Let's use Leetcode

If you're on your journey to mastering data manipulation, especially using **Pandas**, I've got something exciting for you. One of the best ways to sharpen your skills is by solving real-world problems, and **LeetCode** has introduced an excellent **Introduction to Pandas Study Plan** that I highly recommend you dive into. You can check it out here.

Why should you spend time solving these problems?

1. Strengthen Your Pandas Skills

Pandas is the go-to library for data manipulation in Python, and it's essential for anyone working in data science. This study plan will help you build a solid foundation in Pandas, from basic data manipulation to more advanced operations like grouping, merging, and reshaping data. You'll face challenges that mimic real-world tasks, making it a practical learning experience.

2. Learn By Doing

Reading tutorials and watching videos is great, but nothing beats hands-on problem-solving. As you solve each problem, you'll become more confident in applying Pandas functions and writing efficient code. The interactive coding platform on LeetCode also provides instant feedback, which helps in refining your approach and learning better practices.

3. Prepare for Interviews

Pandas-related questions are common in data science and data analyst interviews. By tackling these problems, you'll be preparing yourself for common data challenges asked in technical interviews. It's a double win — you're learning and also gearing up for interviews!

4. Build Problem-Solving Mindset

LeetCode is known for its vast collection of problems that not only focus on technical skills but also encourage a strong problem-solving mindset. As you progress, you'll notice an improvement in your ability to break down a problem, figure out the best approach, and implement a solution efficiently — all key skills for a successful data science career.

I've Got Solutions to Share!

To give you a head start, I've already solved a few problems from this study plan, and I'm excited to share my solutions with you. I've worked through some interesting challenges and would be happy to discuss the thought process and approaches I used. If you're stuck on a problem, feel free to reach out or compare your solution with mine — it's a great way to learn from different perspectives.

Take Action!

So, if you want to strengthen your Pandas knowledge while boosting your problem-solving skills, head over to the LeetCode Pandas Study Plan and start solving. Whether you're a beginner or someone looking to refresh your skills, this is a fantastic resource.

And remember, it's all about consistency. Even if you solve just one problem a day, that's progress. Stay patient, stay curious, and before you know it, you'll be much more confident in handling data using Pandas.

Q1 : https://leetcode.com/problems/create-a-dataframe-from-list/description/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

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

Q2 : https://leetcode.com/problems/get-the-size-of-a-dataframe/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def getDataframeSize(players: pd.DataFrame) -> List[int]:
    row_num, column_num = players.shape
    return [row_num, column_num]
```

Q3: https://leetcode.com/problems/display-the-first-three-rows/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

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

Q4 : https://leetcode.com/problems/select-data/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

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

Q5 : https://leetcode.com/problems/create-a-new-column/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q6 : https://leetcode.com/problems/drop-duplicate-rows/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q7 : https://leetcode.com/problems/drop-missing-data/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q8 : https://leetcode.com/problems/modify-columns/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q9 : https://leetcode.com/problems/rename-columns/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

def renameColumns(students: pd.DataFrame) -> pd.DataFrame:
    return students.rename(columns={'id': 'student_id', 'first':'first_na
```

Q10 : https://leetcode.com/problems/change-data-type/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q11 : https://leetcode.com/problems/fill-missing-data/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q12 : https://leetcode.com/problems/reshape-data-concatenate/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

def concatenateTables(df1: pd.DataFrame, df2: pd.DataFrame) -> pd.DataFra
    return pd.concat([df1,df2])
```

Q13 : https://leetcode.com/problems/reshape-data-pivot/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q14: https://leetcode.com/problems/reshape-data-melt/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

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

Q15 : https://leetcode.com/problems/method-chaining/?envType=study-plan-v2&envId=introduction-to-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

def findHeavyAnimals(animals: pd.DataFrame) -> pd.DataFrame:
    heavy_animals = animals[animals['weight'] > 100].sort_values(by='weig
    return pd.DataFrame(heavy_animals['name'])
```

