39_data_wrangling

April 17, 2022

[ ]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
```

[ ]:
```python
kashti = sns.load_dataset('titanic')
```

[ ]:
```python
ks1 = kashti.copy()
ks2 = kashti.copy()
```

[ ]:
```python
kashti.head()
```

[ ]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

0.0.1 Simple Operations (Math operator)

[ ]:
```python
# Simply add 1 in the whole series of age
(kashti['age']+1).head()
```

[ ]:
```
0    23.0
1    39.0
2    27.0
3    36.0
4    36.0
Name: age, dtype: float64
```

1 Dealing with missing values

- in a data set missing values are either ? or N/A or NaN, or 0 or a blank cell.

  Steps: 1. Try to recollect the data if possible to remove the error or missing values 2. If the column with missing values is not important in data, remove the whole column 3. Replace the missing values: 1. How to replace the missing values? - Average value of entire variable or similar data point - Frequency or MODE replacement - Replace based on other functions (Data sampler knows that) - ML algorithm can also be used to figure out the missing values - Leave it as it is

2. Why to replace the missing values?
   - It's better to have less lost data and more valueable data
   - Data with missing values is less accurate.

[ ]:
```
# Where exactly missing values are in our DataFrame?
# DF.isnull().sum()
kashti.isnull().sum()
```

[ ]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

[ ]:
```
# Dropping missing values

# check the shape of data before removing missing values
print(kashti.shape)

# drop rows in deck column with missing values; axis=0 means drop rows
kashti.dropna(subset=['deck'], axis=0, inplace=True)
print(kashti.shape)
```

```
(891, 15)
(203, 15)
```

[ ]:
```
# dropping a whole column
ks1.copy().head().drop(columns=['deck']).columns
```

[ ]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'embark_town', 'alive',
       'alone'],
      dtype='object')
```

[ ]:
```
kashti.head()
```

[ ]:
```
    survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
1          1       1  female  38.0      1      0  71.2833        C  First
3          1       1  female  35.0      1      0  53.1000        S  First
6          0       1    male  54.0      0      0  51.8625        S  First
10         1       3  female   4.0      1      1  16.7000        S  Third
11         1       1  female  58.0      0      0  26.5500        S  First

      who  adult_male deck  embark_town alive  alone
1   woman       False    C    Cherbourg   yes  False
3   woman       False    C  Southampton   yes  False
6     man        True    E  Southampton    no   True
10  child       False    G  Southampton   yes  False
11  woman       False    C  Southampton   yes   True
```

[ ]:
```
kashti.isnull().sum()
```

[ ]:
```
survived        0
pclass          0
sex             0
age            19
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck            0
embark_town     2
alive           0
alone           0
dtype: int64
```

[ ]:
```
# dropping all na values
# caution: this may dramatically reduce the data size if called with no␣
↪arguments
```

```
# as it will remove all the rows containing any null value in any column of the␣
↪data set
# DF.dropna()

kashti.dropna(inplace=True)
kashti.isnull().sum()
```

```
[ ]: survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

```
[ ]: # see, dropna can reduced data from 891 rows to 182 rows in this particular␣
↪data set
kashti.shape
```

```
[ ]: (182, 15)
```

```
[ ]: mean_age = ks1['age'].mean()
mean_age
```

```
[ ]: 29.69911764705882
```

```
[ ]: # ks2.copy()['age'].replace(np.nan, mean_age)
ks2.loc[ks2['age'] == np.nan]   # this returns zero records, then how the␣
↪replace command is working?
```

```
[ ]: Empty DataFrame
Columns: [survived, pclass, sex, age, sibsp, parch, fare, embarked, class, who,
adult_male, deck, embark_town, alive, alone]
Index: []
```

```
[ ]: ks1['age'].replace(np.nan, mean_age)   # this actually replaces the nan but the␣
↪above shows zero records, how is it possible?
ks1['age'].isnull().sum()
```

[ ]: 177

[ ]:
```
# replacing values of age column with average value of the same column
ks1['age'] = ks1['age'].replace(np.nan, mean_age)
```

[ ]:
```
# See age null values has been replaced with mean value, so null values are␣
↪zero now
ks1.isnull().sum()
```

[ ]:
```
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

[ ]:
```
# replacing with mean value, saves us from dropping 177 records of data
ks1.shape
```

[ ]: (891, 15)

[ ]:
```
deck_mode = ks1['deck'].mode().values[0]
deck_mode
```

[ ]: 'C'

[ ]:
```
# since deck value is a string value, so we can't compute its mean
# we will replace it with mode
# ks1['deck'] = ks2['deck']
# ks1['deck'].isnull()
# ks2.loc[ks2['age'].isnull()]
# ks1.loc[ks1['deck'].isnull()]
# ks1['deck'].value_counts()
# replace is not working in 'deck' column
# ks1['deck'].replace(to_replace=np.nan, value=deck_mode)
# ks1['deck'][0] = 'Test'
# type(ks1['deck'])
# ks1['deck'][0]
```

```
# ks1['deck']
# kashti.dtypes
# ks1['deck'].astype(str).replace(to_replace='nan', value=deck_mode).
 ↪astype('category')
# (ks1['deck'].astype(str)).replace(to_replace=np.nan, value=deck_mode)
# ks1['deck'].replace(to_replace=np.nan, value=deck_mode)
# ks1['deck']
# ks1['deck']
# deck_mode
# kashti.head()
```

