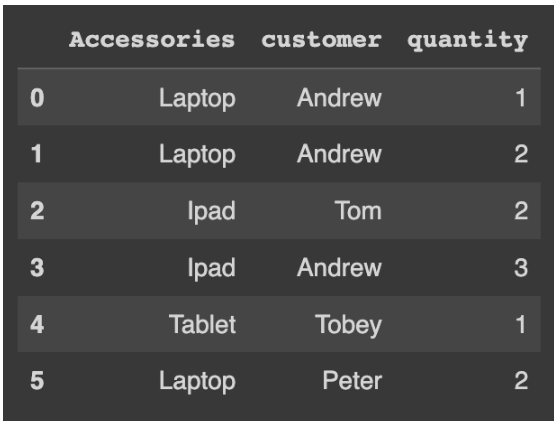
Q1. Pivot pandas

Which of the given option(s) are the right method and syntax for pivoting the below-mentioned dataframe such that each customer would have a value that is the total quantity of a particular type of accessories he holds?

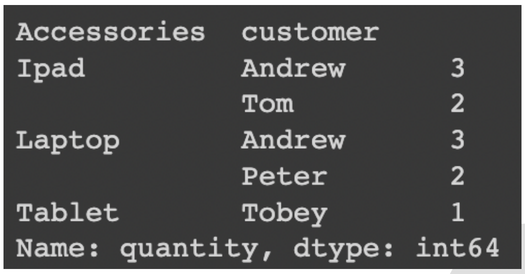
Input:



	Accessories	customer	quantity
0	Laptop	Andrew	1
1	Laptop	Andrew	2
2	Ipad	Tom	2
3	Ipad	Andrew	3
4	Tablet	Tobey	1
5	Laptop	Peter	2

The output for the same, after the required operation, will look something like this:

Output:



```
Accessories  customer
Ipad         Andrew      3
             Tom         2
Laptop       Andrew      3
             Peter       2
Tablet       Tobey       1
Name: quantity, dtype: int64
```

Use the code below in order to create the given dataframe:

import pandas as pd

df = pd.DataFrame({

"Accessories": ["Laptop", "Laptop", "Ipad", "Ipad", "Tablet", "Laptop"],

"customer": ["Andrew", "Andrew", "Tom", "Andrew", "Tobey", "Peter"],

"quantity": [1, 2, 2, 3, 1, 2],

})

Options:

A. **df.pivot(index="Accessories", columns="customer", values="quantity")**
B. **df.groupby(['Accessories', 'customer']).quantity.sum()**
C. **df.set_index(["Accessories", "customer"])["quantity"]**

Q2. Find NaN Rows

You need to find out **all** the rows containing **one or more** NaN values in a dataframe **df**.

Choose the correct code that will do the job in Pandas.

A. df[df.isna()]
B. df.isna()
C. df[df.isnull().any(axis=1)]
D. df[df.isnull(axis=1)]

Q3. Apply pandas

Given a function '**is_null**' with the following implementation.

```
def is_null(x):

  return sum(x.isnull())
```

Choose the correct statement for a dataframe '**df**' when we execute the following code:

```
df.apply(is_null, axis=1)
```

A. Returns the number of missing values of df for each column
B. Returns the number of missing values of df for each row
C. Returns the total number of missing values in the dataframe df
D. Returns None

Q4. Preprocess the dataframe

Problem Description

Given a dataframe **df** as input, perform the following steps for preprocessing:

1. Remove the row if **all** the columns have missing values.

2. Replace the missing values of **Roll_ID** column with 0 and **Name** column with "Anonymous".

3. Replace the missing values in **Marks** column with the **median** value of the column.

4. Change the numerical columns (**Roll_ID** and **Marks**) to **int** datatype in the output.

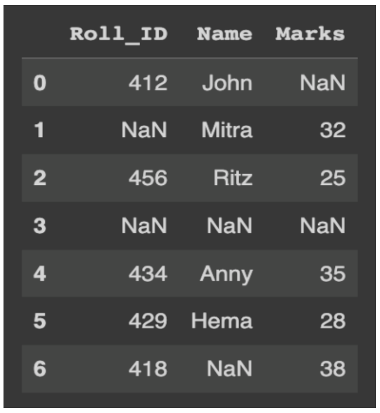
5. Reset the index of the dataframe returned as shown in the sample output.

Note: Every column of the input dataframe **df** is currently of object type.

