367.dhl\_b = pd.read\_excel('W:\Team Spaces\AWS Cloud Logistics\Logistics\Data Extract for Tableau\APAC\Snow Spares Direct WBR\Weekly reports\AMZ\_Web\_Services\_Performance\_by\_EDD.xlsx', sheet\_name = 'Shipment\_Detail', header = 1)

Explan:-

1. dhl\_b :-as a pandas DataFrame.

2.pd.read\_excel():-This function is from the **pandas** library (imported as pd). It's used to read an Excel file into a pandas DataFrame.

3. File Path:- 'W:\Team Spaces\AWS Cloud Logistics\Logistics\Data Extract for Tableau\APAC\Snow Spares Direct WBR\Weekly reports\AMZ\_Web\_Services\_Performance\_by\_EDD.xlsx':

4. sheet\_name = 'Shipment\_Detail'

5. header = 1:- By default, pandas assumes the first row (index 0) contains headers. In this case, you are telling it to skip the first row and treat the second one as the header row.

368. dhl = dhl\_b.copy()

Explan:-

1.  dhl\_b: This is the pandas DataFrame that you loaded in the previous line using pd.read\_excel(). It contains the data from the Excel sheet 'Shipment\_Detail'.

 **.**copy():

 This is a method used in pandas to create a **shallow copy** of the DataFrame.

 A **shallow copy** means that dhl will contain the same data as dhl\_b, but they will be two separate objects in memory.

 Modifying dhl will not affect dhl\_b and vice versa. If you did not use .copy(), and instead did something like dhl = dhl\_b, then dhl would just be a reference to the same object as dhl\_b. Any changes to one would affect the other.

# Mistake 369. #dhl = dhl[dhl['Shipper Company Name'] == 'HVYE SOLUTIONS SINGAPORE PTE LTD']#

Explan:-

This is a **filtering operation**.

**370. dhl['Startclock Date'] = dhl['Startclock Date'].dt.date**

Explan:-

 dhl['Startclock Date']:

* This accesses the 'Startclock Date' column in the dhl DataFrame. It's assumed that this column contains datetime objects (i.e., data with both date and time values).

 .dt:

* The .dt accessor is a pandas attribute that allows you to work with datetime-like data (such as timestamps or datetime columns). It provides various attributes and methods to extract or modify parts of the datetime data (like the year, month, day, hour, minute, etc.).

 .date:

* The .date is an attribute that extracts just the date part of each value in the 'Startclock Date' column, which means it strips off the time portion of the datetime objects.
* For example, if the value in the column was 2025-01-30 12:34:56, it would be converted to 2025-01-30, keeping only the date (year, month, day), and discarding the time (hours, minutes, seconds).

dhl\_b = pd.read\_excel('W:\Team Spaces\AWS Cloud Logistics\Logistics\Data Extract for Tableau\APAC\Snow Spares Direct WBR\Weekly reports\AMZ\_Web\_Services\_Performance\_by\_EDD.xlsx', sheet\_name = 'Shipment\_Detail', header = 1) please this code needs an explanation

dhl\_b = pd.read\_excel('W:\Team Spaces\AWS Cloud Logistics\Logistics\Data Extract for Tableau\APAC\Snow Spares Direct WBR\Weekly reports\AMZ\_Web\_Services\_Performance\_by\_EDD.xlsx', sheet\_name = 'Shipment\_Detail', header = 1)

**Explanation:**

1. **pd.read\_excel()**:
   * This function is from the **pandas** library (imported as pd). It's used to read an Excel file into a pandas DataFrame.
2. **File Path**:
   * 'W:\Team Spaces\AWS Cloud Logistics\Logistics\Data Extract for Tableau\APAC\Snow Spares Direct WBR\Weekly reports\AMZ\_Web\_Services\_Performance\_by\_EDD.xlsx':
     + This is the file path of the Excel file you're trying to read. It seems to be stored in a network location or shared drive (W: drive).
3. **sheet\_name = 'Shipment\_Detail'**:
   * This specifies the name of the sheet you want to read from the Excel file. In this case, you're reading the sheet called 'Shipment\_Detail'.
4. **header = 1**:
   * This tells pandas that the second row (index 1) in the sheet contains the column names (headers). By default, pandas assumes the first row (index 0) contains headers. In this case, you are telling it to skip the first row and treat the second one as the header row.

**Summary:**

This code reads an Excel file located at the specified path, loads the sheet named 'Shipment\_Detail', and assumes the second row contains the headers (column names). The resulting data is stored in the variable dhl\_b as a pandas DataFrame.

Does that help clarify things?

dhl = dhl\_b.copy() please this code needs an explanation

dhl = dhl\_b.copy()

**Explanation:**

* **dhl\_b**: This is the pandas DataFrame that you loaded in the previous line using pd.read\_excel(). It contains the data from the Excel sheet 'Shipment\_Detail'.
* **.copy()**:
  + This is a method used in pandas to create a **shallow copy** of the DataFrame.
  + A **shallow copy** means that dhl will contain the same data as dhl\_b, but they will be two separate objects in memory.
  + Modifying dhl will not affect dhl\_b and vice versa. If you did not use .copy(), and instead did something like dhl = dhl\_b, then dhl would just be a reference to the same object as dhl\_b. Any changes to one would affect the other.

**Summary:**

This line of code creates a copy of the dhl\_b DataFrame and stores it in the variable dhl. This is useful if you want to work with the data without altering the original DataFrame (dhl\_b).

Let me know if that clears things up!

#dhl = dhl[dhl['Shipper Company Name'] == 'HVYE SOLUTIONS SINGAPORE PTE LTD'] please this code needs an explanation

dhl = dhl[dhl['Shipper Company Name'] == 'HVYE SOLUTIONS SINGAPORE PTE LTD']

**Breakdown:**

* **dhl**:
  + This is the pandas DataFrame you're working with. In the previous line, you created a copy of dhl\_b and assigned it to dhl.
* **dhl['Shipper Company Name']**:
  + This accesses the **'Shipper Company Name'** column in the dhl DataFrame. It returns a pandas Series containing the values from that column.
* **== 'HVYE SOLUTIONS SINGAPORE PTE LTD'**:
  + This is a condition that checks if each value in the 'Shipper Company Name' column is equal to the string 'HVYE SOLUTIONS SINGAPORE PTE LTD'.
  + The result of this comparison is a **boolean Series**: each entry will be True if the condition is met, or False if it's not.
* **dhl[dhl['Shipper Company Name'] == 'HVYE SOLUTIONS SINGAPORE PTE LTD']**:
  + This is a **filtering operation**. It filters the dhl DataFrame and only keeps the rows where the condition ('Shipper Company Name' == 'HVYE SOLUTIONS SINGAPORE PTE LTD') is True.
  + So, it returns a new DataFrame with just the rows where the 'Shipper Company Name' is 'HVYE SOLUTIONS SINGAPORE PTE LTD'.
* **Assignment (dhl = ...)**:
  + The filtered DataFrame is then assigned back to dhl, so now dhl contains only the rows where the 'Shipper Company Name' is 'HVYE SOLUTIONS SINGAPORE PTE LTD'.

**Summary:**

This line of code filters the dhl DataFrame to include only the rows where the 'Shipper Company Name' is exactly 'HVYE SOLUTIONS SINGAPORE PTE LTD', and assigns the filtered data back to the variable dhl.

dhl['Startclock Date'] = dhl['Startclock Date'].dt.date dhl['Startclock Date'] = dhl['Startclock Date'].dt.date

dhl['Startclock Date'] = dhl['Startclock Date'].dt.date

**Breakdown:**

* **dhl['Startclock Date']**:
  + This accesses the **'Startclock Date'** column of the dhl DataFrame. Assuming this column contains date and/or time values, it's likely a datetime column (i.e., a column of datetime objects).
* **.dt**:
  + This is a **pandas accessor** for datetime-like columns. It allows you to extract specific components of datetime values, like the year, month, day, hour, minute, etc.
* **.date**:
  + This is a **datetime method** that extracts only the **date part** of each datetime value, discarding the time component. It converts each datetime object in the 'Startclock Date' column into a date object, which only includes the year, month, and day, but not the time.
* **Assignment (dhl['Startclock Date'] = ...)**:
  + This assigns the modified 'Startclock Date' column (which now contains only the date part, with the time removed) back to the same column in the dhl DataFrame.

**Effect:**

This line of code modifies the 'Startclock Date' column in the dhl DataFrame, stripping off the time part of the datetime values, so only the date (year, month, day) remains.

