UDACITY

Back to Data Analyst Nanodegree

Explore US Bikeshare Data

REVIEW

CODE REVIEW 5

HISTORY

home/bikeshare.py 5

```
1 import time
2 import pandas as pd
```

AWESOME

Pandas and numpy are useful packages for analysis of data.
We want our students to learn the usage of these packages..

```
3 import numpy as np
4
5 CITY_DATA = { 'chicago': 'chicago.csv',
6               'new york city': 'new_york_city.csv',
7               'washington': 'washington.csv' }
8 months = ['january', 'february', 'march', 'april', 'may', 'june']
9 days = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']
10 isMonthSpecified = False
11 isDayOfWeekSpecified = False
12
13 def get_filters():
14     """
15     Asks user to specify a city, month, and day to analyze.
16
17     Returns:
           (str) city - name of the city to analyze
18         (str) month - name of the month to filter by, or "all" to apply no month filte
20         (str) day - name of the day of week to filter by, or "all" to apply no day fi
```

```
21      """
22
23      print('Hello! Let\'s explore some US bikeshare data!')
24      # TO DO: get user input for city (chicago, new york city, washington). HINT: Use a
25      print ('\n\nWhich of the below city data would you like to explore?')
26      print ('Chicago, New York City, Washington')
27
28      while True:
29          try:
30              city = input('Type City name to begin: ')
31              city = city.lower()
```

AWESOME

By using the lower() function you have made the user inputs case agnost.
this feature increases the robustness of user input and makes the code more usable..

```
32              CITY_DATA[city]
33              break
34          except KeyError:
35              print ('Incorrect Entry, please try again')
36
37      # TO DO: get user input for month (all, january, february, ... , june)
38      print ('\n\nWhich of the below months are you interested in ?')
39      print ('January, February, March, April, May, June or all')
40      months = ['january', 'february', 'march', 'april', 'may', 'june']
41      while True:
42          try:
43              month = input('Type Month name to begin: ')
44              month = month.lower()
45              if month == 'all':
46                  break
47              months.index(month)
48              global isMonthSpecified
49              isMonthSpecified = True
50              break
51          except ValueError:
52              print ('Incorrect Entry, please try again')
53
54      # TO DO: get user input for day of week (all, monday, tuesday, ... sunday)
55      print ('\n\nWhich of the below days would you like to look into ?')
56      print ('Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday or all')
57      while True:
58          try:
59              day = input('Type Day name to begin: ')
60              day = day.lower()
61              if day == 'all':
62                  break
63              days.index(day.lower())
64              global isDayOfWeekSpecified
65              isDayOfWeekSpecified = True
66              break
67          except ValueError:
68              print ('Incorrect Entry, please try again')
69
70      print('-'*40)

71      return city, month, day
72
```

```
73
74 def load_data(city, month, day):
75     """
76     Loads data for the specified city and filters by month and day if applicable.
77
78     Args:
79         (str) city - name of the city to analyze
```

AWESOME

I see that you have added comments..
However I do have a suggestion..
There is a scope for more comments. Comments increase the readability of the code.

```
80         (str) month - name of the month to filter by, or "all" to apply no month filte
81         (str) day - name of the day of week to filter by, or "all" to apply no day fi
82     Returns:
83         df - pandas DataFrame containing city data filtered by month and day
84     """
85     start_time = time.time()
86     # load data file into a dataframe
87     df = pd.read_csv(CITY_DATA[city])
88
89     # convert the Start Time column to datetime
90     df['Start Time'] = pd.to_datetime(df['Start Time'])
91
92     # extract month and day of week from Start Time to create new columns
93     df['month'] = df['Start Time'].dt.month
94     df['day_of_week'] = df['Start Time'].dt.dayofweek
95
96     # filter by month if applicable
97     if month != 'all':
98         # use the index of the months list to get the corresponding int
99         month = months.index(month)
100        month += 1
101
102        # filter by month to create the new dataframe
103        is_it_month = df['month']==month
104        df = df[is_it_month]
105        #print ('\nPrint:\n')
106        #print (df.head())
107
108    # filter by day of week if applicable
109    if day != 'all':
110        # filter by day of week to create the new dataframe
111        days = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', '
112        day = days.index(day)
113
114        # filter by day to create the new dataframe
115        is_it_day = df['day_of_week']==day
116        df = df[is_it_day]
117
118    print("\nThis took %s seconds." % (time.time() - start_time))
119    #print (df.head())
120    return df
121
122 def time_stats(df):
```

