Introduction to Pandas Leetcode solution

Pandas Data Structures
- Create a DataFrame from List

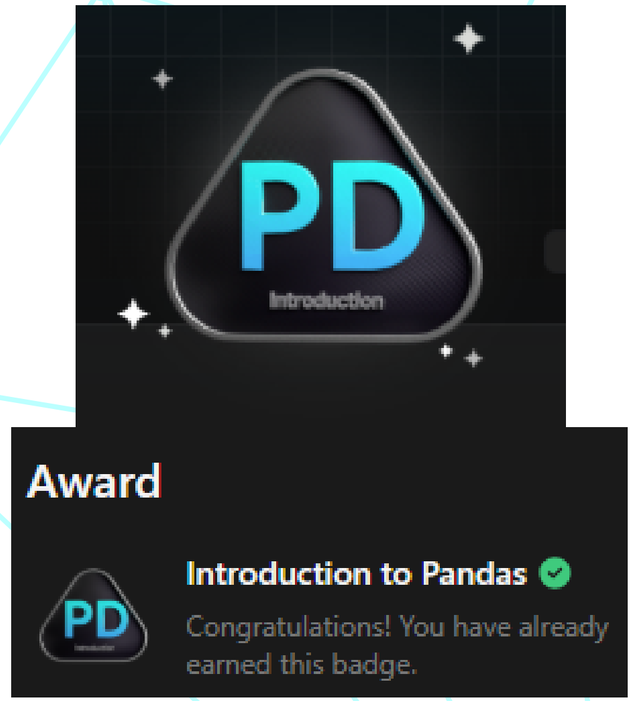
Data Inspection
- Get the Size of a DataFrame
- Display the First Three Rows

Data Selecting
- Select Data
- Create a New Column

Data Cleaning
- Drop Duplicate Rows
- Drop Missing Data
- Modify Columns
- Rename Columns
- Change Data Type
- Fill Missing Data

Table Reshaping
- Reshape Data: Concatenate
- Reshape Data: Pivot
- Reshape Data: Melt

Advanced Techniques
- Method Chaining



- Cheet sheat
- Some basic syntax

All the code are given inside this solution you can write this code same it will work

Pandas Data Structures

- Create a DataFrame from List

See in leetcode you don't need to write full code only complete the function.

So crete Dataset using list in pandas is like this

```
import pandas as pd

data = [[1, 15], [2, 11], [3, 11], [4, 20]]
df = pd.DataFrame(data, columns=['student_id',
'age'])
print(df)
```

Output →

	student_id	age
0	1	15
0	2	11
0	3	11
0	4	20

So in question 2877. Create a DataFrame from List

We need to complete the function give that
Write a solution to create a DataFrame from a 2D list called student_data. This 2D list contains the IDs and ages of some students. The DataFrame should have two columns, student_id and age, and be in the same order as the original 2D list.

```python
import pandas as pd

def createDataframe(student_data: List[List[int]]) -> pd.DataFrame:
    return pd.DataFrame(student_data, columns=['student_id', 'age'])

```

Data Inspection

- Get the Size of a DataFrame
- Display the First Three Rows

- To get size of dataframe we use df.shape df → dataframe
- To display the specific data row we use df.head(number)

```
import pandas as pd

# Create DataFrame
data = [[1, 15], [2, 11], [3, 11], [4, 20]]
df = pd.DataFrame(data, columns=['student_id', 'age'])

# Get size
rows, cols = df.shape
print([rows, cols])  # Output: [4, 2]

# Display first 3 rows
print(df.head(3))
```

Output

```
Size : [4, 2]
   student_id  age
0           1   15
1           2   11
2           3   11
```

2878. Get the Size of a DataFrame

```python
import pandas as pd

def getDataframeSize(players: pd.DataFrame) -> List[int]:
    return list(players.shape)

```

Write a solution to calculate and display the number of rows and columns of players.

