😊 Useful Methods and Functions

In this lab, we will learn about some useful methods and functions built into Pandas.

1. apply() method

The apply() method is used to apply a function along an axis of the DataFrame (rows or columns). It allows for element-wise operations or aggregation based on custom logic. It is very flexible and works with both built-in and user-defined functions.

2. apply() with a function

When apply() is used with a function, the function is applied to each row or column, depending on the axis specified. This is useful for performing row-wise or column-wise computations that cannot be done using vectorized operations.

3. apply() with a lambda expression

You can pass a lambda function to apply() for inline, one-off operations. This is especially handy when the logic is simple and you do not want to define a separate function. It enhances the readability and conciseness of the code.

4. apply() on multiple columns

apply() can also work on multiple columns if you set the axis to rows and write a function that accepts a row object. This allows you to use the values from several columns to compute a new result for each row.

5. describe()

The describe() method provides summary statistics of the numerical columns in a DataFrame. This includes count, mean, standard deviation, minimum, maximum, and quartile values. It helps in getting a quick overview of the data distribution.

6. sort_values()

This method is used to sort a DataFrame or Series based on the values in one or more columns. It supports ascending or descending sorting and can handle missing values appropriately.

7. corr()

The corr() method computes pairwise correlation of columns, typically for numerical columns. It returns a correlation matrix that indicates how strongly the values in one column are related to those in another.

8. idxmin and idxmax

These methods return the index of the minimum and maximum values, respectively, in a Series or along a DataFrame axis. They help identify the position where the lowest or highest value occurs.

9. value_counts

This function counts the unique values in a Series and returns them in descending order of frequency. It is helpful in understanding the distribution of categorical or discrete values.

10. replace

The replace() method is used to substitute values in a Series or DataFrame. It can replace single values, multiple values, or use dictionaries and regular expressions for more complex replacements.

---

11. unique and nunique

- unique() returns an array of unique values in a Series.
- nunique() returns the count of unique values.
  These methods are useful for exploring distinct entries and understanding data variety.

12. map

The map() function is used to substitute each value in a Series according to a dictionary, function, or Series. It is commonly used for data transformation or formatting individual elements.

13. duplicated and drop_duplicates

- duplicated() identifies duplicate rows based on specified columns.
- drop_duplicates() removes duplicate rows and keeps the first (or last) occurrence based on the provided settings.
  These are useful for data cleaning and ensuring data integrity.

14. between

The between() method checks whether each element in a Series falls within a specified range, inclusive of the boundaries. It returns a boolean Series indicating the result for each element.

15. sample

This method is used to randomly select a specified number or fraction of rows from a DataFrame or elements from a Series. It is helpful for testing, data exploration, or creating train-test splits.

16. nlargest

The nlargest() method returns the top N rows with the largest values in a specified column. It is efficient and useful when you want to retrieve high-ranking entries based on a numerical column.

😄 To begin with the Lab

1. We start by importing Pandas and NumPy in our Jupyter Notebook, then we create a data frame for the tips CSV file and display the first few lines using the head command.

[4]:
```
import pandas as pd
import numpy as np
```

[5]:
```
df = pd.read_csv('tips.csv')
```

[6]:
```
df.head()
```

[6]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251

2. This code defines a function called last_four that takes a number, converts it to a string, and returns the last four characters.
3. It then applies this function to the CC Number column of the DataFrame to extract the last four digits of each credit card number.
4. A new column called last_four is created in the DataFrame to store these values.
5. This is useful for displaying or storing only partial credit card information for privacy or security purposes.

