Exercise1

Finding consistency

In this exercise and throughout this chapter, you'll be working with the airlines DataFrame which contains survey responses on the San Francisco Airport from airline customers.

The DataFrame contains flight metadata such as the airline, the destination, waiting times as well as answers to key questions regarding cleanliness, safety, and satisfaction. Another DataFrame named categories was created, containing all correct possible values for the survey columns.

In this exercise, you will use both of these DataFrames to find survey answers with inconsistent values, and drop them, effectively performing an outer and inner join on both these DataFrames as seen in the video exercise. The pandas package has been imported as pd, and the airlines and categories DataFrames are in your environment.

Instructions

- Print the categories DataFrame and take a close look at all possible correct categories of the survey columns.
- Print the unique values of the survey columns in airlines using the .unique() method.
- Create a set out of the cleanliness column in airlines using set() and find the inconsistent category by finding the difference in the cleanliness column of categories.
- Find rows of airlines with a cleanliness value not in categories and print the output.
- Print the rows with the consistent categories of cleanliness only.

In [ ]:
```
# Print categories DataFrame
print(____)

# Print unique values of survey columns in airlines
print('Cleanliness: ', airlines['cleanliness'].____, "\n")
print('Safety: ', ____, "\n")
print('Satisfaction: ', ____, "\n")

# Find the cleanliness category in airlines not in categories
cat_clean = ____(____).____(____)

# Find rows with that category
cat_clean_rows = airlines['cleanliness'].____(____)

# Print rows with inconsistent category
print(airlines[____])

# Print rows with consistent categories only
print(airlines[____])

#_____________________#
#Solutions

# Print categories DataFrame
print(categories)

# Print unique values of survey columns in airlines
print('Cleanliness: ', airlines['cleanliness'].unique(), "\n")
print('Safety: ', airlines['safety'].unique(), "\n")
print('Satisfaction: ', airlines['satisfaction'].unique(),"\n")

# Find the cleanliness category in airlines not in categories
cat_clean = set(airlines['cleanliness']).difference(categories['cleanliness'])

# Find rows with that category
cat_clean_rows = airlines['cleanliness'].isin(cat_clean)

# Print rows with inconsistent category
print(airlines[cat_clean_rows])

# Print rows with consistent categories only
print(airlines[~cat_clean_rows])
```

Exercise2

Inconsistent categories

In this exercise, you'll be revisiting the airlines DataFrame from the previous lesson.

As a reminder, the DataFrame contains flight metadata such as the airline, the destination, waiting times as well as answers to key questions regarding cleanliness, safety, and satisfaction on the San Francisco Airport.

In this exercise, you will examine two categorical columns from this DataFrame, dest_region and dest_size respectively, assess how to address them and make sure that they are cleaned and ready for analysis. The pandas package has been imported as pd, and the airlines DataFrame is in your environment.

Instructions

- Print the unique values in dest_region and dest_size respectively.
- Change the capitalization of all values of dest_region to lowercase.
- Replace the 'eur' with 'europe' in dest_region using the .replace() method.
- Strip white spaces from the dest_size column using the .strip() method.
- Verify that the changes have been into effect by printing the unique values of the columns using .unique().

In [ ]:
```
# Print unique values of both columns
print(airlines['dest_region'].____())
print(airlines['____'].____())

# Lower dest_region column and then replace "eur" with "europe"
airlines['dest_region'] = airlines['dest_region'].____.____()
airlines['dest_region'] = airlines['dest_region'].____({'____':'____'})

# Remove white spaces from `dest_size`
airlines['dest_size'] = airlines['dest_size'].____.____()

# Verify changes have been effected
print(____)

#_____________________#
#Solutions

# Print unique values of both columns
print(airlines['dest_region'].unique())
print(airlines['dest_size'].unique())

# Lower dest_region column and then replace "eur" with "europe"
airlines['dest_region'] = airlines['dest_region'].str.lower()
airlines['dest_region'] = airlines['dest_region'].replace({'eur':'europe'})

# Remove white spaces from `dest_size`
airlines['dest_size'] = airlines['dest_size'].str.strip()

# