Data Cleaning Process

Source file: cyclistic_data_cleaning.py

Loading pandas library:

```
import pandas as pd
```

Importing raw data:

```
okt2021 = pd.read_csv('workfiles/202110-divvy-tripdata.csv', index_col=0)
nov2021 = pd.read_csv('workfiles/202111-divvy-tripdata.csv', index_col=0)
dec2021 = pd.read_csv('workfiles/202112-divvy-tripdata.csv', index_col=0)
jan2022 = pd.read_csv('workfiles/202201-divvy-tripdata.csv', index_col=0)
feb2022 = pd.read_csv('workfiles/202202-divvy-tripdata.csv', index_col=0)
mar2022 = pd.read_csv('workfiles/202203-divvy-tripdata.csv', index_col=0)
apr2022 = pd.read_csv('workfiles/202204-divvy-tripdata.csv', index_col=0)
may2022 = pd.read_csv('workfiles/202205-divvy-tripdata.csv', index_col=0)
jun2022 = pd.read_csv('workfiles/202206-divvy-tripdata.csv', index_col=0)
jul2022 = pd.read_csv('workfiles/202207-divvy-tripdata.csv', index_col=0)
aug2022 = pd.read_csv('workfiles/202208-divvy-tripdata.csv', index_col=0)
sep2022 = pd.read_csv('workfiles/202209-divvy-tripdata.csv', index_col=0)
```

Merging all datasets:

```
# Merging all datasets into one:
dataframe = [okt2021, nov2021, dec2021, jan2022, feb2022,
             mar2022, apr2022, may2022, jun2022, jul2022, aug2022, sep2022]
df_complete = pd.concat(dataframe)
```

Checking the structure of all dataset files:

```
# Checking number and columns and its names in datasets:
for _ in dataframe:
    print(_.shape)
    print(_.columns)
```

All tables should have the same structure to complete the merging step successfully. Check showed that all 12 datasets have a countable number of rows and 12 columns with the same index names.

Checking the structure of the merged dataset:

```
# Raw datasets contain 12 same columns. Tables are consistent.
df_complete.shape
df_complete.columns
# Merged raw dataset has 5828235 rows and 12 columns.
```

Merged table keeps the structure of the single raw dataset. It has 5.828.235 rows and 12 columns.

Pushing “ride_id” index at the beginning of the table::

```
# Pushing "ride_id" index as first column
df_complete = df_complete.reset_index(level=0)
```

Checking data types:

```
df_complete.dtypes
```

Changing data types to datetype format:

```
# Columns "started_at" and "ended_at" should be in datetype format:
df_complete['started_at'] = pd.to_datetime(df_complete['started_at'])
df_complete['ended_at'] = pd.to_datetime(df_complete['ended_at'])
```

Looking for “typos” or “misspellings” in the whole set:

```
# Checking for "typos" or "misspellings" in the whole dataset:
df_complete.rideable_type.unique()
# There are 3 types of bikes: "electric bike", "docked bike" and "classic
bike". No errors.
df_complete.member_casual.unique()
# There are 2 types of users: "casual", "member". No errors.
df_complete.start_station_id.value_counts()
df_complete.start_station_name.value_counts()
df_complete.end_station_id.value_counts()
df_complete.end_station_name.value_counts()
# There are many names and id's that occur just once.
```

No such errors were found.

There are many station names and id's that occur just once. They could be error inputs or new/deleted stations. This problem has been left at this point to eventual further investigation.

Checking length of the ride id’s and looking for its duplicates:

```
df_complete['ride_id'].map(len).unique()
# All id's have 16 characters

df_complete.duplicated(subset=['ride_id']).value_counts()
# There are no duplicates of ride id’s
```

Checking for empty cells, null values in single columns:

```
df_complete['ride_id'].isnull().value_counts()
df_complete['rideable_type'].isnull().value_counts()
df_complete['started_at'].isnull().value_counts()
df_complete['ended_at'].isnull().value_counts()
# No empty cells or null values

df_complete['start_station_name'].isnull().value_counts()
df_complete['start_station_id'].isnull().value_counts()
df_complete['start_station_name'].isnull().value_counts(
) & df_complete['start_station_id'].isnull().value_counts()
# 895032 rows with empty cells or null values in start station description
columns
# start_station_id inconsistent length and format

df_complete['end_station_name'].isnull().value_counts()
df_complete['end_station_id'].isnull().value_counts()
df_complete['end_station_name'].isnull().value_counts(
) & df_complete['end_station_id'].isnull().value_counts()
# 958227 rows with empty cells or null values in start station description
columns
# end_station_id inconsistent length and format

df_complete['start_station_name'].isnull().value_counts(
) & df_complete['end_station_name'].isnull().value_counts()
# 821264 rows where both start and end station is not given

df_complete['start_lat'].isnull().value_counts()
df_complete['start_lng'].isnull().value_counts()
# No empty cells or null values

df_complete['end_lat'].isnull().value_counts()
df_complete['end_lng'].isnull().value_counts()
# 5844 rows with empty cells or null values in end coordinates columns

df_complete['member_casual'].isnull().value_counts()
# No empty cells or null values
```

There were few inputs without recorded station description (names or id's) or coordinates.