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   total_bill        244 non-null    float64
 1   tip               244 non-null    float64
 2   sex               244 non-null    object
 3   smoker            244 non-null    object
 4   day               244 non-null    object
 5   time              244 non-null    object
 6   size              244 non-null    int64
 7   price_per_person  244 non-null    float64
 8   Payer Name        244 non-null    object
 9   CC Number         244 non-null    int64
 10  Payment ID        244 non-null    object
dtypes: float64(3), int64(2), object(6)
memory usage: 21.1+ KB
```

[9]:
```
def last_four(num):
    return str(num)[-4:]
```

[10]:
```
df['CC Number'][0]
```

[10]:
```
3560325168603410
```

[11]:
```
last_four(3560325168603410)
```

[11]:
```
'3410'
```

[12]:
```
df['last_four'] = df['CC Number'].apply(last_four)
```

[13]:
```
df.head()
```

[13]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221

6. This code calculates the average (mean) of the total_bill column in the DataFrame.
7. It then defines a function named yelp that categorizes the bill amount into price ranges: bills less than $10 are labeled with a single dollar sign ($), bills between $10 and $30 with two dollar signs ($$), and bills $30 or more with three dollar signs ($$$).
8. This function is applied to the total_bill column using the .apply() method, and the results are stored in a new column called Expensive, effectively classifying each bill based on cost level.

[15]:
```
df['total_bill'].mean()
```

[15]:
```
19.78594262295082
```

[16]:
```
def yelp(price):
    if price < 10:
        return '$'
    elif price >= 10 and price < 30:
        return '$$'
    else:
        return '$$$'
```

[17]:
```
df['Expensive'] = df['total_bill'].apply(yelp)
```

9. This code first defines a regular function simple that doubles a given number.
10. Then, it shows a more concise version using a **lambda function**, which is an anonymous function that does the same thing in a single line.
11. Finally, it uses a lambda function to calculate an 18% tip for each value in the total_bill column of the DataFrame by multiplying each bill by 0.18.
12. This is done using the .apply() method, which applies the lambda to each entry in the column, often for quick, inline calculations or transformations.

[20]:
```
def simple(num):
    return num*2
```

[21]:
```
lambda num: num*2
```

[21]:
```
<function __main__.<lambda>(num)>
```

[22]:
```
df['total_bill'].apply(lambda bill:bill*0.18)
```

[22]:
```
0      3.0582
1      1.8612
2      3.7818
3      4.2624
4      4.4262
        ...
239    5.2254
240    4.8924
241    4.0806
242    3.2076
243    3.3804
Name: total_bill, Length: 244, dtype: float64
```

13. This code compares the performance of two methods for applying a custom function (quality) to a pandas DataFrame.
14. The quality function labels a tip as "Generous" if it is more than 25% of the total bill, otherwise it labels it as "Other".
15. The first method uses .apply() with axis=1, which applies the function row-by-row, while the second uses NumPy's vectorize() to apply it in a more optimized way over entire columns.
16. The timeit module is used to measure how long each method takes when executed 1000 times.
17. This comparison helps identify which approach is more efficient for larger datasets.

[24]:
```
df.head()
```

[24]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$

[25]:
```
def quality(total_bill,tip):
    if tip/total_bill  > 0.25:
        return "Generous"
    else:
        return "Other"
```

[26]:
```
df['Tip Quality'] = df[['total_bill','tip']].apply(lambda df: quality(df['total_bill'],df['tip']),axis=1)
```

[27]:
```
df.head()
```

[27]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Other
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Other
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$	Other
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$	Other
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$	Other

[28]:
```
import numpy as np
```

[29]:
```
df['Tip Quality'] = np.vectorize(quality)(df['total_bill'], df['tip'])
```

[30]:
```
df.head()
```

[30]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Other
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Other
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$	Other
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$	Other
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$	Other

[32]:
```
import timeit

# code snippet to be executed only once
setup = '''
import numpy as np
import pandas as pd
df = pd.read_csv('tips.csv')
def quality(total_bill,tip):
    if tip/total_bill  > 0.25:
        return "Generous"
    else:
        return "Other"
'''