2879. Display the First Three Rows

Write a solution to display the first 3 rows of this DataFrame.

```python
import pandas as pd

def selectFirstRows(employees: pd.DataFrame) -> pd.DataFrame:
    return employees.head(3)

```

Data Selecting

- Select Data
- Create a New Column

- To add a new column we use `df['new_column_name'] = values`
- To select data → two condition

1. Select Specific Columns:
   `df[['name', 'bonus']]`
1. Select Rows by Condition:
   `df[df['bonus'] > 2000]`

```
import pandas as pd

# Create DataFrame
data = {'name': ['Alice', 'Bob', 'Charlie'], 'salary': [1000, 1500, 2000]}
df = pd.DataFrame(data)

# Create a new column 'bonus' (double the salary)
df['bonus'] = df['salary'] * 2

# Select specific columns
print(df[['name', 'bonus']])

# Select specific row(s) where bonus > 2500
print(df[df['bonus'] > 2500])
```

Output →

```
      name  bonus
0    Alice   2000
1      Bob   3000
2  Charlie   4000
```

```
      name  salary  bonus
2  Charlie    2000   4000
```

```python
import pandas as pd

def createBonusColumn(employees: pd.DataFrame) -> pd.DataFrame:
    employees['bonus'] = employees['salary'] * 2
    return employees
```

2881. Create a New Column

A company plans to provide its employees with a bonus.
Write a solution to create a new column name bonus that contains the **doubled values** of the salary column.

2880. Select Data

Write a solution to select the name and age of the student with student_id = 101.

```python
import pandas as pd

def selectData(students: pd.DataFrame) -> pd.DataFrame:
    return students.loc[students['student_id'] == 101, ['name', 'age']]

```

Data Cleaning

- Drop Duplicate Rows
- Drop Missing Data
- Modify Columns
- Rename Columns
- Change Data Type
- Fill Missing Data

💡 Covers:

- drop_duplicates()
- dropna()
- astype() for type conversion
- rename() for column renaming
- fillna() to fill missing values
- Column modification using operations like * 2

- drop_duplicates()
- Removes duplicate rows from the DataFrame.

```
df.drop_duplicates
()
```

- dropna()
- Removes rows with missing (NaN) values.
- You can specify a column too.

```
df.dropna()  # Drop rows with any missing values

df.dropna(subset=['col'])  # Drop rows only if 'col' is missing
```

- astype()
- Converts the data type of a column.

```
df['age'] = df['age'].astype(int)
```

- rename()
- Renames columns using a dictionary.

```
df.rename(columns={'old_name': 'new_name'})
```

- fillna()
- Fills missing values with a specified value.

```
df['age'] = df['age'].fillna(0)
```

These functions are essential for data cleaning in pandas.

```
import pandas as pd

# Sample data with duplicates, missing values, and incorrect types
data = {
    'id': [1, 2, 2, 4],
    'name': ['Alice', 'Bob', 'Bob', None],
    'age': [25, None, None, 40],
    'salary': ['1000', '1500', '1500', '2000']
}

df = pd.DataFrame(data)

#  Drop duplicate rows based on all columns
df = df.drop_duplicates()

# Drop rows with missing values in 'name' column
df = df.dropna(subset=['name'])

# Modify 'salary' column by doubling it (after converting to int)
df['salary'] = df['salary'].astype(int) * 2

# Rename columns
df = df.rename(columns={
    'id': 'employee_id',
    'name': 'employee_name',
    'age': 'employee_age',
    'salary': 'employee_salary'
})

#  Change data type of age to int (after filling missing)
df['employee_age'] = df['employee_age'].fillna(0).astype(int)

print(df)
```

Output →

	employee_id	employee_name	employee_age	employee_salary
0	1	Alice	25	2000
1	2	Bob	0	3000
3	4	None	40	4000

🔍 Explanation:

- Row 2 was a duplicate → removed.
- Row with missing name (None) is still present since dropna was called before renaming.
- salary doubled.
- age converted to int with fillna(0).

```python
import pandas as pd

def dropDuplicateEmails(customers: pd.DataFrame) -> pd.DataFrame:
    return customers.drop_duplicates(subset='email', keep='first')

```

2882. Drop Duplicate Rows There are some duplicate rows in the DataFrame based on the email column.
Write a solution to remove these duplicate rows and keep only the first occurrence.

```python
import pandas as pd

def dropMissingData(students: pd.DataFrame) -> pd.DataFrame:
    return students.dropna(subset=['name'])
```

