目录

项目说明

本次项目是依据文档信息来解决相应的问题，主要是看文档字符串了解函数作用，查看参数信息了解参数的数据类型以及参数的信息。特别注意⚠️脚本中的 `main` 函数和最后的 `if` 条件语句部分不用修改——可以进行优化，但是不要修改文件结构

1. 变量设置

变量使用，根据变量的作用来判断使用何种方式的变量——全局变量以及局部变量。例如 MONTHS 和 CITY_DATA 等可能在全局使用的可以在文件开始申明并赋值

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import time
import pandas as pd
import numpy as np

CITY_DATA = {'chicago': 'chicago.csv',
             'new york city': 'new_york_city.csv',
             'washington': 'washington.csv'}

MONTHS = ['january', 'february', 'march', 'april', 'may', 'june']
```

2. `get_filters` 函数

根据文档字符串信息，确认需要筛选出来的哪些数据，三个结果需要一定限定条件才能输出，另一个需要解决的是用户交互问题——怎么保障用户输入是正确的值，筛选三个值都是一样的结构，这里隐藏优化点——使用函数。所以该问题需要考虑使用循环以及 input 函数来控制交互，使用条件语句去控制循环

```
city = input("Choose city:")
CITY_DATA = { 'chicago': 'chicago.csv',
              'new york city': 'new_york_city.csv',
              'washington': 'washington.csv' }

while True:
    if city in CITY_DATA:
        break
    else:
        print("Your value is valide")
        city = input("Choose city:")
```

使用函数进行优化：

```
def option(conditions, tip_message, error_message):
    value = input(tip_message)

    while True:
        if value in conditions:
            break
        else:
            print(error_message)
            value = input(tip_message)
    return value


city = option(CITY_DATA, "Choose city:", "Your value is valide")
month = option(MONTHS, "Choose month:", "Your value is valide")
```

2. `load_data` 函数

需要根据前面 `get_filter` 函数得到的 city、month 和 day 来收集和筛选数据，其中还需要使用像一个相应的日期进行处理，输出相应的值

```
# 读取数据
df = pd.read_csv(file)

# 列表数据得到对应值——解决某个值在列表中的第一个索引值，在本例中列表是唯一值，所以可以筛选出对应的月份和周数的数字值
months = ["a", "b", "c", "d"]
month = 'a'
months.index(month)

# output
0

# 因为需要对应 dataframe 中对应的月份和周数，所以需要加 1
```

```
months.index(month) + 1
# output
1

# 数据筛选可以使用一般掩码筛选
df[df["month"] == 1]

# 对于多条件筛选的方式，月数是 1 周数是 3 的数据
df[(df["month"] == 1) and (df["day"] == 3)]
```

3. 其他函数说明

- 针对**最常见**的数值处理——包括最常见月份、周数、起始时间、起始和终止站点，需要筛选出数据中的众数

```
# 针对筛选对应列的众数
a = pd.DataFrame({'a':['a', 'b', 'a', 'c', 'b', 'a', 'b', 'c', 'd',
        'a'], 'b':['c', 'c', 'd', 'a', 'b', 'a', 'a','b', 'a',
'd']})

a
# output

   a  b
0  a  c
1  b  c
2  a  d
3  c  a
4  b  b
5  a  a
6  b  a
7  c  b
8  d  a
9  a  d

# 筛选 a 中 a 列的众数
In [23]: a['a'].mode()
Out[23]:
0    a
dtype: object

# 可以筛选出对应的行值
In [24]: a['a'].mode()[0]
Out[24]: 'a'
```

- 常见的起点站和终点站组合，这里的语句已经说明了是需要计算频繁的车程，可以想一下现实情况中怎么表现频繁线段——是否可以考虑为计算线段的两头端点的组合频繁。假设上海，北京，成都以及广州等城市之间的频繁车程：我们可以考虑为统计起始点和终点 `北京-成都`，`北京-上海`，`成都-广州`，`成都-北京`，`成都-上海` 等组合方式的统计数量就可以计算出来了。这里的解决方式，可以将两个端点数据构造为一个新的列再计算众数

```
In [15]: a['c'] = a['a'] + a['b']

In [16]: a
Out[16]:
   a  b   c
0  a  c  ac
1  b  c  bc
2  a  d  ad
3  c  a  ca
4  b  b  bb
5  a  a  aa
6  b  a  ba
7  c  b  cb
8  d  a  da
9  a  d  ad

In [20]: a['c'].mode()[0]
Out[20]: 'ad'
```

- 总旅行和平均旅行时间，这个是需要通过聚合计算，也就是使用 `Series` 的 `sum` 和 `mean` 方法

```
In [28]: b = pd.Series([1,2, 3, 4, 5,6], name="trip")

In [29]: b.mean()
Out[29]: 3.5

In [30]: b.sum()
Out[30]: 21
```

- 分别统计用户信息的数量——用户的类型和用户性别的统计量，可以使用 `Series` 的 `value_counts` 方法进行统计

```
In [26]: a["a"].value_counts()
Out[26]:
a    4
b    3
c    2
d    1
Name: a, dtype: int64
```