AWESOME

Function implementation looks good.
Methods are articulated well and used effectively.

```
123     """Displays statistics on the most frequent times of travel."""
124
125     print('\nCalculating The Most Frequent Times of Travel...\n')
126     start_time = time.time()
127
128     if isMonthSpecified == False:
129         # display the most common month
130         df['month'] = df['Start Time'].dt.month
131         #print (df['month'].value_counts().idxmax())
132         most_common_month_index = df['month'].value_counts().idxmax()
133         print (months[most_common_month_index-1].title(), 'is the most common month')
134
135     if isDayOfWeekSpecified == False:
136         # display the most common day of week
137         df['day_of_week'] = df['Start Time'].dt.dayofweek
138         #print (df['day_of_week'].value_counts().idxmax())
139         most_common_day_of_week_index = df['day_of_week'].value_counts().idxmax()
140         print (days[most_common_day_of_week_index].title(), 'is the most common day o
141
142     # display the most common start hour
143     df['hour'] = df['Start Time'].dt.hour
144     #print (df['hour'].value_counts().idxmax())
145     most_common_hour_index = df['hour'].value_counts().idxmax()
146     print (most_common_hour_index, 'is the most common hour of the day')
147     print ('\n')
148
149
150     print("\nThis took %s seconds." % (time.time() - start_time))
151     print('-'*40)
152
153
154 def station_stats(df):
155     """Displays statistics on the most popular stations and trip."""
156
157     print('\nCalculating The Most Popular Stations and Trip...\n')
158     start_time = time.time()
159
160     # display most commonly used start station
161     most_common_start_station_index = df['Start Station'].value_counts().idxmax()
162     print (most_common_start_station_index, 'is the commonly used Start Station')
163
164     # display most commonly used end station
165     most_common_end_station_index = df['End Station'].value_counts().idxmax()
166     print (most_common_end_station_index, 'is the commonly used End Station')
167     print ('\n')
168
169     # display most frequent combination of start station and end station trip
170     df = df.groupby(['Start Station', 'End Station']).size().reset_index().rename(colu
171     #print(df['count'].head())
172     most_common_start_end_station_index = df['count'].value_counts().idxmax()
173     start_station = df.iloc[most_common_start_end_station_index, df.columns.get_loc("S
174     end_station = df.iloc[most_common_start_end_station_index, df.columns.get_loc("End
175     print (start_station, '-', end_station, 'are the commonly used combination of Star
```

```
    print ('\n')
    #print (most_common_start_end_station_index, ' is the commonly used Start-End Stat
    #print (df.iloc[most_common_start_end_station_index, df.columns.get_loc("Start Sta
    #print (df.columns.get_loc("Start Station"))

    print("\nThis took %s seconds." % (time.time() - start_time))
    print('-'*40)


def trip_duration_stats(df):
    """Displays statistics on the total and average trip duration."""

    print('\nCalculating Trip Duration...\n')
    start_time = time.time()

    # display total travel time
    print (df['Trip Duration'].sum(), 'seconds is the total travel time')

    # display mean travel time
    print (df['Trip Duration'].mean(), 'seconds is the mean travel time')
    print ('\n')

    print("\nThis took %s seconds." % (time.time() - start_time))
    print('-'*40)


def user_stats(df):
    """Displays statistics on bikeshare users."""

    print('\nCalculating User Stats...\n')
    start_time = time.time()

    # Display counts of user types
    #print (df.groupby(['User Type']).size().reset_index().rename(columns={0:'count'}
    if 'User Type' in df:
        mod_df = df.groupby(['User Type']).size().reset_index().rename(columns={0:'cou
        #print (mod_df.head())
        #column_loc = mod_df.columns.get_loc("User Type")
        #print (mod_df.iloc[0, column_loc], "are", mod_df.iloc[0, column_loc+1], "in n
        #print (mod_df.iloc[1, column_loc], "are", mod_df.iloc[1, column_loc+1], "in n

        for index, row in mod_df.iterrows():
            print(row['User Type'], "are", row['count'], "in number")
        print ('\n')

    # Display counts of gender
    if 'Gender' in df:
        mod_df = df.groupby(['Gender']).size().reset_index().rename(columns={0:'count
        #print (mod_df.head())
        #print (mod_df.iloc[0, column_loc], "are", mod_df.iloc[0, column_loc+1], "in n
        #print (mod_df.iloc[1, column_loc], "are", mod_df.iloc[1, column_loc+1], "in n
        for index, row in mod_df.iterrows():
            print(row['Gender'], "are", row['count'], "in number")
        print ('\n')

    # Display earliest, most recent, and most common year of birth
    if 'Birth Year' in df:
        print (df['Birth Year'].min(), 'is the oldest person\'s year of birth')
        print (df['Birth Year'].max(), 'is the youngest person\'s year of birth')
        print (df['Birth Year'].value_counts().idxmax(), 'is the common person\'s year
        print ('\n')
```

```
237
238     print("\nThis took %s seconds." % (time.time() - start_time))
239     print('-'*40)
240
241
242 def main():
243     while True:
244         global isDayOfWeekSpecified
245         isDayOfWeekSpecified = False
246         global isMonthSpecified
247         isMonthSpecified = False
248
249         city, month, day = get_filters()
250         #city = 'chicago'
251         #month = 'all'
252         #day = 'all'
253         df = load_data(city, month, day)
```

AWESOME

Looks good. But I would provide a suggestion,
sometimes when we display raw data it is difficult to read 100 odd rows of data, therefore it is advisable tc
If you have time, write a code that will display 5 lines at a time and would the user "if he needs to read mo

```
254
255         time_stats(df)
256         station_stats(df)
257         trip_duration_stats(df)
258         user_stats(df)
259
260         restart = input('\nWould you like to restart? Enter yes or no.\n')
261         if restart.lower() != 'yes':
262             break
263
264
265 if __name__ == "__main__":
266     main()
267
```

RETURN TO PATH