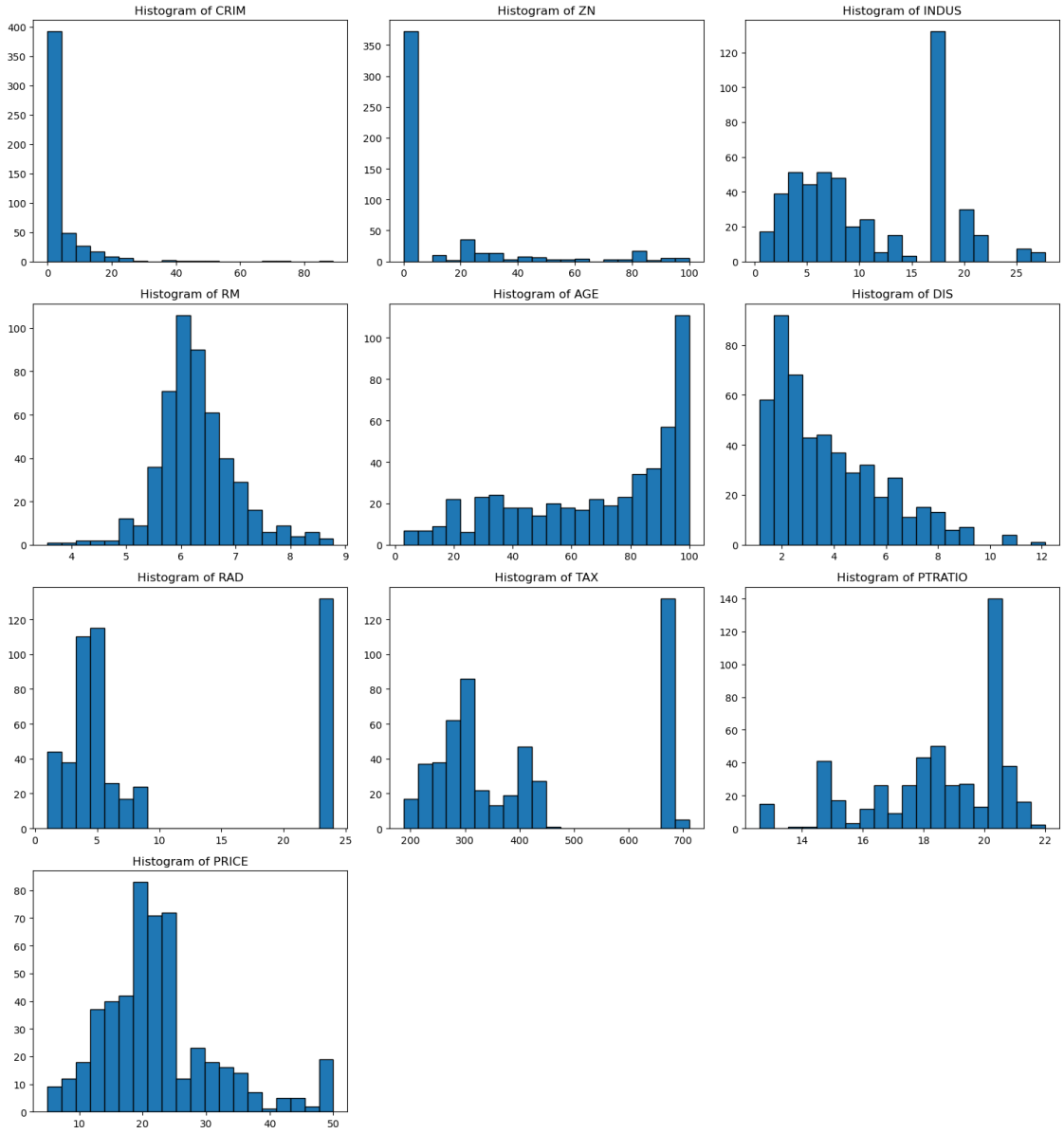
[305]:
```
# Plotting histograms of all the variables (columns)
df_new.hist(figsize=(12, 10), bins=20, edgecolor='black')
plt.suptitle("Histograms of all the Variables (columns) in the new Boston␣
 ↪Housing DataFrame")
plt.show()
```



