22. Pandas - Part 5

October 28, 2022

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

1 How to find and remove duplicate rows?

```
[ ]: # read a dataset of movie reviewers into a DataFrame
     user_cols = ['user_id', 'age', 'gender', 'occupation', 'zip_code']
```

```
[ ]: users_df = pd.read_table('../data/u.user', sep='|', header=None,␣
      ↪names=user_cols, index_col='user_id')
```

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

```
[ ]: users_df.shape
```

```
[ ]: # detect duplicate zip codes: True if an item is identical to a previous item
     users_df.zip_code.duplicated()
```

```
[ ]: # count the duplicate items (True becomes 1, False becomes 0)
     users_df.zip_code.duplicated().sum()
```

```
[ ]: # detect duplicate DataFrame rows: True if an entire row is identical to a␣
      ↪previous row
     users_df.duplicated()
```

```
[ ]: # count the duplicate rows
     users_df.duplicated().sum()
```

Logic for `duplicated`:

- **`keep='first'`** (default): Mark duplicates as True except for the first occurrence.
- **`keep='last'`**: Mark duplicates as True except for the last occurrence.
- **`keep=False`**: Mark all duplicates as True.

```
[ ]: # examine the duplicate rows (ignoring the first occurrence)
     users_df.loc[users_df.duplicated(keep='first'), :]
```

```
[ ]: # examine the duplicate rows (ignoring the last occurrence)
     users_df.loc[users_df.duplicated(keep='last'), :]
```

```
[ ]: # examine the duplicate rows (including all duplicates)
     users_df.loc[users_df.duplicated(keep=False), :]
```

```
[ ]: # drop the duplicate rows (inplace=False by default)
     users_df.drop_duplicates(keep='first')
```

```
[ ]: users_df.drop_duplicates(keep='first').shape
```

```
[ ]: users_df.shape
```

```
[ ]: users_df.drop_duplicates(keep='last').shape
```

```
[ ]: users_df.drop_duplicates(keep=False).shape
```

```
[ ]: # only consider a subset of columns when identifying duplicates
     users_df.duplicated(subset=['age', 'zip_code'])#.tail()
```

```
[ ]: # only consider a subset of columns when dropping duplicates
     users_df.drop_duplicates(subset=['age', 'zip_code'])#,inplace=True)
```

2 Creating DataFrame from another object

```
[ ]: # create a DataFrame from a dictionary (keys become column names, values become␣
      ↪data)
     pd.DataFrame({'id':[100, 101, 102], 'color':['red', 'blue', 'red']})
```

```
[ ]: # optionally specify the order of columns and define the index
     df = pd.DataFrame({'id':[100, 101, 102], 'color':['red', 'blue', 'red']},
                       columns=['id', 'color'], index=['a', 'b', 'c'])
```

```
[ ]: df
```

```
[ ]: # create a DataFrame from a list of lists (each inner list becomes a row)
     pd.DataFrame([[100, 'red'], [101, 'blue'], [102, 'red']], columns=['id',␣
      ↪'color'])
```

```
[ ]: # create a new Series using the Series constructor
     shape = pd.Series(['round', 'square'], index=['c', 'b'], name='shape')
```

```
[ ]: shape
```

```
[ ]: df
```

[ ]:
```
# concatenate the DataFrame and the Series (use axis=1 to concatenate columns)
pd.concat([df, shape], axis=1)
```

Notes:

- The Series name became the column name in the DataFrame.
- The Series data was aligned to the DataFrame by its index.
- The 'shape' for row 'a' was marked as a missing value (NaN) because that index was not present in the Series.

3 How to apply a function to a Series or DataFrame?

[ ]:
```
train_df = pd.read_csv('../data/titanic_train.csv')
```

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

3.1 Map the existing values of a Series to a different set of values

Method:`map`

[ ]:
```
train_df.Gender.map({'female':0, 'male':1})
```

[ ]:
```
# map 'female' to 0 and 'male' to 1
train_df['Gender_num'] = train_df.Gender.map({'female':0, 'male':1})
```

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

[ ]:
```
train_df.loc[0:4, ['Gender', 'Gender_num']]
```

3.2 Apply a function to each element in a Series

Method: `apply`

Note: **`map`** can be substituted for **`apply`** in many cases, but **`apply`** is more flexible and thus is recommended

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

[ ]:
```
# calculate the length of each string in the 'Name' Series
train_df['Name_length'] = train_df.Name.apply(len)
```

[ ]:
```
train_df.loc[0:4, ['Name', 'Name_length']]
```

[ ]:
```
# round up each element in the 'Fare' Series to the next integer
import numpy as np
```

[ ]:
```
train_df['Fare_ceil'] = train_df.Fare.apply(np.ceil)
```

[ ]:
```
train_df.loc[0:4, ['Fare', 'Fare_ceil']]
```

[ ]:
```
# we want to extract the last name of each person
train_df.Name.head()
```

[ ]:
```
# use a string method to split the 'Name' Series at commas (returns a Series of␣
  ↪lists)
train_df.Name.str.split(',')
```

[ ]:
```
def lname(s):
    s = s.split(',')
    return s[0]
```

[ ]:
```
train_df.Name.apply(lname)
```

[ ]:
```
# define a function that returns an element from a list based on position
def get_element(my_list, position):
    return my_list[position]
```

[ ]:
```
# apply the 'get_element' function and pass 'position' as a keyword argument
train_df.Name.str.split(',').apply(get_element, position=0)
```

[ ]:
```
# alternatively, use a lambda function
train_df.Name.str.split(',').apply(lambda x: x[0])
```

3.3 Apply a function along either axis of a DataFrame

[ ]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks_df = pd.read_csv('../data/drinks.csv')
```

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

[ ]:
```
# select a subset of the DataFrame to work with
drinks_df.loc[:, 'beer_servings':'wine_servings']
```

[ ]:
```
# apply the 'max' function along axis 0 to calculate the maximum value in each␣
  ↪column
drinks_df.loc[:, 'beer_servings':'wine_servings'].apply(max, axis=0)
```

[ ]:
```
# apply the 'max' function along axis 1 to calculate the maximum value in each␣
  ↪row
drinks_df.loc[:, 'beer_servings':'wine_servings'].apply(max, axis=1)
```

[ ]:
```
drinks_df.loc[:, 'beer_servings':'wine_servings']
```

[ ]:
```
# convert every DataFrame element into a float
drinks_df.loc[:, 'beer_servings':'wine_servings'].applymap(float)
```

- **Difference between map, applymap and apply methods**

Link 1

Link 2

4 pd.to_datetime()

[ ]:
```
pd.to_datetime() # for converting strings in date column to datetime objects

# try this on Time column of ufo dataset.
```

For more details on this click here