**Why you might use this:**

This is typically done when you only care about the date and not the specific time of day, especially if you want to

\dhl['Startclock Date'] = dhl['Startclock Date'].dt.date please this code needs an explanation

dhl['Startclock Date'] = dhl['Startclock Date'].dt.date

**Breakdown:**

* **dhl['Startclock Date']**:
  + This accesses the **'Startclock Date'** column in the dhl DataFrame. It's assumed that this column contains **datetime objects** (i.e., data with both date and time values).
* **.dt**:
  + The .dt accessor is a **pandas attribute** that allows you to work with **datetime-like data** (such as timestamps or datetime columns). It provides various attributes and methods to extract or modify parts of the datetime data (like the year, month, day, hour, minute, etc.).
* **.date**:
  + The .date is an attribute that extracts just the **date part** of each value in the 'Startclock Date' column, which means it strips off the **time portion** of the datetime objects.
  + For example, if the value in the column was 2025-01-30 12:34:56, it would be converted to 2025-01-30, keeping only the date (year, month, day), and discarding the time (hours, minutes, seconds).
* **Assignment (dhl['Startclock Date'] = ...)**:
  + This updates the 'Startclock Date' column in the dhl DataFrame with the **new** version of the dates where the time part has been removed.

**What happens:**

* The 'Startclock Date' column, originally containing datetime values (like 2025-01-30 12:34:56), will now contain only the date part (like 2025-01-30).

**Why this is useful:**

* Often, when you're dealing with datetime data, you might want to ignore the time of day and only focus on the date itself.
* This operation is useful for situations like:
  + When you want to group by or compare just the **date** without being concerned about the specific time.
  + When the time portion of the datetime is irrelevant to your analysis.

**Example:**

If the Startclock Date column originally had values like:

2025-01-30 12:34:56

2025-01-30 08:15:30

2025-01-29 17:05:10

After applying this line of code, it would be transformed to:

2025-01-30

2025-01-30

2025-01-29

371. c1 = (dhl['Billing Shipper Reference'].str.contains('DHL',na=False))

**Explanation:**

1. **dhl['Billing Shipper Reference']**: This refers to the column 'Billing Shipper Reference' in the DataFrame dhl. So, you are working with the data from this specific column.
2. **.str.contains('DHL', na=False)**:
   * The .str accessor is used to apply string operations to a whole series (or column).
   * The contains('DHL') method checks if the string 'DHL' is present in each entry of the 'Billing Shipper Reference' column.
   * The na=False part ensures that if there are any NaN (Not a Number, i.e., missing or undefined) values in that column, they are treated as False. If you don’t include na=False, NaN values would be treated as NaN, which might affect your results.
3. **The result (c1)**:
   * The expression will return a Boolean series (True or False values).
   * For each row in the 'Billing Shipper Reference' column, if the string 'DHL' is found, the result will be True. If it's not found, it will be False.

**Example:**

Suppose the 'Billing Shipper Reference' column contains these values:

* 'DHL123'
* 'ABC456'
* 'DHL789'
* None (which is a NaN)

After running the code:

c1 = dhl['Billing Shipper Reference'].str.contains('DHL', na=False)

c1 will contain:

* True (because 'DHL123' contains 'DHL')
* False (because 'ABC456' does not contain 'DHL')
* True (because 'DHL789' contains 'DHL')
* False (because None/NaN is treated as False)

The result in c1 will be:

[True, False, True, False]

Assume we have the following DataFrame dhl:

import pandas as pd

# Sample DataFrame

data = {

data

'Billing Shipper Reference': ['DHL123', 'ABC456', 'DHL789', None, 'DHL456']

}

dhl = pd.DataFrame(data)

}

dhl = pd.DataFrame(data

}

dhl =

print(dhl)

`

This would output

css

Copy

Billing Shipper Reference

0 DHL123

1 ABC456

2 DHL789

3 None

4 DHL456

Now, when

c1 = dhl['Billing Shipper Reference'].str.contains('DHL', na=False)

This checks each value in the 'Billing Shipper Reference' col'DHL' an

**Explanation of what happens**:

* For the first row ('DHL123'), it'DHL',True.
* For the second row ('ABC456'),'DHL', soFalse.
* For the third row ('DHL789'),'DHL', so theTrue.
* For the fourth row (None), it'sNaN (missingna=False, itFalse.
* For the fifth row ('DHL456'), it contains'DHL', so the resultTrue.

Thus, after running the code, the value of c1 wi

python

Copy

0 True

1 False

2 True

3 False

4 True

Name: Billing Shipper Reference, dtype:

Name: Billing Shipper Reference, dtype: bo

Name: Billing Shipper Reference, dtype:

Name: Billing Shipper

Name: Billing Ship

Name: Billing S

Name: Billin

Name: Bil

Name:

bool

This Boolean series is telling you whether the string 'DHL' appears in'Billing Shipper Reference' column. The `TrueTrue and `FalseFalse values correspond'DHL' was found or not

**Full Code**

python

Copy

import pandas as pd

# Sample DataFrame

data = {

data = {

data = {

'Billing Shipper Reference': ['DHL123', 'ABC456', 'DHL789', None, 'DHL456']

}

dhl = pd.DataFrame(data)

}

dhl = pd.DataFrame(data

}

dhl =

# Check if 'DHL' appears in the 'Billing Shipper Reference' column

c1 = dhl[

c1 = dhl[

c1 = d

'Billing Shipper Reference'].str.contains('DHL', na=False)

# Output the result

print(c1)

**Output:**

0 True

1 False

2 True

3 False

4 True

Name: Billing Shipper Reference, dtype: bool

This c1 can be used for filtering or further operations, depending on your needs. For example, if you wanted to filter out the rows where 'DHL' appears, you could do:

python

Copy

dhl\_filtered = dhl[c1]

print(dhl\_filtered)

Which would return:

Copy

Billing Shipper Reference

0 DHL123

2 DHL789

4 DHL456

372. # c2 = (dhl['Billing Account Number'] == 309321147)

c2 = (dhl['Billing Account Number'] == 309321147)

**Explanation of the Code:**

1. **dhl['Billing Account Number']**:
   * This refers to the column 'Billing Account Number' in the dhl DataFrame. You’re accessing the values in this column, which could contain numbers, strings, or other data types.
2. **==**:
   * This is the **equality operator**. It checks whether the values in 'Billing Account Number' are equal to the number 309321147.
   * The result of this comparison will be a **Boolean** value for each entry: True if the value matches 309321147, and False if it doesn't.
3. **309321147**:
   * This is the value you are comparing each entry in the 'Billing Account Number' column against. In this case, it’s a specific integer.
4. **c2 =**:
   * This assigns the result of the comparison (the Boolean values) to a new variable c2.
   * c2 will be a **Boolean series** of True and False values, indicating which rows have the 'Billing Account Number' equal to 309321147.

**What Happens in Detail:**

* The code will check **each row** of the 'Billing Account Number' column to see if the value is **equal** to 309321147.
* It will return a **True** for rows where the value is 309321147 and **False** for rows where it is not.

**Example:**

Let’s say your DataFrame dhl looks like this:

python

Copy

import pandas as pd

# Sample DataFrame

data = {

'Billing Account Number': [309321147, 123456789, 309321147, 987654321]

}

dhl = pd.DataFrame(data)

print(dhl)

**Output of the DataFrame:**

javascript

Copy

Billing Account Number

0 309321147

1 123456789

2 309321147

3 987654321

Now, when we apply the code:

python

Copy

c2 = (dhl['Billing Account Number'] == 309321147)

print(c2)

**Output of c2:**

yaml

Copy

0 True

1 False

2 True

3 False

Name: Billing Account Number, dtype: bool

**Explanation of the Output:**

* For the first row, the value 309321147 is equal to 309321147, so the result is True.
* For the second row, the value 123456789 is not equal to 309321147, so the result is False.
* For the third row, the value 309321147 is again equal to 309321147, so the result is True.
* For the fourth row, the value 987654321 is not equal to 309321147, so the result is False.

**Use Case:**

* **Filtering**: You can use the Boolean series (c2) to filter the rows in the DataFrame where the 'Billing Account Number' is 309321147. For example:

filtered\_df = dhl[c2]

print(filtered\_df)

**Output:**

Billing Account Number

0 309321147

2 309321147

373.c2 = (dhl['Billing Account Number'].isin(['309321147', '309991023']))

c2 = (dhl['Billing Account Number'].isin(['309321147', '309991023']))

**Explanation of Each Part:**

1. **dhl['Billing Account Number']**:
   * This is accessing the column 'Billing Account Number' in the DataFrame dhl. You are working with the values in this column, which could be numbers or strings.
2. **.isin()**:
   * The .isin() method is a **vectorized** function in pandas. It checks whether each element in the given series (in this case, 'Billing Account Number') is **present** in the list or sequence provided.
   * It returns a Boolean series where each entry is True if the value in the 'Billing Account Number' column matches any of the values in the provided list (e.g., ['309321147', '309991023']), and False if it does not.
3. **['309321147', '309991023']**:
   * This is a **list** containing the values you're checking against.
   * You are looking to see if each value in the 'Billing Account Number' column matches either '309321147' **or** '309991023'.
   * It's important to note that the values in the list are **strings**. If the 'Billing Account Number' column contains numbers, you'll need to ensure that they match the correct type. If the column contains integers, the values in the list should be integers as well.
4. **c2 =**:
   * This assigns the result of the .isin() method (which is a Boolean series) to the variable c2.
   * c2 will be a Boolean series indicating whether each value in the 'Billing Account Number' column is in the list ['309321147', '309991023'].