Series.replace is not working on a series of Categorical data to replace NaN values. In the above code, I tested it with different scenarios, and finally got a "work around" by converting it to string (`NaN` values converted into `nan` string), and then replacing `"nan"` with deck_mode and then converting the column back into category data type to store it back to 'deck' series

[ ]:
```
ks1['deck'] = ks1['deck'].astype(str).replace(to_replace='nan',␣
 ↪value=deck_mode).astype('category')
```

[ ]:
```
ks1['deck']
```

[ ]:
```
0      C
1      C
2      C
3      C
4      C
      ..
886    C
887    B
888    C
889    C
890    C
Name: deck, Length: 891, dtype: category
Categories (7, object): ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

[ ]:
```
print(ks1.isnull().sum())
print(ks1.shape)
```

```
survived      0
pclass        0
sex           0
age           0
sibsp         0
parch         0
fare          0
embarked      2
class         0
who           0
```

```
adult_male      0
deck            0
embark_town     2
alive           0
alone           0
dtype: int64
(891, 15)
```

as shown in the above output, now we have less missing values with more data i.e. 891 rows lets see embarked and embark town now

[ ]:
```
# ks1.loc[:, ['embarked', 'embark_town']]
ks1.loc[ks1['embarked'].isnull()].loc[:, ['embarked', 'embark_town']]
```

[ ]:
```
    embarked embark_town
61       NaN         NaN
829      NaN         NaN
```

[ ]:
```
# embarked and embark_town are also string, so lets replace them with mode as␣
 ↪well
ks1.embark_town
```

[ ]:
```
0      Southampton
1        Cherbourg
2      Southampton
3      Southampton
4      Southampton
          …
886    Southampton
887    Southampton
888    Southampton
889      Cherbourg
890     Queenstown
Name: embark_town, Length: 891, dtype: object
```

[ ]:
```
# ks1.embarked.mode().values[0]
# ks1.embarked = ks2.embarked
```

[ ]:
```
embarked_mode = ks1['embarked'].mode().values[0]
embark_town_mode = ks1['embark_town'].mode().values[0]
ks1['embarked'] = ks1['embarked'].astype(str).replace(to_replace='nan',␣
 ↪value=embarked_mode).astype('category')
ks1['embark_town'] = ks1['embark_town'].astype(str).replace(to_replace='nan',␣
 ↪value=embark_town_mode).astype('category')
```

[ ]:
```
ks1.isnull().sum()
```

[ ]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

[ ]:
```
ks1.shape
```

[ ]:
```
(891, 15)
```

Now all the rows in the data has been preserved with zero `NaN` values. Used `mean` in numerical data i.e. `age` to replace the `NaN` values, and used `mode` in categorical data e.g. `deck, embarked, embark_town` etc.

2 Data Formatting

- Converting data into a common standard unit (e.g. if height is in cm, inches, and feets, convert all of them into one common unit in the whole data)
- Ensuring data is consistent and understandable e.g. don't mix both short and long form for same type fo data.
  - Easy to gather
  - Easy to work with
    - Faisalabad (FSD)
    - Lahore (LHR)
    - Islamabad(ISB)
    - Karachi (KHI)
    - Peshawar (PWR)

[ ]:
```
kashti.dtypes
```

[ ]:
```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
```

```
embarked          object
class           category
who               object
adult_male          bool
deck            category
embark_town       object
alive             object
alone               bool
dtype: object
```

[ ]:
```
# astype method to convert data type from one to another format
kashti['survived'] = kashti['survived'].astype(np.float64)
kashti['survived'] = kashti['survived'].astype(np.int64)
kashti.dtypes
```

[ ]:
```
survived           int64
pclass             int64
sex               object
age              float64
sibsp              int64
parch              int64
fare             float64
embarked          object
class           category
who               object
adult_male          bool
deck            category
embark_town       object
alive             object
alone               bool
dtype: object
```

[ ]:
```
# Applying an operation to whole column (converting whole column into another␣
↪unit)
ks1['age'] = (ks1['age'] * 365).astype(np.int64) # converted from years into␣
↪days now.
ks1['age']
```

[ ]:
```
0        8030
1       13870
2        9490
3       12775
4       12775
        …
886      9855
887      6935
888     10840
```