30 Day Challenge: https://leetcode.com/studyplan/30-days-of-pandas/

Q : https://leetcode.com/problems/big-countries/submissions/1395277988/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def big_countries(world: pd.DataFrame) -> pd.DataFrame:
    return world[(world.area >=3000000) | (world.population >= 25000000)]
```

Q : https://leetcode.com/problems/recyclable-and-low-fat-products/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def find_products(products: pd.DataFrame) -> pd.DataFrame:
    return products[(products.low_fats == 'Y') & (products.recyclable ==
```

Q : https://leetcode.com/problems/customers-who-never-order/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def find_customers(customers: pd.DataFrame, orders: pd.DataFrame) -> pd.D
    cust = customers[~customers['id'].isin(orders['customerId'])][['name'
    return cust.rename(columns = {"name":'Customers'})
```

Q : https://leetcode.com/problems/article-views-i/submissions/1395339239/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def article_views(views: pd.DataFrame) -> pd.DataFrame:
    sameAuthor = views[(views['author_id']==views['viewer_id'])].sort_val
    return sameAuthor.rename(columns={'viewer_id': 'id'})
```

Q: https://leetcode.com/problems/invalid-tweets/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def invalid_tweets(tweets: pd.DataFrame) -> pd.DataFrame:
    return tweets[tweets['content'].str.len() >15][['tweet_id']]
```

Q : https://leetcode.com/problems/calculate-special-bonus/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def calculate_special_bonus(employees: pd.DataFrame) -> pd.DataFrame:
    # Apply the bonus condition to each row
    employees['bonus'] = employees.apply(
        lambda row: row['salary'] if row['employee_id'] % 2 == 1 and not
    )

    # Select only the employee_id and bonus columns
    result = employees[['employee_id', 'bonus']].sort_values(by='employee

    return result
```

Q : https://leetcode.com/problems/fix-names-in-a-table/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

def fix_names(users: pd.DataFrame) -> pd.DataFrame:
    users['name'] = users['name'].str.capitalize()
    return users.sort_values('user_id')
```

Q : https://leetcode.com/problems/find-users-with-valid-e-mails/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

def valid_emails(users: pd.DataFrame) -> pd.DataFrame:
    return users[users['mail'].str.match(r"^[a-zA-Z][a-zA-Z0-9_.-]*\@leet
```

Q : https://leetcode.com/problems/patients-with-a-condition/description/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```
import pandas as pd

def find_patients(patients: pd.DataFrame) -> pd.DataFrame:
    return patients[patients['conditions'].str.contains(r"\bDIAB1")]
```

Question:

You are given a dataset with two columns: `file_name` and `content`. Each row represents a file with its name and the textual content within that file. You need to determine how many files contain at least one occurrence of the words **"bull"** and **"bear"** as **standalone words**.

Task:

1. Write a solution to count the number of files that have at least one standalone occurrence of the word **"bull"** and at least one standalone occurrence of the word **"bear"**, respectively.
    - A word is considered standalone if it has spaces or punctuation marks on both sides, or it appears at the beginning or end of the content without being part of another word. For example, the word "bull" in "bull market" should be counted, but not in "bullish". Similarly, count "bear" as a standalone word, but not "bearish".
2. The output should return a table with two rows:
    - One row for the word **"bull"**, showing how many files contain the word as a standalone occurrence.
    - One row for the word **"bear"**, showing how many files contain the word as a standalone occurrence.

3. The table should have the following columns:
    - `word` : Either **"bull"** or **"bear"**.
    - `occurrences` : The number of files that contain the word at least once as a standalone occurrence.
4. Implement the solution in Python using **Pandas** and **regular expressions**.

Example:

Given the following `Employees` dataset:

file_name	content
file1	The bull market is strong, but beware of the bear market
file2	Investors see a bear approaching but no bull in sight
file3	bull and bear are often used to describe market trends
file4	The bullfight was exciting, but no mention of bear
file5	A bullish trend may turn bearish at any moment

Output:

The output should be a DataFrame like this:

word	occurrences
bull	4
bear	4

Additional Clarification:

- **"bull"** and **"bear"** should be counted only when they appear as standalone words, not part of other words like "bullish" or "bearish".
- Case sensitivity should be ignored (i.e., "Bull" and "bull" should both be considered valid occurrences).

Constraints:

- You are required to use **Pandas** for data manipulation and regular expressions for pattern matching.
- Implement the solution in Python, returning the counts of occurrences of the words **"bull"** and **"bear"** in the specified format.

In [10]:
```
import pandas as pd