**What This Code Does:**

The code is checking whether each entry in the 'Billing Account Number' column is either 309321147 or 309991023. It returns a series of True or False values for each row in the column, depending on whether the value is in the list of values you're checking against.

**Example:**

Let’s assume your dhl DataFrame looks like this:

python

Copy

import pandas as pd

# Sample DataFrame

data = {

'Billing Account Number': ['309321147', '123456789', '309991023', '987654321', '309321147']

}

dhl = pd.DataFrame(data)

print(dhl)

**Output of the DataFrame:**

javascript

Copy

Billing Account Number

0 309321147

1 123456789

2 309991023

3 987654321

4 309321147

Now, when we run the following code:

python

Copy

c2 = (dhl['Billing Account Number'].isin(['309321147', '309991023']))

print(c2)

**Output of c2:**

yaml

Copy

0 True

1 False

2 True

3 False

4 True

Name: Billing Account Number, dtype: bool

**Explanation of the Output:**

* For the first row, the value '309321147' is in the list ['309321147', '309991023'], so the result is True.
* For the second row, the value '123456789' is not in the list, so the result is False.
* For the third row, the value '309991023' is in the list, so the result is True.
* For the fourth row, the value '987654321' is not in the list, so the result is False.
* For the fifth row, the value '309321147' is in the list, so the result is True.

**Use Case Example:**

You can use this Boolean series to **filter** the rows of the DataFrame. For example, if you only want to keep the rows where the 'Billing Account Number' is either '309321147' or '309991023', you can do:

python

Copy

filtered\_df = dhl[c2]

print(filtered\_df)

**Output:**

javascript

Copy

Billing Account Number

0 309321147

2 309991023

4 309321147

Now, filtered\_df contains only the rows where 'Billing Account Number' is either '309321147' or '309991023'.

**Why Use .isin()?**

The .isin() method is very useful when you need to check if a value exists in a list, and you don’t want to write multiple OR conditions manually. It simplifies the code and makes it more readable when you're comparing against a list of potential values.

**374.dhl = dhl[c1|c2].reset\_index(drop=True)**

dhl = dhl[c1 | c2].reset\_index(drop=True)

Explanation of Each Part:

1. dhl[c1 | c2]:
   * This part is filtering the dhl DataFrame based on the Boolean conditions c1 and c2.
   * c1 and c2 are Boolean series, which are the result of previous conditions (for example, they could be checks for whether certain values are present in specific columns, like True/False results).
   * The | (bitwise OR) operator is used to combine these two Boolean series (c1 and c2). Here's what happens:
     + For each row, if either c1 or c2 is True, the row is kept.
     + If both c1 and c2 are False for a row, that row is excluded.

In essence, dhl[c1 | c2] selects all rows where either condition c1 or c2 is True.

1. .reset\_index(drop=True):
   * After filtering the DataFrame using dhl[c1 | c2], the indices of the resulting rows might no longer be in a consecutive order. They may contain gaps because rows have been filtered out.
   * .reset\_index() is used to reindex the resulting DataFrame. This means the indices will be reset to start from 0 and increment by 1.
   * The drop=True argument ensures that the old indices are not added as a column. If you don't use drop=True, the old index values will be added as a new column called 'index'.
   * So, drop=True is used to avoid keeping the old index and just getting a clean DataFrame with a new index.
2. dhl =:
   * This assigns the resulting DataFrame (after applying the filter and resetting the index) back to the dhl variable. This updates dhl with the new filtered and reindexed DataFrame.

What This Code Does:

* Filters the dhl DataFrame to keep only the rows where either c1 or c2 is True.
* Resets the index of the filtered DataFrame so that it starts from 0 and increments by 1, removing any gaps in the index.
* Reassigns the resulting DataFrame back to the dhl variable, updating it.

Example:

Let’s say you have the following DataFrame dhl and conditions c1 and c2:

import pandas as pd

# Sample DataFrame

data = {

'Billing Account Number': [309321147, 123456789, 309991023, 987654321, 309321147],

'Amount': [100, 200, 150, 250, 100]

}

dhl = pd.DataFrame(data)

# Sample conditions (Booleans for example)

c1 = (dhl['Billing Account Number'] == 309321147) # Match '309321147'

c2 = (dhl['Amount'] > 150) # Amount greater than 150

print("Original DataFrame:")

print(dhl)

Output (Original DataFrame):

Billing Account Number Amount

0 309321147 100

1 123456789 200

2 309991023 150

3 987654321 250

4 309321147 100

Now, apply the code:

dhl = dhl[c1 | c2].reset\_index(drop=True)

print("Filtered and Reindexed DataFrame:")

print(dhl)

Output (After Filtering and Resetting Index):

Billing Account Number Amount

0 309321147 200

1 123456789 200

2 987654321 250

3 309321147 100

Explanation:

* Filtering (c1 | c2):
  + Rows where c1 is True (where 'Billing Account Number' == 309321147) or c2 is True (where 'Amount' > 150) are kept.
  + Specifically:
    - The first row (309321147, 100) is excluded because the amount is not greater than 150.
    - The second row (123456789, 200) is included because the amount is greater than 150.
    - The third row (309991023, 150) is excluded because the amount is not greater than 150.
    - The fourth row (987654321, 250) is included because the amount is greater than 150.
    - The fifth row (309321147, 100) is excluded because the amount is not greater than 150.
* Resetting the Index:
  + After filtering, the index of the rows is no longer in a simple, consecutive order.
  + reset\_index(drop=True) reorders the index so that it starts from 0 and increments by 1 without keeping the old index.

**375.dhl['mot'] = 'AIR'**

1. **Result**:
   * After running this line of code, the DataFrame dhl will have a new column 'mot' (or it will update the existing one) where each row has the value 'AIR'.

**Example:**

Let’s look at an example DataFrame dhl before and after applying the code:

python

Copy

import pandas as pd

# Sample DataFrame

data = {

'Billing Account Number': [309321147, 123456789, 309991023],

'Amount': [100, 200, 150]

}

dhl = pd.DataFrame(data)

print("Original DataFrame:")

print(dhl)

# Adding or updating the 'mot' column with 'AIR'

dhl['mot'] = 'AIR'

print("\nDataFrame After Adding 'mot' Column:")

print(dhl)

**Output:**

**Original DataFrame:**

Billing Account Number Amount

0 309321147 100

1 123456789 200

2 309991023 150

**After Adding the 'mot' Column:**

Billing Account Number Amount mot

0 309321147 100 AIR

1 123456789 200 AIR

2 309991023 150 AIR

**Key Points:**

* The **'mot' column** is added to the DataFrame.
* Every row in the **'mot' column** gets the value 'AIR'.

**Possible Use Case:**

* You might use this to add a default value for the **mode of transportation** (such as 'AIR') for all rows, which could later be updated with specific values if necessary.
* For example, you could later update 'mot' for certain rows to 'SEA' or 'GROUND' depending on other conditions in the DataFrame.

