2.1. PROBLEM STATEMENT : NUMPY

Problem Statement 1:

Write a function so that the columns of the output matrix are powers of the input vector.

The order of the powers is determined by the increasing boolean argument.Specifically, when increasing is False, the i-th output column is the input vector raised element-wise power of N-i-1.

HINT: Such a matrix with a geometric progression in each row is named for Alexandre Theophile Vandermonde

In [1]:
```
import numpy as np

x=np.array([1,2,3,4,5])

def vander_matrix(x):
    N=4                      #N is the size of the column

    mat= np.column_stack([x**(N-i-1) for i in range(N)])
                                             #iterating throu
gh the elements in array x for its power raised to N-i-1 as i
n the question column_stack is used to stack 1-D array as col
umns in a 2-D array

    return mat
```

In [2]:
```
vander_matrix(x)
```

Out[2]:
```
array([[  1,   1,   1,   1],
       [  8,   4,   2,   1],
       [ 27,   9,   3,   1],
       [ 64,  16,   4,   1],
       [125,  25,   5,   1]], dtype=int32)
```

Problem Statement 2:

Write a function to find moving average in an array over a window:

Test it over [3,5,7,2,8,10,11,65,72,81,99,100,150] and window of 3

In [3]:

```python
import numpy as np

seq=[3,5,7,2,8,10,11,65,72,81,99,100,150] # here seq is the given input list of elements
w=3
def moving_average(seq,w):
    n=len(seq)    # n is the length of the list

    for i in range(n-w+1):  #iterating throught the list to find the average for a window of 3 till n-w+1,ie., 11 values should be displayed

        x=sum(seq[i:i+w])   #find the sum of the elements in seq for index equal to window,ie., during first iteration, 3 elements from the list will be taken--> (3+5+7)will be value of x

        mavg=x/w           #finding moving average by dividing sum by window.With respect to the above step, for 1st iteration it becomes (3+5+7)/3=15/3=5.0 which is the first output

        print(mavg)
```

In [4]:

```python
moving_average(seq,w)
```

```
5.0
4.666666666666667
5.666666666666667
6.666666666666667
9.666666666666666
28.666666666666668
49.333333333333336
72.66666666666667
84.0
93.33333333333333
116.33333333333333
```

2.2. PROBLEM STATEMENT: PANDAS

Problem Statement 1:

Qn.1. How-to-count-the-distance-to-the-previous-zero

For each value, count the difference of the distance from the previous zero(or the start of the Series, whichever is closer) and if there are no previous zeros, print the position

Consider a DataFrame df where there is an integer column{'X':[7,2,0,3,4,2,5,0,3,4]}

The values should therefore be [1,2,0,1,2,3,4,0,1,2]. Make this a new column 'Y'/

In [5]:

```
import numpy as np
import pandas as pd
```

In [6]:

```
df=pd.DataFrame({'X':[7,2,0,3,4,2,5,0,3,4]})
```

In [7]:

```
df
```

Out[7]:

	X
0	7
1	2
2	0
3	3
4	4
5	2
6	5
7	0
8	3
9	4

In [8]:

```
#Counting the distance to the previous zero

X=[7,2,0,3,4,2,5,0,3,4]

count=1
Y=[]
for items in X:
    if items==0:
        count=0

    Y.append(count)
    count+=1
print('Y = ',Y)
```

```
Y =  [1, 2, 0, 1, 2, 3, 4, 0, 1, 2]
```

In [9]:

```
df=pd.DataFrame({'X':[7,2,0,3,4,2,5,0,3,4],'Y':[1, 2, 0, 1, 2
, 3, 4, 0, 1, 2]})
```

In [10]:

```
df
```

Out[10]:

	X	Y
0	7	1
1	2	2
2	0	0
3	3	1
4	4	2
5	2	3
6	5	4
7	0	0
8	3	1
9	4	2

Qn.2. Create a Datetimeindex that contains each business day of 2015 and use it to index a Series of random numbers

In [11]:

```
DtIndex = pd.date_range('2015-01-01', '2015-12-31',freq='B')
 #freq='B' stands for 'Business day' starting from 1st Jan to
 31st Dec of 2015
```

In [12]:

```
DtIndex
```

Out[12]:

```
DatetimeIndex(['2015-01-01', '2015-01-02', '2015-01-05', '20
15-01-06',
               '2015-01-07', '2015-01-08', '2015-01-09', '20
15-01-12',
               '2015-01-13', '2015-01-14',
               ...
               '2015-12-18', '2015-12-21', '2015-12-22', '20
15-12-23',
               '2015-12-24', '2015-12-25', '2015-12-28', '20
15-12-29',
               '2015-12-30', '2015-12-31'],
              dtype='datetime64[ns]', length=261, freq='B')
```

In [13]:

```
#making s as the series of random numbers and using my DtInde
x to index s
```

In [14]:

```
s=pd.Series(np.random.rand(261),index=DtIndex)
             #created a DatetimeIndex that contains each busin
essday of 2015 and that has been used to index to a series of
 random numbers
```

In [15]:

```
s
```

```
Out[15]: 2015-01-01    0.180522
         2015-01-02    0.384560
         2015-01-05    0.837433
         2015-01-06    0.404452
         2015-01-07    0.581959
         2015-01-08    0.483476
         2015-01-09    0.128267
         2015-01-12    0.965145
         2015-01-13    0.203820
         2015-01-14    0.201965
         2015-01-15    0.238237
         2015-01-16    0.483211
         2015-01-19    0.489988
         2015-01-20    0.604303
         2015-01-21    0.743370
         2015-01-22    0.747126
         2015-01-23    0.855448
         2015-01-26    0.135125
         2015-01-27    0.967107
         2015-01-28    0.359201
         2015-01-29    0.332033
         2015-01-30    0.339583
         2015-02-02    0.256298
         2015-02-03    0.974069
         2015-02-04    0.127657
         2015-02-05    0.445769
         2015-02-06    0.682649
         2015-02-09    0.718425
         2015-02-10    0.894220
         2015-02-11    0.997396
                         ...
         2015-11-20    0.349118
         2015-11-23    0.219259
         2015-11-24    0.484825
         2015-11-25    0.183250
         2015-11-26    0.580114
         2015-11-27    0.219244
         2015-11-30    0.940591
         2015-12-01    0.058943
         2015-12-02    0.030501
         2015-12-03    0.654721
         2015-12-04    0.168405
         2015-12-07    0.904779
         2015-12-08    0.387871
         2015-12-09    0.917678
         2015-12-10    0.434431
         2015-12-11    0.857675
         2015-12-14    0.094920
         2015-12-15    0.081586
         2015-12-16    0.889119
```

```
2015-12-17    0.406439
2015-12-18    0.601203
2015-12-21    0.901803
2015-12-22    0.496089
2015-12-23    0.549698
2015-12-24    0.087471
2015-12-25    0.365286
2015-12-28    0.046043
2015-12-29    0.562544
2015-12-30    0.427023
2015-12-31    0.129502
Freq: B, Length: 261, dtype: float64
```