```
889     9490
890    11680
Name: age, Length: 891, dtype: int64
```

[ ]:
```
# renaming column names
ks1.rename(columns={'age': 'age in days'}, inplace=True)
ks1.head()
```

[ ]:
```
   survived  pclass     sex  age in days  sibsp  parch     fare embarked  \
0         0       3    male         8030      1      0   7.2500        S
1         1       1  female        13870      1      0  71.2833        C
2         1       3  female         9490      0      0   7.9250        S
3         1       1  female        12775      1      0  53.1000        S
4         0       3    male        12775      0      0   8.0500        S

   class    who  adult_male deck  embark_town alive  alone
0  Third    man        True    C  Southampton    no  False
1  First  woman       False    C    Cherbourg   yes  False
2  Third  woman       False    C  Southampton   yes   True
3  First  woman       False    C  Southampton   yes  False
4  Third    man        True    C  Southampton    no   True
```

3 Data Normalization

- Uniform the data
- Making sure that all the data have same impact
- Easy to understand relation in normalized data
- Helps in computations as well

[ ]:
```
ks4 = ks1[['age in days', 'fare']]
ks4.head()
```

[ ]:
```
   age in days     fare
0         8030   7.2500
1        13870  71.2833
2         9490   7.9250
3        12775  53.1000
4        12775   8.0500
```

- The above data is really in wide range annd we need to normalize it to make it easy to compare the data
- Normalization changes the values to the range of 0-1 (to have both variables similar influenece on our models)

3.1 Method of normalization

1. Simple feature scaling
    - x(new) = x(current) / x(max)

2. Min-Max method
3. Z-Score (standard score) -3 to +3
4. Log transformation

[ ]:
```
# ks4['fare'] = ks1['fare']
ks4['fare'] = ks4['fare']/ks4['fare'].max()
ks4['age in days'] = ks4['age in days']/ks4['age in days'].max()
ks4.head()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\667730025.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['fare'] = ks4['fare']/ks4['fare'].max()
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\667730025.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['age in days'] = ks4['age in days']/ks4['age in days'].max()
```

[ ]:

	age in days	fare
0	0.2750	0.014151
1	0.4750	0.139136
2	0.3250	0.015469
3	0.4375	0.103644
4	0.4375	0.015713

[ ]:
```
# Min - Max method
ks4['fare'] = ks1['fare']
ks4['fare'] = (ks4['fare'] - ks4['fare'].min()) / (ks4['fare'].max() -␣
↪ks4['fare'].min())
ks4.head()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\3940022684.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['fare'] = ks1['fare']
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\3940022684.py:3:
SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['fare'] = (ks4['fare'] - ks4['fare'].min()) / (ks4['fare'].max() -
ks4['fare'].min())
```

[ ]:

	age in days	fare
0	0.2750	0.014151
1	0.4750	0.139136
2	0.3250	0.015469
3	0.4375	0.103644
4	0.4375	0.015713

[ ]:

```python
# Z-Score method
# reset fare, and age to original values
ks4['fare'] = ks1['fare']
ks4['age in days'] = ks1['age in days']

# Apply Z-Score method formulae
ks4['fare'] = (ks4['fare'] - ks4['fare'].mean()) / (ks4['fare'].std())
ks4['age in days'] = (ks4['age in days'] - ks4['age in days'].mean()) /␣
↪(ks4['age in days'].std())
ks4.head()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\3664100664.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['fare'] = ks1['fare']
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\3664100664.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['age in days'] = ks1['age in days']
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\3664100664.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['fare'] = (ks4['fare'] - ks4['fare'].mean()) / (ks4['fare'].std())
C:\Users\hp\AppData\Local\Temp\ipykernel_10012\3664100664.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['age in days'] = (ks4['age in days'] - ks4['age in days'].mean()) /
(ks4['age in days'].std())
```

[ ]:

```
   age in days      fare
0    -0.592136 -0.502163
1     0.638440  0.786404
2    -0.284492 -0.488580
3     0.407707  0.420494
4     0.407707 -0.486064
```

[ ]:

```
ks3 = ks2.copy()
ks3.head()
```

[ ]:

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

[ ]:

```
ks3['fare'] = np.log(ks3['fare'])
ks3['age'] = np.log(ks3['age'])
ks3.head()
```

```
C:\Python310\lib\site-packages\pandas\core\arraylike.py:397: RuntimeWarning:
divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[ ]:

```
   survived  pclass     sex       age  sibsp  parch      fare embarked  class  \
0         0       3    male  3.091042      1      0  1.981001        S  Third
1         1       1  female  3.637586      1      0  4.266662        C  First
2         1       3  female  3.258097      0      0  2.070022        S  Third
```

```
3         1        1  female  3.555348      1      0  3.972177        S  First
4         0        3    male  3.555348      0      0  2.085672        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

3.2 Binning

- Grouping of values into smaller number of values (bins)
- Convert numeric into categories (Child, Teen, Young, Mature, Old) or 1-12, 13-19, 19-25, 26-40, 40-60 etc.
- Another example is products with prices
  - low
  - mid
  - high

[ ]:
```python
help(np.linspace)
bins = np.linspace(min(ks1['age']), max(ks1['age']), 4)
bins
```

```
Help on function linspace in module numpy:

linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)
    Return evenly spaced numbers over a specified interval.

    Returns `num` evenly spaced samples, calculated over the
    interval [`start`, `stop`].

    The endpoint of the interval can optionally be excluded.

    .. versionchanged:: 1.16.0
        Non-scalar `start` and `stop` are now supported.

