UDACITY

< Return to Classroom

Explore US Bikeshare Data

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear Student,
You have put dedicated effort into this project and it paid off. Congratulations on meeting all the specifications of the project! You have demonstrated a very good **python** coding skills, **producing some interesting statistics like most popular stations, peak day, peak hour of bike use etc., which are really helpful for future bike users.**

You also did a fantastic job of incorporating **the previous reviewer** suggestions. **As a different reviewer, I have left some additional comments.** I made these comments marked as **Suggestions** to help you improve the project. **I**t does not require you to resubmit the project. You have already passed the project. **Congratulations!** If you are uploading this project to your portfolio or sharing it with your potential employer, it is a good idea to address these comments. It also gives you an opportunity to appreciate the complete essence of this project. Keep up all the great work you are doing. Good luck with your future projects!

Here are a few resources that may help your continued learning:

- To further develop your skills in **python**, you can practice with HackerRank, which challenges you with increasingly difficult problems.
- **PEP8** is the style guide for python. This style guide provides guidelines and best practices on how to write Python code to improve the readability of code and make it consistent across the wide spectrum of Python code. You can take a look at this guide here; https://www.python.org/dev/peps/pep-0008/ and should strive to adhere to these guidelines.

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.**

Excellent job writing functional code, which ran without any errors. You have appropriately handled the unavailability of gender and birth year columns in Washington data.

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. You can handle this the way you handled invalid city. **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.**

Please correct the following input message. It is about days, not month as you mentioned.

```
Accepted input: Full name of month or all in title or lower case e.g. "Tuesday" or "tuesday".
```

Only when a user have entered an invalid city or month, your code asks them to choose only a city from available cities or a month from first 6 months. If you tell this in first input message itself, it provides a better user experience. Because a user may not know what cities and months data is available.

✓

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

If you would like to know more about data structures and data types, 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. You did a really good job wrapping the code to provide raw data into the following function.

```
def display_data(df):
    """
    Load the first 10 rows of data for the specified city if asked by users.
    """

    print('\nWould you like to view 10 rows of individual trip data?')
    print('\nAccepted input: "Yes" or "No" in title or lower case.')
    view_data = input('Enter "Yes" or "No".\n').title()
    start_loc = 0
    while view_data == 'Yes':
        print(df.iloc[start_loc:start_loc + 10])
        start_loc += 10
        continue_view = input('Do you wish to continue?').title()
        if continue_view != 'Yes':
            break
    print('-'*40)
```

✓

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. 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.

Suggestions

Please note that, you should handle all user input in a case-insensitive manner. While you have handled some of the user inputs in a case-insensitive manner, other user inputs are not handled in the same manner. For instance, if I enter `CHICAGO` instead of `chicago`, your code does not accept the input. In all cases, you should make use of lower() function to convert the user input to lower case before making comparisons.

Note: I saw that you ask users to enter name in lower or title case. But there is no need to do this as this is not user friendly code. Instead with just use of lower() function you can make your code case-insensitive, which is much more user friendly.

✓

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 calculating all the required statistics.
Good job prompting user if they want to see 10 lines of raw data and displaying as much data as user wanted. It is nice to see that you have interacted with users in a very efficient manner. This is a challenging task, where many students struggle. Nice job!
Note: project requirement is 5 lines of raw data, but you printed 10 lines, which is fine.

The data is there for Sunday as well. So you should not exclude Sunday.

DOWNLOAD PROJECT

RETURN TO PATH