In [16]:

```
BD = pd.DataFrame(data = s, index= DtIndex)

#Created a dataframe with DtIndex as 'index' and 's' as attri
bute
```

In [17]:

```
BD.columns=['s']        #naming column as 's'
```

In [18]:

```
BD.head()
```

Out[18]:

	s
2015-01-01	0.180522
2015-01-02	0.384560
2015-01-05	0.837433
2015-01-06	0.404452
2015-01-07	0.581959

In [19]:

```
BD.shape
```

Out[19]:

```
(261, 1)
```

Qn.3 Find the sum of the values in s for every Wednesday

In [20]:

```
#I have created the dataframe BD with attribute 's'. So takin
g sum of all wednesdays in 2015 from BD
```

In [21]:

```
BD.head()
```

Out[21]:

	s
2015-01-01	0.180522
2015-01-02	0.384560
2015-01-05	0.837433
2015-01-06	0.404452
2015-01-07	0.581959

In [22]:

```
BD['Day']=DtIndex.day_name() #creating a column for showing t
he corresponding days for the date
```

In [23]:

```
BD.head(7)
```

Out[23]:

	s	Day
2015-01-01	0.180522	Thursday
2015-01-02	0.384560	Friday
2015-01-05	0.837433	Monday
2015-01-06	0.404452	Tuesday
2015-01-07	0.581959	Wednesday
2015-01-08	0.483476	Thursday
2015-01-09	0.128267	Friday

In [24]:

```
BD1=BD.copy()
```

In [25]:

```
BD1=BD1.groupby(['Day']).sum()
```

In [26]:

```
BD1
```

Out[26]:

	s
Day	
Friday	24.784556
Monday	28.025872
Thursday	25.100930
Tuesday	26.532080
Wednesday	25.892034

In [27]:
```
BD1.iloc[4] #gives the sum of all wednesdays
```

Out[27]:
```
s    25.892034
Name: Wednesday, dtype: float64
```

In [28]:
```
#Another method for finding the sum of s for all Wednesday
```

In [29]:
```
BD2=BD.copy() #copying the data to another dataframe
```

In [30]:
```
sum_of_s=BD2[BD2.Day=="Wednesday"].s.sum()
```

In [31]:
```
sum_of_s
```

Out[31]:
```
25.892033950143148
```

Qn.4.Average for each calendar month

In [32]:
```
BD.resample('M').mean() #finding the average for each month in 2015
```

Out[32]:

	s
2015-01-31	0.484833
2015-02-28	0.529426
2015-03-31	0.487405
2015-04-30	0.597809
2015-05-31	0.457258
2015-06-30	0.575066
2015-07-31	0.464190
2015-08-31	0.505761
2015-09-30	0.509543
2015-10-31	0.475586
2015-11-30	0.472767
2015-12-31	0.437119

Qn.5. For each group of four consecutive calendar months in s, find the date on which the highest value occurred

In [33]:

```
BD.head()
```

Out[33]:

	s	Day
2015-01-01	0.180522	Thursday
2015-01-02	0.384560	Friday
2015-01-05	0.837433	Monday
2015-01-06	0.404452	Tuesday
2015-01-07	0.581959	Wednesday

In [34]:

```
BD.groupby(pd.Grouper(freq='M')).max()[:4] #finding date on which s was highest for 4 consecutive months
```

Out[34]:

	s	Day
2015-01-31	0.967107	Wednesday
2015-02-28	0.997396	Wednesday
2015-03-31	0.911611	Wednesday
2015-04-30	0.983615	Wednesday

Problem Statement 2:

Read the dataset from the below link

https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/06_

In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
df=pd.read_csv('https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/06_Stats/US_Baby_Names/US_Baby_Names_right.csv')
```

In [3]:

```
df.shape  #to know the number of rows and columns for the data in the dataframe
```

Out[3]:

```
(1016395, 7)
```

In [4]:

```
df.head() #Gives the first 5 rows in the dataframe.Here df is my dataframe
```

Out[4]:

	Unnamed: 0	Id	Name	Year	Gender	State	Count
0	11349	11350	Emma	2004	F	AK	62
1	11350	11351	Madison	2004	F	AK	48
2	11351	11352	Hannah	2004	F	AK	46
3	11352	11353	Grace	2004	F	AK	44
4	11353	11354	Emily	2004	F	AK	41

Questions Answered:

Qn.1. Delete unnamed columns

In [5]:
```
df.columns                    #To know the unnamed column by knowing the column names
```

Out[5]:
```
Index(['Unnamed: 0', 'Id', 'Name', 'Year', 'Gender', 'State', 'Count'], dtype='object')
```

In [6]:
```
df1=df.copy()                 #Since the original dataframe should not be changed copying the df to another dataframe named df1
```

In [7]:
```
df1.drop("Unnamed: 0",axis=1,inplace=True) #Dropping the unnamed column using drop method.
```

In [8]:
```
df1.head() #Printing the first 5 rows to check if unnamed has been removed
```

Out[8]:

	Id	Name	Year	Gender	State	Count
0	11350	Emma	2004	F	AK	62
1	11351	Madison	2004	F	AK	48
2	11352	Hannah	2004	F	AK	46
3	11353	Grace	2004	F	AK	44
4	11354	Emily	2004	F	AK	41

Qn.2. Show the distribution of male and female

In [9]:
```
both_genders=df1['Gender'].value_counts()
```

In [10]:
```
both_genders
```

Out[10]:
```
F    558846
M    457549
Name: Gender, dtype: int64
```

In [11]:
```
genders=df1.groupby(['Gender'])
```

In [12]:
```
genders.describe()
```

Out[12]:

	Count								Id
	count	mean	std	min	25%	50%	75%	max	co
Gender									
F	558846.0	29.310925	75.962992	5.0	6.0	10.0	23.0	3634.0	55
M	457549.0	41.615650	118.074308	5.0	7.0	12.0	29.0	4167.0	45

2 rows × 24 columns

Qn.3 Show the top 5 most preferred names

In [13]:
```
preferred_name=df1['Name'].value_counts()
preferred_name.head()                              #knowing the 5
 most preferred name by knowing the value_counts for the 'Nam
e 'column in df1 dataframe
```

Out[13]:
```
Riley     1112
Avery     1080
Jordan    1073
Peyton    1064
Hayden    1049
Name: Name, dtype: int64
```

Qn.4. What is the median name occurence in the dataset

In [14]:
```
df1.head()
```

Out[14]:

	Id	Name	Year	Gender	State	Count
0	11350	Emma	2004	F	AK	62
1	11351	Madison	2004	F	AK	48
2	11352	Hannah	2004	F	AK	46
3	11353	Grace	2004	F	AK	44
4	11354	Emily	2004	F	AK	41

In [15]:

```
df1.Id.median()
```

Out[15]:

```
2811921.0
```

In [16]:

```
median_name=df1[df1['Id']==df1.Id.median()]
```

In [17]:

```
median_name
```

Out[17]:

	Id	Name	Year	Gender	State	Count
508197	2811921	Kasey	2010	F	MO	6

Qn.5.Distribution of male and female born count by states

In [18]:

```
distribution=df1.groupby(['Gender','State']).count()
#Showing the distribution of male and female with respect to
 each State
```

In [19]:

```
distribution.loc[: , 'Count']
```

```
Out[19]: Gender  State
         F       AK         2404
                 AL         9878
                 AR         7171
                 AZ        14518
                 CA        45144
                 CO        11424
                 CT         6575
                 DC         3053
                 DE         2549
                 FL        25781
                 GA        19385
                 HI         3255
                 IA         7131
                 ID         4918
                 IL        21268
                 IN        13056
                 KS         7753
                 KY         8817
                 LA        10510
                 MA        10580
                 MD        11276
                 ME         2976
                 MI        16038
                 MN        10677
                 MO        11948
                 MS         7235
                 MT         2690
                 NC        17357
                 ND         2399
                 NE         5370
                                ...
         M       ME         2777
                 MI        13243
                 MN         9004
                 MO         9917
                 MS         6862
                 MT         2986
                 NC        13530
                 ND         2581
                 NE         5029
                 NH         2659
                 NJ        12274
                 NM         4966
                 NV         6024
                 NY        22585
                 OH        14318
                 OK         8138
                 OR         7333
                 PA        14171
```

```
        RI        2468
        SC        8195
        SD        2908
        TN       10588
        TX       27791
        UT        8233
        VA       11997
        VT        1618
        WA       11049
        WI        8940
        WV        3733
        WY        1904
Name: Count, Length: 102, dtype: int64
```

In [20]:

```
pd.crosstab(index=df1.Gender,columns=df1.State) #distribution
 in a more formatted way
```

Out[20]:

State	AK	AL	AR	AZ	CA	CO	CT	DC	DE	FL
Gender										
F	2404	9878	7171	14518	45144	11424	6575	3053	2549	25781
M	2587	8419	6475	10820	31637	9183	5733	3000	2440	20070

2 rows × 51 columns

2.3. PROBLEM STATEMENT: USE CASES OF NUMPY AND PANDAS

Qn.1.Write a Python program which accepts a list named: randomList=['a',0,2]Use exception handling using try-catch

In [1]:

```
import sys

randomList=['a',0,2]

for item in randomList:
    try:
        print("The entry is",item)
        r=1/item
        break
    except:

        print("Oops!",sys.exc_info()[0],"occurred")
        print("Next entry")
        print()

print("The reciprocal of",item,"is",r)
```

```
The entry is a
Oops! <class 'TypeError'> occurred
Next entry

The entry is 0
Oops! <class 'ZeroDivisionError'> occurred
Next entry

The entry is 2
The reciprocal of 2 is 0.5
```

Qn.2.Write a Python program to give exception "Array Out of Bound" if the user wants to access the elements beyond the list size(use try and except)

In [2]:

```
import sys
l=[1,2,3,4]
i=0
for i in l:
    try:
        print("Value at index",i)
        print("is",l[i])

    except:
        print("\nYour list does not contain anymore elements"
)
        print("Aray out of bound ",sys.exc_info()[0])
```

```
Value at index 1
is 2
Value at index 2
is 3
Value at index 3
is 4
Value at index 4

Your list does not contain anymore elements
Aray out of bound  <class 'IndexError'>
```

Qn.3 Write a python module script that contains fib2() method to calculate the Fibonacci series till 1000 and save it as fibo.py

In [3]:

```
def fib2(n):
    a=0
    b=1
    print(a)
    while b<n:
        print(b)
        a, b=b,a+b
```

In [4]:

```
#To run this program as a module, I need to import the module
 as
#import fibo
#fibo.fib2(1000)


fib2(1000)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
```

Qn.4.Write a Python module script that contains is palindrome() method to calculate the input string as palindrome string or not and save it as palindrome.py

In [5]:

```python
def ispalindrome(s):
        s=str(s)
        s=s.lower()
        rev=reversed(s)
        if list(s) == list(rev):
            return True
        return False
```

In [6]:

```python
# To run this program I need to import the module as
        #import palindrome
        #ispalindrome('malayalam')
        #ispalindrome(121)
        #ispalindrome('any')
ispalindrome('malayalam')
```

Out[6]:

```
True
```

In [7]:

```python
ispalindrome(121)
```

Out[7]:

```
True
```

In [8]:

```python
ispalindrome('any')
```

Out[8]:

```
False
```

Qn.5.Write a program in Python with one class called Cipher.

In [13]:

```
import numpy as np
class Cipher:
    L2I = dict(zip("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmno
pqrstuvwxyz0123456789",range(62)))
    I2L = dict(zip(range(62),"ABCDEFGHIJKLMNOPQRSTUVWXYZabcde
fghijklmnopqrstuvwxyz0123456789"))

    def __init__(self,Instr=""):
        self.Instr=str(input("Enter the input string "))

    def encrypt(self,key):
        ciphertext = ""
        Instr=self.Instr
        for c in Instr:
            if c.isalnum():
                ciphertext += self.I2L[ (self.L2I[c] + key) ]
            else:
                ciphertext += c
        return ciphertext

    def decrypt(self,Enstr,key):
        plaintext2 = ""
        for c in Enstr:
            if c.isalnum(): plaintext2 += self.I2L[ (self.L2I
[c] - key)]
            else: plaintext2 += c
        return plaintext2

k=np.random.randint(1,50,1)
key=k[0]
c=Cipher()
encryptstr=c.encrypt(key)
decryptstr=c.decrypt(encryptstr,key)
print("\n Input String is :\t"+c.Instr)
print("\nEncryption value of given string is :\t"+encryptstr)
print("\nDecrypted value is:\t"+decryptstr)
```

```
Enter the input string acadgild

 Input String is :       acadgild

Encryption value of given string is :    oqoruwzr

Decrypted value is:      acadgild
```

Qn.6. Get Data from the following link:

```
http://files.grouplens.org/datasets/movielens/ml-20m.zip
```

In [1]:

```
import numpy as np
import pandas as pd
```

Qn1.Read the dataset using pandas

In [2]:

```
import os
import pandas as pd
os.chdir("c:\data") #since the data is stored in my system im
porting it to the jupyter notebook
```

In [3]:

```
movies=pd.read_csv("movies.csv")
```

In [4]:

```
tags= pd.read_csv("tags.csv")
```

In [5]:

```
ratings=pd.read_csv("ratings.csv")
```

In [6]:

```
movies.head()
```

Out[6]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [7]:

```
tags.head()
```

Out[7]:

	userId	movieId	tag	timestamp
0	18	4141	Mark Waters	1240597180
1	65	208	dark hero	1368150078
2	65	353	dark hero	1368150079
3	65	521	noir thriller	1368149983
4	65	592	dark hero	1368150078

In [8]:

```
ratings.head()
```

Out[8]:

	userId	movieId	rating	timestamp
0	1	2	3.5	1112486027
1	1	29	3.5	1112484676
2	1	32	3.5	1112484819
3	1	47	3.5	1112484727
4	1	50	3.5	1112484580

Qn.2.Extract first row from tags and print its type

In [9]:

```
tagsfirstrow=tags.iloc[0]
```

In [10]:

```
tagsfirstrow
```

Out[10]:

```
userId                18
movieId             4141
tag          Mark Waters
timestamp     1240597180
Name: 0, dtype: object
```

In [11]:

```
print(type(tagsfirstrow))
```

```
<class 'pandas.core.series.Series'>
```

Qn.3.Extract row 0,11,2000 from tags dataframe

In [12]:

```
tagsrows= tags.iloc[[0,11,2000]]
```

In [13]:

```
tagsrows #displaying tag rows
```

Out[13]:

	userId	movieId	tag	timestamp
0	18	4141	Mark Waters	1240597180
11	65	1783	noir thriller	1368149983
2000	910	68554	conspiracy theory	1368043943

Qn.4. Print index,columns of the dataframe

In [14]:

```
ratings.columns
```

Out[14]:

```
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='o
bject')
```

In [15]:

```
movies.columns
```

Out[15]:

```
Index(['movieId', 'title', 'genres'], dtype='object')
```

In [16]:

```
tags.columns
```

Out[16]:

```
Index(['userId', 'movieId', 'tag', 'timestamp'], dtype='obje
ct')
```

Qn.5.Calculate the descriptive statistics for the 'rating' column of the ratings dataframe.Verify using describe

In [17]:

```
ratings.head()
```

Out[17]:

	userId	movieId	rating	timestamp
0	1	2	3.5	1112486027
1	1	29	3.5	1112484676
2	1	32	3.5	1112484819
3	1	47	3.5	1112484727
4	1	50	3.5	1112484580

```
In [18]: print("The descriptive statistics for the rating column is:")
         print("count= ",ratings["rating"].count())
         print("mean= ",ratings["rating"].mean())
         print("standard deviation=", ratings["rating"].std())
         print("min",ratings["rating"].min())
         print("median= ",ratings["rating"].median())
         print("max= ",ratings["rating"].max())
```

```
The descriptive statistics for the rating column is:
count=  20000263
mean=  3.5255285642993797
standard deviation= 1.051988919275684
min 0.5
median=  3.5
max=  5.0
```

```
In [19]: ratings["rating"].describe()#verifying using desscribe
```

```
Out[19]: count    2.000026e+07
         mean     3.525529e+00
         std      1.051989e+00
         min      5.000000e-01
         25%      3.000000e+00
         50%      3.500000e+00
         75%      4.000000e+00
         max      5.000000e+00
         Name: rating, dtype: float64
```

Qn.6. Filter out ratings with rating >5

```
In [20]: r=ratings[ratings["rating"]>5]
```

```
In [21]: print(r)
```

```
Empty DataFrame
Columns: [userId, movieId, rating, timestamp]
Index: []
```

Qn.7.Find how many null values,missing values are present.Deal with them.Print out how many rows have been modified

