Test task
Nazar Karpiuk
Data Engineer position

Task 1. Consider influenza epidemics for 2-person families. The probability is 21% that at least one has disease. The probability that the husband has contracted influenza is 15% while the probability that both the wife and husband have contracted the disease is 10%. What is the probability that the wife has influenza?

Solution:
P(h or w) = 0,21
P(h) = 0,15
P(h and w) = 0,10

This is not mutually exclusive events, so**:**

P(h or w) = P(h) + P(w) - P(h and w)
0,21 = 0,15 + P(w) - 0,10
P(w) = 0,16

Task 2. Here's an example of database structures:

FRUITS_EXPORT (Schema):

seller_info (table):
- seller_id
- fruit_id
- fruit_weight (tons)

consumption_info (table):
- fruit_id
- seller_id
- client_id
- quantity_purchased_fruit (tons)

Write a PostgreSQL query to find the following:
2.1. How many tons worth of fruit does an average seller have?

2.2. How many sellers have at least one client who purchased their fruit?

Solution:

2.1.

```
SELECT AVG(SumWeight) AS AvgWeight
FROM
   (SELECT SUM(fruit_weight) AS SumWeight
   FROM seller_info
   GROUP BY seller_id
   ) AS avg_result;
```

2.2.

```
SELECT COUNT(DISTINCT(seller_id))
FROM consumption_info
WHERE quantity_purchased_fruit > 0;
```

Task 3.

Publish the solution in lucky.py file

We shall call a sequence of successive rolls in the trial a lucky series, if it contains only 5s
and 6s but not all same digits.

For example, 5656556565 is a lucky series, with a length of 10, but 566436 or 55555 not.

Find the longest lucky series in the trial.

If there are more than one “most frequent” lucky series lengths, print the longest.

If there are no lucky series in the trial, print zero.

Example: 4556432455665334 -> 55665.

Solution:

```
# First option

def insert_zeros(seq):
   """delim lucky series by zeros"""
   lst = [int(x) for x in str(seq)]
   for i in range(len(lst)):
      if lst[i] != 5 and lst[i] != 6:
         lst[i] = 0
```

```
    temp = ''.join(str(e) for e in lst)
    return temp

def list_of_seq(str):
    """keep list with not all same digits"""
    un_list = []
    for i in str.split('0'):
        if len(set(i)) > 1:
            un_list.append(i)
    return un_list

def get_longest(lst):
    """get longest lucky series"""
    if len(lst) > 0:
        return max(lst, key=len)
    else:
        return 0

var = input("Enter sequence: ")
print("Longest lucky series:{0}".format(get_longest(list_of_seq(insert_zeros(var)))))

# Second option

import re

def get_lucky_seq(seq):
    """get longest lucky series"""
    temp = re.split(r'[0-47-9]', str(seq))
    temp2 = [i for i in temp if len(set(i)) == 2]
    if len(temp2) > 0:
        return max(temp2, key=len)
    else:
        return 0

var = input("Enter sequence: ")
print("Longest lucky series: {0}".format(get_lucky_seq(var)))
```

Task 4. Complete all tasks listed in the weather_processing.py file

Solution:

weather_processing

November 14, 2022

0.0.1 # TODO Import the necessary libraries

[1]:
```
import pandas as pd
import numpy as np
import os
```

0.0.2 # TODO Import the dataset

[2]:
```
path = r'./data/weather_dataset.data'
```

[3]:
```
data = pd.read_csv(path, sep='\s+')
data.head()
```

[3]:

	Yr	Mo	Dy	loc1	loc2	loc3	loc4	loc5	loc6	loc7	loc8	loc9	loc10	loc11	loc12
0	61	1	1	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1	61	1	2	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
2	61	1	3	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
3	61	1	4	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
4	61	1	5	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83

0.0.3 # TODO Write a function in order to fix date (this relate only to the year info) and apply it

[4]:
```
def fix_year_col(val):
    val = 1900 + val
    return val
```

[5]:
```
#Fixed year
data['Yr'] = data['Yr'].apply(fix_year_col)
```

0.0.4 # TODO Assign it to a variable called data and replace the first 3 columns by a proper datetime index

[6]:
```
data['date']=pd.to_datetime(data[['Yr', 'Mo', 'Dy']].astype(str).agg('-'.join,␣
↪axis=1))
data.drop(['Yr', 'Mo', 'Dy'], axis=1, inplace=True)
data.insert(0, 'date', data.pop('date'))
data.head()
```