7. Plot them all at once using a for loop. Try to add a unique title to the plot.

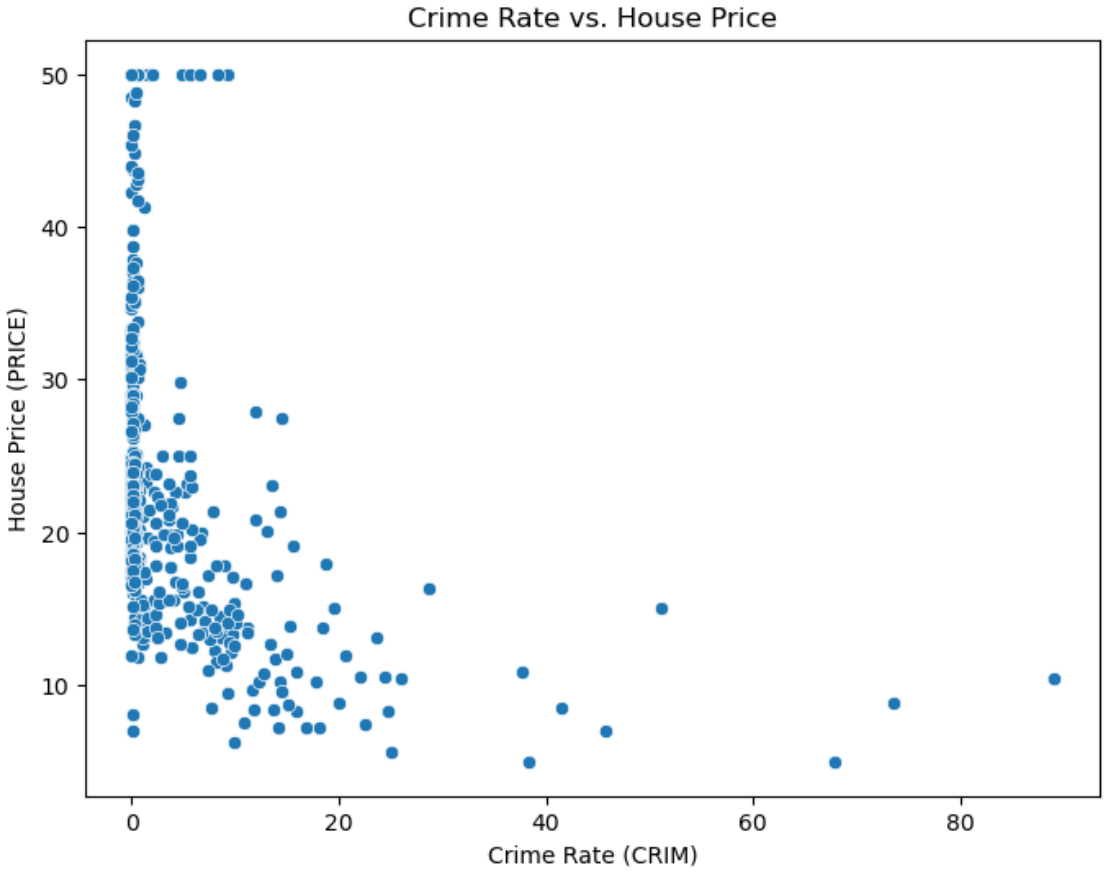
[306]:
```
# Numbering of columns in the DataFrame
num_cols = len(df_new.columns)

# Calculating optimal grid size
num_rows = int(np.ceil(num_cols / 3))  # 3 columns per row

# Creating subplots dynamically
fig, axes = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 4 * num_rows))

# Flattening the axes array for easy iteration
axes = axes.flatten()

# Plotting histograms
for i, col in enumerate(df_new.columns):
```

```
    axes[i].hist(df_new[col], bins=20, edgecolor='black')
    axes[i].set_title(f"Histogram of {col}")

# Hiding unused subplots
for i in range(num_cols, len(axes)):  # Hiding any extra empty subplots
    fig.delaxes(axes[i])

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



8. Create a scatter plot of crime rate versus price.

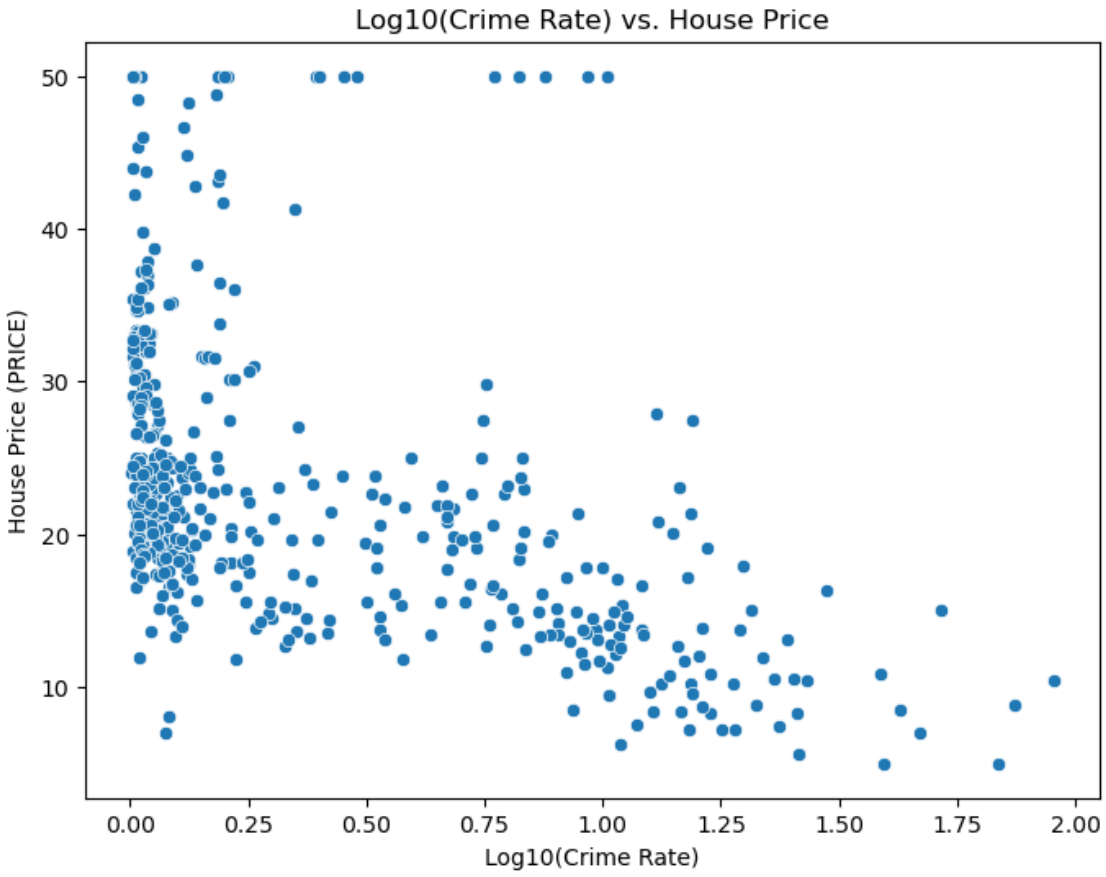
[307]:
```
# Plotting scatter plot of crime rate vs. price
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['CRIM'], y=df['PRICE'])
plt.xlabel("Crime Rate (CRIM)")
plt.ylabel("House Price (PRICE)")
plt.title("Crime Rate vs. House Price")