# code snippet whose execution time is to be measured
stmt_one = '''
df['Tip Quality'] = df[['total_bill','tip']].apply(lambda df: quality(df['total_bill'],df['tip']),axis=1)
'''

stmt_two = '''
df['Tip Quality'] = np.vectorize(quality)(df['total_bill'], df['tip'])
'''
```

[33]:
```
timeit.timeit(setup = setup,
              stmt = stmt_one,
              number = 1000)
```

[33]:
```
1.7549014000005627
```

[34]:
```
timeit.timeit(setup = setup,
              stmt = stmt_two,
              number = 1000)
```

[34]:
```
0.18351369999982126
```

18. The first line, `df.describe()`, generates summary statistics for all numeric columns in the DataFrame, such as count, mean, standard deviation, minimum, maximum, and quartile values.
19. The second line, `df.describe().transpose()`, transposes this summary so that each statistic becomes a row and each column becomes a column header.
20. This makes it easier to read and compare statistics across multiple columns.

```
[37]: df.describe()
```

[37]:

	total_bill	tip	size	price_per_person	CC Number
count	244.000000	244.000000	244.000000	244.000000	2.440000e+02
mean	19.785943	2.998279	2.569672	7.888197	2.563496e+15
std	8.902412	1.383638	0.951100	2.914234	2.369340e+15
min	3.070000	1.000000	1.000000	2.880000	6.040679e+10
25%	13.347500	2.000000	2.000000	5.800000	3.040731e+13
50%	17.795000	2.900000	2.000000	7.255000	3.525318e+15
75%	24.127500	3.562500	3.000000	9.390000	4.553675e+15
max	50.810000	10.000000	6.000000	20.270000	6.596454e+15

```
[38]: df.describe().transpose()
```

[38]:

	count	mean	std	min	25%	50%	75%	max
total_bill	244.0	1.978594e+01	8.902412e+00	3.070000e+00	1.334750e+01	1.779500e+01	2.412750e+01	5.081000e+01
tip	244.0	2.998279e+00	1.383638e+00	1.000000e+00	2.000000e+00	2.900000e+00	3.562500e+00	1.000000e+01
size	244.0	2.569672e+00	9.510998e-01	1.000000e+00	2.000000e+00	2.000000e+00	3.000000e+00	6.000000e+00
price_per_person	244.0	7.888197e+00	2.914234e+00	2.880000e+00	5.800000e+00	7.255000e+00	9.390000e+00	2.027000e+01
CC Number	244.0	2.563496e+15	2.369340e+15	6.040679e+10	3.040731e+13	3.525318e+15	4.553675e+15	6.596454e+15

21. The first line, df.sort_values('tip'), sorts the DataFrame in ascending order based on the values in the 'tip' column. This helps identify the lowest and highest tips easily.
22. The second line, df.sort_values(['tip','size']), sorts the DataFrame first by 'tip' and then by 'size' as a secondary sort key.
23. This is useful when you want to break ties in the 'tip' column by sorting based on another column, such as the group size.

