UDACITY

Return to Classroom

Explore US Bikeshare Data

REVIEW

CODE REVIEW

HISTORY

Requires Changes

2 specifications require changes

Dear Student,
You have done a great job writing functional python code for most part, **producing some interesting statistics like most popular stations, peak day, peak hour of bike use etc., which are really helpful for future bike users.** However, you need to work a little more on this project to meet all the specifications (please see below). Since you have already addressed most of the requirements, it is just a matter of paying attention to some finer details. I am sure you will be able to quickly get this project to meet all specifications as you have a very good coding skills in **python**.

To pass this project, you need to address only the comments marked as **Requires Changes**. The comments marked as **Suggestions** are optional and you do not need to address them to pass this project. But if you address these suggestions, it will improve your project.

Mainly, you need to **make the following changes** before resubmission, details of which can be found in the corresponding rubric item below.

- **all invalid user inputs** should be handled graciously without producing execution error. Prompt user to enter correct input if they enter invalid input.

- as **per project instruction**, you need to prompt the user whether he/she would like to see the raw data and **print raw data** in an interactive manner.

Code Quality

All code cells can be run without error.

Tips: Implement safeguards against invalid user inputs that can potentially break the codes. Please refer to the "Solicit and handle raw user input" rubric item for further details.

Requires Changes

For month, if I enter **December**, which is a valid month, your code produced the **following error.** There are two issues with this. Please note that you have data only for first 6 months in this dataset and a user would not know this. So you should mention this in your message and ask user to select only one of these 6 months. For instance, your input message could be ***"Enter any one of the first 6 months or enter All to select all 6 months."***. Second, even if a user enter a wrong month, a good script should **handle any invalid user input graciously**, because users make all sorts of mistakes. If a user enter invalid input, your code should let the user know that the input is invalid and ask them to provide the correct input. It should not produce error for invalid input, which leads to poor user experience.

```
Traceback (most recent call last):
  File "bikeshare.py", line 309, in <module>
    main()
  File "bikeshare.py", line 298, in main
    time_stats(df,city, month, day)
  File "bikeshare.py", line 161, in time_stats
    print("*> Weekday that had high frequency travel is : ",df.weekdayname.mode().va
lues[0])
IndexError: index 0 is out of bounds for axis 0 with size 0
```

Suggestions

To your yes/no question related to **restart**, if I type `ye` by mistake instead of `yes`, your code simply treat this response as `no`. Instead, you should let the user know that it is an invalid input and seek valid input. Basically, you should consider any input other than `yes` or `no` as invalid, because it could just be a mistake. **I agree that this code is already given as a starter code. But you can improve upon the given code to make it more robust to mistakes in user input.**

In input message, you should tell the user what all cities they can choose from. Otherwise how would they know what city to enter? For instance, your input message could be "**Please enter Chicago, Washington or New York City for your analysis**".

Appropriate data types (e.g. strings, floats) and data structures (e.g. lists, dictionaries) are chosen to carry out the required analysis tasks.

You handled the **datatypes** like string well and appropriately used **data structures** like list and dictionaries. If you would like to know more about data types and data structures, which are fundamental for mastering any programming language, here are some good resources.
Data Structures
Data Types

Loops and conditional statements are used to process the data correctly.

Packages are used to carry out advanced tasks.

Functions are used to reduce repetitive code.

Good job with writing appropriate **functions** to achieve modularity of code and avoid repetition. When the code is functional like this, it is much easier to **update** the code, as you have to update in only one place. This makes code **maintenance** easier.

Docstrings, comments, and variable names enable the readability of the code.

Tips: Please refer to the Python's documentation PEP 257 – Docstring Conventions. Example of docstring conventions:

```
def function(a, b):
    """Do X and return a list."""
```

Good job including a **docstring** in all functions to explain the purpose of a function. **It makes it easy to follow your function**. If you would like to know more about what docstrings are, how to write good one-line or multi-line docstrings please see the following link; https://www.python.org/dev/peps/pep-0257/.

Script and Questions

Raw input is solicited and handled correctly to guide the interactive question-answering experience; no errors are thrown when unexpected input is entered.

User inputs should be made case insensitive, which means the input should accept the string of "Chicago" and its case variants, such as "chicago", "CHICAGO", or "cHicAgo".

You should also implement error handlings so your program does not throw any errors due to invalid inputs. For example, if the user enters "Los Angeles" for the city, the error handling should reject the user input and avoid breaking the codes.

Good job handling user input in a **case-insensitive manner**, which is a good practice as users do not always supply input in a case-sensitive manner. User inputs like `Chicago`, `chicago`, `CHICAGO` all work as you have appropriately converted case of user input before making comparisons.

Descriptive statistics are correctly computed and used to answer the questions posed about the data.

Raw data is displayed upon request by the user in the following manner:

- Your script should prompt the user if they want to see 5 lines of raw data,
- Display that data if the answer is 'yes',
- Continue iterating these prompts and displaying the next 5 lines of raw data at each iteration,
- Stop the program when the user says 'no' or there is no more raw data to display.

Tips: you can implement the `while` loop and track the row index in order to display the continuous raw data.

Good job computing all the **required statistics**.

Well done formatting **years** appropriately instead of using default **decimal (float)** format.

Requires Changes

Your script **does not display raw data** in an interactive manner. As specified in the section **An Interactive Experience** in Project: Explore US Bikeshare Data - Sublesson 3. Code Walkthrough, *your script also needs to prompt the user whether they would like to see the raw data. If the user answers 'yes,' then the script should print 5 rows of the data at a time, then ask the user if they would like to see 5 more rows of the data. The script should continue prompting and printing the next 5 rows at a time until the user chooses 'no'.*

So please write a function which could be something like the one given below. **Then call the function at appropriate place inside the `main()` function.**

```
def display_raw_data(df):
    """ Your docstring here """
    i = 0
    raw = input("<your input message here>") # TO DO: convert the user input to lowe
r case using lower() function
    pd.set_option('display.max_columns',200)

    while True:
        if raw == 'no':
            break
        elif raw == 'yes':
            print(df[???]) # TO DO: appropriately subset/slice your dataframe to dis
play next five rows
            raw = input("<your input message here>") # TO DO: convert the user input
to lower case using lower() function
            i += 5
        else:
            raw = input("\nYour input is invalid. Please enter only 'yes' or 'n
o'\n").lower()
```

Suggestions

Please format your numbers related to time using round() function. It does not make much sense to use decimals like `7.55738383` with seconds/minutes. Instead it should be formatted to `7.6` seconds.

Please print statistics related to **Popular times of travel (Total travel time and Average travel time)** with **time unit** (is it seconds, minutes, hours or days?). Printing just numbers without time unit is not useful.

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

START