```
In [22]: ratings.isnull().sum()#finding null values in ratings column
```

Out[22]:
```
userId       0
movieId      0
rating       0
timestamp    0
dtype: int64
```

In [23]:
```
movies.isnull().sum()#finding null values in movies column
```

Out[23]:
```
movieId    0
title      0
genres     0
dtype: int64
```

In [24]:
```
tags.isnull().sum()#finding null values in tags column
```

Out[24]:
```
userId        0
movieId       0
tag          16
timestamp     0
dtype: int64
```

In [25]:
```
#Only tags data has null values for'tag' and there are 16 nul
l values
```

In [26]:
```
tags.fillna(method = "ffill", inplace=True)# using forward fi
ll to fill in the missing null values in tags
```

In [27]:
```
tags.isnull().sum() #thus null values removed using forward f
ill
```

Out[27]:
```
userId       0
movieId      0
tag          0
timestamp    0
dtype: int64
```

Qn.8.Filter out movies from the movies dataframe that are of type'Animation'

In [28]:
```
animatedmovies=movies[movies.genres =="Animation"]
```

In [29]:
```
animatedmovies
```

Out[29]:

	movieId	title	genres
2503	2588	Cloudland (1998)	Animation
4906	5002	Fritz the Cat (1972)	Animation
4907	5003	Nine Lives of Fritz the Cat, The (1974)	Animation
9455	27738	Cathedral, The (Katedra) (2002)	Animation
9989	32840	Vincent (1982)	Animation
13444	66335	Afro Samurai: Resurrection (2009)	Animation
13858	69469	Garfield's Pet Force (2009)	Animation
14184	71158	Immigrants (L.A. Dolce Vita) (2008)	Animation
14492	72603	Merry Madagascar (2009)	Animation
14578	72927	Donkey Xote (2007)	Animation
14580	72931	Tango (1981)	Animation
14876	74486	$9.99 (2008)	Animation
14938	74791	Town Called Panic, A (Panique au village) (2009)	Animation
15085	76709	Spider-Man: The Ultimate Villain Showdown (2002)	Animation
15860	80469	Superman/Batman: Apocalypse (2010)	Animation
15990	81018	Illusionist, The (L'illusionniste) (2010)	Animation
16551	83603	Fern flowers (Fleur de fougère) (1949)	Animation
16649	84192	Corto Maltese: Ballad of the Salt Sea (Corto M...	Animation
18144	90843	Lavatory Lovestory (Ubornaya istoriya - lyubov...	Animation
18145	90845	Fall of the House of Usher, The (Zánik domu Us...	Animation
18224	91187	Millhaven (2010)	Animation
18695	93083	Live Music (2009)	Animation
18980	94423	Disney Princess Collection: Jasmine's Enchante...	Animation
19562	96897	Bleach: Fade to Black (Bur&#299;chi Fade to Bl...	Animation
20310	99820	Pokémon the Movie: Black - Victini and Reshira...	Animation
20311	99822	Pokémon the Movie: White - Victini and Zekrom ...	Animation
20314	99832	Hand, The (Ruka) (1966)	Animation
20527	100509	Tale of Tales (Skazka skazok) (1979)	Animation
20871	102007	Invincible Iron Man, The (2007)	Animation
20872	102009	Thor: Tales of Asgard (2011)	Animation
...	...	...	...

	movieId	title	genres
25207	118948	Blackbird (1959)	Animation
25208	118950	A Phantasy (1952)	Animation
25211	118956	Lines: Horizontal (1962)	Animation
25212	118958	Mosaic (1966)	Animation
25213	118960	Begone Dull Care (1949)	Animation
25214	118962	Synchromy (1971)	Animation
25538	120853	Fresh Guacamole (2012)	Animation
25539	120855	Adam and Dog (2012)	Animation
25663	121302	Someone's Gaze (2013)	Animation
25718	121600	Bosko's Parlor Pranks (1934)	Animation
26052	124889	The Adventures of Tom Thumb & Thumbelina (2002)	Animation
26163	125924	The Periwig-Maker (1999)	Animation
26169	125936	Crac (1981)	Animation
26171	125940	Syrinx (1966)	Animation
26181	125960	The Trip to Squash Land (1967)	Animation
26185	125968	It's Christmastime Again, Charlie Brown (1992)	Animation
26189	125976	Nocturna Artificialia (1979)	Animation
26192	125982	Trick or Treat (1952)	Animation
26207	126012	The Fat Albert Halloween Special (1977)	Animation
26226	126050	Stille Nacht I: Dramolet (1988)	Animation
26230	126058	Rehearsals for Extinct Anatomies (1987)	Animation
26236	126070	Chainsaw Maid (2007)	Animation
26237	126072	The Little Matchgirl (2006)	Animation
26246	126090	Hedgehog in the Fog (1975)	Animation
26247	126092	The Cat's Out (1931)	Animation
26248	126094	Claymation Comedy of Horrors (1991)	Animation
26319	126405	The Adventures of André and Wally B. (1984)	Animation
26809	128864	Four Sahibjade (2014)	Animation
27103	130394	The Mascot (1934)	Animation
27155	130644	The Garden of Sinners - Chapter 5: Paradox Par...	Animation

83 rows × 3 columns

Qn.9.Find the average ratings of movies

In [30]:

```
ratings.rating.mean()
```

Out[30]:

```
3.5255285642993797
```

Qn.10.Perform an inner join of movies and tags based on movieId

In [31]:

```
result=pd.merge(movies,tags, how = "inner", on = "movieId")
```

In [32]:

```
result.head()
```

Out[32]:

	movieId	title	genres	userId	
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1644	Watc
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	comp anima
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	Dis anim fea
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	F anima
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	T L does st m

Qn.11.Print out the 5 movies that belong to the Comedy genre and having rating greater than 4

In [33]:

```
df1=pd.merge(movies,ratings, how = "inner", on = "movieId")#p
erforming inner join based on movieid and creating a datafram
e df1
```

In [34]:

```
c1=df1[(df1["genres"] =="Comedy") & (df1["rating"] >4)] # app
lying condition for df1
```

In [35]:
```
c1.head()
```

Out[35]:

	movieId	title	genres	userId	rating	timestamp
87435	5	Father of the Bride Part II (1995)	Comedy	117	5.0	861553146
87437	5	Father of the Bride Part II (1995)	Comedy	127	5.0	847127740
87455	5	Father of the Bride Part II (1995)	Comedy	350	5.0	1360209812
87460	5	Father of the Bride Part II (1995)	Comedy	390	5.0	836139583
87462	5	Father of the Bride Part II (1995)	Comedy	401	5.0	847049988

Qn.12.Split 'genres' into multiple columns

In [36]:
```
new = movies["genres"].str.split("|", expand = True)
```

In [37]:
```
new.head()
```

Out[37]:

	0	1	2	3	4	5	6	7	
0	Adventure	Animation	Children	Comedy	Fantasy	None	None	None	Noi
1	Adventure	Children	Fantasy	None	None	None	None	None	Noi
2	Comedy	Romance	None	None	None	None	None	None	Noi
3	Comedy	Drama	Romance	None	None	None	None	None	Noi
4	Comedy	None	None	None	None	None	None	None	Noi

Qn.13. Extract year from title, e.g.(1995)

In [38]:
```
movies['year'] = movies['title']
movies['year']=movies.year.str[-6:]
```

In [39]:
```
movies.head()
```

Out[39]:

	movieId	title	genres	year
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	(1995)
1	2	Jumanji (1995)	Adventure|Children|Fantasy	(1995)
2	3	Grumpier Old Men (1995)	Comedy|Romance	(1995)
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	(1995)
4	5	Father of the Bride Part II (1995)	Comedy	(1995)

Qn.14. Select rows based on timestamps later than 2015-02-01

In [40]:

```
tags["times"]=pd.to_datetime(tags["timestamp"], format = '%Y-%m-%d', exact = True)
```

In [41]:

```
ratings["times"]=pd.to_datetime(ratings["timestamp"], format = '%Y-%m-%d', exact = True)
```

In [42]:

```
print(tags[tags["times"]> '2015-02-01'])
```

```
Empty DataFrame
Columns: [userId, movieId, tag, timestamp, times]
Index: []
```

In [43]:

```
print(ratings[ratings["times"]> '2015-02-01'])
```

```
Empty DataFrame
Columns: [userId, movieId, rating, timestamp, times]
Index: []
```

Qn.15. Sort the tags dataframe based on timestamp

In [44]:

```
sorted_tags=tags.sort_values(by=['timestamp'])
```

In [45]:

```
sorted_tags.head()
```

Out[45]:

	userId	movieId	tag	timestamp	times
333932	100371	2788	monty python	1135429210	1970-01-01 00:00:01.135429210
333927	100371	1732	coen brothers	1135429236	1970-01-01 00:00:01.135429236
333924	100371	1206	stanley kubrick	1135429248	1970-01-01 00:00:01.135429248
333923	100371	1193	jack nicholson	1135429371	1970-01-01 00:00:01.135429371
333939	100371	5004	peter sellers	1135429399	1970-01-01 00:00:01.135429399

2.4. PROBLEM STATEMENT: SCIPY AND MATPLOTLIB

Scipy:

We have the min and max temperatures in a city in India for each months of the year. We would like to find a function to describe this and show it graphically, the dataset given below:

Task:

1. fitting it to the periodic function

2. plot the fit

Data

Max=39,41,43,47,49,51,45,38,37,29,27,25

Min=21,23,27,28,32,35,31,28,21,19,17,18

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
%matplotlib inline
```

In [3]:
```
Max=np.array([39,41,43,47,49,51,45,38,37,29,27,25])
Min=np.array([21,23,27,28,32,35,31,28,21,19,17,18])
Months=np.arange(12)
```

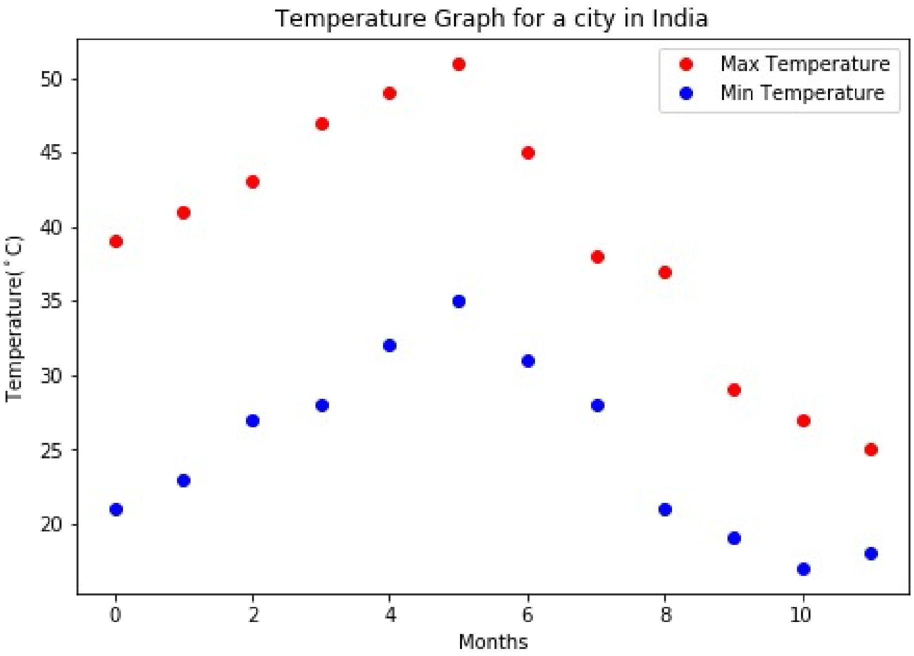
In [4]:
```
Max
```

Out[4]:
```
array([39, 41, 43, 47, 49, 51, 45, 38, 37, 29, 27, 25])
```

In [5]:
```
Min
```

Out[5]:
```
array([21, 23, 27, 28, 32, 35, 31, 28, 21, 19, 17, 18])
```

In [6]:
```
fig=plt.figure()
axes=fig.add_axes([0.1,0.1,1,1])

axes.plot(Months,Max,'ro',label='Max Temperature')
axes.plot(Months,Min,'bo',label='Min Temperature')

axes.set_xlabel('Months')
axes.set_ylabel('Temperature($^\circ$C)')
axes.set_title('Temperature Graph for a city in India')
axes.legend(loc=0)

plt.show()
```



In [7]:

```
#Writing the periodic function for curve fitting
```

In [8]:

```
from scipy import optimize
```

In [9]:

```
def yearly_temp(times, avg, ampl, time_offset):
    return (avg+ ampl * np.cos((times + time_offset) * 2 * np
.pi / times.max()))
```

In [10]:

```
res_max, cov_max = optimize.curve_fit(yearly_temp, Months,Max
, [30, 20, 0])
res_min, cov_min =optimize.curve_fit(yearly_temp, Months,Min,
 [50, 10, 0])
optimize.curve_fit(yearly_temp, Months,Min, [40, 10, 0])
```

Out[10]:

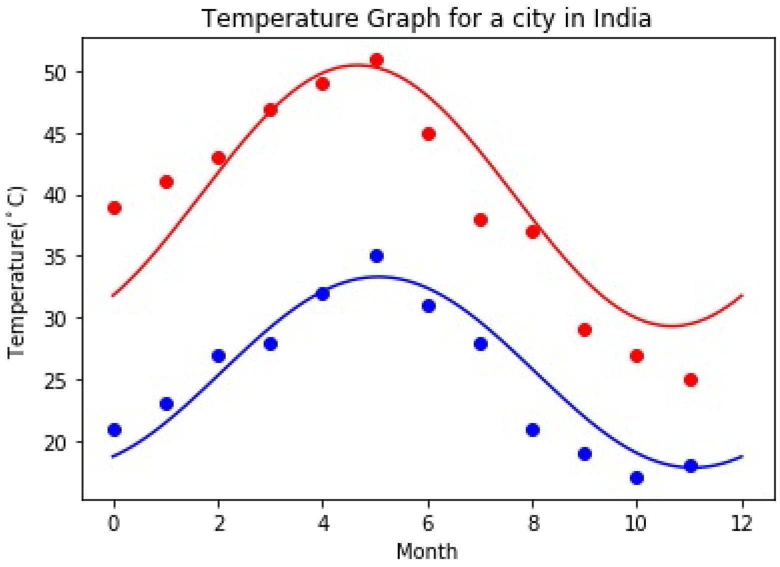
```
(array([25.55626462, -7.74472964,  0.93101294]),
 array([[ 0.19941393, -0.02644226, -0.00351662],
        [-0.02644226,  0.38392581, -0.00606194],
        [-0.00351662, -0.00606194,  0.02114125]]))
```

In [11]:

```
#to plot the fit
```

In [12]:

```
days=np.linspace(0,12,num=365)
fig=plt.figure()
axes=fig.add_axes([0.1,0.1,0.8,0.8])
axes.plot(Months, Max, 'ro')
axes.plot(days, yearly_temp(days, *res_max), 'r-')
axes.plot(Months, Min, 'bo')
axes.plot(days, yearly_temp(days, *res_min), 'b-')
axes.set_xlabel('Month')
axes.set_ylabel('Temperature($^\circ$C)')
axes.set_title('Temperature Graph for a city in India')
plt.show()
```



Matplotlib

Qn. This assignment is for visualization using matplotlib:

data to use:

url=https://raw.githubusercontent.com/Geoyi/Cleaning-Titanic-Data/master/titanic_original.csv

titanic=pd.read_csv(url)

Charts to plot:

1. Create a pie chart representing the male/female proportion

2. Create a scatterplot with the Fare paid and the Age, differ the plot

color by gender

In [13]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [14]:

```
%matplotlib inline
```

In [15]:

```
titanic=pd.read_csv("https://raw.githubusercontent.com/Geoyi/
Cleaning-Titanic-Data/master/titanic_original.csv")
```

In [16]:

```
titanic.head()
```

Out[16]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5
3	1.0	0.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5
4	1.0	0.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5

In [17]:

```
titanic.shape
```

Out[17]:

```
(1310, 14)
```

In [18]:

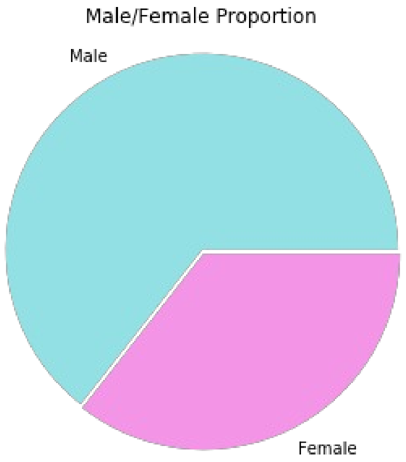
```
gender=titanic['sex'].value_counts() #gives the male female p
roportion
```

In [19]:

```
gender
```

Out[19]:
```
male      843
female    466
Name: sex, dtype: int64
```

In [20]:
```
#Pie chart representing the male/female proportion

labels = 'Male','Female'
sizes= gender         #gender=titanic['sex'].value_counts()
colors= ['#92DFE4','#F394E7'] #got from RGB Hex code

explode = (0,0.02)  # to "explode" the 2nd slice,ie.,showing
 the 2nd slice as a separated piece

fig,ax = plt.subplots(figsize=(10,5))   #Creates a figure and
 one subplot
ax.set_title("Male/Female Proportion")

ax.pie(sizes, explode=explode, labels=labels,colors=colors)
ax.axis('equal')  # Equal aspect ratio ensures that pie is dr
awn as a circle

plt.show()
```



Qn.2.Create a scatterplot with the Fare paid and the Age, differ the plot color by gender

In [21]:
```
titanic.head()
```

Out[21]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.
3	1.0	0.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.
4	1.0	0.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.

In [22]:

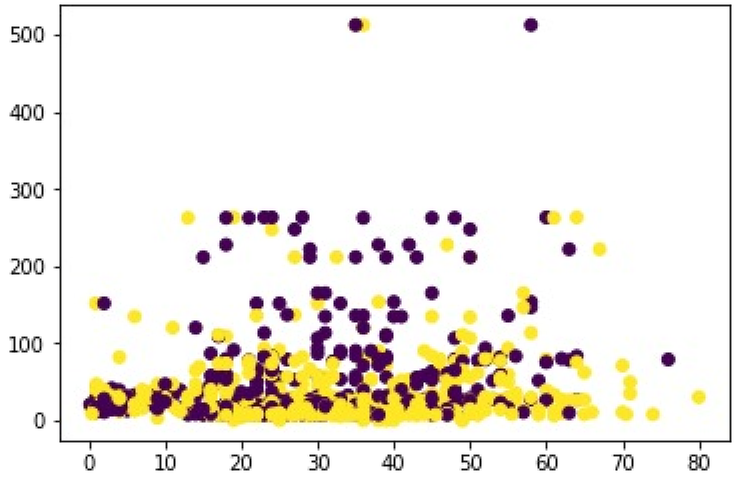
```
titanic.columns
```

Out[22]:

```
Index(['pclass', 'survived', 'name', 'sex', 'age', 'sibsp',
'parch', 'ticket',
       'fare', 'cabin', 'embarked', 'boat', 'body', 'home.de
st'],
      dtype='object')
```

In [23]:

```
#have to get color from titanic['sex']. so have to split the
 male and female to labels of 1 and 0 and hence using factori
ze

fig, ax = plt.subplots()
ax.scatter(titanic['age'], titanic['fare'], c=pd.factorize(ti
tanic['sex']) [0])

plt.show()
```



2.5. PROBLEM STATEMENT: DATA CLEANING

Qn1. Some Values in the FlightNumber column are missing. These numbers are meant to increase by 10 with each row so 10055 and 10075 need to be put in place. Fill in these missing numbers and make the column an integer column(instead of a float column).