**Conclusion:**

* This line of code creates or updates the 'mot' column in the dhl DataFrame and assigns the string 'AIR' to every entry in that column.

please an example to 'SEA''GROUND'

Certainly! Let's create an example where we initially set the 'mot' column to'AIR' a'SEA' or `''GROUND' based on

**Exa**

Let’s assume we have a DataFrame that tracks shipments, and the mode of transport ('mot') i'AIR' by default'SEA' for'GROUND' for o**Amount** or another

import pandas as pd

# Sample DataFrame

data = {

data = {

data = {

data

'Billing Account Number': [309321147, 123456789, 309991023, 987654321],

'Amount': [100, 500, 300, 150]

}

}

# Create the DataFrame

dhl = pd.DataFrame(data)

dhl = pd.DataFrame(dat

dhl = pd.DataFram

dhl = pd.Dat

dhl =

# Initially setting 'mot' to 'AIR' for all rows

dhl[

dhl[

'mot'] = 'AIR'

prin

p

print("Original DataFrame:")

prin

print(dhl)

# Now let's update the 'mot' column based on conditions:

# If the Amount is greater than 400, set 'mot' to 'SEA'

dhl.loc[dhl[

dhl.loc

'Amount'] > 400, 'mot'] = 'SEA'

# If the Amount is between 150 and 400 (inclusive), set 'mot' to 'GROUND'

dhl.loc[(dhl[

dhl.loc[(dhl[

dhl.loc[(

dhl.

dh

'Amount'] <= 400) & (dhl['Amount'] >= 150), 'mot'] = 'GROUND'

# Print the updated DataFrame

print("\nUpdated DataFrame with Mode of Transport:")

print(dhl)

**Outp**

**Original DataFrame:**

Billing Account Number Amount mot

0 309321147 100 AIR

1 123456789 500 AIR

2 309991023 300 AIR

3 987654321 150 AIR

**Updated DataFrame with Mode of Transport:**

mathematica

Copy

Billing Account Number Amount mot

0 309321147 100 AIR

1 123456789 500 SEA

2 309991023 300 GROUND

3 987654321 150 GROUND

**E**

1. **Initial State**:
   * The 'mot' colu'AIR' for all rows.
2. **First Condition**:
   * We update the 'mot' column to 'SEA' for rows where the 'Amount' is greater than 400.
   * So, the second row (Amount = 500) gets its 'mot' value updated to 'SEA'.
3. **Second Condition**:
   * We update the 'mot' column to 'GROUND' for rows where the 'Amount' is between 150 and 400 (inclusive).
   * This updates the third (Amount = 300) and fourth (Amount = 150) rows to have 'GROUND' as their mode of transport.
4. **Final Result**:
   * The updated DataFrame shows that the mode of transport is 'AIR', 'SEA', or 'GROUND' depending on the amount value.

**Customizing the Logic:**

You can adjust the conditions based on different factors, such as:

* **Distance** (e.g., shipments over a certain distance might be by sea or ground).
* **Priority** (e.g., express shipments might always be air).
* **Customer type** (e.g., certain customers always use a specific mode of transport).

Top of Form

376.dhl['department'] = 'Spares

**Explanation of Each Part:**

1. **dhl**:
   * This is your pandas DataFrame. It's assumed to already exist and contain data.
   * The DataFrame could represent various types of data, such as shipment records, product details, etc.
2. **['department']**:
   * This refers to the column named 'department' in the dhl DataFrame.
   * If the 'department' column already exists in the DataFrame, this code will update all of its values.
   * If the 'department' column does not exist, this line will create a new column named 'department' and assign values to it.
3. **= 'Spares'**:
   * This part assigns the value 'Spares' to the 'department' column.
   * The key point here is that **all rows** in the 'department' column will be set to the string 'Spares'.
   * It does not matter what other values might be in the DataFrame; every entry in the 'department' column will get the value 'Spares'.
4. **Result**:
   * After this line runs, the DataFrame will have a new column 'department' where every row is labeled with the string 'Spares'.

**Example:**

import pandas as pd

# Sample DataFrame

data = {

'Billing Account Number': [309321147, 123456789, 309991023],

'Amount': [100, 200, 150]

}

dhl = pd.DataFrame(data)

print("Original DataFrame:")

print(dhl)

# Adding the 'department' column with the value 'Spares' for every row

dhl['department'] = 'Spares'

print("\nDataFrame After Adding 'department' Column:")

print(dhl)

**Output:**

**Original DataFrame:**

Billing Account Number Amount

0 309321147 100

1 123456789 200

2 309991023 150

**After Adding the 'department' Column:**

Billing Account Number Amount department

0 309321147 100 Spares

1 123456789 200 Spares

2 309991023 150 Spares

**Key Points:**

* **Adding a Column**: If the 'department' column didn’t exist before, it would now be created with the value 'Spares' for every row.
* **Updating Existing Column**: If'department' column alr'Spares'.
* Bottom of Form

**377.dhl['Program'] = 'Spares'**

**Explanation of Each Part:**

1. **dhl**:
   * This refers to your pandas DataFrame. It's assumed that you have a DataFrame already named dhl, which contains data such as shipments, products, or any other related information.
2. **['Program']**:
   * This specifies the column **'Program'** in the dhl DataFrame.
   * If the column 'Program' already exists in the DataFrame, this code will update **all** values in that column.
   * If the column 'Program' doesn't exist in the DataFrame, this line will **create a new column** called 'Program' and assign values to it.
3. **= 'Spares'**:
   * This assigns the string 'Spares' to every row in the 'Program' column.
   * **Every row** in the 'Program' column will now have the value 'Spares'.
   * The value 'Spares' could be used to categorize records i
4. **Result**:
   * After running this line of code, the DataFrame will have a new column 'Program' (or up'Spares'.

**E**

Let’s

p

Copy

import pandas as pd

# Sample DataFrame

data = {

data =

'Billing Account Number': [309321147, 123456789, 309991023],

'Amount': [100, 200, 150]

}

dhl = pd.DataFrame(data)

}

dhl = pd.Data

}

dhl = pd.D

}

dhl = p

}

dhl

}

d

print("Original DataFrame:")

print(dhl)

# Adding the 'Program' column with the value 'Spares' for every row

dhl['Program'] = 'Spares'

print("\nDataFrame After Adding 'Program' Column:")

print(dhl)

**Ou**

**Original DataFrame:**

Billing Account Number Amount

0 309321147 100

1 123456789 200

2 309991023 150

**After Adding the 'Program' Column:**

Billing Account Number Amount Program

0 309321147 100 Spares

1 123456789 200 Spares

2 309991023 150 Spares

378 To 384.dhl = dhl.rename({'Waybill Number': 'tracking\_number', 'Billing Shipper Reference': 'PO', 'Origin Service Area Code': 'origin\_cluster', 'Destination Service Area Code':'destination\_cluster', 'Startclock Date':'pup', 'Stopclock Date': 'delivery\_date\_sanitised'},axis=1)

**Explanation of Each Part:**

1. **dhl.rename({...}, axis=1)**:
   * This is the rename() method from pandas, which allows you to rename columns in a DataFrame.
   * The {...} part contains a **dictionary** where the keys are the **current column names** that you want to change, and the values are the **new column names** you want to assign to those columns.
   * axis=1 specifies that the renaming is happening on **columns** (not rows). In pandas, axis=1 refers to columns, and axis=0 refers to rows.

385.Null

**386. dhl['LSP'] = 'DHL'**

1. **['LSP']**:
   * This specifies a column in the DataFrame called 'LSP'.
   * If the column 'LSP' already exists in the dhl DataFrame, this line will **overwrite** all the values in that column with 'DHL'.
   * If the column 'LSP' does not exist, this line will **create a new column** called 'LSP' and set all its values to 'DHL'.
2. **= 'DHL'**:
   * This part assigns the string 'DHL' to every row in the 'LSP' column.
   * This means **every row** in the 'LSP' column will have the same value 'DHL'.
   * 'DHL' could represent a label, such as the name of the logistics service provider (LSP), in the context of shipments or logistics.
3. **Result**:
   * After running this line of code, the dhl DataFrame will either have a new 'LSP' column or it will overwrite the existing one, and every entry in that column will be set to 'DHL'.

**Example:**

import pandas as pd

# Sample DataFrame

data = {

'Billing Account Number': [309321147, 123456789, 309991023],

'Amount': [100, 200, 150]

}

dhl = pd.DataFrame(data)

print("Original DataFrame:")

print(dhl)

# Adding the 'LSP' column with the value 'DHL' for every row

dhl['LSP'] = 'DHL'

print("\nDataFrame After Adding 'LSP' Column:")

print(dhl)

**Output:**

**Original DataFrame**:

Billing Account Number Amount

0 309321147 100

1 123456789 200

2 309991023 150

**After Adding the 'LSP' Column**:

Billing Account Number Amount LSP

0 309321147 100 DHL

1 123456789 200 DHL

2 309991023 150 DHL

**Key Points:**

* The **'LSP' column** is **added** to the DataFrame if it doesn't exist, or **overwritten** if it already exists.
* Every row in the 'LSP' column gets the value 'DHL', which might represent the **logistics service provider**.

**Possible Use Cases:**

1. **Default Value**:
   * If you're working with shipment data and all the records are related to DHL, you might want to tag every row with 'DHL' as the d'LSP' column.