2883. Drop Missing Data There are some rows having missing values in the name column.
Write a solution to remove the rows with missing values.

```python
import pandas as pd

def modifySalaryColumn(employees: pd.DataFrame) -> pd.DataFrame:
    employees['salary'] = employees['salary'] * 2
    return employees
```

2884. Modify Columns A company intends to give its employees a pay rise.
Write a solution to modify the salary column by multiplying each salary by 2.

```python
import pandas as pd

def renameColumns(students: pd.DataFrame) -> pd.DataFrame:
    students = students.rename(columns={
        'id': 'student_id',
        'first': 'first_name',
        'last': 'last_name',
        'age': 'age_in_years'
    })
    return students
```

2885. Rename Columns
Write a solution to rename the columns as follows:
id to student_id
first to first_name
last to last_name
age to age_in_years

```python
import pandas as pd

def changeDatatype(students: pd.DataFrame) -> pd.DataFrame:
    students['grade'] = students['grade'].astype(int)
    return students
```

2886. Change Data Type Write a solution to correct the errors:
The grade column is stored as floats, convert it to integers.

```python
import pandas as pd

def fillMissingValues(products: pd.DataFrame) -> pd.DataFrame:
    products['quantity'] = products['quantity'].fillna(0)
    return products
```

2887. Fill Missing Data Write a solution to fill in the missing value as 0 in the quantity column.

Table Reshaping

- Reshape Data: Concatenate
- Reshape Data: Pivot
- Reshape Data: Melt

💡 Covers:

1. Concatenate:
   - Vertical Join (axis=0): Adds rows.
   - Horizontal Join (axis=1): Adds columns.
2. Pivot: Converts rows to columns (wide format).
3. Melt: Converts columns to rows (long format).

- Concatenate (Vertical/Horizontal Join)
- Joins DataFrames side by side (adds columns).
- Stacks DataFrames vertically (adds rows).

```
# Vertical stack (same columns)
pd.concat([df1, df2], axis=0)

# Horizontal join (side by side)
pd.concat([df1, df2], axis=1)
```

- Pivot (Rows → Columns)
- Converts row values into column headers (reshape data wide).

```
pd.melt(df, id_vars=['product'], var_name='quarter', value_name='sales')
```

- Melt (Columns → Rows)
- Converts columns into rows (reshape data long).

```
pd.melt(df, id_vars=['product'], var_name='quarter', value_name='sales')
```

```python
import pandas as pd

def concatenateTables(df1: pd.DataFrame, df2: pd.DataFrame) -> pd.DataFrame:
    return pd.concat([df1, df2], axis=0).reset_index(drop=True)

```

2888. Reshape Data: Concatenate Write a solution to concatenate these two DataFrames vertically into one DataFrame.

```python
import pandas as pd

def pivotTable(weather: pd.DataFrame) -> pd.DataFrame:
    return weather.pivot(index='month', columns='city', values='temperature')

```

2889. Reshape Data: Pivot Write a solution to pivot the data so that each row represents temperatures for a specific month, and each city is a separate column.

```python
import pandas as pd

def meltTable(report: pd.DataFrame) -> pd.DataFrame:
    return pd.melt(report, id_vars=['product'], var_name='quarter', value_name='sales')

```

2890. Reshape Data: Melt Write a solution to reshape the data so that each row represents sales data for a product in a specific quarter.

Advanced Techniques

- Method Chaining

Method Chaining in pandas is a technique where you call multiple methods in a single line, allowing you to perform multiple operations one after another without creating intermediate variables. It makes the code cleaner and more concise.

```
import pandas as pd

df = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Charlie'],
    'age': [25, 30, 35],
    'salary': [50000, 60000, 70000]
})

# Method Chaining Example:
Filter, rename, and calculate
df = (df[df['age'] > 28]
      .rename(columns={'name': 'emp_name'})
      .assign(salary_inc=df['salary'] * 1.1))

print(df)
```

Output →

	emp_name	age	salary	salary_inc
1	Bob	30	60000	66000
2	Charlie	35	70000	77000

This example filters rows, renames a column, and adds a new column using a single chain of methods.

```python
import pandas as pd

def findHeavyAnimals(animals: pd.DataFrame) -> pd.DataFrame:
    heavy = animals[animals['weight'] > 100]
    return heavy.sort_values(by='weight', ascending=False)[['name']]
```

2891. Method Chaining Write a solution to list the names of animals that weigh strictly more than 100 kilograms.
Return the animals sorted by weight in descending order.