[6]:

	date	loc1	loc2	loc3	loc4	loc5	loc6	loc7	loc8	loc9	loc10	loc11	loc12
0	1961-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1	1961-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
2	1961-01-03	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
3	1961-01-04	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
4	1961-01-05	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83

0.0.5 # TODO Check if everything is okay with the data. Create functions to delete/fix rows with strange cases and apply them

- Checking data types

[7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6574 entries, 0 to 6573
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    6574 non-null   datetime64[ns]
 1   loc1    6568 non-null   object
 2   loc2    6571 non-null   object
 3   loc3    6571 non-null   object
 4   loc4    6569 non-null   object
 5   loc5    6572 non-null   object
 6   loc6    6574 non-null   object
```

```
 7   loc7    6571 non-null   object
 8   loc8    6572 non-null   object
 9   loc9    6570 non-null   object
 10  loc10   6573 non-null   object
 11  loc11   6574 non-null   object
 12  loc12   6570 non-null   object
dtypes: datetime64[ns](1), object(12)
memory usage: 667.8+ KB
```

- Change type of columns from 'object' to 'float64'

[8]:
```
cols = data.columns[data.dtypes.eq('object')]
data[cols] = data[cols].apply(pd.to_numeric, errors='coerce')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6574 entries, 0 to 6573
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    6574 non-null   datetime64[ns]
 1   loc1    6567 non-null   float64
 2   loc2    6569 non-null   float64
 3   loc3    6570 non-null   float64
 4   loc4    6567 non-null   float64
 5   loc5    6569 non-null   float64
 6   loc6    6573 non-null   float64
 7   loc7    6570 non-null   float64
 8   loc8    6571 non-null   float64
 9   loc9    6569 non-null   float64
 10  loc10   6572 non-null   float64
 11  loc11   6573 non-null   float64
 12  loc12   6569 non-null   float64
dtypes: datetime64[ns](1), float64(12)
memory usage: 667.8 KB
```

- Created a function to remove abnormal values of wind speed

[9]:
```
def remove_abnormal(df):
    for col in df.columns:
        df[col].mask((df[col]<0) | (df[col]>113), np.nan, inplace=True)
    return df

data.iloc[:, 1:12] = remove_abnormal(data.iloc[:, 1:12])
```

0.0.6 # TODO Set the right dates as the index. Pay attention at the data type, it should be datetime64[ns]

[10]:
```
data = data.set_index('date')
data.head()
```

[10]:
```
             loc1   loc2   loc3   loc4   loc5  loc6   loc7   loc8   loc9  \
date
1961-01-01  15.04  14.96  13.17   9.29    NaN  9.87  13.67  10.25  10.83
1961-01-02  14.71    NaN  10.83   6.50  12.62  7.67  11.50  10.04   9.79
1961-01-03  18.50  16.88  12.33  10.13  11.17  6.17  11.25    NaN   8.50
1961-01-04  10.58   6.63  11.75   4.58   4.54  2.88   8.63   1.79   5.83
1961-01-05  13.33  13.25  11.42   6.17  10.71  8.21  11.92   6.54  10.92

            loc10  loc11  loc12
date
1961-01-01  12.58  18.50  15.04
1961-01-02   9.67  17.54  13.83
1961-01-03   7.67  12.75  12.71
1961-01-04   5.88   5.46  10.88
1961-01-05  10.34  12.92  11.83
```

0.0.7 # TODO Compute how many values are missing for each location over the entire record

[11]:
```
data.isnull().sum()
```

[11]:
```
loc1     7
loc2     5
loc3     4
loc4     7
loc5     5
loc6     1
loc7     4
loc8     3
loc9     7
loc10    2
loc11    1
loc12    5
dtype: int64
```

0.0.8 # TODO Compute how many non-missing values there are in total

[12]:
```
data.notnull().sum().sum()
```

[12]:
```
78837
```

0.0.9 # TODO Calculate the mean windspeeds of the windspeeds over all the locations and all the times

[13]:
```
data.iloc[:, 0:12].sum().sum() / data.iloc[:, 0:12].notna().sum().sum()
```

[13]:
```
10.226930134327791
```

0.0.10 # TODO Create a DataFrame called loc_stats and calculate the min, max and mean windspeeds and standard deviations of the windspeeds at each location over all the days

[14]:
```
loc_stats = data.describe().drop(['count','25%','50%','75%']).
 ↪reindex(['min','max','mean','std'])
loc_stats
```

[14]:

	loc1	loc2	loc3	loc4	loc5	loc6	\
min	0.670000	0.210000	1.000000e-17	0.000000	0.130000	0.000000	
max	35.800000	33.370000	3.384000e+01	28.460000	37.540000	26.160000	
mean	12.362814	10.644645	1.165808e+01	6.304810	10.457089	7.092090	
std	5.618823	5.267991	5.010262e+00	3.606568	4.936183	3.968963	

	loc7	loc8	loc9	loc10	loc11	loc12
min	0.000000	0.000000	0.000000	0.040000	0.130000	0.670000
max	30.370000	31.080000	25.880000	28.210000	42.380000	42.540000
mean	9.794537	8.492586	8.492264	8.707115	13.119168	15.598111
std	4.979986	4.499032	4.167172	4.504263	5.833575	6.699844

0.0.11 # TODO Find the average windspeed in January for each location

[15]:
```
january_mean = pd.DataFrame(data[pd.DatetimeIndex(data.index).month==1].
 ↪reset_index(drop=True).mean(),
                            index=data.columns, columns=['january_mean'])
january_mean
```

[15]:

	january_mean
loc1	14.847325
loc2	12.914560