```
[40]: df.sort_values('tip')
```

[40]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
67	3.07	1.00	Female	Yes	Sat	Dinner	1	3.07	Tiffany Brock	4359488526995267	Sat3455	5267	$	Generous
236	12.60	1.00	Male	Yes	Sat	Dinner	2	6.30	Matthew Myers	3543676378973965	Sat5032	3965	$$	Other
92	5.75	1.00	Female	Yes	Fri	Dinner	2	2.88	Leah Ramirez	3508911676966392	Fri3780	6392	$	Other
111	7.25	1.00	Female	No	Sat	Dinner	1	7.25	Terri Jones	3559221007826887	Sat4801	6887	$	Other
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Other
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
141	34.30	6.70	Male	No	Thur	Lunch	6	5.72	Steven Carlson	3526515703718508	Thur1025	8508	$$$	Other
59	48.27	6.73	Male	No	Sat	Dinner	4	12.07	Brian Ortiz	6596453823950595	Sat8139	0595	$$$	Other
23	39.42	7.58	Male	No	Sat	Dinner	4	9.86	Lance Peterson	3542584061609808	Sat239	9808	$$$	Other

[41]:
```
# Helpful if you want to reorder after a sort
# https://stackoverflow.com/questions/13148429/how-to-change-the-order-of-dataframe-columns
df.sort_values(['tip','size'])
```

[41]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
67	3.07	1.00	Female	Yes	Sat	Dinner	1	3.07	Tiffany Brock	4359488526995267	Sat3455	5267	$	Generous
111	7.25	1.00	Female	No	Sat	Dinner	1	7.25	Terri Jones	3559221007826887	Sat4801	6887	$	Other
92	5.75	1.00	Female	Yes	Fri	Dinner	2	2.88	Leah Ramirez	3508911676966392	Fri3780	6392	$	Other
236	12.60	1.00	Male	Yes	Sat	Dinner	2	6.30	Matthew Myers	3543676378973965	Sat5032	3965	$$	Other
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Other
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
141	34.30	6.70	Male	No	Thur	Lunch	6	5.72	Steven Carlson	3526515703718508	Thur1025	8508	$$$	Other
59	48.27	6.73	Male	No	Sat	Dinner	4	12.07	Brian Ortiz	6596453823950595	Sat8139	0595	$$$	Other
23	39.42	7.58	Male	No	Sat	Dinner	4	9.86	Lance Peterson	3542584061609808	Sat239	9808	$$$	Other
212	48.33	9.00	Male	No	Sat	Dinner	4	12.08	Alex Williamson	676218815212	Sat4590	5212	$$$	Other
170	50.81	10.00	Male	Yes	Sat	Dinner	3	16.94	Gregory Clark	5473850968388236	Sat1954	8236	$$$	Other

244 rows × 14 columns

24. The first line, df.select_dtypes(include='number').corr(), computes the correlation matrix for all numeric columns in the DataFrame, helping identify relationships between numerical variables.
25. The second line, df[['total_bill','tip']].corr(), specifically calculates the correlation between the 'total_bill' and 'tip' columns, showing how closely the tip amount is related to the total bill.

[43]:
```
df.select_dtypes(include='number').corr()
```

[43]:

	total_bill	tip	size	price_per_person	CC Number
total_bill	1.000000	0.675734	0.598315	0.647554	0.104576
tip	0.675734	1.000000	0.489299	0.347405	0.110857
size	0.598315	0.489299	1.000000	-0.175359	-0.030239
price_per_person	0.647554	0.347405	-0.175359	1.000000	0.135240
CC Number	0.104576	0.110857	-0.030239	0.135240	1.000000

[44]:
```
df[['total_bill','tip']].corr()
```

[44]:

	total_bill	tip
total_bill	1.000000	0.675734
tip	0.675734	1.000000

26. The code you're using is working with a DataFrame (df) in pandas, a Python library. df.head() shows the first 5 rows of the DataFrame, giving you a quick preview of the data. df['total_bill'].max() finds the highest value in the "total_bill" column, while df['total_bill'].idxmax() gives you the index (or row) where that highest value is located.
27. Similarly, df['total_bill'].idxmin() finds the index of the lowest value in the "total_bill" column. Finally, df.iloc[67] and df.iloc[170] are used to retrieve the data from the 68th and 171st rows of the DataFrame, respectively.

[46]:
```
df.head()
```

[46]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Other
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Other
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$	Other
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$	Other
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$	Other

```
[47]: df['total_bill'].max()
```

```
[47]: 50.81
```

```
[48]: df['total_bill'].idxmax()
```

```
[48]: 170
```

```
[49]: df['total_bill'].idxmin()
```

```
[49]: 67
```

```
[50]: df.iloc[67]
```

```
[50]: total_bill                    3.07
      tip                           1.0
      sex                        Female
      smoker                        Yes
      day                           Sat
      time                       Dinner
      size                            1
      price_per_person             3.07
      Payer Name          Tiffany Brock
      CC Number         4359488526995267
      Payment ID                Sat3455
      last_four                    5267
      Expensive                       $
      Tip Quality              Generous
      Name: 67, dtype: object
```