# Displaying a plot
plt.show()
```



9. Plot log10(crime) versus price.

[308]:
```
# Plotting scatter plot of log10(crime) vs. price
plt.figure(figsize=(8, 6))

# Adding 1 to avoid log(0) errors
sns.scatterplot(x=np.log10(df['CRIM'] + 1), y=df['PRICE'])
plt.xlabel("Log10(Crime Rate)")
plt.ylabel("House Price (PRICE)")
```

```
plt.title("Log10(Crime Rate) vs. House Price")

# Displaying the plot
plt.show()
```



10. Calculate some useful statistics, such as mean rooms per dwelling, median age, mean distances to five Boston employment centers, and the percentage of houses with a low price ($< \$20{,}000$).

[309]:
```
# Calculating required statistics
mean_rooms_per_dwelling = df['RM'].mean()
median_age = df['AGE'].median()
mean_distance_to_5_boston_emp_centers = df['DIS'].mean()

# Converting to percentage
percentage_of_houses_with_low_price = (df['PRICE'] < 20).mean() * 100

print(f"Mean rooms per dwelling: {mean_rooms_per_dwelling}")
print(f"Median age: {median_age}")
```

```
print(f"Mean distances to five Boston employment centers:␣
  ↪{mean_distance_to_5_boston_emp_centers}")
print(f"Percentage of houses with a low price (<$20,000):␣
  ↪{percentage_of_houses_with_low_price}")
```

```
Mean rooms per dwelling: 6.284634387351779
Median age: 77.5
Mean distances to five Boston employment centers: 3.795042687747036
Percentage of houses with a low price (<$20,000): 41.50197628458498
```

2. The Data Wrangling Workshop: Activity 4.01, page 233

1. Load the necessary libraries.

[310]:
```
# Loding the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Read the adult income dataset from the following URL: https://packt.live/2N9lRUU. Will download and load dataset from local folder instead.