```
loc3      13.299624
loc4       7.199498
loc5      11.667734
loc6       8.054839
loc7      11.819355
loc8       9.512047
loc9       9.543208
loc10     10.053566
loc11     14.550520
loc12     18.028763
```

0.0.12 # TODO Downsample the record to a yearly frequency for each location

[16]:
```
mean_by_year = data.resample('1Y').mean()
mean_by_year
```

[16]:

date	loc1	loc2	loc3	loc4	loc5	loc6
1961-12-31	12.299583	10.351796	11.314696	6.923722	10.882597	7.729726
1962-12-31	12.246923	10.110438	11.732712	6.960440	10.657918	7.393068
1963-12-31	12.813452	10.836986	12.541151	7.330055	11.724110	8.434712
1964-12-31	12.363661	10.920164	12.104372	6.787787	11.454481	7.570874
1965-12-31	12.451370	11.075534	11.848767	6.858466	11.024795	7.478110
1966-12-31	13.461973	11.557205	12.020630	7.345726	11.805041	7.793671
1967-12-31	12.737151	10.990986	11.739397	7.143425	11.630740	7.368164
1968-12-31	11.835628	10.468197	11.409754	6.477678	10.760765	6.067322
1969-12-31	11.166356	9.723699	10.902000	5.767973	9.873918	6.189973
1970-12-31	12.600329	10.726932	11.730247	6.217178	10.567370	7.609452
1971-12-31	11.273123	9.095178	11.088329	5.241507	9.440329	6.097151
1972-12-31	12.463962	10.561311	12.058333	5.929699	9.430410	6.358825
1973-12-31	11.828466	10.680493	10.680493	5.547863	9.640877	6.548740
1974-12-31	13.643096	11.811781	12.336356	6.427041	11.110986	6.809781
1975-12-31	12.008575	10.293836	11.564712	5.269096	9.190082	5.668521
1976-12-31	11.737842	10.210767	10.761230	5.109426	8.864740	6.311038
1977-12-31	13.098516	11.142940	12.632885	6.079313	10.002720	8.587582
1978-12-31	12.504356	11.044274	11.380000	6.082356	10.167233	7.650658

date	loc7	loc8	loc9	loc10	loc11	loc12
1961-12-31	9.733923	8.858788	8.647652	9.835577	13.502795	13.680773
1962-12-31	11.020712	8.793753	8.316822	9.676247	12.930685	14.323956
1963-12-31	11.075699	10.336548	8.903589	10.224438	13.638877	14.999014
1964-12-31	10.259153	9.467350	7.789016	10.207951	13.740546	14.910301
1965-12-31	10.618712	8.879918	7.907425	9.918082	12.964247	15.591644
1966-12-31	10.579808	8.835096	8.514438	9.768959	14.265836	16.307260
1967-12-31	10.652027	9.325616	8.645014	9.547425	14.774548	17.135945