Removing incomplete instances and checking the number of deleted rows:

```
# Deleting rows with missing data:
no_nan_data = df_complete.dropna()
no_nan_data.shape
# The dataset without NAN values contains 4474141 rows with data.
# In relation to the whole dataset over 76% of data is complete.
```

The dataset without NAN values contains 4474141 rows which is 76% of the whole set.

Looking for unwanted data – test data:

There were few instances with "TEST" in start and end station id's columns. All should not be considered. After deletion there were 4472680 rows left (about 75% of the whole dataset).

```
# Searching for the unwanted data - test data.
# Context: TEST found in one of the start stations id's "Hubbard Bike-
checking (LBS-WH-TEST)"
no_nan_data['start_station_id'].str.contains('TEST').value_counts()
no_nan_data['start_station_name'].str.contains('TEST').value_counts()
# 1207 test rides found in start station id's. All should not be
considered:
# Deleting test rides:
no_test_start_data =
no_nan_data[no_nan_data['start_station_id'].str.contains(
    'TEST') != True]

no_test_start_data['end_station_id'].str.contains('TEST').value_counts()
no_test_start_data['end_station_name'].str.contains('TEST').value_counts()
# 254 test rides found in rest end station id's. All should not be
considered:
# Deleting test rides:
no_nan_no_test_data =
no_test_start_data[no_test_start_data['end_station_id'].str.contains(
    'TEST') != True]
# 4472680 rows left
```

Preparing the data:

Adding new column to calculate ride time:

```
# Dataframe with no test rides and no NaN values:
no_nan_no_test_data.head()

# Creating copy of the dataframe to add new column:
df_ridetime = no_nan_no_test_data.copy(deep=True)

# Adding new column with ride time:
df_ridetime.insert(
    loc=4, column='ride_time[s]', value=df_ridetime['ended_at'] -
df_ridetime['started_at'])
# Changing ride time to seconds:
df_ridetime['ride_time[s]'] = df_ridetime['ride_time[s]'].astype(
    'timedelta64[s]')
```

All time values were calculated into seconds for further analysis.

Checking for negative and irrelevant ride times:

Negative ride time value means an input error which should not be considered. For the analysis purpose also the ride time below 60 seconds won’t be taken into account.

```
# Sorting data by ride time:
df_rt_sorted = df_ridetime.sort_values(by='ride_time[s]')

# Checking for negative ride time values:
df_rt_sorted[df_rt_sorted['ride_time[s]'] < 0]
df_rt_sorted[df_rt_sorted['started_at'] > df_rt_sorted['ended_at']]
# 71 wrong inputs with negative ride times. Deleting:
df_rt_no_neg = df_rt_sorted[df_rt_sorted['ride_time[s]'] > 0]
# Checking for the low ride time values, assuming these are incorrect or
irrelevant records (below 60s):
df_rt_no_neg[df_rt_no_neg['ride_time[s]'] < 60]
# There are 73439 inputs with ride time below 60s. Deleting:
df_rt_cleaned = df_rt_sorted[df_rt_sorted['ride_time[s]'] >= 60]

# Data sorted by date:
df_sort_date = df_rt_cleaned.sort_values(by="started_at")
```

Deleting irrelevant columns such as ride id's, station id's and station coordinates:

```
# For the analysis purposes ride id's, station id's and coordinates are
irrelevant.
df_sort_date.columns
columns_to_be_dropped = ['ride_id', 'start_station_id',
                         'end_station_id', 'start_lat', 'start_lng',
'end_lat', 'end_lng']
df_drop_columns = df_sort_date.drop(columns=columns_to_be_dropped)
```

Adding columns with the day of the week for each ride (0 – Monday, 6 – Sunday):

```
## Adding columns with the day of the week that each ride started
# 0 - Monday, 6 - Sunday:
df_drop_columns.insert(loc=4, column='weekday',
value=df_ridetime['started_at'].dt.weekday)

all_rides = df_drop_columns.copy(deep=True)
```

Creating new column names for cleaned dataset:

```
all_cleaned_rides_new_columns = all_rides.copy()

all_cleaned_rides_new_columns.columns = ['BikeType',
                                        'RideStart',
                                        'RideEnd',
                                        'RideTime[s]',
                                        'Weekday',
                                        'StartStation',
                                        'EndStation',
                                        'UserType']
```

Exporting clean dataset, ready for analysis:

```
# EXPORTING CLEAN DATA:
all_cleaned_rides_new_columns.to_csv('cleaned_data/cyclistic_202110-
202209_cleaned.csv')
```