Lab 2 - Data Wrangling

Rushabh Barbhaya

- Recall from our last on-campus meeting, I asked you to do any 50 problems from the following website. https://github.com/ajcr/100-pandas-puzzles/blob/master/100-pandas-puzzles.ipynb
- Your deliverable is a single Python script. Please ensure you clearly mark your questions with the question number from the website and document your code through ample comments. Don't make me guess what you're trying to do- tell me.

--- This submission is in the form of a python notebook

Question 1. import pandas with alian `pd`

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

Qustion 2. print the current version of pandas that is being imported

In [ ]:
```
print(f"Pandas version :: {pd.__version__}")
```

```
Pandas version :: 1.4.2
```

Note: remember to import numpy using:

```
import numpy as np
```

Consider the following Python dictionary `data` and Python list `labels`:

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat',
'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no',
'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

(This is just some meaningless data I made up with the theme of animals and trips to a vet.)

Question 3. Create a DataFrame `df` from this dictionary `data` which has the index `labels`.

In [ ]:
```
import numpy as np

data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'nc

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(data, index=labels)
print(df)
```

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  2.0       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no
```

Question 4. Display a summary of the basic information about this DataFrame and its data

In [ ]:
```
print("\nCore information about the dataframe")
print(df.info()) # Information about the dataframe
print("\nBasic stats about the dataset")
print(df.describe()) # Stats on the dataframe
```

```
Core information about the dataframe
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   animal    10 non-null     object
 1   age       8 non-null      float64
 2   visits    10 non-null     int64
 3   priority  10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
None

Basic stats about the dataset
             age     visits
count  8.000000  10.000000
mean   3.437500   1.900000
std    2.007797   0.875595
min    0.500000   1.000000
25%    2.375000   1.000000
50%    3.000000   2.000000
75%    4.625000   2.750000
max    7.000000   3.000000
```

Question 5. Display the first 3 rows of the DataFrame

In [ ]:
```
print(df.head(3))
```

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
```

Question 6. Dispay the last 2 rows of the DataFrame

In [ ]:
```
print(df.tail(2))
```

```
  animal  age  visits priority
i    dog  7.0       2       no
j    dog  3.0       1       no
```

Question 7. Select just the 'animals' and 'age' columns from the DataFrame

In [ ]:
```
print(df[['animal', 'age']])
```

```
  animal  age
a    cat  2.5
b    cat  3.0
c  snake  0.5
d    dog  NaN
e    dog  5.0
f    cat  2.0
g  snake  4.5
h    cat  NaN
i    dog  7.0
j    dog  3.0
```

Question 8. Select the data in rows `[3, 4, 8]` *and* in columns `['animal', 'age']`.

In [ ]:
```
print(df.loc[df.index[[3,4,8]], ['animal', 'age']])
```

```
  animal  age
d    dog  NaN
e    dog  5.0
i    dog  7.0
```

Question 9. Select only the rows where the number of visits is greater than 2

In [ ]:
```
print(df[df['visits'] > 2])
```

```
  animal  age  visits priority
b    cat  3.0       3      yes
d    dog  NaN       3      yes
f    cat  2.0       3       no
```

Question 10. Select the rows where the age is `null` i.e. `NaN`

In [ ]:
```
print(df[df['age'].isnull()])
```

```
  animal  age  visits priority
d    dog  NaN       3      yes
h    cat  NaN       1      yes
```

Question 11. Select the rows where the animal is a cat *and* the age is less than 3.

In [ ]:
```
print(df[(df['animal'] == 'cat') & (df['age'] < 3)])
```

```
  animal  age  visits priority
a    cat  2.5       1      yes
f    cat  2.0       3       no
```

Question 12. Select the rows the age is between 2 and 4 (inclusive).

In [ ]:
```
print(df[df['age'].between(2, 4)])
```

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
f    cat  2.0       3       no
j    dog  3.0       1       no
```

Question 13. Change the age in row 'f' to 1.5

In [ ]:
```
df.loc['f', 'age'] = 1.5
print(df)
```

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  1.5       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no
```

Question 14. Caculate the sum of all visits in DataFrame

In [ ]:
```
print(f"Sum of 'visits' column :: {df['visits'].sum()}")
```

```
Sum of 'visits' column :: 19
```

Question 15. Calculate the mean age for each different animal in DataFrame

In [ ]:
```
print(df.groupby('animal')['age'].mean())
```

```
animal
cat      2.333333
dog      5.000000
snake    2.500000
Name: age, dtype: float64
```

Question 16. Append a new row 'k' to DataFrame with your choice of values for each column. then delete that row.