```
1968-12-31   8.859180   8.255519  7.224945   7.832978  12.808634  15.017486
1969-12-31   8.564493   7.711397  7.924521   7.754384  12.621233  15.762904
1970-12-31   9.609890   8.334630  9.297616   8.289808  13.183644  16.456027
1971-12-31   8.385890   6.757315  7.915370   7.229753  12.208932  15.025233
1972-12-31   9.704508   7.680792  8.357295   7.515273  12.727377  15.028716
1973-12-31   8.482110   7.614274  8.245534   7.812411  12.169699  15.441096
1974-12-31  10.084603   9.896986  9.331753   8.736356  13.252959  16.947671
1975-12-31   8.562603   7.843836  8.797945   7.382822  12.631671  15.307863
1976-12-31   9.149126   7.146202  8.868843   7.876932  12.332377  15.471448
1977-12-31  11.477310   8.333516  9.093984   8.821616  13.426786  16.576099
1978-12-31   9.489342   8.800466  9.089753   8.301699  12.967397  16.771370
```

0.0.13 # TODO Downsample the record to a monthly frequency for each location

[17]:
```
mean_by_month = data.resample('1M').mean()
mean_by_month
```

[17]:
```
                loc1       loc2       loc3      loc4       loc5       loc6  \
date
1961-01-31  14.841333  11.988333  13.431613  7.736774  11.072759   8.588065
1961-02-28  16.269286  14.975357  14.250385  9.230741  13.852143  10.937500
1961-03-31  10.890000  11.296452  10.752903  7.284000  10.507000   8.866774
1961-04-30  10.722667   9.427667   9.998000  5.830667   8.435000   6.495000
1961-05-31   9.860968   8.850000  10.818065  5.905333   9.490323   6.574839
...               ...        ...        ...       ...        ...        ...
1978-08-31   9.645161   8.259355   9.032258  4.502903   7.368065   5.935161
1978-09-30  10.913667  10.895000  10.635000  5.725000  10.372000   9.278333
1978-10-31   9.897742   8.670968   9.295806  4.721290   8.525161   6.774194
1978-11-30  16.151667  14.802667  13.508000  7.317333  11.475000   8.743000
1978-12-31  16.175484  13.748065  15.635161  7.094839  11.398710   9.241613

                loc7       loc8       loc9      loc10      loc11      loc12
date
1961-01-31  11.184839   9.245333   9.085806  10.107419  13.880968  14.703226
1961-02-28  11.890714  11.846071  11.821429  12.714286  18.583214  15.411786
1961-03-31   9.644194   9.829677  10.294138  11.251935  16.410968  15.720000
1961-04-30   6.925333   7.094667   7.342333   7.237000  11.147333  10.278333
1961-05-31   7.604000   8.177097   8.039355   8.499355  11.900323  12.011613
...               ...        ...        ...        ...        ...        ...
1978-08-31   5.650323   5.417742   7.241290   5.536774  10.466774  12.054194
1978-09-30  10.790333   9.583000  10.069333   8.939000  15.680333  19.391333
1978-10-31   8.115484   7.337742   8.297742   8.243871  13.776774  17.150000
1978-11-30  11.492333   9.657333  10.701333  10.676000  17.404667  20.723000
1978-12-31  12.077419  10.194839  10.616774  11.028710  13.859677  21.371613