Cheat sheat and basic syntax of panda

📦 Basics

```
import pandas as pd
df = pd.DataFrame(data)    # Create DataFrame
df.head(n)              # First n rows
df.tail(n)              # Last n rows
df.shape                # (rows, columns)
df.columns              # Column names
df.dtypes               # Data types
df.info()               # DataFrame summary
df.describe()           # Stats summary
```

🔍 Selection & Filtering

```
df['col']                  # Single column
df[['col1', 'col2']]       # Multiple columns
df.loc[row_label]          # Row by label
df.iloc[row_index]         # Row by index
df[5:10]                   # Slice rows
df[df['age'] > 25]         # Conditional filter
df.query('age > 25')       # Query syntax
```

🛠️ Modifying Data

```
df['new'] = df['a'] + df['b']          # Add column
df['col'] = df['col'].astype(int)      # Change type
df.rename(columns={'old': 'new'})      # Rename
df.drop(columns=['col'])               # Drop column
df.sort_values(by='col')               # Sort
```

🧼 Cleaning Data

```
df.dropna()                    # Drop missing
df.fillna(0)                   # Fill missing
df.drop_duplicates()           # Remove
duplicates
df.replace('old', 'new')       # Replace values
```

📊 Grouping & Aggregating

```
df.groupby('col')['val'].sum()        # Group + sum
df.groupby('col').agg(['min', 'max'])  # Multiple agg
df.pivot_table(index='col', values='val', aggfunc='mean')  #
Pivot
```

Cheat sheat and basic syntax of panda

🔁 Reshaping

```
pd.concat([df1, df2])                  # Vertical stack
pd.concat([df1, df2], axis=1)          # Horizontal join
pd.melt(df, id_vars='product', var_name='quarter', value_name='sales')  # Melt
df.pivot(index='month', columns='city', values='temperature')  # Pivot
```

💾 Input/Output

```
pd.read_csv('file.csv')              # Load CSV
df.to_csv('file.csv', index=False)   # Save CSV
pd.read_excel('file.xlsx')           # Load Excel
df.to_excel('file.xlsx')             # Save Excel
```

🔗 Method Chaining

```
(df[df['age'] > 25]
 .sort_values('salary')
 .rename(columns={'name': 'employee_name'}))
```

useful Pandas tips & tricks for beginners

🔹 1. Quick Look at Data

- df.head() # First 5 rows
- df.tail() # Last 5 rows
- df.info() # Summary (columns, non-null count, dtypes)
- df.describe() # Stats summary (mean, std, min, etc.)

🔹 2. Selecting Data

- df['column'] # Select single column
- df[['col1', 'col2']] # Select multiple columns
- df.loc[2] # Select row by label/index
- df.iloc[0:3] # Select rows by position
- df[df['age'] > 25] # Conditional filtering

🔹 3. Modifying Data

- df['new_col'] = df['a'] + df['b'] # Add new column
- df.drop('col', axis=1) # Drop column
- df.rename(columns={'old':'new'}) # Rename column
-

🔹 4. Handling Missing Data

- df.dropna() # Drop missing rows
- df.fillna(0) # Replace missing with 0
- df['age'].fillna(df['age'].mean()) # Fill with mean

🔹 5. Sorting & Grouping

- df.sort_values(by='salary', ascending=False) # Sort
- df.groupby('department').mean() # Group & aggregate

🔹 6. Useful Functions

- df.isnull().sum() # Count missing values
- df.duplicated() # Find duplicate rows
- df.value_counts() # Count unique values

🔹 7. Export/Import

- df.to_csv('file.csv') # Save to CSV
- df = pd.read_csv('file.csv') # Load from CSV

Thank you