# Step 1: Create the DataFrame
data = {
    'file_name': ['file1', 'file2', 'file3', 'file4', 'file5'],
    'content': [
        'The bull market is strong, but beware of the bear market',
```

```
        'Investors see a bear approaching but no bull in sight',
        'bull and bear are often used to describe market trends',
        'The bullfight was exciting, but no mention of bear',
        'A bullish trend may turn bearish at any moment'
    ]
}

df = pd.DataFrame(data)

# Step 2: Define function to find standalone occurrences of 'bull' and 'b
def count_occurrences(df):
    # Use regular expressions with word boundaries to count standalone 'b
    bull_count = df['content'].str.contains(r'\bbull\b', case=False).sum(
    bear_count = df['content'].str.contains(r'\bbear\b', case=False).sum(

    # Return the counts as a DataFrame
    result_df = pd.DataFrame({
        'word': ['bull', 'bear'],
        'occurrences': [bull_count, bear_count]
    })

    return result_df

# Step 3: Get the result
result = count_occurrences(df)
print(result)
```

```
  word  occurrences
0  bull           3
1  bear           4
```

Question:

You are given a dataset representing bills, with three columns: `bill_id`, `customer_id`, and `amount`. Each row corresponds to a bill for a customer, and the `amount` column contains the bill amount.

Your task is to report the **number of distinct customers** who have at least one bill with an `amount` strictly greater than **500**.

Input:

- **bill_id**: Unique identifier for each bill.
- **customer_id**: Identifier for the customer who received the bill.
- **amount**: The monetary amount for the bill.

Output:

The result should be a single number representing the **count of distinct customers** who have at least one bill with an amount greater than **500**.

Example:

Given the following dataset:

bill_id	customer_id	amount
1	101	400
2	102	600
3	103	550
4	101	700
5	104	450
6	102	200

In this example:

- Customer **101** has one bill (bill 4) with an amount greater than 500.
- Customer **102** has one bill (bill 2) with an amount greater than 500.
- Customer **103** has one bill (bill 3) with an amount greater than 500.

Therefore, the output should be **3** because customers 101, 102, and 103 have bills greater than 500.

Constraints:

- The solution must use **Pandas** to manipulate the dataset and determine the number of distinct customers with qualifying bills.

In [11]:
```
import pandas as pd

# Sample dataset
data = {
    'bill_id': [1, 2, 3, 4, 5, 6],
    'customer_id': [101, 102, 103, 101, 104, 102],
    'amount': [400, 600, 550, 700, 450, 200]
}

# Create DataFrame
df = pd.DataFrame(data)

# Filter bills where amount > 500
filtered_bills = df[df['amount'] > 500]

# Find the number of distinct customers with at least one qualifying bill
distinct_customers_count = filtered_bills['customer_id'].nunique()

# Print the result
print("Number of distinct customers with a bill amount greater than 500:"
```

```
Number of distinct customers with a bill amount greater than 500: 3
```

Explanation:

1. We create a DataFrame from the given `data` that contains columns `bill_id`, `customer_id`, and `amount`.

2. We filter the rows where the `amount` column has values strictly greater than 500 using `df['amount'] > 500`.
3. Using the `nunique()` function on the `customer_id` column of the filtered DataFrame, we count the number of distinct customers who had at least one bill with an amount greater than 500.
4. Finally, we print the result, which represents the number of distinct customers.

Problem Statement:

You are given a DataFrame `delivery` containing the following columns:

- `order_id`: Integer, representing the unique order ID.
- `customer_id`: Integer, representing the unique customer ID.
- `order_date`: Date (YYYY-MM-DD format), representing the date the order was placed.
- `customer_pref_delivery_date`: Date (YYYY-MM-DD format), representing the date the customer preferred to have the order delivered.
- `is_express_delivery`: Binary integer (1 or 0), representing whether the delivery was express (1) or standard (0).

Write a query to find the percentage of orders that were delivered on the same day as the order date, rounded to two decimal places.

Return a DataFrame with a single column:

- `immediate_percentage`: Float, the percentage of orders that were delivered on the same day as they were placed.

Expected Output:

```
plaintext
+--------------------+
| immediate_percentage|
+--------------------+
| 66.67              |
+--------------------+
```

Solution using Pandas and Python:

In [12]:
```
import pandas as pd

# Sample data for the delivery DataFrame
data = {
    'order_id': [1, 2, 3],
    'customer_id': [1, 2, 1],
    'order_date': ['2019-08-01', '2019-08-02', '2019-08-11'],
    'customer_pref_delivery_date': ['2019-08-02', '2019-08-02', '2019-08-
    'is_express_delivery': [0, 1, 0]
}
```

```
# Create DataFrame
delivery = pd.DataFrame(data)

# Convert 'order_date' and 'customer_pref_delivery_date' to datetime form
delivery['order_date'] = pd.to_datetime(delivery['order_date'])
delivery['customer_pref_delivery_date'] = pd.to_datetime(delivery['custom

# Find the number of orders where the delivery happened on the same day
same_day_deliveries = delivery[delivery['order_date'] == delivery['custom

# Calculate the total number of orders
total_orders = delivery.shape[0]

# Calculate the percentage of same-day deliveries
immediate_percentage = round((same_day_deliveries / total_orders) * 100,

# Create the final output DataFrame
result = pd.DataFrame({
    'immediate_percentage': [immediate_percentage]
})

# Output the result DataFrame
print(result)
```

```
   immediate_percentage
0                 66.67
```

Explanation:

1. **Data Preparation**:
   - The input data is placed in a dictionary and then converted into a DataFrame using `pd.DataFrame()`.
   - We convert the `order_date` and `customer_pref_delivery_date` columns into `datetime` format using `pd.to_datetime()`.
2. **Filtering for Same-Day Deliveries**:
   - We use a condition to check where the `order_date` is equal to the `customer_pref_delivery_date`.
   - The `.shape[0]` method is used to count how many orders meet this condition.
3. **Total Orders**:
   - The total number of rows (orders) in the `delivery` DataFrame is calculated using `shape[0]`.
4. **Calculating the Percentage**:
   - The percentage of same-day deliveries is calculated by dividing `same_day_deliveries` by `total_orders`, multiplying by 100, and rounding the result to two decimal places with `round()`.
5. **Final Output**:
   - The result is placed in a new DataFrame `result` with one column: `immediate_percentage`.

Expected Output:

```
plaintext
   immediate_percentage
0                 66.67
```

This code provides a clean solution using Python and Pandas for calculating the immediate food delivery percentage.

Q : https://leetcode.com/problems/count-salary-categories/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

In [ ]:
```python
import pandas as pd

def count_salary_categories(accounts: pd.DataFrame) -> pd.DataFrame:
    # make categories as per given constraints
    salary_low = (accounts['income'] < 20000).sum()
    salary_avg = ((accounts['income'] >= 20000) & (accounts['income'] <=
    salary_high = (accounts['income'] > 50000).sum()
    # make dataframe to display the final results
    df = pd.DataFrame({'category':['Low Salary', 'Average Salary', 'High

    return df
```

"Ads Performance" (LeetCode problem 1322)

Problem Statement:

You are given a DataFrame `ads` with the following columns:

- `ads_id`: Integer, the unique ID of each ad.
- `user_id`: Integer, the unique ID of the user who clicked the ad.
- `action`: String, either `'Clicked'` or `'Purchased'`, representing the user's interaction with the ad.

Write a query to find the **conversion rate** for each ad. The conversion rate is defined as the ratio of the number of users who purchased the ad to the number of users who clicked on the ad.

Return the result DataFrame with the following columns:

- `ads_id`: Integer, the unique ID of each ad.
- `conversion_rate`: Float, the conversion rate of the ad, rounded to two decimal places.

Sort the result by `ads_id`.

Example Input:

```
plaintext
ads DataFrame:
+--------+---------+-----------+
| ads_id | user_id | action    |
+--------+---------+-----------+
| 1      | 1       | Clicked   |
| 1      | 2       | Clicked   |
| 1      | 3       | Purchased |
| 2      | 4       | Clicked   |
| 2      | 5       | Clicked   |
| 2      | 6       | Clicked   |
| 2      | 7       | Purchased |
+--------+---------+-----------+
```

Expected Output:

```
plaintext
+--------+-----------------+
| ads_id | conversion_rate |
+--------+-----------------+
| 1      | 0.33            |
| 2      | 0.25            |
+--------+-----------------+
```

Explanation:

For `ads_id` 1, there are 3 clicks (user 1, 2, and 3), and 1 purchase (user 3), so the conversion rate is 1/3 = 0.33.
For `ads_id` 2, there are 4 clicks (user 4, 5, 6, and 7), and 1 purchase (user 7), so the conversion rate is 1/4 = 0.25.

Solution using Python and Pandas:

In [13]:
```python
import pandas as pd

# Sample data for the ads DataFrame
data = {
    'ads_id': [1, 1, 1, 2, 2, 2, 2],
    'user_id': [1, 2, 3, 4, 5, 6, 7],
    'action': ['Clicked', 'Clicked', 'Purchased', 'Clicked', 'Clicked', '
}

# Create the DataFrame
ads = pd.DataFrame(data)

# Count the number of clicks for each ad
clicks = ads[ads['action'] == 'Clicked'].groupby('ads_id')['user_id'].cou

# Count the number of purchases for each ad
purchases = ads[ads['action'] == 'Purchased'].groupby('ads_id')['user_id'

# Merge the two DataFrames on ads_id to get the click and purchase counts
merged = pd.merge(clicks, purchases, on='ads_id', how='left')
```

```
# Fill NaN values in purchase_count with 0 (if no purchases were made for
merged['purchase_count'] = merged['purchase_count'].fillna(0)

# Calculate the conversion rate for each ad
merged['conversion_rate'] = merged['purchase_count'] / merged['click_coun

# Round the conversion rate to two decimal places
merged['conversion_rate'] = merged['conversion_rate'].round(2)

# Select the required columns for the final output
result = merged[['ads_id', 'conversion_rate']]

# Output the result DataFrame
print(result)
```

```
   ads_id  conversion_rate
0       1             0.50
1       2             0.33
```

Explanation of the Solution:

1. **Data Preparation**:
   - We prepare the `ads` DataFrame using the sample data provided. It has three columns: `ads_id`, `user_id`, and `action`.
2. **Counting Clicks**:
   - We filter the DataFrame to include only rows where the `action` is `'Clicked'`, and then use `groupby('ads_id')` to count the number of clicks per `ads_id`. This count is stored in the `click_count` column.
3. **Counting Purchases**:
   - Similarly, we filter the DataFrame for rows where the `action` is `'Purchased'`, and group by `ads_id` to count the number of purchases per `ads_id`. This count is stored in the `purchase_count` column.
4. **Merging Clicks and Purchases**:
   - We merge the `clicks` and `purchases` DataFrames using `pd.merge()` on `ads_id`. We perform a left join so that every ad with clicks is retained, even if there are no purchases.
5. **Handling Missing Purchase Data**:
   - After the merge, we fill any missing values in the `purchase_count` column with 0 using `fillna()`. This ensures that ads with no purchases get a purchase count of 0.
6. **Calculating the Conversion Rate**:
   - The conversion rate is calculated as the ratio of `purchase_count` to `click_count` for each `ads_id`.
7. **Rounding and Final Output**:
   - We round the `conversion_rate` to two decimal places and return the final DataFrame, selecting only the `ads_id` and `conversion_rate`

columns.

Expected Output:

```
plaintext
   ads_id  conversion_rate
0       1             0.33
1       2             0.25
```

This solution efficiently calculates the conversion rate for each ad using Python and Pandas, while handling edge cases like missing purchases.