    .. versionchanged:: 1.20.0
        Values are rounded towards ``-inf`` instead of ``0`` when an
        integer ``dtype`` is specified. The old behavior can
        still be obtained with ``np.linspace(start, stop, num).astype(int)``

    Parameters
    ----------
    start : array_like
        The starting value of the sequence.
    stop : array_like
        The end value of the sequence, unless `endpoint` is set to False.
```

```
    In that case, the sequence consists of all but the last of ``num + 1``
    evenly spaced samples, so that `stop` is excluded.  Note that the step
    size changes when `endpoint` is False.
num : int, optional
    Number of samples to generate. Default is 50. Must be non-negative.
endpoint : bool, optional
    If True, `stop` is the last sample. Otherwise, it is not included.
    Default is True.
retstep : bool, optional
    If True, return (`samples`, `step`), where `step` is the spacing
    between samples.
dtype : dtype, optional
    The type of the output array.  If `dtype` is not given, the data type
    is inferred from `start` and `stop`. The inferred dtype will never be
    an integer; `float` is chosen even if the arguments would produce an
    array of integers.

    .. versionadded:: 1.9.0

axis : int, optional
    The axis in the result to store the samples.  Relevant only if start
    or stop are array-like.  By default (0), the samples will be along a
    new axis inserted at the beginning. Use -1 to get an axis at the end.

    .. versionadded:: 1.16.0

Returns
-------
samples : ndarray
    There are `num` equally spaced samples in the closed interval
    ``[start, stop]`` or the half-open interval ``[start, stop)``
    (depending on whether `endpoint` is True or False).
step : float, optional
    Only returned if `retstep` is True

    Size of spacing between samples.


See Also
--------
arange : Similar to `linspace`, but uses a step size (instead of the
         number of samples).
geomspace : Similar to `linspace`, but with numbers spaced evenly on a log
            scale (a geometric progression).
logspace : Similar to `geomspace`, but with the end points specified as
           logarithms.

Examples
```

```
--------
>>> np.linspace(2.0, 3.0, num=5)
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
>>> np.linspace(2.0, 3.0, num=5, endpoint=False)
array([2. ,  2.2,  2.4,  2.6,  2.8])
>>> np.linspace(2.0, 3.0, num=5, retstep=True)
(array([2.  ,  2.25,  2.5 ,  2.75,  3.  ]), 0.25)

Graphical illustration:

>>> import matplotlib.pyplot as plt
>>> N = 8
>>> y = np.zeros(N)
>>> x1 = np.linspace(0, 10, N, endpoint=True)
>>> x2 = np.linspace(0, 10, N, endpoint=False)
>>> plt.plot(x1, y, 'o')
[<matplotlib.lines.Line2D object at 0x…>]
>>> plt.plot(x2, y + 0.5, 'o')
[<matplotlib.lines.Line2D object at 0x…>]
>>> plt.ylim([-0.5, 1])
(-0.5, 1)
>>> plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File C:\Python310\lib\site-packages\pandas\core\indexes\base.py:3621, in Index.
↪get_loc(self, key, method, tolerance)
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.py?
↪line=3619'>3620</a> try:
-> <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.py?
↪line=3620'>3621</a>     return self._engine.get_loc(casted_key)
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.py?
↪line=3621'>3622</a> except KeyError as err:

File C:\Python310\lib\site-packages\pandas\_libs\index.pyx:136, in pandas._libs
↪index.IndexEngine.get_loc()

File C:\Python310\lib\site-packages\pandas\_libs\index.pyx:163, in pandas._libs
↪index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:5198, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:5206, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()
```

```
KeyError: 'age'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
e:\learning\python\python_ka_chilla_ammar\sessions\39\39_data_wrangling.ipynb␣
 ↪Cell 50' in <cell line: 2>()
      <a href='vscode-notebook-cell:/e%3A/learning/python/python_ka_chilla_amma /
 ↪sessions/39/39_data_wrangling.ipynb#ch0000049?line=0'>1</a> help(np.linspace)
----> <a href='vscode-notebook-cell:/e%3A/learning/python/python_ka_chilla_amma /
 ↪sessions/39/39_data_wrangling.ipynb#ch0000049?line=1'>2</a> bins = np.
 ↪linspace(min(ks1['age']), max(ks1['age']), 4)
      <a href='vscode-notebook-cell:/e%3A/learning/python/python_ka_chilla_amma /
 ↪sessions/39/39_data_wrangling.ipynb#ch0000049?line=2'>3</a> bins

File C:\Python310\lib\site-packages\pandas\core\frame.py:3505, in DataFrame.
 ↪__getitem__(self, key)
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/frame.py?
 ↪line=3502'>3503</a> if self.columns.nlevels > 1:
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/frame.py?
 ↪line=3503'>3504</a>     return self._getitem_multilevel(key)
-> <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/frame.py?
 ↪line=3504'>3505</a> indexer = self.columns.get_loc(key)
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/frame.py?
 ↪line=3505'>3506</a> if is_integer(indexer):
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/frame.py?
 ↪line=3506'>3507</a>     indexer = [indexer]

