Lab4

September 24, 2024

1 STOR 320: Introduction to Data Science

2 Lab 4

Name: Ivy Nangalia

Instructions: Fill in the blanks as necessary and complete the questions below.

Remember to submit the lab to gradescope.

[ ]:
```
# Just run this cell
import numpy as np
import pandas as pd

rng = np.random.default_rng(42)
```

2.1 Handling Missing Data

0. What are the two main approaches to dealing with missing data and a trade-off of each?

Answer: 1. Deleting the data with missing values could potentially remove valuable information with it 2. Performing calculations with missing data could affect the accuracy of calculations

1. What are the two modes of storing and manipulating null data in Pandas?

Answer: 1. Using "NaN" 2. Using "None"

2. Any array containing `None` must have what dtype?

Answer: "Object" dtype

3. Run the cells below. Why does the operation on `dtype=object` take so much longer than `dtype=int`?

[ ]:
```
%timeit np.arange(1E6, dtype=int).sum()
```

```
837 µs ± 40.2 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

[ ]:
```
%timeit np.arange(1E6, dtype=object).sum()
```

```
35.2 ms ± 868 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Answer: Doing an operation on integers is quicker than objects because integers can be dealt with at larger scale and faster, whereas objects need to be dealt with one at a time.

4. Try to run the code cells below. Why does the `.sum()` call on `vals1` throw an error?

[ ]:
```
# Create an array with None object
vals1 = np.array([1, None, 2, 3])
vals1
```

[ ]:
```
array([1, None, 2, 3], dtype=object)
```

[ ]:
```
vals1.sum()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[5], line 1
----> 1 vals1.sum()

File ~/Library/Python/3.12/lib/python/site-packages/numpy/core/_methods.py:49,␣
↪in _sum(a, axis, dtype, out, keepdims, initial, where)
     47 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
     48          initial=_NoValue, where=True):
---> 49     return umr_sum(a, axis, dtype, out, keepdims, initial, where)

TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

Answer: Because the None object can't be added to other values

5. What is the main downside to `NaN`?

Answer: The NaN data type only supports floats. So it doesn't really work with non-float dypes.

6. Run the code cells below. What two changes did Pandas automatically make when you ran `x[0] = None`? Why?

[ ]:
```
x = pd.Series(range(2), dtype=int)
x
```

[ ]:
```
0    0
1    1
dtype: int64
```

[ ]:
```
x[0] = None
x
```

[ ]:
```
0    NaN
1    1.0
dtype: float64
```

Answer: It made x[0] = NaN and it changed the data type of the series to a float.

7. Fill in the blank: In Pandas, strings are always stored with a/an ____________ dtype.

Answer: Object

8. You are given the following DataFrame `df` which contains information about students' test scores in different subjects. Some of the data is missing.

1. Count the number of missing values in each column and display the missing counts.
2. Fill the missing values in the 'Math' column with the median value of the 'Math' column and display `df`.
3. Fill missing values in the 'Science' and 'History' columns with the mean value of their respective columns and display `df`. There should only be one null value left in `df`.
4. Create a new column 'Total_Score' which is the sum of the scores in all subjects for each student. Handle missing values by treating them as zeros in the summation. Display `df`.
5. Interpolate missing value linearly in the 'English' column. Display `df`. Explain the difference between forward fill and linear interpolation and what would have resulted if we would have used forward fill instead of interpolation. Hint: Look at using `.interpolate`

[ ]:
```
data = {
    'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Math': [85, 92, np.nan, 74, np.nan],
    'Science': [np.nan, 88, 93, 81, np.nan],
    'English': [79, np.nan, 85, 90, 87],
    'History': [88, 76, np.nan, np.nan, 80]
}

df = pd.DataFrame(data)
df
```

[ ]:

	Student	Math	Science	English	History
0	Alice	85.0	NaN	79.0	88.0
1	Bob	92.0	88.0	NaN	76.0
2	Charlie	NaN	93.0	85.0	NaN
3	David	74.0	81.0	90.0	NaN
4	Eve	NaN	NaN	87.0	80.0

[ ]:
```
# Code Solution Here
df = pd.DataFrame(data)

missing_counts = df.isnull().sum()
print(missing_counts)

df["Math"] = df["Math"].fillna(df["Math"].dropna().median())
display(df)

df["Science"] = df["Science"].fillna(df["Science"].dropna().mean())
df["History"] = df["History"].fillna(df["History"].dropna().mean())
display(df)
```

```
df["Total Score"] = df.select_dtypes(include=np.number).sum(axis=1)
display(df)

#df["English"] = df["English"].fillna(df["English"].dropna().mean())
df["English"] = df["English"].interpolate()
display(df)
```

```
Student    0
Math       2
Science    2
English    1
History    2
dtype: int64
```

```
   Student  Math  Science  English  History
0    Alice  85.0      NaN     79.0     88.0
1      Bob  92.0     88.0      NaN     76.0
2  Charlie  85.0     93.0     85.0      NaN
3    David  74.0     81.0     90.0      NaN
4      Eve  85.0      NaN     87.0     80.0
```