```
[51]: df.iloc[170]
```

```
[51]: total_bill                   50.81
      tip                           10.0
      sex                           Male
      smoker                         Yes
      day                            Sat
      time                        Dinner
      size                             3
      price_per_person             16.94
      Payer Name          Gregory Clark
      CC Number         5473850968388236
      Payment ID                Sat1954
      last_four                    8236
      Expensive                     $$$
      Tip Quality                 Other
      Name: 170, dtype: object
```

28. The code you're using shows the first 5 rows of the DataFrame with df.head() to give you an overview of the data.
29. Then, df['sex'].value_counts() counts how many times each unique value appears in the "sex" column, such as how many entries are labeled as "male" or "female," and provides a summary of the distribution of those values in the dataset.

```
[53]: df.head()
```

[53]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Other
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Other
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$	Other
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$	Other
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$	Other

```
[54]: df['sex'].value_counts()
```

```
[54]: sex
      Male      157
      Female     87
      Name: count, dtype: int64
```

30. The code first shows the first 5 rows of the DataFrame using df.head().
31. Then, it attempts to replace the value "Other" with "Ok" in the "Tip Quality" column, but without making permanent changes yet.
32. To actually make the change, the code reassigns the modified column back to df['Tip Quality'], replacing "Other" with "Ok."
33. Finally, it displays the first 5 rows again, showing the updated data with the changes applied.

```
[56]: df.head()
```

[56]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Other
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Other
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$	Other
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$	Other
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$	Other

```
[57]: df['Tip Quality'].replace(to_replace='Other',value='Ok')
```

```
[57]: 0      Ok
      1      Ok
      2      Ok
      3      Ok
      4      Ok
             ..
      239    Ok
      240    Ok
      241    Ok
      242    Ok
      243    Ok
      Name: Tip Quality, Length: 244, dtype: object
```

```
[58]: df['Tip Quality'] = df['Tip Quality'].replace(to_replace='Other',value='Ok')
```

```
[59]: df.head()
```

[59]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Ok
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Ok
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$	Ok
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$	Ok
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$	Ok

34. The code is using pandas methods to explore the unique values in specific columns of the DataFrame. df['size'].unique() returns an array of all unique values found in the "size" column, helping you see what distinct values are present. df['size'].nunique() counts how many unique values are in the "size" column, giving you the total number of distinct entries.

35. Similarly, df['time'].unique() returns an array of unique values in the "time" column, showing the distinct time-related entries. These methods are useful for quickly understanding the variety of data in a column and checking for uniqueness.

```
[61]: df['size'].unique()
```

```
[61]: array([2, 3, 4, 1, 6, 5], dtype=int64)
```

```
[62]: df['size'].nunique()
```

```
[62]: 6
```

```
[63]: df['time'].unique()
```

```
[63]: array(['Dinner', 'Lunch'], dtype=object)
```

36. The code defines a dictionary my_map that maps "Dinner" to "D" and "Lunch" to "L". Then, it uses the map() function on the "time" column of the DataFrame to replace the values "Dinner" with "D" and "Lunch" with "L", based on the dictionary.
37. The map() function is used to perform element-wise transformations in a column, replacing each value according to the specified mapping.
38. Any value in the "time" column that isn't in the dictionary will be replaced with NaN.
39. Finally, df.head() displays the first 5 rows of the DataFrame, showing the updated "time" column with the new values.