**387. Null**

**388.** **dhl['lane'] = dhl['origin\_cluster'] + '-' + dhl['destination\_cluster']**

**Explanation of Each Part:**

1. **dhl:**
   * **This refers to your pandas DataFrame, which contains the data you are working with. It's assumed that the dhl DataFrame already has the columns 'origin\_cluster' and 'destination\_cluster'.**
2. **['lane']:**
   * **This specifies the new column 'lane' in the dhl DataFrame. If the column 'lane' does not exist yet, it will be created.**
   * **If the column already exists, it will be overwritten with the new values derived from the operation.**
3. **dhl['origin\_cluster']:**
   * **This accesses the 'origin\_cluster' column in the dhl DataFrame.**
   * **This column likely contains some categorical data (e.g., area codes or regions that represent the origin of a shipment).**
4. **dhl['destination\_cluster']:**
   * **Similarly, this accesses the 'destination\_cluster' column in the dhl DataFrame.**
   * **This column likely contains data about the destination of the shipment (e.g., area codes or regions).**
5. **+ '-' +:**
   * **The + operator is used to concatenate (combine) the values of the 'origin\_cluster' and 'destination\_cluster' columns.**
   * **The '-' is a string literal that is placed between the origin and destination to form a "lane" identifier (for example, NY-SF or LA-DC).**
   * **This creates a new string value by combining the two columns, with a hyphen ('-') separating them.**
6. **Result:**
   * **After running this line of code, the 'lane' column will be created or updated to contain a string value for each row, where the format is origin\_cluster-destination\_cluster.**
   * **For instance, if origin\_cluster is "NY" and destination\_cluster is "SF", the 'lane' will be "NY-SF".**

**Example:**

**import pandas as pd**

**# Sample DataFrame**

**data = {**

**'origin\_cluster': ['NY', 'LA', 'SF'],**

**'destination\_cluster': ['SF', 'DC', 'LA'],**

**'Amount': [100, 200, 150]**

**}**

**dhl = pd.DataFrame(data)**

**print("Original DataFrame:")**

**print(dhl)**

**# Creating the 'lane' column by concatenating 'origin\_cluster' and 'destination\_cluster'**

**dhl['lane'] = dhl['origin\_cluster'] + '-' + dhl['destination\_cluster']**

**print("\nDataFrame After Adding 'lane' Column:")**

**print(dhl)**

**Output:**

**Original DataFrame:**

**origin\_cluster destination\_cluster Amount**

**0 NY SF 100**

**1 LA DC 200**

**2 SF LA 150**

**After Adding the 'lane' Column:**

**origin\_cluster destination\_cluster Amount lane**

**0 NY SF 100 NY-SF**

**1 LA DC 200 LA-DC**

**2 SF LA 150 SF-LA**

**Key Points:**

* **Concatenation: The + operator is used to concatenate the values from the 'origin\_cluster' and 'destination\_cluster' columns, separated by a hyphen ('-').**
* **New Column 'lane': The 'lane' column is created (or updated) to represent a string combining the origin and destination clusters.**
  + **In this case, for the first row, the 'lane' value becomes "NY-SF".**

**Possible Use Cases:**

1. **Shipping Routes:**
   * **You might use this for creating a "lane" identifier in logistics, where the origin and destination of a shipment define the shipping route or "lane."**
   * **For example, the lane "NY-SF" could represent a specific route that shipments follow from New York to San Francisco.**
2. **Group Shipments by Lane:**
   * **After creating the 'lane' column, you could use it to group or filter shipments by their origin-destination pairs, making it easier to analyze or track shipments along specific lanes.**

**Conclusion:**

* **The line of code creates a new column 'lane' that combines the 'origin\_cluster' and 'destination\_cluster' columns, separated by a hyphen ('-').**
* **This helps represent a "lane" or route for each shipment or record.**

**389.Null**

**390.** **# Mapping the transit time**

**dhl = dhl.merge(sla, how = 'left', left\_on = ['origin\_cluster','destination\_cluster'], right\_on = ['origin\_site','destination\_site']).drop(['origin\_site','destination\_site'],axis=1)**dhl.merge(...):

* dhl: This is your base pandas DataFrame that you are merging with another DataFrame (sla).
* merge(): This is the pandas method used to combine two DataFrames based on certain conditions. It's similar to SQL joins (e.g., INNER JOIN, LEFT JOIN, etc.).

**392.Null**

**393.** **c1 = pd.to\_datetime(dhl.pup) >= pd.to\_datetime(dhl.effective\_date)**

**Explanation of Each Part:**

1. **pd.to\_datetime(dhl.pup):**
   * **pd.to\_datetime() is a pandas function used to convert values into datetime format.**
   * **dhl.pup refers to the column pup in the dhl DataFrame. The values in this column are presumably date-like (e.g., strings like '2025-01-01' or '2025-01-15').**
   * **This function converts the values in dhl.pup into pandas datetime objects, which makes it easier to perform date-based operations such as comparisons, filtering, etc.**
2. **pd.to\_datetime(dhl.effective\_date):**
   * **Similarly, dhl.effective\_date is another column in the dhl DataFrame, which also contains date-like values.**
   * **The pd.to\_datetime() function is used here to convert the effective\_date values into pandas datetime objects as well.**
3. **>= (Comparison Operator):**
   * **This operator checks whether the values in dhl.pup (after being converted to datetime) are greater than or equal to the values in dhl.effective\_date (also converted to datetime).**
   * **The result of this comparison is a boolean series (a sequence of True or False values), where True means that the date in dhl.pup is on or after the date in dhl.effective\_date, and False means it is before.**
4. **c1 = ...:**
   * **The result of the comparison (a series of True and False values) is assigned to the variable c1.**
   * **This variable c1 can then be used in further filtering, analysis, or logic within the code.**

**What This Code Does:**

* **The code checks, for each row in the dhl DataFrame, whether the pup (pick-up date) is greater than or equal to the effective\_date.**
* **It returns a boolean series where each entry is True if the condition holds (i.e., the pick-up date is on or after the effective date), and False if it does not.**

**Example:**

**Let’s say the dhl DataFrame looks like this:**

**import pandas as pd**

**# Sample DataFrame**

**data = {**

**'pup': ['2025-01-01', '2025-02-15', '2025-03-10'],**

**'effective\_date': ['2025-01-01', '2025-02-10', '2025-03-05']**

**}**

**dhl = pd.DataFrame(data)**

**print("Original DataFrame:")**

**print(dhl)**

**# Applying the code**

**c1 = pd.to\_datetime(dhl.pup) >= pd.to\_datetime(dhl.effective\_date)**

**print("\nBoolean Series Result (c1):")**

**print(c1)**

**Output:**

**Original DataFrame:**

**pup effective\_date**

**0 2025-01-01 2025-01-01**

**1 2025-02-15 2025-02-10**

**2 2025-03-10 2025-03-05**

**Result of c1 (Boolean Series):**

**0 True**

**1 True**

**2 True**

**dtype: bool**

**How It Works:**

* **Row 1: The pup date (2025-01-01) is equal to the effective\_date (2025-01-01), so the result is True.**
* **Row 2: The pup date (2025-02-15) is later than the effective\_date (2025-02-10), so the result is True.**
* **Row 3: The pup date (2025-03-10) is later than the effective\_date (2025-03-05), so the result is True.**

**Key Points:**

* **Date Conversion: The pd.to\_datetime() function ensures that the pup and effective\_date columns are in the correct datetime format for comparison.**
* **Boolean Series: The result is a pandas Series with True or False values, depending on whether the pup date is greater than or equal to the effective\_date.**
* **Use Case: You might use this comparison to filter records where the pick-up date occurs on or after the effective date (e.g., for validation or analysis).**

**394.c2 = pd.to\_datetime(dhl.pup) <= pd.to\_datetime(dhl.expiration\_date)**

Explanation of Each Part:

1. pd.to\_datetime(dhl.pup):
   * dhl.pup refers to the pup column in the dhl DataFrame, which likely contains date-like values (e.g., '2025-01-01').
   * pd.to\_datetime() is a pandas function that converts the values in the pup column into datetime objects. This allows you to perform date comparisons easily.
   * The result of this operation is a datetime series: every value in the pup column is converted into a pandas datetime object.
2. pd.to\_datetime(dhl.expiration\_date):
   * Similarly, dhl.expiration\_date refers to the expiration\_date column in the dhl DataFrame, which also contains date-like values.
   * pd.to\_datetime() is used again to convert these values into pandas datetime objects.
3. <= (Comparison Operator):
   * The <= operator checks whether the pick-up date (pup) is less than or equal to the expiration date (expiration\_date).
   * This comparison will return a boolean series (i.e., a series of True or False values):
     + True if the pick-up date is on or before the expiration date.
     + False if the pick-up date is after the expiration date.
4. c2 = ...:
   * The result of the comparison (a series of True or False values) is stored in the variable c2.
   * c2 is a boolean series that indicates, for each row in the DataFrame, whether the condition (pup <= expiration\_date) is satisfied.

What This Code Does:

* For each row in the dhl DataFrame, this code compares the pick-up date (pup) with the expiration date (expiration\_date).
* It returns a boolean series where:
  + True means the pick-up date is on or before the expiration date.
  + False means the pick-up date is after the expiration date.