File C:\Python310\lib\site-packages\pandas\core\indexes\base.py:3623, in Index.
 ↪get_loc(self, key, method, tolerance)
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3620'>3621</a>     return self._engine.get_loc(casted_key)
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3621'>3622</a> except KeyError as err:
-> <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3622'>3623</a>     raise KeyError(key) from err
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3623'>3624</a> except TypeError:
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3624'>3625</a>     # If we have a listlike key, _check_indexing_error␣
 ↪will raise
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3625'>3626</a>     #  InvalidIndexError. Otherwise we fall through and␣
 ↪re-raise
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3626'>3627</a>     #  the TypeError.
   <a href='file:///c%3A/Python310/lib/site-packages/pandas/core/indexes/base.p ?
 ↪line=3627'>3628</a>     self._check_indexing_error(key)
```

```
KeyError: 'age'
```

[ ]:
```
help(pd.cut)
```

```
Help on function cut in module pandas.core.reshape.tile:

cut(x, bins, right: 'bool' = True, labels=None, retbins: 'bool' = False,
precision: 'int' = 3, include_lowest: 'bool' = False, duplicates: 'str' =
'raise', ordered: 'bool' = True)
    Bin values into discrete intervals.

    Use `cut` when you need to segment and sort data values into bins. This
    function is also useful for going from a continuous variable to a
    categorical variable. For example, `cut` could convert ages to groups of
    age ranges. Supports binning into an equal number of bins, or a
    pre-specified array of bins.

    Parameters
    ----------
    x : array-like
        The input array to be binned. Must be 1-dimensional.
    bins : int, sequence of scalars, or IntervalIndex
        The criteria to bin by.

        * int : Defines the number of equal-width bins in the range of `x`. The
          range of `x` is extended by .1% on each side to include the minimum
          and maximum values of `x`.
        * sequence of scalars : Defines the bin edges allowing for non-uniform
          width. No extension of the range of `x` is done.
        * IntervalIndex : Defines the exact bins to be used. Note that
          IntervalIndex for `bins` must be non-overlapping.

    right : bool, default True
        Indicates whether `bins` includes the rightmost edge or not. If
        ``right == True`` (the default), then the `bins` ``[1, 2, 3, 4]``
        indicate (1,2], (2,3], (3,4]. This argument is ignored when
        `bins` is an IntervalIndex.
    labels : array or False, default None
        Specifies the labels for the returned bins. Must be the same length as
        the resulting bins. If False, returns only integer indicators of the
        bins. This affects the type of the output container (see below).
        This argument is ignored when `bins` is an IntervalIndex. If True,
        raises an error. When `ordered=False`, labels must be provided.
    retbins : bool, default False
        Whether to return the bins or not. Useful when bins is provided
        as a scalar.
```

```
precision : int, default 3
    The precision at which to store and display the bins labels.
include_lowest : bool, default False
    Whether the first interval should be left-inclusive or not.
duplicates : {default 'raise', 'drop'}, optional
    If bin edges are not unique, raise ValueError or drop non-uniques.
ordered : bool, default True
    Whether the labels are ordered or not. Applies to returned types
    Categorical and Series (with Categorical dtype). If True,
    the resulting categorical will be ordered. If False, the resulting
    categorical will be unordered (labels must be provided).

    .. versionadded:: 1.1.0

Returns
-------
out : Categorical, Series, or ndarray
    An array-like object representing the respective bin for each value
    of `x`. The type depends on the value of `labels`.

    * None (default) : returns a Series for Series `x` or a
      Categorical for all other inputs. The values stored within
      are Interval dtype.

    * sequence of scalars : returns a Series for Series `x` or a
      Categorical for all other inputs. The values stored within
      are whatever the type in the sequence is.

    * False : returns an ndarray of integers.

bins : numpy.ndarray or IntervalIndex.
    The computed or specified bins. Only returned when `retbins=True`.
    For scalar or sequence `bins`, this is an ndarray with the computed
    bins. If set `duplicates=drop`, `bins` will drop non-unique bin. For
    an IntervalIndex `bins`, this is equal to `bins`.

See Also
--------
qcut : Discretize variable into equal-sized buckets based on rank
    or based on sample quantiles.
Categorical : Array type for storing data that come from a
    fixed set of values.
Series : One-dimensional array with axis labels (including time series).
IntervalIndex : Immutable Index implementing an ordered, sliceable set.

Notes
-----
Any NA values will be NA in the result. Out of bounds values will be NA in
```

```
    the resulting Series or Categorical object.

    Reference :ref:`the user guide <reshaping.tile.cut>` for more examples.

    Examples
    --------
    Discretize into three equal-sized bins.