[311]:
```
import pandas as pd

# Loading dataset from local folder instead of link because there is a␣
  ↪restriction to access file using the  URL: https://packt.live/2N9lRUU
file_path = r"C:\Users\maxim\OneDrive\Desktop\BU\DSC 540\adult_income_data.csv"
df = pd.read_csv(file_path, sep=",", skipinitialspace=True, header=None)

# Assigning correct column names
df.columns = ["age", "workclass", "fnlwgt", "education", "education-num",
              "marital-status", "occupation", "relationship", "sex",
              "capital-gain", "capital-loss", "hours-per-week",
              "native-country", "income"]

# Checking the first 5 rows to get insight and structure of the dataset
print(df.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     sex  capital-gain  \
0       Never-married       Adm-clerical  Not-in-family    Male          2174
1  Married-civ-spouse    Exec-managerial        Husband    Male             0
2            Divorced  Handlers-cleaners  Not-in-family    Male             0
```

```
3  Married-civ-spouse  Handlers-cleaners        Husband    Male             0
4  Married-civ-spouse     Prof-specialty           Wife  Female             0

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

3. Create a script that will read a text file line by line.

[312]:
```
# Opening the file and reading lines
with open(file_path, 'r') as file:
    lines = file.readlines()
    for i, line in enumerate(lines[:10]):  # Printing only the first 10 lines
        print(f"Line {i+1}: {line.strip()}")
```

```
Line 1: 39, State-gov,77516, Bachelors,13, Never-married, Adm-clerical, Not-in-
family, Male,2174,0,40, United-States, <=50K
Line 2: 50, Self-emp-not-inc,83311, Bachelors,13, Married-civ-spouse, Exec-
managerial, Husband, Male,0,0,13, United-States, <=50K
Line 3: 38, Private,215646, HS-grad,9, Divorced, Handlers-cleaners, Not-in-
family, Male,0,0,40, United-States, <=50K
Line 4: 53, Private,234721, 11th,7, Married-civ-spouse, Handlers-cleaners,
Husband, Male,0,0,40, United-States, <=50K
Line 5: 28, Private,338409, Bachelors,13, Married-civ-spouse, Prof-specialty,
Wife, Female,0,0,40, Cuba, <=50K
Line 6: 37, Private,284582, Masters,14, Married-civ-spouse, Exec-managerial,
Wife, Female,0,0,40, United-States, <=50K
Line 7: 49, Private,160187, 9th,5, Married-spouse-absent, Other-service, Not-in-
family, Female,0,0,16, Jamaica, <=50K
Line 8: 52, Self-emp-not-inc,209642, HS-grad,9, Married-civ-spouse, Exec-
managerial, Husband, Male,0,0,45, United-States, >50K
Line 9: 31, Private,45781, Masters,14, Never-married, Prof-specialty, Not-in-
family, Female,14084,0,50, United-States, >50K
Line 10: 42, Private,159449, Bachelors,13, Married-civ-spouse, Exec-managerial,
Husband, Male,5178,0,40, United-States, >50K
```

4. Add a name of Income for the response variable to the dataset.

[313]:
```
# Since in question 2 I already defined column N as income, I will just rename␣
 ↪the 'income' column to 'Income'
df.rename(columns={'income': 'Income'}, inplace=True)

# Confirming if the 'income' column indeed renamed to 'Income'
print(df.columns)
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'sex', 'capital-gain',
```

```
       'capital-loss', 'hours-per-week', 'native-country', 'Income'],
      dtype='object')
```

5. Find the missing values.

[314]:
```
# Checking for missing values in the dataset using boolean method isnull()
missing_values = df.isnull().sum()