Example:

import pandas as pd

# Sample data

data = {

'pup': ['2025-01-01', '2025-02-15', '2025-03-10'],

'expiration\_date': ['2025-01-10', '2025-02-14', '2025-03-15']

}

# Create DataFrame

dhl = pd.DataFrame(data)

# Display the original DataFrame

print("Original DataFrame:")

print(dhl)

# Apply the comparison to check if pup <= expiration\_date

c2 = pd.to\_datetime(dhl.pup) <= pd.to\_datetime(dhl.expiration\_date)

# Display the result of the comparison

print("\nResult of Comparison (c2):")

print(c2)

Output:

Original DataFrame:

pup expiration\_date

0 2025-01-01 2025-01-10

1 2025-02-15 2025-02-14

2 2025-03-10 2025-03-15

Result of Comparison (c2):

0 True

1 False

2 True

dtype: bool

Explanation of the Comparison Results:

* Row 1:
  + pup = 2025-01-01, expiration\_date = 2025-01-10
  + Since the pick-up date (2025-01-01) is before the expiration date (2025-01-10), the result is True.
* Row 2:
  + pup = 2025-02-15, expiration\_date = 2025-02-14
  + Since the pick-up date (2025-02-15) is after the expiration date (2025-02-14), the result is False.
* Row 3:
  + pup = 2025-03-10, expiration\_date = 2025-03-15
  + Since the pick-up date (2025-03-10) is before the expiration date (2025-03-15), the result is True.

Conclusion:

* The c2 series tells you whether the pick-up date (pup) is on or before the expiration date (expiration\_date) for each row in the dhl DataFrame.
* If the pick-up date is earlier than or equal to the expiration date, the comparison returns True; otherwise, it returns False.

This boolean series can be useful for filtering, validating, or analyzing the data based on this condition.

**395.Null**

**396. dhl = dhl[c1&c2]**

* **dhl**: This is your original pandas DataFrame.
* **c1 & c2**: This is a **boolean condition** that is used to filter the rows of the dhl DataFrame. It combines two conditions, c1 and c2, using the bitwise & (AND) operator.
* **dhl[c1 & c2]**: This is a **filtered version of the dhl DataFrame** that includes only the rows where both c1 and c2 are True (i.e., the conditions for both c1 and c2 must be satisfied for the row to be included).

**Detailed Breakdown:**

**1. c1 and c2:**

Both c1 and c2 are expected to be **boolean series**. In pandas, a **boolean series** is a sequence of True or False values that can be used to filter rows in a DataFrame.

For example, if dhl is a DataFrame with columns pup (pick-up date) and expiration\_date, c1 and c2 could be the following conditions:

* **c1**: This could be a condition like dhl['pup'] >= dhl['effective\_date'], which checks if the pick-up date is greater than or equal to the effective date.
* **c2**: This could be a condition like dhl['pup'] <= dhl['expiration\_date'], which checks if the pick-up date is less than or equal to the expiration date.

Each of these conditions would result in a **boolean series** (True or False for each row in dhl).

**2. & (Bitwise AND Operator):**

* The **& operator** is used to combine the two boolean series (c1 and c2).
* The result will be a new boolean series where:
  + True means **both c1 and c2 are True** for that row.
  + False means **either c1 or c2 is False** for that row.

**3. dhl[c1 & c2] (Filtering the DataFrame):**

* This is the main operation.
* **dhl[c1 & c2]** applies the boolean series (c1 & c2) to filter the dhl DataFrame.
  + Only the rows where the combined condition (c1 & c2) is True will be kept.
  + Rows where the combined condition is False will be excluded.

**4. Reassigning to dhl:**

* The filtered DataFrame is then reassigned to the dhl variable, which means dhl now only contains rows that satisfy both conditions (c1 and c2).

**Example:**

Let’s walk through an example to make this clearer. Suppose we have the following dhl DataFrame:

import pandas as pd

# Sample data

data = {

'pup': ['2025-01-01', '2025-02-15', '2025-03-10'],

'effective\_date': ['2025-01-01', '2025-02-10', '2025-03-05'],

'expiration\_date': ['2025-01-10', '2025-02-14', '2025-03-15']

}

# Create DataFrame

dhl = pd.DataFrame(data)

# Convert 'pup', 'effective\_date', and 'expiration\_date' to datetime

dhl['pup'] = pd.to\_datetime(dhl['pup'])

dhl['effective\_date'] = pd.to\_datetime(dhl['effective\_date'])

dhl['expiration\_date'] = pd.to\_datetime(dhl['expiration\_date'])

# Create c1 and c2 as boolean conditions

c1 = dhl['pup'] >= dhl['effective\_date'] # Pick-up date >= Effective date

c2 = dhl['pup'] <= dhl['expiration\_date'] # Pick-up date <= Expiration date

# Filter the DataFrame using c1 and c2

dhl = dhl[c1 & c2]

print(dhl)

**Breakdown of the Conditions (c1 and c2):**

* **Row 1**:
  + pup = 2025-01-01
  + effective\_date = 2025-01-01
  + expiration\_date = 2025-01-10
  + **c1 (pup >= effective\_date)**: True (since 2025-01-01 >= 2025-01-01)
  + **c2 (pup <= expiration\_date)**: True (since 2025-01-01 <= 2025-01-10)
  + **c1 & c2**: True (since both conditions are True)
* **Row 2**:
  + pup = 2025-02-15
  + effective\_date = 2025-02-10
  + expiration\_date = 2025-02-14
  + **c1 (pup >= effective\_date)**: True (since 2025-02-15 >= 2025-02-10)
  + **c2 (pup <= expiration\_date)**: False (since 2025-02-15 > 2025-02-14)
  + **c1 & c2**: False (since c2 is False)
* **Row 3**:
  + pup = 2025-03-10
  + effective\_date = 2025-03-05
  + expiration\_date = 2025-03-15
  + **c1 (pup >= effective\_date)**: True (since 2025-03-10 >= 2025-03-05)
  + **c2 (pup <= expiration\_date)**: True (since 2025-03-10 <= 2025-03-15)
  + **c1 & c2**: True (since both conditions are True)

**Filtered DataFrame:**

After applying dhl = dhl[c1 & c2], the filtered dhl DataFrame will only include the rows where both conditions are True. The result will be:

pup effective\_date expiration\_date

0 2025-01-01 2025-01-01 2025-01-10

2 2025-03-10 2025-03-05 2025-03-15

**Summary:**

* **dhl = dhl[c1 & c2]** filters the rows of the dhl DataFrame by combining two conditions using the **AND** operator (&).
* It keeps only the rows where **both conditions** (c1 and c2) are True.
  + In the example, it keeps rows where the pick-up date (pup) is between the effective date and expiration date (inclusive).
* This filtered DataFrame is then reassigned back to the dhl variable.

Bottom of Form

**397. Null**

**398.** **dhl = dhl.rename({'time':'Time'},axis=1)**

Rename to column

**Example:**

Let’s say we have the following dhl DataFrame:

import pandas as pd

# Sample data

data = {

'time': ['2025-01-01 10:00:00', '2025-01-02 14:30:00'],

'location': ['NYC', 'LA']

}

# Create DataFrame

dhl = pd.DataFrame(data)

dhl = pd.DataFram

# Display original DataFrame

pr

print("Original DataFrame:")

print(dhl)

# Rename the 'time' column to 'Time'

dhl = dhl.rename({

dhl = dhl.rename({

dhl = dhl.r

dhl

'time': 'Time'}, axis=1)

# Display the updated DataFrame

p

print("\nUpdated DataFrame:")

pr

print(dhl)

**O**

**Original DataFrame:**

markdown

time location

0 2025-01-01 10:00:00 NYC

1 2025-01-02 14:30:00 LA

**Updated DataFrame:**

markdown

Time location

0 2025-01-01 10:00:00 NYC

1 2025-01-02 14:30:00 LA

**399.Null**

**400.** **print('Lanes that do not have mapping')**

**Explanation:**

1. **print() function:**
   * **The print() function in Python is used to output information to the console or terminal.**
   * **When you call print(), you provide it with an argument (in this case, a string), and it will display that argument in the console.**
2. **'Lanes that do not have mapping':**
   * **This is a string — a sequence of characters enclosed in single or double quotes.**
   * **The string represents a message or a label. In this case, it's describing a scenario where certain "lanes" (likely referring to routes, paths, or some other concept in your application) do not have an associated mapping (perhaps some data or relationship that is missing).**
3. **What happens when the code runs:**
   * **When this line of code is executed, it simply prints the message "Lanes that do not have mapping" to the console, so the user will see this message.**
   * **This could be a placeholder message, or it could be part of a larger program where the next steps involve identifying and working with lanes that lack mappings.**

**Practical Example:**

**In the context of a larger data processing script, the print() statement might be used as a heading or label before displaying specific lanes that are missing their corresponding mapping. Here's how that could look:**

**# Sample data: A dictionary where the key is the lane and the value is the mapping**

**lanes = {'NYC-LA': 'Mapped', 'LA-SF': None, 'SF-NYC': 'Mapped', 'NYC-Boston': None}**

**# Find lanes that do not have a mapping**

**no\_mapping = [lane for lane, mapping in lanes.items() if mapping is None]**

**# Print a message indicating that we are showing lanes without mappings**

**print('Lanes that do not have mapping')**

**# Print the lanes that have no mapping**

**print(no\_mapping)**

**Output:**

**Lanes that do not have mapping**

**['LA-SF', 'NYC-Boston']**

**In this case:**

* **'Lanes that do not have mapping' is printed as a heading.**
* **The lanes LA-SF and NYC-Boston are identified as those without mappings, which is printed next.**

**Summary:**

* **The line print('Lanes that do not have mapping') is simply displaying a message on the screen, which could be part of a larger process where you are identifying lanes that lack certain data (such as a mapping).**
* **It helps inform the user about the next step in the program, or it could serve as a label for the data that follows.**

**401.** **#print(dhl[dhl['Time'].isna()]['lane'].unique())**

**dhl['Time'].isna():**

* **dhl['Time']: This refers to the column Time in the dhl DataFrame.**
* **isna(): The isna() function is a pandas method that checks for missing (NaN) values in a DataFrame or Series. It returns a boolean series where True indicates that a value is missing (NaN), and False indicates that the value is present (not NaN).**
* **So, dhl['Time'].isna() creates a boolean series where each element corresponds to whether the value in the Time column is missing (NaN) for that particular row in the dhl DataFrame.**

**2. dhl[dhl['Time'].isna()]:**

* **This is a filtering operation.**
* **The dhl['Time'].isna() returns a boolean series. By passing it inside square brackets (dhl[...]), you're filtering the rows of the dhl DataFrame where the Time column is missing (NaN).**
* **Essentially, this will give you a subset of dhl that contains only the rows where the Time value is missing (NaN).**

**3. ['lane']:**

* **After filtering the rows where Time is NaN, you then access the lane column of the filtered DataFrame.**
* **This selects just the lane values from the rows where Time is missing.**

**4. .unique():**

* **unique() is a pandas method that returns the unique values in a given column or Series.**
* **So, dhl[dhl['Time'].isna()]['lane'].unique() will return an array of the unique values in the lane column, but only for the rows where the Time value is missing.**

**5. #:**

* **The # at the beginning of the line indicates that this line of code is commented out, which means it will not run unless the # is removed. It’s probably included here for debugging or as a placeholder for a future operation.**

**Putting It All Together:**

**The code is trying to:**

1. **Filter the rows in dhl where the Time column has missing (NaN) values.**
2. **Access the lane column of the filtered rows.**
3. **Get the unique values of the lane column for those rows.**

**If the code were uncommented and executed, it would output a list (or array) of unique lane names where the Time value is missing.**

**Example:**

**Let's consider a simplified example to see how this works.**

**import pandas as pd**

**import numpy as np**

**# Sample data**

**data = {**

**'Time': [np.nan, '2025-01-01', np.nan, '2025-02-01', np.nan],**

**'lane': ['NYC-LA', 'LA-SF', 'SF-NYC', 'NYC-Boston', 'LA-NYC']**

**}**

**# Create DataFrame**

**dhl = pd.DataFrame(data)**

**# Print lanes where Time is NaN**

**print(dhl[dhl['Time'].isna()]['lane'].unique())**

**Output:**

**css**

**['NYC-LA' 'SF-NYC' 'LA-NYC']**

**Breakdown:**

1. **Filtering: The code selects rows where the Time column is NaN. This filters out rows 0, 2, and 4 (those with NaN in the Time column).**
2. **Selecting lane: From the filtered rows, the lane column values are selected: ['NYC-LA', 'SF-NYC', 'LA-NYC'].**
3. **Unique lanes: The .unique() function returns the unique lanes in those rows: ['NYC-LA', 'SF-NYC', 'LA-NYC'].**

**Summary:**

* **This code finds the rows in the dhl DataFrame where the Time column has missing (NaN) values.**
* **Then, it selects the unique values from the lane column for those rows.**
* **The # means the code is commented out and won't execute unless the # is removed.**

**402.** **dhl.loc[dhl['Time'].isna(),'Time'] = 0**

* **dhl: This is the name of the pandas DataFrame. You have data stored in this DataFrame.**
* **dhl['Time']: This refers to the 'Time' column in the dhl DataFrame.**
* **dhl['Time'].isna(): This checks for missing values (NaN) in the 'Time' column. It returns a boolean Series where each element is True if the value in the corresponding row of the 'Time' column is NaN (missing), and False otherwise.**
* **dhl.loc[...]: This is a way to access specific rows and columns in a DataFrame using labels. The condition inside the loc[] is used to filter rows that meet a certain condition.**
* **dhl.loc[dhl['Time'].isna(), 'Time']: This selects all rows where the 'Time' column has a missing value (NaN). The second part of this statement specifies that we want to modify the 'Time' column.**
* **= 0: This assigns a value of 0 to the selected rows where the 'Time' column has a NaN value.**

**So, in summary:**

**This code looks for any rows in the 'Time' column that have a missing value (NaN), and then it fills those missing values with 0.**

**Exa**

**We’ll create a DataFrame called dhl thaNaN)Time column.**

**pyt**

**Copy**

**import pandas as pd**

**import numpy as np**

**# Create a DataFrame with missing (NaN) values**

**dhl = pd.DataFrame({**

**dhl = pd.DataFrame({**

**dhl = pd.DataFrame**

**'OrderID': [1, 2, 3, 4, 5],**

**'Time': [10, np.nan, 20, np.nan, 30]**

**})**

**})**

**prin**

**})**

**print("Original DataFrame:")**

**print(dhl)**

**Original**

**r**

**Copy**

**OrderID Time**

**0 1 10.0**

**1 2 NaN**

**2 3 20.0**

**3 4 NaN**

**4 5 30.0**

**Applying the Code**

**Now, let's apply the code:**

**python**

**Copy**

**# Replace NaN values in the 'Time' column with 0**

**dhl.loc[dhl['Time'].isna(), 'Time'] = 0**

**print("\nDataFrame after replacing NaN values with 0:")**

**print(dhl)**

**What Happens Step-by-Step?**

1. **Original DataFrame:**
   * **The Time column contains two NaN values (at index 1 and index 3).**
2. **Condition: dhl['Time'].isna():**
   * **The .isna() function checks for missing values (NaN).**
   * **It returns a boolean Series:**

**yaml**

**Copy**

**0 False**

**1 True**

**2 False**

**3 True**

**4 False**

**Name: Time, dtype: bool**

1. **Selecting the Rows with NaN Values:**
   * **dhl.loc[dhl['Time'].isna(), 'Time'] selects the rows where the Time column is NaN.**
   * **This selects rows at index 1 and index 3.**
2. **Assigning 0:**
   * **Finally, the code replaces the NaN values at index 1 and index 3 with 0.**

**Output:**

**The DataFrame after applying the code will look like this:**

**css**

**Copy**

**OrderID Time**

**0 1 10.0**

**1 2 0.0**

**2 3 20.0**

**3 4 0.0**

**4 5 30.0**

**403.Null**

**404.** **dhl = dhl.merge(iata,how='left',left\_on='origin\_cluster',right\_on='iata\_code').drop(['iata\_code','timezoneExplanation:**

**This code performs multiple operations on the dhl DataFrame by merging it with another DataFrame called iata, and then making some adjustments to the resulting DataFrame.**

**1. dhl.merge(iata, how='left', left\_on='origin\_cluster', right\_on='iata\_code')**

* **dhl.merge(iata, ...):**
  + **This line merges the dhl DataFrame with the iata DataFrame. The merge() function combines rows from two DataFrames based on a common column.**
* **how='left':**
  + **The how parameter specifies the type of join you want to perform.**
  + **'left' means a left join, which means all rows from the left DataFrame (dhl) will be kept, and rows from the right DataFrame (iata) will only be included if they match the corresponding values in the left DataFrame. If no match is found in iata, the resulting row will have NaN for the columns from iata.**
* **left\_on='origin\_cluster':**
  + **This specifies the column in the left DataFrame (dhl) that you want to match with the right DataFrame (iata).**
  + **It says: "Look for matches between the origin\_cluster column in dhl and the iata\_code column in iata."**
* **right\_on='iata\_code':**
  + **This specifies the column in the right DataFrame (iata) that should be matched with the origin\_cluster column in dhl.**

**What happens in the merge step?**

**The merge will combine rows where the origin\_cluster in dhl matches iata\_code in iata. The result will include all rows from dhl, along with matching rows from iata. If no match is found for a row in dhl, the corresponding columns from iata will have NaN values.**