    >>> pd.cut(np.array([1, 7, 5, 4, 6, 3]), 3)
    … # doctest: +ELLIPSIS
    [(0.994, 3.0], (5.0, 7.0], (3.0, 5.0], (3.0, 5.0], (5.0, 7.0], …
    Categories (3, interval[float64, right]): [(0.994, 3.0] < (3.0, 5.0] …

    >>> pd.cut(np.array([1, 7, 5, 4, 6, 3]), 3, retbins=True)
    … # doctest: +ELLIPSIS
    ([(0.994, 3.0], (5.0, 7.0], (3.0, 5.0], (3.0, 5.0], (5.0, 7.0], …
    Categories (3, interval[float64, right]): [(0.994, 3.0] < (3.0, 5.0] …
    array([0.994, 3.   , 5.   , 7.   ]))

    Discovers the same bins, but assign them specific labels. Notice that
    the returned Categorical's categories are `labels` and is ordered.

    >>> pd.cut(np.array([1, 7, 5, 4, 6, 3]),
    …        3, labels=["bad", "medium", "good"])
    ['bad', 'good', 'medium', 'medium', 'good', 'bad']
    Categories (3, object): ['bad' < 'medium' < 'good']

    ``ordered=False`` will result in unordered categories when labels are
passed.
    This parameter can be used to allow non-unique labels:

    >>> pd.cut(np.array([1, 7, 5, 4, 6, 3]), 3,
    …        labels=["B", "A", "B"], ordered=False)
    ['B', 'B', 'A', 'A', 'B', 'B']
    Categories (2, object): ['A', 'B']

    ``labels=False`` implies you just want the bins back.

    >>> pd.cut([0, 1, 1, 2], bins=4, labels=False)
    array([0, 1, 1, 3])

    Passing a Series as an input returns a Series with categorical dtype:

    >>> s = pd.Series(np.array([2, 4, 6, 8, 10]),
    …               index=['a', 'b', 'c', 'd', 'e'])
    >>> pd.cut(s, 3)
    … # doctest: +ELLIPSIS
    a    (1.992, 4.667]
```

```
b    (1.992, 4.667]
c    (4.667, 7.333]
d     (7.333, 10.0]
e     (7.333, 10.0]
dtype: category
Categories (3, interval[float64, right]): [(1.992, 4.667] < (4.667, …
```

Passing a Series as an input returns a Series with mapping value.
It is used to map numerically to intervals based on bins.

```
>>> s = pd.Series(np.array([2, 4, 6, 8, 10]),
…                 index=['a', 'b', 'c', 'd', 'e'])
>>> pd.cut(s, [0, 2, 4, 6, 8, 10], labels=False, retbins=True, right=False)
… # doctest: +ELLIPSIS
(a    1.0
 b    2.0
 c    3.0
 d    4.0
 e    NaN
 dtype: float64,
 array([ 0,  2,  4,  6,  8, 10]))
```

Use `drop` optional when bins is not unique

```
>>> pd.cut(s, [0, 2, 4, 6, 10, 10], labels=False, retbins=True,
…        right=False, duplicates='drop')
… # doctest: +ELLIPSIS
(a    1.0
 b    2.0
 c    3.0
 d    3.0
 e    NaN
 dtype: float64,
 array([ 0,  2,  4,  6, 10]))
```

Passing an IntervalIndex for `bins` results in those categories exactly.
Notice that values not covered by the IntervalIndex are set to NaN. 0
is to the left of the first bin (which is closed on the right), and 1.5
falls between two bins.

```
>>> bins = pd.IntervalIndex.from_tuples([(0, 1), (2, 3), (4, 5)])
>>> pd.cut([0, 0.5, 1.5, 2.5, 4.5], bins)
[NaN, (0.0, 1.0], NaN, (2.0, 3.0], (4.0, 5.0]]
Categories (3, interval[int64, right]): [(0, 1] < (2, 3] < (4, 5]]
```

[ ]:
```python
age_groups = ["Child", "Teen", "Young", "Mature", "Old"]
# 1-12, 13-19, 19-25, 26-40, 40-45, 45-END OF LIFE
ks3['age'] = age_converted_to_categorical_variable = pd.cut(x=ks1['age in␣
↪days'] // 365,
               bins=[1,13,19,26,40,45], labels=age_groups, include_lowest=True)
```

[ ]:
```python
ks3['age']
```

[ ]:
```
0        Young
1       Mature
2        Young
3       Mature
4       Mature
         …
886     Mature
887       Teen
888     Mature
889      Young
890     Mature
Name: age, Length: 891, dtype: category
Categories (5, object): ['Child' < 'Teen' < 'Young' < 'Mature' < 'Old']
```

[ ]:
```python
ks3.groupby(['age','class', 'survived']).describe()
```

[ ]:
```
                        pclass                                         sibsp  \
                         count mean  std  min  25%  50%  75%  max  count
age    class  survived
Child  First  0            1.0  1.0  NaN  1.0  1.0  1.0  1.0  1.0    1.0
              1            2.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0    2.0
       Second 1           15.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0   15.0
       Third  0           28.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   28.0
              1           18.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   18.0
Teen   First  0            3.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0    3.0
              1           14.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0   14.0
       Second 0            9.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0    9.0
              1            8.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0    8.0
       Third  0           44.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   44.0
              1           15.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   15.0
Young  First  0            5.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0    5.0
              1           18.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0   18.0
       Second 0           20.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0   20.0
              1           12.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0   12.0
       Third  0           79.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   79.0
              1           21.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   21.0
Mature First  0           34.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0   34.0
              1           62.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0   62.0
```