```
   Student  Math    Science  English    History
0    Alice  85.0  87.333333     79.0  88.000000
1      Bob  92.0  88.000000      NaN  76.000000
2  Charlie  85.0  93.000000     85.0  81.333333
3    David  74.0  81.000000     90.0  81.333333
4      Eve  85.0  87.333333     87.0  80.000000
```

```
   Student  Math    Science  English    History  Total Score
0    Alice  85.0  87.333333     79.0  88.000000   339.333333
1      Bob  92.0  88.000000      NaN  76.000000   256.000000
2  Charlie  85.0  93.000000     85.0  81.333333   344.333333
3    David  74.0  81.000000     90.0  81.333333   326.333333
4      Eve  85.0  87.333333     87.0  80.000000   339.333333
```

```
   Student  Math    Science  English    History  Total Score
0    Alice  85.0  87.333333     79.0  88.000000   339.333333
1      Bob  92.0  88.000000     82.0  76.000000   256.000000
2  Charlie  85.0  93.000000     85.0  81.333333   344.333333
3    David  74.0  81.000000     90.0  81.333333   326.333333
4      Eve  85.0  87.333333     87.0  80.000000   339.333333
```

Forward filling the missing “English” value would have resulted in a higher score than would be expected for that student.

2.2 Heirarchical Indexing

9. Run the code cell below. What do the blank values in the first column represent in the hierarchical representation of the data?

[ ]:
```python
# Use Python tuples as keys
index = [('California', 2010), ('California', 2020),
         ('New York', 2010), ('New York', 2020),
         ('Texas', 2010), ('Texas', 2020)]

populations = [37253956, 39538223,
               19378102, 20201249,
               25145561, 29145505]
pop = pd.Series(populations, index=index)
pop

# Create a multi-index from the tuples
index = pd.MultiIndex.from_tuples(index)

# Hierarchical representation of the data
pop = pop.reindex(index)
pop
```

[ ]:
```
California  2010    37253956
            2020    39538223
New York    2010    19378102
            2020    20201249
Texas       2010    25145561
            2020    29145505
dtype: int64
```

Answer: The blank values imply that the State name is a higher-level index, and that each state corresponds with multiple year values.

10. You are given the following MultiIndexed Series sales which contains quarterly sales data for different regions and products.

1. Display the sales data for ‘North’ region.
2. Display the sales data for ‘Product_B’ across all regions.
3. Display the sales data for ‘North’ region and ‘Product_A’.
4. Display the sales data for ‘South’ and ‘West’ regions only.
5. Display the sales data for ‘North’ and ‘South’ regions and ‘Product_B’.

[ ]:
```python
index = pd.MultiIndex.from_product(
    [['North', 'South', 'East', 'West'], ['Product_A', 'Product_B']],
    names=['Region', 'Product']
)

data = [150, 200, 100, 220, 130, 190, 170, 210]

sales = pd.Series(data, index=index)
sales
```

[ ]:
```
Region  Product
North   Product_A    150
        Product_B    200
South   Product_A    100
        Product_B    220
East    Product_A    130
        Product_B    190
West    Product_A    170
        Product_B    210
dtype: int64
```

[ ]:
```
display(sales.loc["North"])
display(sales.loc[:,"Product_B"])
display(sales.loc["North","Product_A"])
display(sales.loc["South"], sales.loc["West"])
display(sales.loc["North","Product_B"], sales.loc["South","Product_B"])
```

```
Product
Product_A    150
Product_B    200
dtype: int64
```

```
Region
North    200
South    220
East     190
West     210
dtype: int64
```

```
150
```

```
Product
Product_A    100
Product_B    220
dtype: int64
```

```
Product
Product_A    170
Product_B    210
dtype: int64
```

```
200
```

```
220
```

11. The code below throws an errors because we cannot slice within a tuple. Provide the correct version of the code and explain what the code is doing.

[ ]:
```
# hierarchical indices and columns
index = pd.MultiIndex.from_product([[2013, 2014], [1, 2]],
                                   names=['year', 'visit'])
```

```
columns = pd.MultiIndex.from_product([['Bob', 'Guido', 'Sue'],
                                      ['HR', 'Temp']],
                                      names=['subject', 'type'])
# mock some data
data = np.round(np.random.randn(4, 6), 1)
data[:, ::2] *= 10
data += 37

# create the DataFrame
health_data = pd.DataFrame(data, index=index, columns=columns)

# try to slice the DataFrame
health_data.loc[(:, 1), (:, 'HR')]
```

```
  Cell In[28], line 16
    health_data.loc[(:, 1), (:, 'HR')]
                     ^
SyntaxError: invalid syntax
```

[ ]:
```
# Code Solution Here
# hierarchical indices and columns
index = pd.MultiIndex.from_product([[2013, 2014], [1, 2]], # create multi-index
                                    names=['year', 'visit'])
columns = pd.MultiIndex.from_product([['Bob', 'Guido', 'Sue'],
                                      ['HR', 'Temp']],
                                      names=['subject', 'type'])
# mock some data
data = np.round(np.random.randn(4, 6), 1) # generate random values
data[:, ::2] *= 10
data += 37