# Displaying results
print(missing_values)
```

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

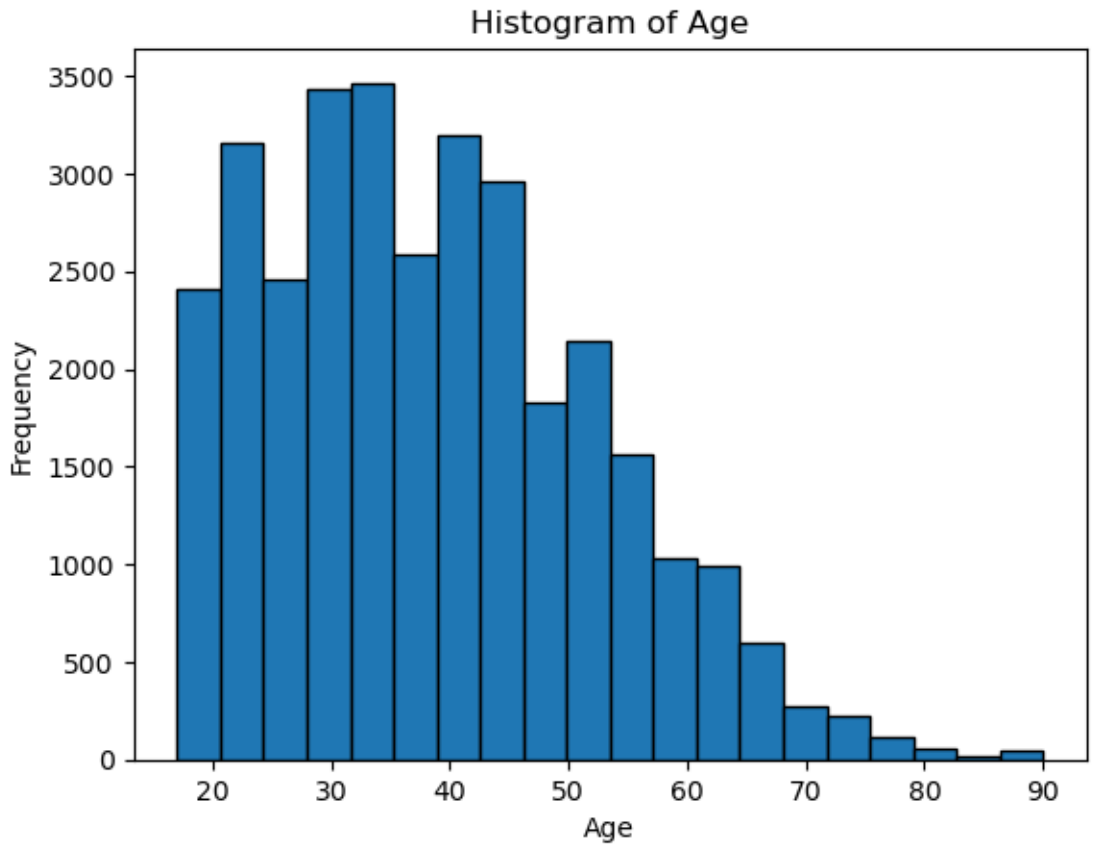
6. Create a DataFrame with only age, education, and occupation by using sub setting.

[315]:
```
# Subsetting the DataFrame to include only 'age', 'education', and 'occupation'
subset_df = df[['age', 'education', 'occupation']]

# Displaying the new DataFrame with only age, education, and occupation
print(subset_df.head(5))
```

```
   age  education         occupation
0   39  Bachelors       Adm-clerical
1   50  Bachelors    Exec-managerial
2   38    HS-grad  Handlers-cleaners
3   53       11th  Handlers-cleaners
4   28  Bachelors     Prof-specialty
```

7. Plot a histogram of age with a bin size of 20.

[316]:
```
# Plotting a histogram of age with bin size of 20
plt.hist(df['age'], bins=20, edgecolor='black')
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



8. Create a function to strip the whitespace characters.

[317]:
```
# Function to strip whitespace from strings
def strip_whitespace(value):
    if isinstance(value, str):
        return value.strip()
    return value

# Applying the function to each column with string values
df = df.apply(lambda col: col.apply(strip_whitespace) if col.dtype == 'object'␣
 ↪else col)

# Checking the first few rows to confirm
print(df.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     sex  capital-gain  \
0       Never-married       Adm-clerical  Not-in-family    Male          2174
1  Married-civ-spouse    Exec-managerial        Husband    Male             0
2            Divorced  Handlers-cleaners  Not-in-family    Male             0
3  Married-civ-spouse  Handlers-cleaners        Husband    Male             0
4  Married-civ-spouse     Prof-specialty           Wife  Female             0

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

9. Use the apply method to apply this function to all the columns with string values, create a new column, copy the values from this new column to the old column, and drop the new column.

[318]:
```
# Applying the strip whitespace function to all columns with string values
def strip_whitespace(value):
    if isinstance(value, str):  # Only apply to string columns
        return value.strip()
    return value