Use the code below in order to create the input dataframe:

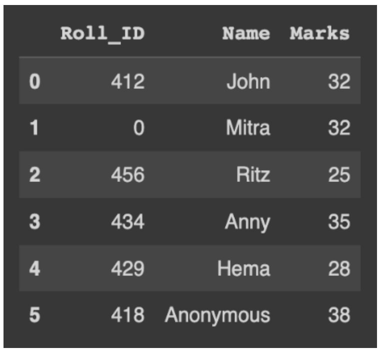
```
import numpy as np

import pandas as pd

data = {

   'Roll_ID': ['412', np.nan, '456', np.nan, '434', '429', '418'],

   'Name': ['John', 'Mitra', 'Ritz', np.nan, 'Anny', 'Hema', np.nan],

   'Marks': [np.nan, '32', '25', np.nan, '35', '28', '38']

}

df = pd.DataFrame(data)
```

Input Dataframe:

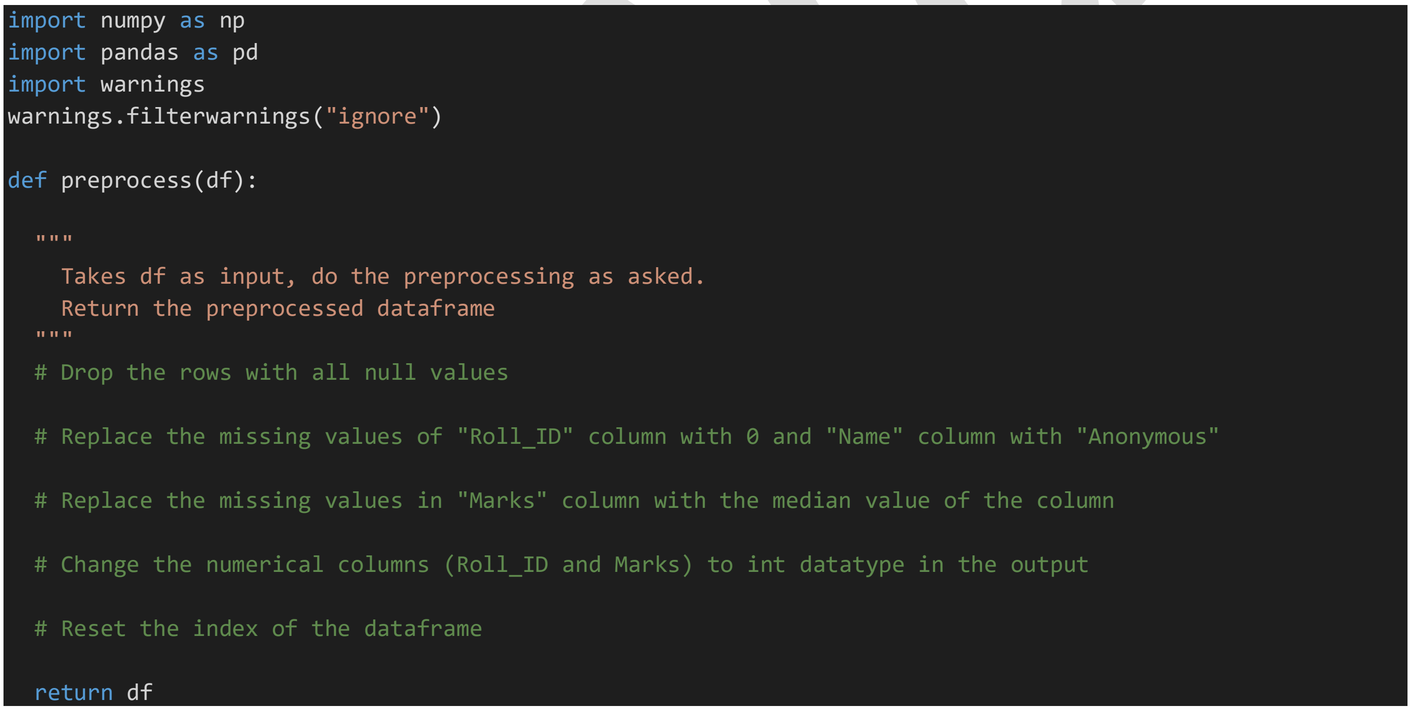


	Roll_ID	Name	Marks
0	412	John	NaN
1	NaN	Mitra	32
2	456	Ritz	25
3	NaN	NaN	NaN
4	434	Anny	35
5	429	Hema	28
6	418	NaN	38