In [ ]:
```
df.loc['k'] = ['crow', 3, 1, 'no']
print(df)

df = df.drop('k')
print(df)
```

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  1.5       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no
k   crow  3.0       1       no
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  1.5       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no
```

Question 17. Count the number of each type of animal in the DataFrame

In [ ]:
```
print(df['animal'].value_counts())
```

```
cat      4
dog      4
snake    2
Name: animal, dtype: int64
```

Question 18. Sort `df` first by the values in the 'age' in *decending* order, then by the value in the 'visits' column in *ascending* order

In [ ]:
```
print(df.sort_values(by=['age', 'visits'], ascending=[False, True]))
```

```
  animal  age  visits priority
i    dog  7.0       2       no
e    dog  5.0       2       no
g  snake  4.5       1       no
j    dog  3.0       1       no
b    cat  3.0       3      yes
a    cat  2.5       1      yes
f    cat  1.5       3       no
c  snake  0.5       2       no
h    cat  NaN       1      yes
d    dog  NaN       3      yes
```

Question 19. The 'priority' column contains the values 'yes' and 'no'. Replace this column with a column of boolean values: 'yes' should be `True` and 'no' should be `False`.

In [ ]:
```
df['priority'] = df['priority'].map({'yes': True, 'no': False})
print(df)
```

```
  animal  age  visits  priority
a    cat  2.5       1      True
b    cat  3.0       3      True
c  snake  0.5       2     False
d    dog  NaN       3      True
e    dog  5.0       2     False
f    cat  1.5       3     False
g  snake  4.5       1     False
h    cat  NaN       1      True
i    dog  7.0       2     False
j    dog  3.0       1     False
```

Question 20. Replace the 'animal' from 'snake' to 'python'

In [ ]:
```
df['animal'] = df['animal'].replace('snake', 'python')
print(df)
```

```
   animal  age  visits  priority
a     cat  2.5       1      True
b     cat  3.0       3      True
c  python  0.5       2     False
d     dog  NaN       3      True
e     dog  5.0       2     False
f     cat  1.5       3     False
g  python  4.5       1     False
h     cat  NaN       1      True
i     dog  7.0       2     False
j     dog  3.0       1     False
```

Question 21. For each animal type and each number of visits, find the mean age. In other words, each row is an animal, each column is a number of visits and the values are the mean ages (*hint: use a pivot table*).

In [ ]:
```
print(df.pivot_table(index='animal', columns='visits', values='age', aggfunc
```

```
visits    1    2     3
animal
cat     2.5  NaN  2.25
dog     3.0  6.0   NaN
python  4.5  0.5   NaN
```

Question 22. You have a DataFrame `df` with a column 'A' of integers. For example:

```
df = pd.DataFrame({'A': [1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 7]})
```

How do you filter out rows which contain the same integer as the row immediately above?

In [ ]:
```
df = pd.DataFrame({'A': [1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 7]})
df = df.loc[df['A'].shift() != df['A']]
# df.drop_duplicates(subset='A')
print(df)
```

```
   A
0  1
1  2
3  3
4  4
5  5
8  6
9  7
```

Question 23. Given a DataFrame of random numeric values:

```
df = pd.DataFrame(np.random.random(size=(5, 3))) # this is a 5x3
DataFrame of float values
```

how do you subtract the row mean from each element in the row?

In [ ]:
```
df = pd.DataFrame(np.random.random(size=(5, 3)))
print(df.sub(df.mean(axis=1), axis=0))
```

```
          0         1         2
0  0.369472  0.069714 -0.439186
1  0.228905 -0.537328  0.308423
2  0.066168  0.341439 -0.407606
3  0.480628 -0.177324 -0.303304
4 -0.268371  0.276157 -0.007787
```

Question 24. Suppose you have DataFrame with 10 columns of real numbers, for example:

```
df = pd.DataFrame(np.random.random(size=(5, 10)),
columns=list('abcdefghij'))
```

Which column of numbers has the smallest sum? Return that column's label.

In [ ]:
```
df = pd.DataFrame(np.random.random(size=(5, 10)), columns=list('abcdefghij'))
print(df.sum().idxmin())
```

```
j
```

Question 25. How do you count how many unique rows a DataFrame has (i.e. ignore all rows that are duplicates)?

In [ ]:
```
df = pd.DataFrame(np.random.randint(0, 2, size=(10, 3)))
print(len(df.drop_duplicates(keep=False)))
```

```
4
```

Question 26. In the cell below, you have a DataFrame `df` that consists of 10 columns of floating-point numbers. Exactly 5 entries in each row are NaN values.

For each row of the DataFrame, find the *column* which contains the *third* NaN value.

You should return a Series of column labels: `e, c, d, h, d`

In [ ]:
```
nan = np.nan
data = [[0.04,  nan,  nan, 0.25,  nan, 0.43, 0.71, 0.51,  nan,  nan],
        [ nan,  nan,  nan, 0.04, 0.76,  nan,  nan, 0.67, 0.76, 0.16],
        [ nan,  nan, 0.5 ,  nan, 0.31, 0.4 ,  nan,  nan, 0.24, 0.01],
        [0.49,  nan,  nan, 0.62, 0.73, 0.26, 0.85,  nan,  nan,  nan],
        [ nan,  nan, 0.41,  nan, 0.05,  nan, 0.61,  nan, 0.48, 0.68]]

columns = list('abcdefghij')

df = pd.DataFrame(data, columns=columns)

print((df.isnull().cumsum(axis=1) == 3).idxmax(axis=1))
```

```
0    e
1    c
2    d
3    h
4    d
dtype: object
```

Question 27. A DataFrame has a column of groups 'grps' and and column of integer values 'vals':

```
df = pd.DataFrame({'grps': list('aaabbcaabcccbbc'),
                   'vals':
[12,345,3,1,45,14,4,52,54,23,235,21,57,3,87]})
```

For each *group*, find the sum of the three greatest values. You should end up with the answer as follows:

```
grps
a    409
b    156
c    345
```

In [ ]:
```
df = pd.DataFrame({
    'grps': list('aaabbcaabcccbbc'),
    'vals': [12,345,3,1,45,14,4,52,54,23,235,21,57,3,87]
})
print(df.groupby('grps')['vals'].nlargest(3).sum(level=0))
```

```
grps
a    409
b    156
c    345
Name: vals, dtype: int64
```

```
/var/folders/hr/9j_kdl5j39n67m7ssgzjq_vr0000gn/T/ipykernel_50637/1395436563.
py:5: FutureWarning: Using the level keyword in DataFrame and Series aggrega
tions is deprecated and will be removed in a future version. Use groupby ins
tead. df.sum(level=1) should use df.groupby(level=1).sum().
  print(df.groupby('grps')['vals'].nlargest(3).sum(level=0))
```

Question 28. The DataFrame `df` constructed below has two integer columns 'A' and 'B'. The values in 'A' are between 1 and 100 (inclusive).

For each group of 10 consecutive integers in 'A' (i.e. `[0, 10]`, `[10, 20]`, ...), calculate the sum of the corresponding values in column 'B'.

The answer should be a Series as follows:

```
A
(0, 10]      635
(10, 20]     360
(20, 30]     315
(30, 40]     306
(40, 50]     750
(50, 60]     284
(60, 70]     424
(70, 80]     526
(80, 90]     835
(90, 100]    852
```

In [ ]:
```
df = pd.DataFrame(np.random.RandomState(8765).randint(1, 101, size=(100, 2)))
print(df.groupby(pd.cut(df['A'], np.arange(0, 101, 10)))['B'].sum())
```

```
A
(0, 10]      635
(10, 20]     360
(20, 30]     315
(30, 40]     306
(40, 50]     750
(50, 60]     284
(60, 70]     424
(70, 80]     526
(80, 90]     835
(90, 100]    852
Name: B, dtype: int64
```

Question 29. Consider a DataFrame `df` where there is an integer column 'X':

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

For each value, count the difference back to the previous zero (or the start of the Series, whichever is closer). These values should therefore be

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

Make this a new column 'Y'.

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

izero = np.r_[-1, (df == 0).values.nonzero()[0]]  # indices of zeros
idx = np.arange(len(df))
y = df['X'] != 0
df['Y'] = idx - izero[np.searchsorted(izero - 1, idx) - 1]
print(df)

# http://stackoverflow.com/questions/30730981/how-to-count-distance-to-the-p
# credit: Behzad Nouri
```

```
   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
```

Question 30. Consider the DataFrame constructed below which contains rows and columns of numerical data.

Create a list of the column-row index locations of the 3 largest values in this DataFrame. In this case, the answer should be:

```
[(5, 7), (6, 4), (2, 5)]
```

In [ ]:
```
df = pd.DataFrame(np.random.RandomState(30).randint(1, 101, size=(8, 8)))
print(df.unstack().sort_values()[-3:].index.tolist())
print(df)

# http://stackoverflow.com/questions/14941261/index-and-column-for-the-max-v
# credit: DSM
```

```
[(5, 7), (6, 4), (2, 5)]
    0   1   2   3   4   5   6   7
0  38  38  46  46  13  24   3  54
1  18  47   4  42   8  66  50  46
2  62  36  19  19  77  17   7  63
3  28  47  46  65  63  12  16  24
4  14  51  34  56  29  59  92  79
5  58  76  96  45  38  76  58  40
6  10  34  48  40  37  23  41  26
7  55  70  91  27  79  92  20  31
```

Question 31. You are given the DataFrame below with a column of group IDs, 'grps', and a column of corresponding integer values, 'vals'.

```
df = pd.DataFrame({"vals": np.random.RandomState(31).randint(-30, 30, size=15),
                   "grps": np.random.RandomState(31).choice(["A", "B"], 15)})
```

Create a new column 'patched_values' which contains the same values as the 'vals' any negative values in 'vals' with the group mean:

	vals	grps	patched_vals
0	-12	A	13.6
1	-7	B	28.0
2	-14	A	13.6
3	4	A	4.0
4	-7	A	13.6
5	28	B	28.0
6	-2	A	13.6
7	-1	A	13.6
8	8	A	8.0
9	-2	B	28.0
10	28	A	28.0
11	12	A	12.0
12	16	A	16.0
13	-24	A	13.6
14	-12	A	13.6

In [ ]:
```
df = pd.DataFrame({"vals": np.random.RandomState(31).randint(-30, 30, size=1
                   "grps": np.random.RandomState(31).choice(["A", "B"], 15)}

def replace(group):
    mask = group<0
    group[mask] = group[~mask].mean()
    return group

ans = df.groupby(['grps'])['vals'].transform(replace)
print(ans)

# http://stackoverflow.com/questions/14760757/replacing-values-with-groupby-
# credit: unutbu
```

```
0     13.6
1     28.0
2     13.6
3      4.0
4     13.6
5     28.0
6     13.6
7     13.6
8      8.0
9     28.0
10    28.0
11    12.0
12    16.0
13    13.6
14    13.6
Name: vals, dtype: float64
```

Question 32. Implement a rolling mean over groups with window size 3, which ignores NaN value. For example consider the following DataFrame:

```
>>> df = pd.DataFrame({'group': list('aabbabbbabab'),
                       'value': [1, 2, 3, np.nan, 2, 3, np.nan, 1,
7, 3, np.nan, 8]})
>>> df
   group  value
0      a    1.0
1      a    2.0
2      b    3.0
3      b    NaN
4      a    2.0
5      b    3.0
6      b    NaN
7      b    1.0
8      a    7.0
9      b    3.0
10     a    NaN
11     b    8.0
```

The goal is to compute the Series:

```
0     1.000000
1     1.500000
2     3.000000
3     3.000000
4     1.666667
5     3.000000
6     3.000000
7     2.000000
8     3.666667
9     2.000000
10    4.500000
11    4.000000
```

E.g. the first window of size three for group 'b' has values 3.0, NaN and 3.0 and occurs at row index 5. Instead of being NaN the value in the new column at this row index should be 3.0 (just the two non-NaN values are used to compute the mean (3+3)/2)

In [ ]:
```
df = pd.DataFrame({'group': list('aabbabbbabab'),
                   'value': [1, 2, 3, np.nan, 2, 3, np.nan, 1, 7, 3, np.nan,

g1 = df.groupby(['group'])['value']              # group values
g2 = df.fillna(0).groupby(['group'])['value']    # fillna, then group values

s = g2.rolling(3, min_periods=1).sum() / g1.rolling(3, min_periods=1).count(
# print(s)
s = s.reset_index(level=0, drop=True).sort_index()  # drop/sort index
print(s)

# http://stackoverflow.com/questions/36988123/pandas-groupby-and-rolling-app
```

```
0     1.000000
1     1.500000
2     3.000000
3     3.000000
4     1.666667
5     3.000000
6     3.000000
7     2.000000
8     3.666667
9     2.000000
10    4.500000
11    4.000000
Name: value, dtype: float64
```

Question 33. Create a DatetimeIndex that contains each business day of 2015 and use it to index a Series of random numbers. Let's call this Series `s`.

In [ ]:
```
dti = pd.date_range(start='2015-01-01', end='2015-12-31', freq='B')
s = pd.Series(np.random.rand(len(dti)), index=dti)
print(s)
```

```
2015-01-01    0.662274
2015-01-02    0.719034
2015-01-05    0.705927
2015-01-06    0.008031
2015-01-07    0.346480
                ...
2015-12-25    0.126708
2015-12-28    0.267464
2015-12-29    0.167496
2015-12-30    0.923966
2015-12-31    0.992976
Freq: B, Length: 261, dtype: float64
```

Question 34. Find the sum of the values in `s` for every Wednesday.

In [ ]:
```
print(s[s.index.weekday == 2].sum())
```

```
27.02626505885619
```

Question 35. For each calendar month in `s`, find the mean of values.

In [ ]:
```
print(s.resample('M').mean())
```

```
2015-01-31    0.539673
2015-02-28    0.564849
2015-03-31    0.514948
2015-04-30    0.589662
2015-05-31    0.434331
2015-06-30    0.550838
2015-07-31    0.454647
2015-08-31    0.511972
2015-09-30    0.580716
2015-10-31    0.475675
2015-11-30    0.501651
2015-12-31    0.529344
Freq: M, dtype: float64
```

Question 36. For each group of four consecutive calendar months in `s`, find the date on which the highest value occurred.

In [ ]:
```
print(s.groupby(pd.Grouper(freq='4M')).idxmax())
```

```
2015-01-31   2015-01-22
2015-05-31   2015-02-03
2015-09-30   2015-08-18
2016-01-31   2015-12-31
Freq: 4M, dtype: datetime64[ns]
```

Question 37. Create a DateTimeIndex consisting of the third Thursday in each month for the years 2015 and 2016.

In [ ]:
```
qq = pd.date_range('2015-01-01', '2016-12-31', freq='WOM-3THU')
print(qq)
```

```
DatetimeIndex(['2015-01-15', '2015-02-19', '2015-03-19', '2015-04-16',
               '2015-05-21', '2015-06-18', '2015-07-16', '2015-08-20',
               '2015-09-17', '2015-10-15', '2015-11-19', '2015-12-17',
               '2016-01-21', '2016-02-18', '2016-03-17', '2016-04-21',
               '2016-05-19', '2016-06-16', '2016-07-21', '2016-08-18',
               '2016-09-15', '2016-10-20', '2016-11-17', '2016-12-15'],
              dtype='datetime64[ns]', freq='WOM-3THU')
```

Question 38. Some values in the the **FlightNumber** column are missing (they are `NaN`). These numbers are meant to increase by 10 with each row so 10055 and 10075 need to be put in place. Modify `df` to fill in these missing numbers and make the column an integer column (instead of a float column).

```
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN',
'londON_StockhOlm',
                               'Budapest_PaRis',
'Brussels_londOn'],
                   'FlightNumber': [10045, np.nan, 10065, np.nan,
10085],
                   'RecentDelays': [[23, 47], [], [24, 43, 87], [13],
[67, 32]],
                        'Airline': ['KLM(!)', '<Air France> (12)',
'(British Airways. )',
                               '12. Air France', '"Swiss Air"']})
```