# Create cleaned columns for all string columns
for col in df.select_dtypes(include=['object']).columns:
    # Create a new column with cleaned values
    df[f'{col}_cleaned'] = df[col].apply(strip_whitespace)

# Now, copy the cleaned columns back to the original columns and drop the new␣
 ↪ones
for col in df.select_dtypes(include=['object']).columns:
    if f'{col}_cleaned' in df.columns:
        df[col] = df[f'{col}_cleaned']  # Copy cleaned values back to the␣
 ↪original column
        df.drop(f'{col}_cleaned', axis=1, inplace=True)  # Drop the temporary␣
 ↪cleaned column

# Display the cleaned DataFrame
print(df.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     sex  capital-gain  \
0       Never-married       Adm-clerical  Not-in-family    Male          2174
1  Married-civ-spouse    Exec-managerial        Husband    Male             0
2            Divorced  Handlers-cleaners  Not-in-family    Male             0
3  Married-civ-spouse  Handlers-cleaners        Husband    Male             0
4  Married-civ-spouse     Prof-specialty           Wife  Female             0

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

10. Find the number of people who are aged between 30 and 50.

[319]:
```
# Filtering rows where age is between 30 and 50
age_range = df[(df['age'] >= 30) & (df['age'] <= 50)]

# Counting the number of people in this age range
num_people = age_range.shape[0]
print(f"Number of people aged between 30 and 50: {num_people:,}")
```

```
Number of people aged between 30 and 50: 16,390
```

11. Group the records based on age and education to find how the mean age is distributed.

[320]:
```
# Grouping by education and calculating the mean age
mean_age_by_education = df.groupby('education')['age'].mean().round().
  ↪astype(int).reset_index()
mean_age_by_education.rename(columns={'age': 'Mean Age'}, inplace=True)

# Displaying the result with formatted output
print(mean_age_by_education)
```

```
       education  Mean Age
0           10th        37
1           11th        32
2           12th        32
3        1st-4th        46
4        5th-6th        43
5        7th-8th        48
6            9th        41
7     Assoc-acdm        37
8      Assoc-voc        39
9      Bachelors        39
10     Doctorate        48
11       HS-grad        39
12       Masters        44
```

```
13      Preschool        43
14   Prof-school        45
15  Some-college        36
```

12. Group by occupation and show the summary statistics of age. Find which profession has the oldest workers on average and which profession has its largest share of the workforce above the 75th percentile.

[321]:
```
# Grouping by occupation and showing summary statistics of age
age_summary = df.groupby('occupation')['age'].describe().round(2)

# Finding which profession has the oldest workers on average
mean_age_by_occupation = df.groupby('occupation')['age'].mean().round(2)
oldest_profession = mean_age_by_occupation.idxmax()
oldest_profession_mean_age = mean_age_by_occupation.max()

# Finding the 75th percentile of age
age_75th_percentile = df['age'].quantile(0.75)

# Finding the percentage of workers in each occupation above the 75th percentile
above_75th_percentile = df[df['age'] > age_75th_percentile]
share_above_75th = above_75th_percentile.groupby('occupation').size() / df.
 ↪groupby('occupation').size()

# Finding the profession with the largest share of workers above the 75th␣
 ↪percentile
largest_share_above_75th = share_above_75th.idxmax()
largest_share_percentage = share_above_75th.max()

# Displaying the results
print("Summary statistics of age grouped by occupation (occupation to 2 decimal␣
 ↪places):")
print(age_summary)
print(f"\nProfession with the oldest workers on average: {oldest_profession}␣
 ↪(Average Age: {oldest_profession_mean_age})")
print(f"Profession with the largest share of the workforce above the 75th␣
 ↪percentile: {largest_share_above_75th} (Share: {largest_share_percentage:.
 ↪2f})")
```

```
Summary statistics of age grouped by occupation (occupation to 2 decimal
places):
                   count   mean    std   min   25%   50%   75%   max
occupation
?                 1843.0  40.88  20.34  17.0  21.0  35.0  61.0  90.0
Adm-clerical      3770.0  36.96  13.36  17.0  26.0  35.0  46.0  90.0
Armed-Forces         9.0  30.22   8.09  23.0  24.0  29.0  34.0  46.0
Craft-repair      4099.0  39.03  11.61  17.0  30.0  38.0  47.0  90.0
Exec-managerial   4066.0  42.17  11.97  17.0  33.0  41.0  50.0  90.0
```

```
Farming-fishing     994.0  41.21  15.07  17.0  29.0  39.0  52.0  90.0
Handlers-cleaners  1370.0  32.17  12.37  17.0  23.0  29.0  39.0  90.0
Machine-op-inspct  2002.0  37.72  12.07  17.0  28.0  36.0  46.0  90.0
Other-service      3295.0  34.95  14.52  17.0  22.0  32.0  45.0  90.0
Priv-house-serv     149.0  41.72  18.63  17.0  24.0  40.0  57.0  81.0
Prof-specialty     4140.0  40.52  12.02  17.0  31.0  40.0  48.0  90.0
Protective-serv     649.0  38.95  12.82  17.0  29.0  36.0  47.0  90.0
Sales              3650.0  37.35  14.19  17.0  25.0  35.0  47.0  90.0
Tech-support        928.0  37.02  11.32  17.0  28.0  36.0  44.0  73.0
Transport-moving   1597.0  40.20  12.45  17.0  30.0  39.0  49.0  90.0

Profession with the oldest workers on average: Exec-managerial (Average Age:
42.17)
Profession with the largest share of the workforce above the 75th percentile: ?
(Share: 0.38)
```