Output Dataframe:



	Roll_ID	Name	Marks
0	412	John	32
1	0	Mitra	32
2	456	Ritz	25
3	434	Anny	35
4	429	Hema	28
5	418	Anonymous	38



```python
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

def preprocess(df):

  """
    Takes df as input, do the preprocessing as asked.
    Return the preprocessed dataframe
  """
  # Drop the rows with all null values

  # Replace the missing values of "Roll_ID" column with 0 and "Name" column with "Anonymous"

  # Replace the missing values in "Marks" column with the median value of the column

  # Change the numerical columns (Roll_ID and Marks) to int datatype in the output

  # Reset the index of the dataframe

  return df
```

Q5. Data Manipulation

Which of the options is **true** about the mentioned statement where **df** is a **pandas** dataframe with column names: "Name", "Gender" and "column_c".

a. df['Gender'].isna().sum() gives the number of null values in the column "**Gender**".

b. df.isna().sum().sum() gives the number of null values in the whole dataframe.

c. df.loc[[3]].isna().sum().sum() gives the number of null values in the **4th** row.

d. df.dropna(axis=1, how='all') and df.dropna(axis=1, how='any') are the methods to drop the **columns** having all or at least one of the values==**Nan** respectively.

e. df.fillna(value=replacement)will fill all the Nan values according to key, value pairs of replacement= {'Name': 0, 'Gender': 'unknown', 'column_c': -99999} where 'Name', 'Gender', and 'column_c' are the column names.

Note:

Refer to Pandas documentation of dropna() to know about the "how" parameter, and fillna() for this question

A. **All the mentioned statements are true except e.**
B. **All the mentioned statements are true except a and b.**
C. **All the mentioned statements are true except c.**
D. **All the mentioned statements are true.**

Q6. fillna return type

Pick the correct statement(s) for the following code snippet.

```
def func(df):
  return df.fillna(1, inplace=True)
```

A. **This function fills the nan values with 1 and returns the dataframe.**
B. **This function fills the nan values with 1 and returns None.**
C. **This function returns the dataframe as it is without any imputation.**
D. **None of these.**

Q7. Org inc.

Problem Statement:

Given a dataframe consisting of organization's access id, return the count of invalid access ids. The valid access ids should contain 'ORG' in the beginning.

For example:

Valid: 'ORG1234' / 'ORG4200'

Invalid: '1234' / 'OR1435'

Input Format:

A DataFrame

Output Format:

An integer

Sample Input:

	access_id
0	ORG6684
1	4564
2	ORG6995
3	2130
4	5839
5	ORG1281
6	ORG2651
7	ORG9870
8	ORG4089
9	ORG3794

Sample Output:

3

Note: Recall String methods in pandas

```
import pandas as pd

def count_ids(df):
    '''
    INPUT: df -> dataframe

    OUTPUT: result -> integer
    '''

    ## STEP1 : Get the valid ids mask

    valid_id_mask = _______

    ## STEP 2: Get invalid ids

    invalid_ids = ______

    ## STEP 3: Get count of invalid ids

    result = _____

    return result
```

Q8. Sanity check

Problem Statement:

Given a dataframe of student's username for batch of 2022,

return the count of username which doesn't follow the below given guidelines:

1. The username should start with 'user_', and
2. The username should end with '_2022'

Input Format: A DataFrame

Output Format: An integer

Sample Input:

	name	username
0	elon	user_spaceboyelon_2022
1	suzlon	user_suzlon123
2	keylon	tothemoon123_2022
3	dusk	user_duskmusk_2022

Sample Output:

2

Note: Recall String methods in pandas

```
import pandas as pd

def validate(df):
    '''
    INPUT: df -> dataframe

    OUTPUT: result -> int
    '''

    ## STEP 1: Get the mask for vaild username

    valid_username_mask = _______

    ## STEP 2: Find the invalid usernames

    invalid_usernames = _____

    ## STEP 3: Get count of invalid usernames

    result = _____

    return result
```

Q9. Location Divide

Problem Description:

Given a dataframe containing a single column "**City\tState**".

Our objective is to process and return this dataframe with names of **City** and **State** separated into two different columns.