In [ ]:
```
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_Stock
                               'Budapest_PaRis', 'Brussels_londOn'],
                   'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
                   'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
                        'Airline': ['KLM(!)', '<Air France> (12)', '(British Airw
                               '12. Air France', '"Swiss Air"']})

df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)
print(df)
```

```
            From_To  FlightNumber  RecentDelays               Airline
0      LoNDon_paris         10045      [23, 47]                KLM(!)
1      MAdrid_miLAN         10055            []    <Air France> (12)
2  londON_StockhOlm         10065  [24, 43, 87]  (British Airways. )
3    Budapest_PaRis         10075          [13]       12. Air France
4   Brussels_londOn         10085      [67, 32]           "Swiss Air"
```

Question 39. The **From_To** column would be better as two separate columns! Split each string on the underscore delimiter `_` to give a new temporary DataFrame called 'df2' with the correct values. Assign the correct column names 'From' and 'To' to this temporary DataFrame.

In [ ]:
```
df2 = df.From_To.str.split('_', expand=True)
df2.columns = ['From', 'To']
print(df2)
```

```
       From         To
0    LoNDon      paris
1    MAdrid      miLAN
2    londON  StockhOlm
3  Budapest      PaRis
4  Brussels     londOn
```

Question 40. Notice how the capitalisation of the city names is all mixed up in this temporary DataFrame 'temp'. Standardise the strings so that only the first letter is uppercase (e.g. "londON" should become "London".)

In [ ]:
```
df2['From'] = df2['From'].str.capitalize()
df2['To'] = df2['To'].str.capitalize()
print(df2)
```

```
       From         To
0    London      Paris
1    Madrid      Milan
2    London  Stockholm
3  Budapest      Paris
4  Brussels     London
```

Question 41. Delete the From_To column `df` and attach the temporary dataset to the main df

In [ ]:
```
df = df.drop('From_To', axis=1)
df = df.join(df2)
print(df)
```

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

Question 42. In the **Airline** column, you can see some extra puctuation and symbols have appeared around the airline names. Pull out just the airline name. E.g. `'(British Airways. )'` should become `'British Airways'`.

In [ ]:
```
df['Airline'] = df['Airline'].str.extract('([a-zA-Z\s]+)', expand=False).str
print(df)
```

```
   FlightNumber  RecentDelays          Airline      From         To
0         10045      [23, 47]              KLM    London      Paris
1         10055            []       Air France    Madrid      Milan
2         10065  [24, 43, 87]  British Airways    London  Stockholm
3         10075          [13]       Air France  Budapest      Paris
4         10085      [67, 32]        Swiss Air  Brussels     London
```

Question 43. In the **RecentDelays** column, the values have been entered into the DataFrame as a list. We would like each first value in its own column, each second value in its own column, and so on. If there isn't an Nth value, the value should be NaN.

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

In [ ]:
```
delays = df['RecentDelays'].apply(pd.Series)
delays.columns = [f'delay_{n+1}' for n in range(len(delays.columns))]
df = df.drop('RecentDelays', axis=1).join(delays)
print(df)
```

```
   FlightNumber          Airline      From          To  delay_1  delay_2  \
0         10045              KLM    London       Paris     23.0     47.0
1         10055       Air France    Madrid       Milan      NaN      NaN
2         10065  British Airways    London  Stockholm     24.0     43.0
3         10075       Air France  Budapest       Paris     13.0      NaN
4         10085        Swiss Air  Brussels      London     67.0     32.0

   delay_3
0      NaN
1      NaN
2     87.0
3      NaN
4      NaN
```

```
/var/folders/hr/9j_kdl5j39n67m7ssgzjq_vr0000gn/T/ipykernel_50637/1085007710.
py:1: FutureWarning: The default dtype for empty Series will be 'object' ins
tead of 'float64' in a future version. Specify a dtype explicitly to silence
this warning.
  delays = df['RecentDelays'].apply(pd.Series)
```

Question 44. Given the lists `letters = ['A', 'B', 'C']` and `numbers = list(range(10))`, construct a MultiIndex object from the product of the two lists. Use it to index a Series of random numbers. Call this Series `s`.

