data_process

September 15, 2020

1 Data Aggregation

[1]:
```
import sys
import pandas as pd
```

[2]:
```
sys.path.append(r'D:\OneDrive\Programming\documents_python\NUS␣
↪Courses\DBA5106\Course-DBA5106\assignment\IndividualAssignment2')
from utils.logger import logger
from utils.config import RAW_DATA_DIR, PROCESSED_DATA_DIR
from utils.data_porter import save_to_csv, read_from_csv
```

[3]:
```
# set hyperparameters
FIRST_DATA_YM = '2015-06'
LAST_DATA_YM = '2020-05'
```

[4]:
```
# generate date list(from FIRST_DATA_YM to LAST_DATA_YM)
date_lst = pd.date_range(FIRST_DATA_YM, LAST_DATA_YM, freq='MS').astype(str).
↪tolist()
date_lst = [''.join(date.split('-'))[:-2] for date in date_lst]
```

[5]:
```
# concat all the monthly data to one dataFrame
tripdata = None
for date in date_lst:
#     logger.debug(f'Processing {date} file.')
    suffix = '-hubway-tripdata.csv' if date <= '201804' else␣
↪'-bluebikes-tripdata.csv'
    monthly_file = date + suffix
    month_df = read_from_csv(monthly_file, RAW_DATA_DIR)
    # default join method is outer join.
    # set sort parameter to silence warning
    tripdata = month_df if tripdata is None else pd.concat([tripdata,␣
↪month_df], ignore_index=True, sort=False)
logger.info('Finish load all monthly file.')
```

```
LOG: 2020-09-15 22:42:53 [INFO] Finish load all monthly file.
```

2 Data Cleaning

[6]:
```
# Check NaN values in every columns
tripdata.isnull().sum()
```

[6]:
```
tripduration                  0
starttime                     0
stoptime                      0
start station id              0
start station name            0
start station latitude        0
start station longitude       0
end station id                0
end station name              0
end station latitude          0
end station longitude         0
bikeid                        0
usertype                      0
birth year               134471
gender                   124879
postal code             8165118
dtype: int64
```

2.1 Drop Features

[7]:
```
# drop postal code column because of massive missing values
tripdata.drop(columns = ['postal code'], inplace = True)
```

2.2 Replace Missing Values

[8]:
```
# Replace all NaN elements with 0s.
tripdata.fillna(0, inplace=True)
# Replace all \N elements with 0s.
tripdata.replace(r'\N', 0, inplace=True)
```

2.3 Specify Feature Type

[9]:
```
# Cast column 'birth year' to a specified dtype(int64).
tripdata['birth year'] = tripdata['birth year'].astype('int64')
# Cast column 'starttime' & 'stoptime' to a specified dtype(datetime64).
tripdata['starttime'] = tripdata['starttime'].astype('datetime64')
tripdata['stoptime'] = tripdata['stoptime'].astype('datetime64')
```

[10]:
```
tripdata.dtypes
```

[10]:
```
tripduration                          int64
starttime                    datetime64[ns]
stoptime                     datetime64[ns]
start station id                      int64
start station name                   object
start station latitude              float64
start station longitude             float64
end station id                        int64
end station name                     object
end station latitude                float64
end station longitude               float64
bikeid                                int64
usertype                             object
birth year                            int64
gender                              float64
dtype: object
```

3 Data Transformation

3.1 Process Station Info

[11]:
```
# extract station information(station id & station name)
start_station_info = tripdata[['start station id', 'start station name',
                               'start station latitude', 'start station␣
 ↪longitude']]
end_station_info = tripdata[['end station id', 'end station name',
                             'end station latitude', 'end station longitude']]
# rename the columns for later concat
station_info_lst = ['station_id','station_name', 'station_latitude',␣
 ↪'station_longitude']
start_station_info.columns = station_info_lst
end_station_info.columns = station_info_lst

# concat and drop duplicates
station_info = pd.concat([start_station_info, end_station_info], join='inner',␣
 ↪ignore_index=True)
station_info.drop_duplicates(subset=None, keep='first', inplace=True)
```

[12]:
```
# If errors = 'coerce', then invalid parsing will be set as NaN.
station_info['station_id'] = pd.to_numeric(station_info['station_id'],␣
 ↪errors='coerce').fillna(0)
station_info.sort_values('station_id', inplace=True)
```

[13]:
```
# drop duplicate id situation(one station have two name, multiple latitude/
 ↪longitude)
station_info_unique_id = station_info.copy()
```

```
station_info_unique_id.drop_duplicates(subset=['station_id'], keep='last',␣
↪inplace = True)
station_info_unique_id.sort_values('station_id', inplace=True)
```

[14]:
```
# drop station name columns from tripdata to reduce file size
tripdata.drop(columns=['start station name', 'end station name',
                       'start station latitude', 'start station longitude',
                       'end station latitude', 'end station longitude'],␣
↪inplace=True)
```

3.2 Process Usertype Info

[15]:
```
# find all unique usertype values
tripdata['usertype'].value_counts()
```

[15]:
```
Subscriber    6540965
Customer      1734651
Name: usertype, dtype: int64
```

[16]:
```
# create usertype map(remain usertype id in the main file)
usertype_dict = {'Subscriber':1,'Customer':2}
tripdata['usertype_id'] = tripdata['usertype'].map(usertype_dict)
```

[17]:
```
# save usertype id map to
usertype_info = pd.DataFrame(list(usertype_dict.items()),columns =␣
↪['usertype','usertype_id'])
```

[18]:
```
# drop usertype columns from tripdata to reduce file size
tripdata.drop(columns = ['usertype'], inplace = True)
```

3.3 Save to File

[19]:
```
# Save station info
save_to_csv(station_info, 'station_info.csv', PROCESSED_DATA_DIR)
save_to_csv(station_info_unique_id, 'station_info_unique_id.csv',␣
↪PROCESSED_DATA_DIR)
logger.info('Save station info to csv file.')

# Save usertype info
save_to_csv(usertype_info, 'usertype_info.csv', PROCESSED_DATA_DIR)
logger.info('Save usertype info to csv file.')

# Save trip data
save_to_csv(tripdata, 'tripdata.csv', PROCESSED_DATA_DIR)
logger.info('Save trip data to csv file.')
```

```
LOG: 2020-09-15 22:43:27 [INFO] Save station info to csv file.
LOG: 2020-09-15 22:43:27 [INFO] Save usertype info to csv file.
LOG: 2020-09-15 22:44:41 [INFO] Save trip data to csv file.
```

[20]:
```
tripdata.dtypes
```

[20]:
```
tripduration                int64
starttime           datetime64[ns]
stoptime            datetime64[ns]
start station id             int64
end station id               int64
bikeid                       int64
birth year                   int64
gender                     float64
usertype_id                  int64
dtype: object
```

[21]:
```
station_info.dtypes
```

[21]:
```
station_id             int64
station_name          object
station_latitude     float64
station_longitude    float64
dtype: object
```

[22]:
```
usertype_info.dtypes
```

[22]:
```
usertype       object
usertype_id     int64
dtype: object
```