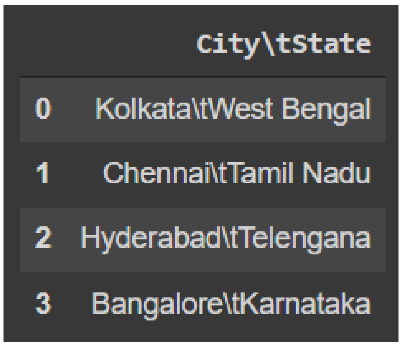
Input Format:

First line of the input contains the dataframe, in the form of a dictionary.

Output Format:

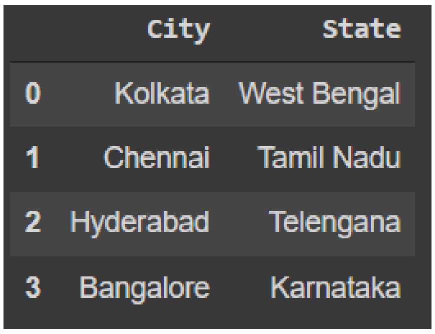
The new dataframe

Sample Input:



	City\tState
0	Kolkata\tWest Bengal
1	Chennai\tTamil Nadu
2	Hyderabad\tTelengana
3	Bangalore\tKarnataka

Sample Output:



	City	State
0	Kolkata	West Bengal
1	Chennai	Tamil Nadu
2	Hyderabad	Telengana
3	Bangalore	Karnataka

Sample Explanation:

The respective **City** and **State** are separated into two different columns as shown in the sample output.

Note: Recall the string methods in Pandas.

Use the code below in order to create the given dataframe:

```
import pandas as pd

df = pd.DataFrame({'City\tState': ["Kolkata\tWest Bengal", "Chennai\tTamil Nadu", "Hyderabad\tTelengana",
"Bangalore\tKarnataka"]})
```

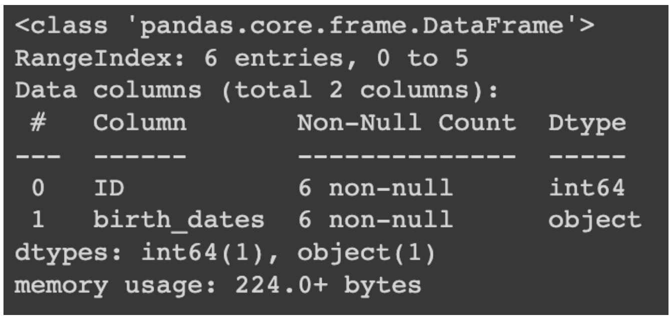
```
import pandas as pd
def location_divide(df):
    '''
    input:
    df -> the dataframe provided to the function

    output:
    df_new -> the new preprocessed dataframe
    '''

    ### STEP 1: Split the values into 2 columns

    df_new = ______

    ### STEP 2: Split the column name into 2

    df_new.columns = _______


    return df_new
```

Q10. compute the age

Given below is an image containing information about a data frame.

The data frame has two columns, **ID** and **birth_dates**.

Assuming that the birth dates are convertible to DateTime format which of the following code snippets will compute the "age" column using the values in the "birth_rates" column?



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           6 non-null      int64
 1   birth_dates  6 non-null      object
dtypes: int64(1), object(1)
memory usage: 224.0+ bytes
```

a.

```
df["birth_dates"]= df["birth_dates"].astype("datetime64")
df["age"] = [pd.to_datetime('today').year-age.year for age in df["birth_dates"]]
```

b.

```
df["age"] = pd.to_datetime("today").year - df["birth_dates"].dt.year
```

c.