```
       Second 0           47.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0   47.0
              1           36.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0   36.0
       Third  0          186.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0  186.0
              1           59.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   59.0
Old    First  0            6.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0    6.0
              1            9.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0    9.0
       Second 0            5.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0    5.0
              1            6.0  2.0  0.0  2.0  2.0  2.0  2.0  2.0    6.0
       Third  0           19.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0   19.0
              1            2.0  3.0  0.0  3.0  3.0  3.0  3.0  3.0    2.0

                                  … parch        fare                      \
                              mean …  75%  max  count      mean        std
age    class  survived             …
Child  First  0           1.000000 …  2.00  2.0    1.0  5.020916        NaN
              1           0.500000 …  2.00  2.0    2.0  4.596241   0.270470
       Second 1           0.800000 …  2.00  2.0   15.0  3.362424   0.253011
       Third  0           3.250000 …  2.00  2.0   28.0  3.354276   0.335218
              1           0.944444 …  2.00  2.0   18.0  2.747955   0.376082
Teen   First  0           1.666667 …  1.00  2.0    3.0  4.744920   0.801379
              1           0.500000 …  2.00  2.0   14.0  4.452389   0.717251
       Second 0           0.111111 …  0.00  1.0    9.0  2.875207   0.690680
              1           0.250000 …  0.25  2.0    8.0  2.850981   0.452423
       Third  0           0.727273 …  0.25  3.0   44.0      -inf        NaN
              1           0.466667 …  0.00  2.0   15.0  2.148876   0.184806
Young  First  0           0.200000 …  1.00  2.0    5.0  4.832375   0.487291
              1           0.666667 …  1.00  2.0   18.0  4.454073   0.619554
       Second 0           0.500000 …  0.00  2.0   20.0  2.915258   0.690140
              1           0.750000 …  2.00  3.0   12.0  3.135687   0.527989
       Third  0           0.215190 …  0.00  2.0   79.0  2.163074   0.304433
              1           0.190476 …  0.00  3.0   21.0      -inf        NaN
Mature First  0           0.147059 …  0.00  2.0   34.0      -inf        NaN
              1           0.403226 …  0.00  2.0   62.0  4.282124   0.737744
       Second 0           0.297872 …  0.00  2.0   47.0      -inf        NaN
              1           0.388889 …  0.00  2.0   36.0  2.849911   0.388839
       Third  0           0.559140 …  0.00  5.0  186.0      -inf        NaN
              1           0.338983 …  0.00  5.0   59.0  2.483870   0.588879
Old    First  0           0.666667 …  0.00  0.0    6.0  3.845678   0.532402
              1           0.222222 …  1.00  1.0    9.0  4.297739   0.909928
       Second 0           0.800000 …  0.00  1.0    5.0  3.128929   0.315653
              1           0.333333 …  0.75  1.0    6.0  2.871461   0.339527
       Third  0           0.210526 …  1.00  6.0   19.0  2.439832   0.619582
              1           0.000000 …  0.00  0.0    2.0  2.077847   0.011066


                               min        25%        50%        75%        max
age    class  survived
```

```
Child  First  0          5.020916  5.020916  5.020916  5.020916  5.020916
              1          4.404990  4.500615  4.596241  4.691866  4.787492
       Second 1          2.931194  3.258097  3.267666  3.607585  3.727600
       Third  0          2.347797  3.292541  3.371597  3.469140  3.848018
              1          1.978128  2.512369  2.761316  3.005718  3.446410
Teen   First  0          3.972177  4.331303  4.690430  5.131292  5.572154
              1          3.268934  4.047310  4.485937  4.767738  5.569775
       Second 0          2.351375  2.442347  2.564949  3.258097  4.297285
              1          2.351375  2.451524  2.850222  3.258097  3.403555
       Third  0              -inf  2.050913  2.092354  2.671989  3.848018
              1          1.977547  2.054371  2.083085  2.160524  2.670985
Young  First  0          4.347532  4.371976  4.909955  5.020916  5.511495
              1          3.401197  4.048700  4.297309  4.675296  5.572154
       Second 0          2.351375  2.534299  2.564949  3.258097  4.297285
              1          2.351375  2.661628  3.258097  3.375771  4.174387
       Third  0          1.389414  2.047157  2.066331  2.169710  3.537330
              1              -inf  2.021548  2.050913  2.756313  4.034166
Mature First  0              -inf  3.289818  3.494668  3.951244  5.427260
              1          3.269094  3.664299  4.356129  4.708478  6.238967
       Second 0              -inf  2.351375  2.564949  3.258097  4.297285
              1          2.351375  2.564949  2.596917  3.258097  3.663562
       Third  0              -inf  2.047693  2.070022  2.673429  4.242046
              1          1.942332  2.049303  2.093406  2.770994  4.034166
Old    First  0          3.279030  3.404811  3.760388  4.306221  4.499810
              1          3.269094  3.322183  4.060084  5.105137  5.427260
       Second 0          2.564949  3.258097  3.258097  3.267666  3.295837
              1          2.564949  2.574384  2.786552  3.186176  3.267666
       Third  0          1.864080  2.028127  2.085672  2.724902  3.848018
              1          2.070022  2.073935  2.077847  2.081760  2.085672