13. Use subset and groupBy to find the outliers.

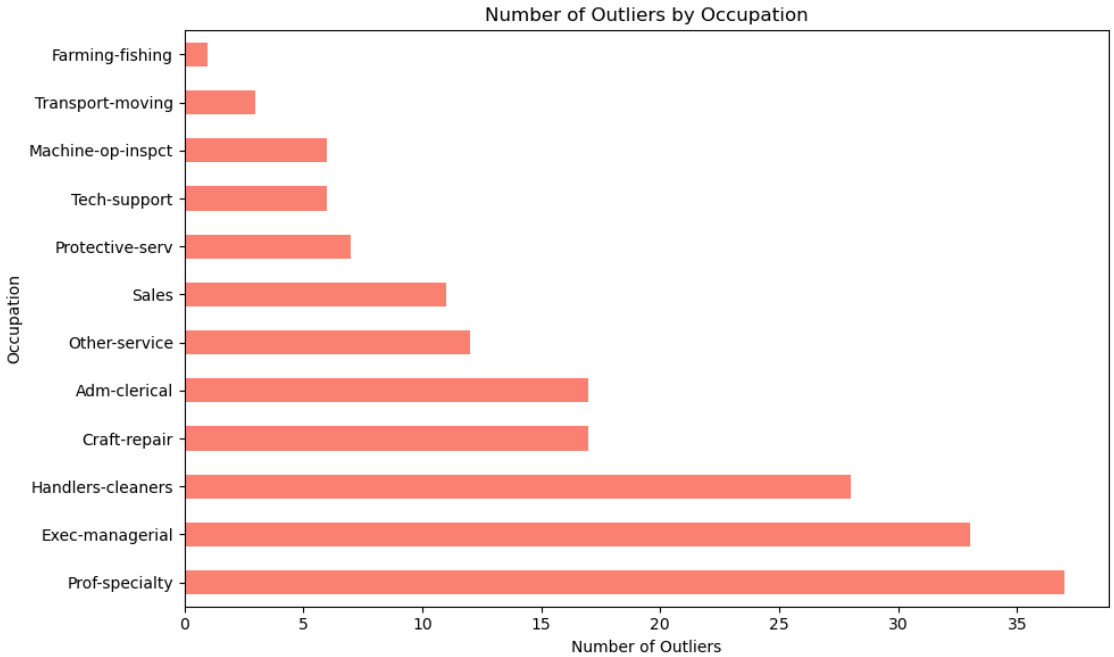
[322]:
```python
# Grouping by occupation and finding the IQR for each group
Q1 = df.groupby('occupation')['age'].quantile(0.25)
Q3 = df.groupby('occupation')['age'].quantile(0.75)
IQR = Q3 - Q1

# Identifying outliers
outliers_lower = Q1 - 1.5 * IQR
outliers_upper = Q3 + 1.5 * IQR

# Applying the IQR thresholds to each row in the dataframe using the occupation␣
 ↪as the key
df['lower_bound'] = df['occupation'].map(outliers_lower)
df['upper_bound'] = df['occupation'].map(outliers_upper)

# Finding outliers
outliers = df[(df['age'] < df['lower_bound']) | (df['age'] > df['upper_bound'])]

# Displaying the outliers
print("Outliers based on IQR:")
print(outliers[['age', 'occupation']])
```

```
Outliers based on IQR:
       age         occupation
74      79     Prof-specialty
100     76    Exec-managerial
144     70       Tech-support
222     90      Other-service
324     76       Craft-repair
…      …                  …
31625   74  Handlers-cleaners
```

```
32277    90        Adm-clerical
32341    74        Craft-repair
32367    90     Protective-serv
32459    85     Exec-managerial

[178 rows x 2 columns]
```