# create the DataFrame
health_data = pd.DataFrame(data, index=index, columns=columns) # put it all␣
↪together to create a dataframe
# try to slice the DataFrame
idx = pd.IndexSlice
health_data.loc[idx[:, 1], idx[:, 'HR']]
```

[ ]:
```
subject      Bob Guido   Sue
type          HR    HR    HR
year visit
2013 1      49.0  20.0  31.0
2014 1      24.0  43.0  43.0
```

Answer: By default you are not able to slice within a tuple, since a tuple is an immutable data type. By creating an IndexSlice, you can get around this. Using regular multi-index notation with the specified `idx`, we are able to isolate the HR for each subject on their first visit every year (which I believe was intended by the original code).

12. Fill in the blank: Partial slices and other similar operations require the levels in the `MultiIndex` to be in sorted (i.e., ________________________) order.

Answer: lexiographic

13. What does the `level=0` vs. `level=1` parameter change in the `.unstack()` function?

Answer: `level=0` makes the `unstack()` function use the outer level's values as new columns, whereas `level=1` uses the inner level's

14. You are given the following MultiIndexed DataFrame sales_data which contains quarterly sales data for different products across multiple regions and years. You should use the original `sales_data` DataFrame to perform each task.

1. Unstack the 'Region' level and display the resulting DataFrame.
2. Swap the levels 'Year' and 'Region' and display the resulting DataFrame. Hint: Look at using `.swaplevels`
3. Slice the data to retrieve sales information for 'Product_A' in the 'South' region for the year 2020 for quarters Q2 and Q3. Display your answer.

[ ]:
```
index = pd.MultiIndex.from_product(
    [['2019', '2020'], ['North', 'South'], ['Product_A', 'Product_B'], ['Q1',␣
 ↪'Q2', 'Q3', 'Q4']],
    names=['Year', 'Region', 'Product', 'Quarter']
)

data = np.random.randint(100, 1000, size=(32, 1))

sales_data = pd.DataFrame(data, index=index, columns=['Sales'])
```

[ ]:
```
# Code Solution Here
display(sales_data)

sales_data_unstacked = sales_data.unstack(level="Region")
display(sales_data_unstacked)

sales_data_swap = sales_data.swaplevel(i="Year", j="Region")
display(sales_data_swap)

sales_data_slice = sales_data.loc[('2020', 'South', 'Product_A', ['Q2', 'Q3']),␣
 ↪:]
display(sales_data_slice)
```

```
                               Sales
Year Region Product   Quarter
2019 North  Product_A Q1         238
                      Q2         763
                      Q3         841
```

```
                    Q4          536
          Product_B Q1          340
                    Q2          114
                    Q3          741
                    Q4          480
     South  Product_A Q1        819
                    Q2          842
                    Q3          173
                    Q4          465
          Product_B Q1          797
                    Q2          256
                    Q3          578
                    Q4          137
2020 North  Product_A Q1        547
                    Q2          884
                    Q3          110
                    Q4          386
          Product_B Q1          479
                    Q2          637
                    Q3          325
                    Q4          398
     South  Product_A Q1        579
                    Q2          264
                    Q3          938
                    Q4          568
          Product_B Q1          592
                    Q2          392
                    Q3          481
                    Q4          476

                       Sales
Region                 North South
Year Product   Quarter
2019 Product_A Q1        238   819
               Q2        763   842
               Q3        841   173
               Q4        536   465
     Product_B Q1        340   797
               Q2        114   256
               Q3        741   578
               Q4        480   137
2020 Product_A Q1        547   579
               Q2        884   264
               Q3        110   938
               Q4        386   568
     Product_B Q1        479   592
               Q2        637   392
               Q3        325   481
```

```
            Q4       398   476
                           Sales
Region Year Product   Quarter
North  2019 Product_A Q1         238
                      Q2         763
                      Q3         841
                      Q4         536
            Product_B Q1         340
                      Q2         114
                      Q3         741
                      Q4         480
South  2019 Product_A Q1         819
                      Q2         842
                      Q3         173
                      Q4         465
            Product_B Q1         797
                      Q2         256
                      Q3         578
                      Q4         137
North  2020 Product_A Q1         547
                      Q2         884
                      Q3         110
                      Q4         386
            Product_B Q1         479
                      Q2         637
                      Q3         325
                      Q4         398
South  2020 Product_A Q1         579
                      Q2         264
                      Q3         938
                      Q4         568
            Product_B Q1         592
                      Q2         392
                      Q3         481
                      Q4         476
                           Sales
Year Region Product   Quarter
2020 South  Product_A Q2         264
                      Q3         938
```