In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
df= pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAn',
'londON_StockhOlm','Budapest_PaRis','Brussels_londOn'],
                              'Flight Number':[10045,np.nan,1
0065,np.nan,10085],
                              'RecentDelays':[[23,47],[],[24,
43,87],[13],[67,32]],
                              'Airline':['KLM(!)','<Air Franc
e>(12)','(BritishAirways.)','12.Air France',"'Swiss Air'"]})
```

In [3]:

```
df
```

Out[3]:

	From_To	Flight Number	RecentDelays	Airline
0	LoNDon_paris	10045.0	[23, 47]	KLM(!)
1	MAdrid_miLAn	NaN	[]	<Air France>(12)
2	londON_StockhOlm	10065.0	[24, 43, 87]	(BritishAirways.)
3	Budapest_PaRis	NaN	[13]	12.Air France
4	Brussels_londOn	10085.0	[67, 32]	'Swiss Air'

In [4]:

```
df['Flight Number']=df['Flight Number'].interpolate().astype(int)

#interpolate increases the number by 10 with each rows in Flight Number column
#astype changes the float value to int
```

In [5]:

```
df
```

Out[5]:

	From_To	Flight Number	RecentDelays	Airline
0	LoNDon_paris	10045	[23, 47]	KLM(!)
1	MAdrid_miLAn	10055	[]	<Air France>(12)
2	londON_StockhOlm	10065	[24, 43, 87]	(BritishAirways.)
3	Budapest_PaRis	10075	[13]	12.Air France
4	Brussels_londOn	10085	[67, 32]	'Swiss Air'

Qn2. The From*To column would be better as two separate columns! Split each string on the underscore delimiter* to give a new temporary DataFrame with the correct values. Assign the correct column names to this temporary DataFrame.

In [6]:

```
df['From'], df['To'] = df['From_To'].str.split('_').str

#splitting From_To to 2 columns and so df is the new temporary dataframe with correct column names 'From' and 'To'
```

In [7]:

```
df
```

Out[7]:

	From_To	Flight Number	RecentDelays	Airline	From	
0	LoNDon_paris	10045	[23, 47]	KLM(!)	LoNDon	
1	MAdrid_miLAn	10055	[]	<Air France> (12)	MAdrid	
2	londON_StockhOlm	10065	[24, 43, 87]	(BritishAirways.)	londON	Sto
3	Budapest_PaRis	10075	[13]	12.Air France	Budapest	
4	Brussels_londOn	10085	[67, 32]	'Swiss Air'	Brussels	

Qn.3. Notice how the capitalisation of the city names is all mixed up in this temporary DataFrame. Standardise the strings so that only the first letter is uppercase(e.g., "londOn" should become"London".)

In [8]:
```
df['From'] = df['From'].str.capitalize()
```

In [9]:
```
df['To'] = df['To'].str.capitalize()
```

In [10]:
```
df
```

Out[10]:

	From_To	Flight Number	RecentDelays	Airline	From	
0	LoNDon_paris	10045	[23, 47]	KLM(!)	London	
1	MAdrid_miLAn	10055	[]	<Air France> (12)	Madrid	
2	londON_StockhOlm	10065	[24, 43, 87]	(BritishAirways.)	London	Sto
3	Budapest_PaRis	10075	[13]	12.Air France	Budapest	
4	Brussels_londOn	10085	[67, 32]	'Swiss Air'	Brussels	l

Qn.4. Delete the From_To column from df and attach the temporary DataFrame from the previous questions

In [11]:
```
df.drop(["From_To"],axis=1,inplace=True) #deleting the From_T
o column
```

In [12]:
```
df
```

Out[12]:

	Flight Number	RecentDelays	Airline	From	To
0	10045	[23, 47]	KLM(!)	London	Paris
1	10055	[]	<Air France>(12)	Madrid	Milan
2	10065	[24, 43, 87]	(BritishAirways.)	London	Stockholm
3	10075	[13]	12.Air France	Budapest	Paris
4	10085	[67, 32]	'Swiss Air'	Brussels	London

Qn.5. In the RecentDelays column, the values have been entered into the DataFrame as a list. We would like each first value in its column, each second value in its column, and so on. If there isn't an Nth value, the value should be NaN.

Expand the Series of lists into a DataFrame named delays, rename the columns delay_1, delay_2, etc. and replace the unwanted RecentDdelays column in df with delays.

In [13]:

```
df['RecentDelays'] #is the column in the dataframe which is represented as a list
```

Out[13]:

```
0        [23, 47]
1              []
2    [24, 43, 87]
3            [13]
4        [67, 32]
Name: RecentDelays, dtype: object
```

In [14]:

```
df[['delay_1','delay_2','delay_3']]=pd.DataFrame(df['RecentDelays'].values.tolist(), index= df.index)

# using the function .values.tolist() to convert list to different columns and storing it in df
```

In [15]:

```
df
```

Out[15]:

	Flight Number	RecentDelays	Airline	From	To	delay_1	de
0	10045	[23, 47]	KLM(!)	London	Paris	23.0	
1	10055	[]	<Air France> (12)	Madrid	Milan	NaN	
2	10065	[24, 43, 87]	(BritishAirways.)	London	Stockholm	24.0	
3	10075	[13]	12.Air France	Budapest	Paris	13.0	
4	10085	[67, 32]	'Swiss Air'	Brussels	London	67.0	

In [16]:

```
df.drop(["RecentDelays"],axis=1,inplace=True) #deleting the
 'RecentDelays' column
```

In [17]:

```
df
```

Out[17]:

	Flight Number	Airline	From	To	delay_1	delay_2	delay_3
0	10045	KLM(!)	London	Paris	23.0	47.0	NaN
1	10055	<Air France> (12)	Madrid	Milan	NaN	NaN	NaN
2	10065	(BritishAirways.)	London	Stockholm	24.0	43.0	87.0
3	10075	12.Air France	Budapest	Paris	13.0	NaN	NaN
4	10085	'Swiss Air'	Brussels	London	67.0	32.0	NaN

In [ ]:

```

```