[216 rows x 12 columns]
```

0.0.14 # TODO Downsample the record to a weekly frequency for each location

[18]:
```
mean_by_week= data.resample('1W').mean()
mean_by_week
```

[18]:

date	loc1	loc2	loc3	loc4	loc5	loc6
1961-01-01	15.040000	14.960000	13.170000	9.290000	NaN	9.870000
1961-01-08	13.541429	11.486667	10.487143	6.417143	9.474286	6.435714
1961-01-15	12.468571	8.967143	11.958571	4.630000	7.351429	5.072857
1961-01-22	13.204286	9.862857	12.982857	6.328571	8.966667	7.417143
1961-01-29	19.880000	16.141429	18.225714	12.720000	17.432857	14.828571
...	...	...	...	...	...	...
1978-12-03	14.934286	11.232857	13.941429	5.565714	10.215714	8.618571
1978-12-10	20.740000	19.190000	17.034286	9.777143	15.287143	12.774286
1978-12-17	16.758571	14.692857	14.987143	6.917143	11.397143	7.272857
1978-12-24	11.155714	8.008571	13.172857	4.004286	7.825714	6.290000
1978-12-31	14.951429	11.801429	16.035714	6.507143	9.660000	8.620000

date	loc7	loc8	loc9	loc10	loc11	loc12
1961-01-01	13.670000	10.250000	10.830000	12.580000	18.500000	15.040000
1961-01-08	11.061429	6.616667	8.434286	8.497143	12.481429	13.238571
1961-01-15	7.535714	6.820000	5.712857	7.571429	11.125714	11.024286
1961-01-22	9.257143	7.875714	7.145714	8.124286	9.821429	11.434286
1961-01-29	15.528571	15.160000	14.480000	15.640000	20.930000	22.530000
...	...	...	...	...	...	...
1978-12-03	9.642857	7.685714	9.011429	9.547143	11.835714	18.728571
1978-12-10	14.437143	12.488571	13.870000	14.082857	18.517143	23.061429
1978-12-17	10.208571	7.967143	9.168571	8.565714	11.102857	15.562857
1978-12-24	7.798571	8.667143	7.151429	8.072857	11.845714	18.977143
1978-12-31	13.708571	10.477143	10.868571	11.471429	12.947143	26.844286

[940 rows x 12 columns]

0.0.15 # TODO Calculate the min, max and mean windspeeds and standard deviations of the windspeeds across all locations for each week (assume that the first week starts on January 2 1961) for the first 21 weeks

[19]:
```
week_stat = data.loc['1961-01-02':, :].resample('1W').
 ↪agg(['min','max','mean','std'])[:21]
week_stat
```

[19]:

	loc1				loc2			
	min	max	mean	std	min	max	mean	

date							
1961-01-08	10.58	18.50	13.541429	2.631321	6.63	16.88	11.486667
1961-01-15	9.04	19.75	12.468571	3.555392	3.54	12.08	8.967143
1961-01-22	4.92	19.83	13.204286	5.337402	3.42	14.37	9.862857
1961-01-29	13.62	25.04	19.880000	4.619061	9.96	23.91	16.141429
1961-02-05	10.58	24.21	16.827143	5.251408	9.46	24.21	15.460000
1961-02-12	16.00	24.54	19.684286	3.587677	11.54	21.42	16.417143
1961-02-19	6.04	22.50	15.130000	5.064609	11.63	20.17	15.091429
1961-02-26	7.79	25.80	15.221429	7.020716	7.08	21.50	13.625714
1961-03-05	10.96	13.33	12.101429	0.997721	8.83	17.00	12.951429
1961-03-12	4.88	14.79	9.376667	3.732263	8.08	16.96	11.578571
1961-03-19	4.92	16.88	11.911429	3.860036	9.46	15.54	13.501429
1961-03-26	6.29	15.00	9.567143	3.613298	2.58	11.63	8.387143
1961-04-02	5.88	18.25	10.757143	5.046922	3.50	16.29	8.852857
1961-04-09	4.50	18.12	11.964286	4.604392	7.04	14.62	10.654286
1961-04-16	4.71	15.50	8.965714	3.937727	4.83	12.25	8.000000
1961-04-23	4.00	21.09	12.621429	5.676655	3.71	15.41	10.438571
1961-04-30	4.08	16.29	10.117143	4.349662	6.50	14.46	9.798571
1961-05-07	9.87	23.00	15.367143	5.025507	10.29	19.79	13.970000
1961-05-14	3.54	12.79	7.772857	3.371022	3.96	15.12	8.712857
1961-05-21	4.88	15.04	8.225714	3.631730	3.58	10.17	5.631667
1961-05-28	4.96	11.79	8.155714	2.739433	3.67	12.50	7.388571

	loc3			...	loc10		loc11		\
	std	min	max	...	mean	std	min	max	
date				...					
1961-01-08	3.949525	7.62	12.33	...	8.497143	1.704941	5.46	17.54	
1961-01-15	3.148945	7.08	19.50	...	7.571429	4.084293	5.25	20.71	
1961-01-22	3.837785	7.29	20.79	...	8.124286	4.783952	6.50	15.92	
