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. It 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.

You have done an excellent job handling **invalid user input.** Handling invalid inputs graciously as you did is quite important in any application as users tend to make all kinds of mistakes while entering input.

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.**
Note: the above comment applies to the question related to **displaying raw data** as well.

You should give an option to filter by both **month and day.** Someone may want to see data for say **Sundays** of **June**.

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.

Suggestions

Please note that each of the task you perform should ideally have its own function so it provides modularity to the code. This makes code update or maintenance much easier. So please wrap the code to print raw data in a separate function and call it inside main() function.

✓

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."""
```

Suggestions

You have added a **Docstring** in all functions except the `user_stats` function. Please add a docstring to this function as well. All functions should have a docstring to explain the purpose of a function, and how it should be used. 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

- 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**.

Excellent job prompting user if they want to see 5 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!

Well done formatting **years** as `int` instead of default `float`.

Suggestions

By default pandas displays only so many columns so when you display the raw data some of the columns are replaced by **ellipses**. To modify this behavior, just before print statement, add this line `pd.set_option('display.max_columns',200)`. It sets the column to be displayed to 200 (or you can set it to as many columns as you have).

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START