```
df["birth_dates"]= df["birth_dates"].astype("datetime64")
df["age"] = 2022-df["birth_dates"].year
```

Q11. Get the Month

As an educational institute, you need to keep a track of all the registered students.

Here you're given the registration IDs and the corresponding dates of a batch of students.

You need to return a DataFrame containing the columns as follows:

- **RID**: Id of the student
- **RDate**: Date of joining of the student
- **RMonth**: Month of joining of the student
- **RYear:** Year of joining of the student
- **RDay:** Day of joining of the student

Input Format:

Number of testcases.

For each testcase there will be two lines of input.

First-line contains the registration IDs of the students, and the following line contains the corresponding registration dates.

Output Format:

Just complete the function that returns a Pandas dataframe object containing the columns mentioned.

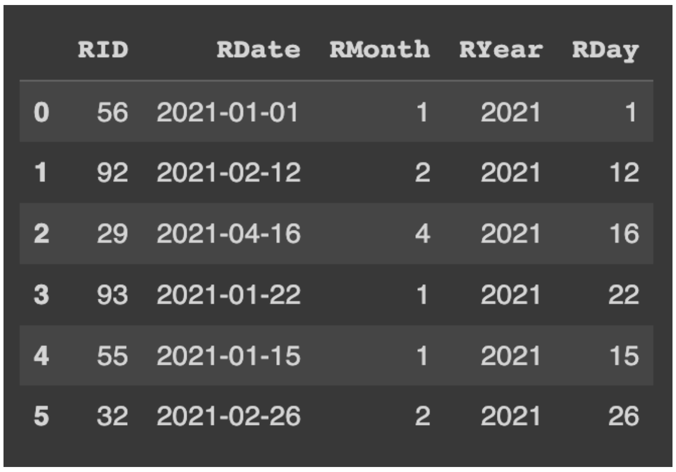
Sample Input:

1

56 92 29 93 55 32

2021-01-01 2021-02-12 2021-04-16 2021-01-22 2021-01-15 2021-02-26

Sample Output:



	RID	RDate	RMonth	RYear	RDay
0	56	2021-01-01	1	2021	1
1	92	2021-02-12	2	2021	12
2	29	2021-04-16	4	2021	16
3	93	2021-01-22	1	2021	22
4	55	2021-01-15	1	2021	15
5	32	2021-02-26	2	2021	26

Sample Explanation:

For 1 test case, here we are given two input lists.

First one consisting of the registration ids and the second one consists of the registration dates of the corresponding registration ids.

Here through the solve function, a dataframe is returned which consists of columns RID, RDate, RMonth, RYear, RDay.

```python
import pandas as pd
def solve(reg_id, reg_dates):
    res_df = pd.DataFrame(zip(reg_id, reg_dates), columns=["RID", "RDate"])
    """
    input:
    reg_id -> id of the input dataframe
    reg_dates -> dates of the input dataframe
    res_df -> the input dataframe

    output:
    return the resultant dataframe after performing the required operation
    """
    # YOUR CODE GOES HERE


    return res_df
```

Q12. Max registrations they asked?

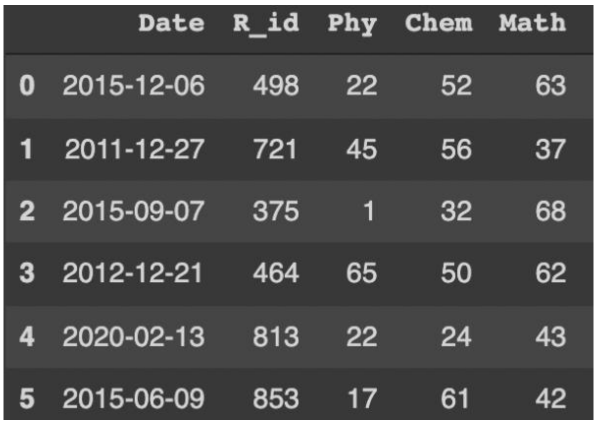
You're given a record of students with their registration dates and registration ids.

You're also given the marks scored by each of the students in Physics, Chemistry, and Mathematics.

You need to find out the month that observed the maximum registrations. Then return the average marks scored in each subject.

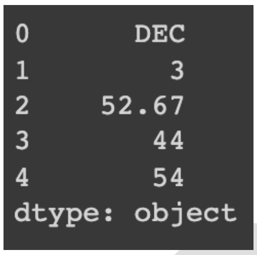
In case of a tie, select the month that comes first (lexicographically).

Sample Input:



	Date	R_id	Phy	Chem	Math
0	2015-12-06	498	22	52	63
1	2011-12-27	721	45	56	37
2	2015-09-07	375	1	32	68
3	2012-12-21	464	65	50	62
4	2020-02-13	813	22	24	43
5	2015-06-09	853	17	61	42

Sample Output:



```
0       DEC
1         3
2     52.67
3        44
4        54
dtype: object
```

- The function is supposed to return a Series.
- The returned Series is just printed with space in the end.

DEC 3 44.0 52.67 54.0

Sample Explanation:

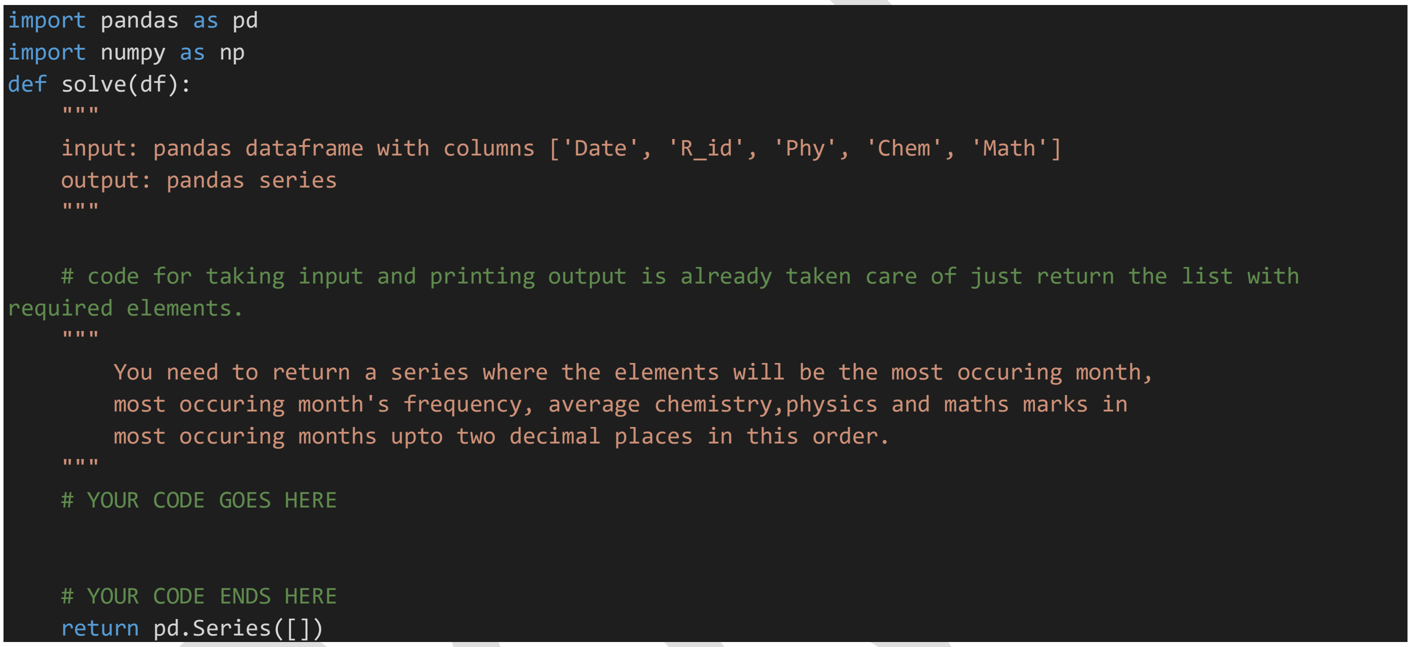
- As it is visible from the dataframe, December is most frequent therefore its first three-letter in upper case are printed first.
- As December's frequency is 3, 3 is printed.
- Then the average marks in **Chemistry, Physics, and Maths** for the registration month December, are printed up to 2 decimal places.
- E.g. The IDs 498, 721, and 464, the average of chemistry is **(52+56+50)/3**.

Note:

1. Round up to **2 decimal points**.
2. Refer to 3-letter abbreviations for each month, i.e. **JAN** for January, **FEB** for February, and so on.
3. You are allowed to use functions like **value_counts()**, **apply()**, **mean()**, **sort_index()**, etc.

Use the code below in order to create the given dataframe:

```
import pandas as pd

df = pd.DataFrame({'Date':["2015-12-06", "2011-12-27", "2015-09-07", "2012-12-21", "2020-02-13", "2015-06-09"],

              'R_id':[498, 721, 375, 464, 813, 853],

              'Phy':[22, 45, 1, 65, 22, 17],

              'Chem':[52, 56, 32, 50, 24, 61],

              'Math':[63, 37, 68, 62, 43, 42]})
```



```
import pandas as pd
import numpy as np
def solve(df):
    """
    input: pandas dataframe with columns ['Date', 'R_id', 'Phy', 'Chem', 'Math']
    output: pandas series
    """

    # code for taking input and printing output is already taken care of just return the list with
required elements.
    """
        You need to return a series where the elements will be the most occuring month,
        most occuring month's frequency, average chemistry,physics and maths marks in
        most occuring months upto two decimal places in this order.
    """
    # YOUR CODE GOES HERE


    # YOUR CODE ENDS HERE
    return pd.Series([])
```