**2. .drop(['iata\_code', 'timezone', 'iso\_code'], axis=1)**

* **.drop([...], axis=1):**
  + **This part of the code removes certain columns from the merged DataFrame.**
  + **axis=1 specifies that we are dropping columns (not rows).**
* **['iata\_code', 'timezone', 'iso\_code']:**
  + **These are the column names to be dropped from the DataFrame.**
  + **After the merge, the DataFrame will include columns from both dhl and iata. This line removes the columns iata\_code, timezone, and iso\_code from the resulting DataFrame because they are no longer needed (perhaps the iata\_code column was merged with origin\_cluster and the others are unnecessary for analysis).**

**What happens in the drop step?**

**After merging, the code removes the columns iata\_code, timezone, and iso\_code from the DataFrame, keeping only the necessary columns.**

**3. .rename({'country': 'origin\_country', 'des\_region': 'origin\_region'}, axis=1)**

* **.rename({...}, axis=1):**
  + **This function renames columns in the DataFrame.**
  + **axis=1 indicates that we are renaming columns, not rows.**
* **{'country': 'origin\_country', 'des\_region': 'origin\_region'}:**
  + **This is a dictionary where:**
    - **'country' will be renamed to 'origin\_country'.**
    - **'des\_region' will be renamed to 'origin\_region'.**

**What happens in the rename step?**

**The rename function renames the columns country and des\_region to origin\_country and origin\_region respectively. This can make the column names more specific and easier to interpret based on the context (perhaps they represent the country and region for the origin of shipments).**

**Final Result:**

**The code performs the following sequence of operations on the dhl DataFrame:**

1. **Merges dhl with iata based on matching values between origin\_cluster in dhl and iata\_code in iata (using a left join).**
2. **Removes unnecessary columns iata\_code, timezone, and iso\_code from the merged result.**
3. **Renames the columns country to origin\_country and des\_region to origin\_region for clarity.**

**Example:**

**Let’s say we have the following example data for dhl and iata:**

**dhl DataFrame:**

| **OrderID** | **origin\_cluster** | **other\_column** |
| --- | --- | --- |
| **1** | **US** | **XYZ** |
| **2** | **FR** | **ABC** |
| **3** | **DE** | **PQR** |

**iata DataFrame:**

| **iata\_code** | **country** | **des\_region** | **timezone** | **iso\_code** |
| --- | --- | --- | --- | --- |
| **US** | **United States** | **North America** | **UTC-5** | **US** |
| **FR** | **France** | **Europe** | **UTC+1** | **FR** |
| **DE** | **Germany** | **Europe** | **UTC+1** | **DE** |

**After applying the code, the resulting dhl DataFrame would look like this:**

**Merged and Updated dhl DataFrame:**

| **OrderID** | **origin\_cluster** | **other\_column** | **origin\_country** | **origin\_region** |
| --- | --- | --- | --- | --- |
| **1** | **US** | **XYZ** | **United States** | **North America** |
| **2** | **FR** | **ABC** | **France** | **Europe** |
| **3** | **DE** | **PQR** | **Germany** | **Europe** |

**Notice:**

* **The origin\_cluster from dhl matched with iata\_code in iata, bringing over the country and des\_region columns.**
* **The unnecessary columns (iata\_code, timezone, iso\_code) were dropped.**
* **The columns country and des\_region were renamed to origin\_country and origin\_region.**

**Let me know if you need any further clarification or examples!**

**You said:**

**please given an example**

**ChatGPT said:**

**ChatGPT**

**Sure! Let's walk through an example with real data to see how the code works. We'll start with two sample DataFrames (dhl and iata), then apply the code step by step.**

**Sample DataFrames**

**dhl DataFrame (before merge):**

**python**

**Copy**

**import pandas as pd**

**dhl = pd.DataFrame({**

**'OrderID': [1, 2, 3, 4],**

**'origin\_cluster': ['US', 'FR', 'DE', 'IN'],**

**'other\_column': ['XYZ', 'ABC', 'PQR', 'LMN']**

**})**

**print("Original DHL DataFrame:")**

**print(dhl)**

**Output of dhl DataFrame:**

**Copy**

**OrderID origin\_cluster other\_column**

**0 1 US XYZ**

**1 2 FR ABC**

**2 3 DE PQR**

**3 4 IN LMN**

**iata DataFrame (we will merge dhl with this DataFrame):**

**python**

**Copy**

**iata = pd.DataFrame({**

**'iata\_code': ['US', 'FR', 'DE', 'GB'],**

**'country': ['United States', 'France', 'Germany', 'United Kingdom'],**

**'des\_region': ['North America', 'Europe', 'Europe', 'Europe'],**

**'timezone': ['UTC-5', 'UTC+1', 'UTC+1', 'UTC+0'],**

**'iso\_code': ['US', 'FR', 'DE', 'GB']**

**})**

**print("\nIATA DataFrame:")**

**print(iata)**

**Output of iata DataFrame:**

**Copy**

**iata\_code country des\_region timezone iso\_code**

**0 US United States North America UTC-5 US**

**1 FR France Europe UTC+1 FR**

**2 DE Germany Europe UTC+1 DE**

**3 GB United Kingdom Europe UTC+0 GB**

**Applying the Code:**

**Now, let’s apply the following code to merge the DataFrames, drop certain columns, and rename some columns:**

**python**

**Copy**

**dhl = dhl.merge(iata, how='left', left\_on='origin\_cluster', right\_on='iata\_code') \**

**.drop(['iata\_code', 'timezone', 'iso\_code'], axis=1) \**

**.rename({'country': 'origin\_country', 'des\_region': 'origin\_region'}, axis=1)**

**print("\nMerged and Processed DHL DataFrame:")**

**print(dhl)**

**Step-by-Step Breakdown:**

**1. Merging the DataFrames (dhl.merge(iata, how='left', left\_on='origin\_cluster', right\_on='iata\_code'))**

* **We're merging dhl with iata on the origin\_cluster column from dhl and the iata\_code column from iata.**
* **Since we use how='left', all rows from dhl will be kept, and we add matching rows from iata. If there is no match, NaN will be used for the columns from iata.**

**After merging, we get the following intermediate DataFrame:**

**r**

**Copy**

**OrderID origin\_cluster other\_column iata\_code country des\_region timezone iso\_code**

**0 1 US XYZ US United States North America UTC-5 US**

**1 2 FR ABC FR France Europe UTC+1 FR**

**2 3 DE PQR DE Germany Europe UTC+1 DE**

**3 4 IN LMN NaN NaN NaN NaN NaN**

* **The IN entry in origin\_cluster doesn’t have a matching iata\_code in iata, so the columns from iata are NaN for that row.**

**2. Dropping the Unnecessary Columns (drop(['iata\_code', 'timezone', 'iso\_code'], axis=1))**

**Next, we drop the iata\_code, timezone, and iso\_code columns since we no longer need them. After dropping them, we get:**

**r**

**Copy**

**OrderID origin\_cluster other\_column country des\_region**

**0 1 US XYZ United States North America**

**1 2 FR ABC France Europe**

**2 3 DE PQR Germany Europe**

**3 4 IN LMN NaN NaN**

**3. Renaming Columns (rename({'country': 'origin\_country', 'des\_region': 'origin\_region'}, axis=1))**

**Now, we rename the columns:**

* **country is renamed to origin\_country**
* **des\_region is renamed to origin\_region**

**After renaming, the final DataFrame looks like this:**

**r**

**Copy**

**OrderID origin\_cluster other\_column origin\_country origin\_region**

**0 1 US XYZ United States North America**

**1 2 FR ABC France Europe**

**2 3 DE PQR Germany Europe**

**3 4 IN LMN NaN NaN**

**Final Output:**

**So, the final dhl DataFrame after all operations will look like this:**

**r**

**Copy**

**OrderID origin\_cluster other\_column origin\_country origin\_region**

**0 1 US XYZ United States North America**

**1 2 FR ABC France Europe**

**2 3 DE PQR Germany Europe**

**3 4 IN LMN NaN NaN**

**Key Points:**

1. **Merging: We merged dhl and iata on the origin\_cluster column and iata\_code column. The how='left' join ensures that all rows from dhl are kept, and matching rows from iata are added.**
2. **Dropping Columns: We dropped the iata\_code, timezone, and iso\_code columns that were no longer necessary after the merge.**
3. **Renaming Columns: We renamed country to origin\_country and des\_region to origin\_region for better clarity.**

**This is the step-by-step flow, and the final DataFrame is now ready for further analysis or use.**

**Let me know if you need further clarification or adjustments!**

**Top of Form**

**Get smarter responses, upload files and images, and more.**

**Log in**

**Sign up**

**Bottom of Form**

**ChatGPT can make mistakes. Check important info.**

**','iso\_code'],axis=1).rename({'country':'origin\_country','des\_region':'origin\_region'},axis=1)**