In [ ]:
```
letters = ['A', 'B', 'C']
numbers = list(range(10))

mi = pd.MultiIndex.from_product([letters, numbers])
s = pd.Series(np.random.rand(30), index=mi)
print(s)
```

```
A  0    0.486294
   1    0.807001
   2    0.445930
   3    0.283969
   4    0.459008
   5    0.938689
   6    0.091611
   7    0.903428
   8    0.197921
   9    0.538806
B  0    0.159563
   1    0.573046
   2    0.775068
   3    0.080871
   4    0.585616
   5    0.679605
   6    0.642874
   7    0.420898
   8    0.912838
   9    0.249856
C  0    0.681370
   1    0.787750
   2    0.944011
   3    0.874280
   4    0.727944
   5    0.510081
   6    0.365921
   7    0.807776
   8    0.918047
   9    0.542862
dtype: float64
```

Question 45. Check the index of `s` is lexicographically sorted (this is a necessary proprty for indexing to work correctly with a MultiIndex).

In [ ]:
```
print(s.index.is_lexsorted())
```

```
True
/var/folders/hr/9j_kdl5j39n67m7ssgzjq_vr0000gn/T/ipykernel_50637/3463388138.
py:1: FutureWarning: MultiIndex.is_lexsorted is deprecated as a public funct
ion, users should use MultiIndex.is_monotonic_increasing instead.
  print(s.index.is_lexsorted())
```

Question 46. Select the labels `1`, `3` and `6` from the second level of the MultiIndexed Series.

In [ ]:
```
print(s.loc[:, [1,3,6]])
```

```
A  1    0.807001
   3    0.283969
   6    0.091611
B  1    0.573046
   3    0.080871
   6    0.642874
C  1    0.787750
   3    0.874280
   6    0.365921
dtype: float64
```

Question 47. Slice the Series `s`; slice up to label 'B' for the first level and from label 5 onwards for the second level.

In [ ]:
```
print(s.loc[pd.IndexSlice[:'B', 5:]])
```

```
A  5    0.938689
   6    0.091611
   7    0.903428
   8    0.197921
   9    0.538806
B  5    0.679605
   6    0.642874
   7    0.420898
   8    0.912838
   9    0.249856
dtype: float64
```

Question 48. Sum the values in `s` for each label in the first level (you should have Series giving you a total for labels A, B and C).

In [ ]:
```
print(s.groupby(level=0).sum())
```

```
A    5.152659
B    5.080236
C    7.160043
dtype: float64
```

Question 49. Suppose that `sum()` (and other methods) did not accept a `level` keyword argument. How else could you perform the equivalent of `s.sum(level=1)`?

In [ ]:
```
print(s.unstack().sum(axis=0))
```

```
0    1.327227
1    2.167798
2    2.165009
3    1.239120
4    1.772568
5    2.128375
6    1.100406
7    2.132103
8    2.028806
9    1.331524
dtype: float64
```

Question 50. Exchange the levels of the MultiIndex so we have an index of the form (letters, numbers). Is this new Series properly lexsorted? If not, sort it.

In [ ]:
```
new_s = s.swaplevel(0, 1)
if not new_s.index.is_lexsorted():
    new_s = new_s.sort_index()

print(new_s)
```

```
0  A    0.486294
   B    0.159563
   C    0.681370
1  A    0.807001
   B    0.573046
   C    0.787750
2  A    0.445930
   B    0.775068
   C    0.944011
3  A    0.283969
   B    0.080871
   C    0.874280
4  A    0.459008
   B    0.585616
   C    0.727944
5  A    0.938689
   B    0.679605
   C    0.510081
6  A    0.091611
   B    0.642874
   C    0.365921
7  A    0.903428
   B    0.420898
   C    0.807776
8  A    0.197921
   B    0.912838
   C    0.918047
9  A    0.538806
   B    0.249856
   C    0.542862
dtype: float64
```

```
/var/folders/hr/9j_kdl5j39n67m7ssgzjq_vr0000gn/T/ipykernel_50637/360456438.p
y:2: FutureWarning: MultiIndex.is_lexsorted is deprecated as a public functi
on, users should use MultiIndex.is_monotonic_increasing instead.
  if not new_s.index.is_lexsorted():
```