[29 rows x 32 columns]
```

3.2.1 Converting categories into dummies

- easy to use for computation e.g.
- Male Female (0, 1)

[ ]:
```
pd.get_dummies(ks1['sex'])
```

[ ]:
```
     female  male
0         0     1
1         1     0
2         1     0
3         1     0
4         0     1
..      …     …
886       0     1
887       1     0
```

```
888        1     0
889        0     1
890        0     1

[891 rows x 2 columns]
```

[ ]:
```python
dummy_male_categories = pd.get_dummies(ks1['sex'])['male']
dummy_female_categories = pd.get_dummies(ks1['sex'])['female']
# Assignment: how to use get dummies to change data inside a DataFrame
```

[ ]:
```python
dummy_male_categories
```

[ ]:
```
0      1
1      0
2      0
3      0
4      1
      ..
886    1
887    0
888    0
889    1
890    1
Name: male, Length: 891, dtype: uint8
```

[ ]:
```python
dummy_female_categories
```

[ ]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    1
889    0
890    0
Name: female, Length: 891, dtype: uint8
```

[ ]:
```python
ks1
```

[ ]:
```
     survived  pclass     sex  age in days  sibsp  parch     fare embarked  \
0           0       3    male         8030      1      0   7.2500        S
1           1       1  female        13870      1      0  71.2833        C
2           1       3  female         9490      0      0   7.9250        S
3           1       1  female        12775      1      0  53.1000        S
```

```
4            0       3     male        12775       0       0    8.0500        S
..         ...     ...      ...         ...      ...     ...       ...     ...
886          0       2     male         9855       0       0   13.0000        S
887          1       1   female         6935       0       0   30.0000        S
888          0       3   female        10840       1       2   23.4500        S
889          1       1     male         9490       0       0   30.0000        C
890          0       3     male        11680       0       0    7.7500        Q

      class    who  adult_male deck  embark_town alive  alone
0     Third    man        True    C  Southampton    no  False
1     First  woman       False    C    Cherbourg   yes  False
2     Third  woman       False    C  Southampton   yes   True
3     First  woman       False    C  Southampton   yes  False
4     Third    man        True    C  Southampton    no   True
..      ...    ...         ...  ...          ...   ...    ...
886  Second    man        True    C  Southampton    no   True
887   First  woman       False    B  Southampton   yes   True
888   Third  woman       False    C  Southampton    no  False
889   First    man        True    C    Cherbourg   yes   True
890   Third    man        True    C   Queenstown    no   True

[891 rows x 15 columns]
```

[ ]:
```
ks5 = ks2.copy()
ks5.head()
```

[ ]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

[ ]:
```
dummy_male_categories
```

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

```
       ..
886    1
887    0
888    0
889    1
890    1
Name: male, Length: 891, dtype: uint8
```

[ ]:
```
ks5['sex']
```

[ ]:
```
0        male
1      female
2      female
3      female
4        male
        …
886      male
887    female
888    female
889      male
890      male
Name: sex, Length: 891, dtype: object
```

[ ]:
```
# Male: sex = 1; Female: sex=0
ks5['sex'] = dummy_male_categories
ks5.head()
```

[ ]:
```
   survived  pclass  sex   age  sibsp  parch     fare embarked  class    who  \
0         0       3    1  22.0      1      0   7.2500        S  Third    man
1         1       1    0  38.0      1      0  71.2833        C  First  woman
2         1       3    0  26.0      0      0   7.9250        S  Third  woman
3         1       1    0  35.0      1      0  53.1000        S  First  woman
4         0       3    1  35.0      0      0   8.0500        S  Third    man

   adult_male deck  embark_town alive  alone
0        True  NaN  Southampton    no  False
1       False    C    Cherbourg   yes  False
2       False  NaN  Southampton   yes   True
3       False    C  Southampton   yes  False
4        True  NaN  Southampton    no   True
```

[ ]:
```
ks5.head()
```

[ ]:
```
   survived  pclass  sex   age  sibsp  parch     fare embarked  class    who  \
0         0       3    1  22.0      1      0   7.2500        S  Third    man
1         1       1    0  38.0      1      0  71.2833        C  First  woman
2         1       3    0  26.0      0      0   7.9250        S  Third  woman
```

```
3         1       1    0  35.0      1      0  53.1000        S  First  woman
4         0       3    1  35.0      0      0   8.0500        S  Third    man

   adult_male deck  embark_town alive  alone
0        True  NaN  Southampton    no  False
1       False    C    Cherbourg   yes  False
2       False  NaN  Southampton   yes   True
3       False    C  Southampton   yes  False
4        True  NaN  Southampton    no   True
```