1961-01-29	5.170224	12.67	25.84	...	15.640000	3.713368	14.04	27.71	
1961-02-05	5.187395	9.04	19.70	...	9.460000	2.839501	9.17	19.33	
1961-02-12	3.608373	13.67	21.34	...	14.440000	1.746749	15.21	26.38	
1961-02-19	3.575012	6.13	17.25	...	13.542857	2.531361	14.09	29.63	
1961-02-26	5.147348	6.08	22.42	...	12.730000	4.920064	9.59	23.21	
1961-03-05	2.851955	8.17	13.67	...	12.370000	1.593685	11.58	23.45	
1961-03-12	3.230167	7.54	16.38	...	10.458571	3.655113	10.21	22.71	
1961-03-19	2.352867	5.25	13.96	...	11.627143	3.099472	11.29	22.79	
1961-03-26	3.657265	4.79	15.63	...	11.481429	2.538224	8.25	21.34	
1961-04-02	4.687315	5.09	14.96	...	9.631429	3.191115	7.21	18.63	
1961-04-09	2.845399	9.29	18.29	...	7.238571	2.336182	7.62	17.16	
1961-04-16	2.607118	3.92	15.79	...	6.178571	2.161137	5.75	16.17	
1961-04-23	4.631736	3.33	17.00	...	9.551429	3.347972	6.75	19.21	
1961-04-30	2.871425	2.54	14.96	...	6.124286	2.840568	5.13	13.04	
1961-05-07	3.750835	8.42	21.21	...	11.585714	3.620819	4.79	28.08	
1961-05-14	3.782947	4.63	12.33	...	7.822857	5.460237	6.54	18.66	
1961-05-21	2.468906	5.91	15.96	...	7.114286	2.216889	6.63	12.00	
1961-05-28	3.378537	3.58	20.96	...	7.535714	2.575661	6.13	14.33	

			loc12			
	mean	std	min	max	mean	std
date						
1961-01-08	12.481429	4.349139	10.88	16.46	13.238571	1.773062
1961-01-15	11.125714	5.552215	5.17	16.92	11.024286	4.692355
1961-01-22	9.821429	3.626584	6.79	17.96	11.434286	4.237239
1961-01-29	20.930000	5.210726	17.50	27.63	22.530000	3.874721
1961-02-05	14.012857	4.210858	7.17	19.25	11.935714	4.336104
1961-02-12	21.832857	4.063753	17.04	21.84	19.155714	1.828705
1961-02-19	21.167143	5.910938	10.96	22.58	16.584286	4.685377
1961-02-26	16.304286	5.091162	6.67	23.87	14.322857	6.182283
1961-03-05	17.842857	4.332331	8.83	17.54	13.951667	3.021387
1961-03-12	16.701429	4.358759	5.54	22.54	14.420000	5.769890
1961-03-19	19.350000	3.779727	11.34	22.95	16.227143	4.331958
1961-03-26	14.037143	4.318069	13.13	22.50	18.134286	3.701846
1961-04-02	13.471429	4.179854	7.17	19.58	13.900000	3.924555
1961-04-09	11.712857	3.147781	7.21	15.34	11.371429	2.598271
1961-04-16	9.482857	3.641464	5.66	12.87	8.690000	2.747842
1961-04-23	13.620000	4.735096	4.96	20.46	12.470000	5.908542
1961-04-30	9.720000	2.948237	2.67	17.50	8.637143	5.108365
1961-05-07	17.548571	8.003490	3.83	26.58	14.571429	7.728504
1961-05-14	10.421429	3.968272	3.33	26.30	10.382857	7.858246
1961-05-21	9.624286	1.975853	5.91	14.96	10.612857	3.310819
1961-05-28	10.518571	3.024524	8.00	17.04	11.697143	3.811818

[21 rows x 48 columns]

Task 5. Publish the solution in the sum_in_array.py file
Input: sorted array and number S
Output: xi and xj values as list from array that in sum gives S (xi + xj =S)

- If digits can't be found, return [–1].
- If there is more than one solution pair return any of them.
  Example: [-3, 1, 4, 6] S=7 -> [1, 6]
  [-3, 1, 4, 6] S=8 -> [-1]

Find the most optimal algorithm by speed and memory.

Solution:

```
# First option

def find_pairs_v1(lst, S):
    res = []
    while lst:
        temp = lst.pop()
        diff = S - temp
        if diff in lst:
            res = ((diff, temp))
            return res
    return -1

# Second option

def find_pairs_v2(lst, S):
    res = []
    for i in range(0, len(lst) - 1):
        for j in range(i + 1, len(lst)):
            if (lst[i] + lst[j] == S):
                res = ((lst[i],lst[j]))
                return res
    return -1

# Third option - BEST ONE

def find_pairs_v3(lst, S):
    res = []
    lst.sort()
```

```
(low, high) = (0, len(lst) - 1)
while low < high:
    if lst[low] + lst[high] == S:
        res  = ((lst[low], lst[high]))
        return res
    if lst[low] + lst[high] < S:
        low = low + 1
    else:
        high = high - 1
return -1
```