```
[65]: my_map = {'Dinner':'D','Lunch':'L'}
```

```
[66]: df['time'].map(my_map)
```

```
[66]: 0      D
      1      D
      2      D
      3      D
      4      D
            ..
      239    D
      240    D
      241    D
      242    D
      243    D
      Name: time, Length: 244, dtype: object
```

```
[67]: df.head()
```

[67]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Ok
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Ok
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458	1322	$$	Ok
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	5994	$$	Ok
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251	7221	$$	Ok

40. The code is used to identify and remove duplicate rows in a pandas DataFrame. df.duplicated() checks for duplicate rows in the DataFrame and returns a boolean Series where True indicates a duplicated row (after the first occurrence).
41. In the simple_df DataFrame, created with values [1, 2, 2], simple_df.duplicated() finds that the second row (with value 2) is a duplicate, returning True for that row.

42. The drop_duplicates() method is then used to remove any duplicate rows from simple_df, keeping only the first occurrence of each unique value.
43. These methods help you detect and clean up duplicate entries in your data.

```
[69]: # Returns True for the 1st instance of a duplicated row
      df.duplicated()
```

```
[69]: 0      False
      1      False
      2      False
      3      False
      4      False
             ...
      239    False
      240    False
      241    False
      242    False
      243    False
      Length: 244, dtype: bool
```

```
[70]: simple_df = pd.DataFrame([1,2,2],['a','b','c'])
```

```
[71]: simple_df
```

[71]:

	0
a	1
b	2
c	2

```
[72]: simple_df.duplicated()
```

```
[72]: a    False
      b    False
      c     True
      dtype: bool
```

```
[73]: simple_df.drop_duplicates()
```

[73]:

	0
a	1
b	2

44. The code checks which rows in the "total_bill" column have values between 10 and 20, including both 10 and 20. The `between()` method is used to create a Boolean Series that marks `True` for values within that range and `False` otherwise.
45. Then, this boolean Series is used to filter the DataFrame, so only the rows where "total_bill" falls between 10 and 20 are returned.
46. This method is a simple and readable way to select data within a specific numeric range.

```
[76]: df['total_bill'].between(10, 20, inclusive='both')
```

```
[76]: 0       True
      1       True
      2      False
      3      False
      4      False
             ...
      239    False
      240    False
      241    False
      242     True
      243     True
      Name: total_bill, Length: 244, dtype: bool
```

```
[80]: df[df['total_bill'].between(10,20,inclusive='both')]
```

[80]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	3410	$$	Ok
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	9230	$$	Ok
8	15.04	1.96	Male	No	Sun	Dinner	2	7.52	Joseph Mcdonald	3522866365840377	Sun6820	0377	$$	Ok
9	14.78	3.23	Male	No	Sun	Dinner	2	7.39	Jerome Abbott	3532124519049786	Sun3775	9786	$$	Ok
10	10.27	1.71	Male	No	Sun	Dinner	2	5.14	William Riley	566287581219	Sun2546	1219	$$	Ok
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
234	15.53	3.00	Male	Yes	Sat	Dinner	2	7.76	Tracy Douglas	4097938155941930	Sat7220	1930	$$	Ok
235	10.07	1.25	Male	No	Sat	Dinner	2	5.04	Sean Gonzalez	3534021246117605	Sat4615	7605	$$	Ok
236	12.60	1.00	Male	Yes	Sat	Dinner	2	6.30	Matthew Myers	3543676378973965	Sat5032	3965	$$	Ok

47. The code is used to randomly select rows from a DataFrame. The first line, df.sample(5), randomly picks 5 rows from the DataFrame.
48. The second line, df.sample(frac=0.1), randomly selects 10% of the total rows from the DataFrame.
49. The sample() method is used to take random samples from the data, either by specifying the exact number of rows with n or by specifying a fraction of the total with frac.
50. This is helpful when you want to work with a smaller, random subset of your data for testing or analysis.