14. Plot the outlier values on a bar chart. It should look something like this:

[323]:
```
# Counting the number of outliers for each occupation
outliers_count = outliers['occupation'].value_counts()

# Sorting the counts in descending order for better visualization
outliers_count = outliers_count.sort_values(ascending=False)

# Creating a horizontal bar plot
plt.figure(figsize=(10, 6))
outliers_count.plot(kind='barh', color='salmon')

# Adding titles and labels
plt.title('Number of Outliers by Occupation')
plt.ylabel('Occupation')
plt.xlabel('Number of Outliers')

# Showing the plot
plt.tight_layout()
plt.show()
```



15. Merge the two DataFrames using common keys to drop duplicate values.

[324]:
```
# Only having one DataFrame and will keep only the necessary columns
df_cleaned = df[['age', 'workclass', 'occupation', 'education']]  # Selecting␣
 ↪necessary columns

# Removing any duplicate rows (if any exist)
df_cleaned = df_cleaned.drop_duplicates()

# Displaying the cleaned DataFrame
print(df_cleaned.head())
```

```
   age          workclass          occupation  education
0   39          State-gov       Adm-clerical  Bachelors
1   50  Self-emp-not-inc    Exec-managerial  Bachelors
2   38            Private  Handlers-cleaners    HS-grad
3   53            Private  Handlers-cleaners       11th
4   28            Private     Prof-specialty  Bachelors
```

Creating second dataframe and merging with the above one.

[325]:
```
# Original DataFrame (df)
data = {
    'age': [39, 50, 38, 53, 28],
    'workclass': ['State-gov', 'Self-emp-not-inc', 'Private', 'Private',␣
 ↪'Private'],
    'occupation': ['Adm-clerical', 'Exec-managerial', 'Handlers-cleaners',␣
 ↪'Handlers-cleaners', 'Prof-specialty'],
    'education': ['Bachelors', 'Bachelors', 'HS-grad', '11th', 'Bachelors']
}
df = pd.DataFrame(data)

# Mocking second DataFrame (df2) with a common key, let's say 'age' for merging
data2 = {
    'age': [39, 50, 60, 28],
    'salary': [50000, 60000, 70000, 80000]
}
df2 = pd.DataFrame(data2)

# Now, merging df and df2 on the common column 'age'
merged_df = pd.merge(df, df2, on='age', how='inner')  # You can use 'left',␣
 ↪'right', or 'outer' depending on needs

# Formating the 'salary' column with commas for better readability
merged_df['salary'] = merged_df['salary'].apply(lambda x: f"{x:,}")

# Displaying the merged DataFrame
```

```
print(merged_df)
```

```
   age         workclass       occupation  education  salary
0   39         State-gov     Adm-clerical  Bachelors  50,000
1   50  Self-emp-not-inc  Exec-managerial  Bachelors  60,000
2   28           Private   Prof-specialty  Bachelors  80,000
```

3. Create a series and practice basic arithmetic steps

Creating series 1

a. Series 1 = 7.3, -2.5, 3.4, 1.5

i. Index = 'a', 'c', 'd', 'e'

[326]:
```
# Creating Series 1 with specified values and index
series1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])

# Displaying Series 1
series1
```

[326]:
```
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
```

Creating series 2

b. Series 2 = -2.1, 3.6, -1.5, 4, 3.1

i. Index = 'a', 'c', 'e', 'f', 'g'

[327]:
```
# Creating Series 2 with specified values and index
series2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1], index=['a', 'c', 'e', 'f', 'g'])

# Displaying Series 2
series2
```

[327]:
```
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

c. Add Series 1 and Series 2 together and print the results

[328]:
```
# Adding Series 1 and Series 2
added_series = series1 + series2

# Displaying the result of addition
```

```
added_series
```

[328]:
```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

d. Subtract Series 1 from Series 2 and print the results

[329]:
```
# Subtracting Series 1 from Series 2
subtracted_series = series2 - series1

# Displaying the result of subtraction
subtracted_series
```

[329]:
```
a   -9.4
c    6.1
d    NaN
e   -3.0
f    NaN
g    NaN
dtype: float64
```