UDACITY

‹ Return to "Data Analyst Nanodegree" in the classroom

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
6 CITY_DATA = { 'chicago': 'chicago.csv',
7               'new york': 'new_york_city.csv',
8               'washington': 'washington.csv' }
9
10 def get_filters():
11     """
12     Asks user to specify a city, month, and day to analyze.
13     Returns:
14         (str) city - name of the city to analyze
```

```
        (str) day - name of the day of week to filter by, or "all" to apply
    """

    invalid_inputs = "Invalid input. Please try again with the mentioned ci

    print('Hey there! Let\'s now explore some US bikeshare data!')
    # TO DO: get user raw_input for city (chicago, new york city, washingto
    while 1 == 1 :
        city = input("\nEnter the name of the city to analyze,City names ar
```

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..

```
        if city in ['chicago', 'new york', 'washington']:
            break
        else:
            print(invalid_inputs)

    # TO DO: get user raw_input for month (all, january, february, ... , ju
    while 1 == 1 :
        month = input("\n Enter the month for which you want to analyze:\nj
            "\napril,\nmay,\njune\nto filter by, or \"all\" to apply no mon
        if month in ["january", "february", "march", "april", "may", "june"
            break
        else:
            print(invalid_inputs)

    # TO DO: get user raw_input for day of week (all, monday, tuesday, ...
    while 1 == 1 :
        day = input("\nEnter the day for which you want the data \nMonday,\
            "\nFriday,\nSaturday,\nSunday\nof week to filter by, or \"all\"
        if day in ["monday", "tuesday", "wednesday", "thursday", "friday",
            break
        else:
            print(invalid_inputs)

    print('-'*40)
    return city, month, day


def load_data(city, month, day):
    """
    Loads data for the specified city and filters by month and day if appli
    Args:
        (str) city - name of the city to analyze
        (str) month - name of the month to filter by, or "all" to apply no
        (str) day - name of the day of week to filter by, or "all" to apply
```

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.

```
    Returns:
        df - Pandas DataFrame containing city data filtered by month and da
    """
    file_name = CITY_DATA[city]
    print ("Accessing data from: " + file_name)
    df = pd.read_csv(file_name)

    # convert the Start Time column to datetime
    df['Start Time'] = pd.to_datetime(arg = df['Start Time'], format = '%Y-

    # filter by month if applicable
    if month != 'all':
        # extract month and day of week from Start Time to create new colum
        df['month'] = df['Start Time'].dt.month

        # use the index of the months list to get the corresponding int
        months = ['january', 'february', 'march', 'april', 'may', 'june']
        month = months.index(month) + 1

        # filter by month to create the new dataframe
        df = df.loc[df['month'] == month]

    # filter by day of week if applicable
    if day != 'all':
        df['day_of_week'] = df['Start Time'].dt.weekday_name

        # filter by day of week to create the new dataframe
        df = df.loc[df['day_of_week'] == day.title()]
    return df


def time_stats(df):
```

AWESOME

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

```
    """Displays statistics on the most frequent times of travel."""

    print('\nMost Frequent Times of Travel...\n')
    start_time = time.time()

    # Convert the Start Time column to datetime
    df['Start Time'] = pd.to_datetime(arg = df['Start Time'], format = '%Y-

    # Create new columns for month, weekday, hour
    month = df['Start Time'].dt.month
```

```
    hour = df['Start Time'].dt.hour

    # TO DO: display the most common month
    most_common_month = month.mode()[0]
    print('Most common month: ', most_common_month)

    # TO DO: display the most common day of week
    most_common_day_of_week = weekday_name.mode()[0]
    print('Most common day of week: ', most_common_day_of_week)

    # TO DO: display the most common start hour
    common_start_hour = hour.mode()[0]
    print('Most frequent start hour: ', common_start_hour)

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


def station_stats(df):
    """Displays statistics on the most popular stations and trip."""

    print('\nMost Popular Stations and Trip...\n')
    start_time = time.time()

    # TO DO: display most commonly used start station
    print('Most commonly used start station:', df['Start Station'].value_co

    # TO DO: display most commonly used end station
    print('Most commonly used end station:', df['End Station'].value_counts

    # TO DO: display most frequent combination of start station and end sta
    combine_stations = df['Start Station'] + "*" + df['End Station']
    common_station = combine_stations.value_counts().idxmax()
    print('Most frequent trips are:\n{} \nto\n{}'.format(common_station.spl

    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()
    # Convert seconds to readable time format
    def secs_to_readable_time(seconds):
        m, s = divmod(seconds,60)
        h, m = divmod(m,60)
        d, h = divmod(h,24)
        y, d = divmod(d,365)
        print('Years: {}, Days: {}, Hours: {}, Mins: {}, Secs: {}'.format(y

    # TO DO: display total travel time
    total_travel_time = df['Trip Duration'].sum()
    print('Total travel time:\n')
    secs_to_readable_time(total_travel_time)

```

```
    mean_travel_time = df['Trip Duration'].mean()
    print('\nMean travel time: {} seconds'.format(mean_travel_time))

    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()

    # TO DO: Display counts of user types
    user_types = df['User Type'].value_counts()
    print(user_types)

    # TO DO: Display counts of gender
    if 'Gender' in df.columns:
        gender_count = df['Gender'].value_counts()
        print(gender_count)

    # TO DO: Display earliest, most recent, and most common year of birth
    if 'Birth Year' in df.columns:
        earliest_birth_year = df['Birth Year'].min()
        most_recent_birth_year = df['Birth Year'].max()
        common_birth_year = df['Birth Year'].mode()[0]
        print("\nEarliest year of birth: " + str(earliest_birth_year))
        print("\nMost recent year of birth: " + str(most_recent_birth_year))
        print("\nMost common year of birth: " + str(common_birth_year))

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

def raw_data(df):
    user_input = input('Do you want to see raw data? Enter yes or no.\n')
    line_number = 0

    while 1 == 1 :
        if user_input.lower() != 'no':
            print(df.iloc[line_number : line_number + 5])
            line_number += 5
```

AWESOME

Looks good.

Sometimes when we display raw data it is difficult to read 100 odd rows of data, therefore it is

You have written the code that will display 5 lines at a time and would the user "if he needs to

```
            user_input = input('\nDo you want to see more raw data? Enter y
        else:
            break

```

```
    while 1 == 1 :
        city, month, day = get_filters()
        df = load_data(city, month, day)

        time_stats(df)
        station_stats(df)
        trip_duration_stats(df)
        user_stats(df)
        raw_data(df)
        restart = input('\nWould you like to restart? Enter yes or no.\n')
        if restart.lower() != 'yes':
            break


if __name__ == "__main__":
    main()
```

RETURN TO PATH