```
[82]: df.sample(5)
```

[82]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
191	19.81	4.19	Female	Yes	Thur	Lunch	2	9.90	Kristy Boyd	4317015327600068	Thur967	0068	$$	Ok
204	20.53	4.00	Male	Yes	Thur	Lunch	4	5.13	Scott Kim	3570611756827620	Thur2160	7620	$$	Ok
207	38.73	3.00	Male	Yes	Sat	Dinner	4	9.68	Ricky Ramirez	347817964484033	Sat4505	4033	$$$	Ok
212	48.33	9.00	Male	No	Sat	Dinner	4	12.08	Alex Williamson	676218815212	Sat4590	5212	$$$	Ok
183	23.17	6.50	Male	Yes	Sun	Dinner	4	5.79	Dr. Michael James	4718501859162	Sun6059	9162	$$	Generous

```
[84]: df.sample(frac=0.1)
```

[84]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
142	41.19	5.00	Male	No	Thur	Lunch	5	8.24	Eric Andrews	4356531761046453	Thur3621	6453	$$$	Ok
172	7.25	5.15	Male	Yes	Sun	Dinner	2	3.62	Larry White	30432617123103	Sun9209	3103	$	Generous
103	22.42	3.48	Female	Yes	Sat	Dinner	2	11.21	Kathleen Hawkins	348009865484721	Sat1015	4721	$$	Ok
77	27.20	4.00	Male	No	Thur	Lunch	4	6.80	John Davis	30344778738589	Thur4924	8589	$$	Ok
74	14.73	2.20	Female	No	Sat	Dinner	2	7.36	Ashley Harris	501828723483	Sat6548	3483	$$	Ok
67	3.07	1.00	Female	Yes	Sat	Dinner	1	3.07	Tiffany Brock	4359488526995267	Sat3455	5267	$	Generous
178	9.60	4.00	Female	Yes	Sun	Dinner	2	4.80	Melanie Gray	4211808859168	Sun4598	9168	$	Generous
82	10.07	1.83	Female	No	Thur	Lunch	1	10.07	Julie Moody	630413282843	Thur4909	2843	$$	Ok
7	26.88	3.12	Male	No	Sun	Dinner	4	6.72	Robert Buck	3514785077705092	Sun8157	5092	$$	Ok

51. The code `df.nlargest(10, 'tip')` returns the top 10 rows from the DataFrame `df` that have the highest values in the "tip" column.

52. The `nlargest()` method is used to quickly find the rows with the largest values in a specific column.
53. In this case, it helps you identify the 10 biggest tips recorded in the dataset, which can be useful for analysis like spotting high-value transactions or outliers.

[86]:
```
df.nlargest(10,'tip')
```

[86]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID	last_four	Expensive	Tip Quality
170	50.81	10.00	Male	Yes	Sat	Dinner	3	16.94	Gregory Clark	5473850968388236	Sat1954	8236	$$$	Ok
212	48.33	9.00	Male	No	Sat	Dinner	4	12.08	Alex Williamson	676218815212	Sat4590	5212	$$$	Ok
23	39.42	7.58	Male	No	Sat	Dinner	4	9.86	Lance Peterson	3542584061609808	Sat239	9808	$$$	Ok
59	48.27	6.73	Male	No	Sat	Dinner	4	12.07	Brian Ortiz	6596453823950595	Sat8139	0595	$$$	Ok
141	34.30	6.70	Male	No	Thur	Lunch	6	5.72	Steven Carlson	3526515703718508	Thur1025	8508	$$$	Ok
183	23.17	6.50	Male	Yes	Sun	Dinner	4	5.79	Dr. Michael James	4718501859162	Sun6059	9162	$$	Generous
214	28.17	6.50	Female	Yes	Sat	Dinner	3	9.39	Marissa Jackson	4922302538691962	Sat3374	1962	$$	Ok
47	32.40	6.00	Male	No	Sun	Dinner	4	8.10	James Barnes	3552002592874186	Sun9677	4186	$$$	Ok
239	29.03	5.92	Male	No	Sat	Dinner	3	9.68	Michael Avila	5296068606052842	Sat2657	2842	$$	Ok
88	24.71	5.85	Male	No	Thur	Lunch	2	12.36	Roger Taylor	4410248629955	Thur9003	9955	$$	Ok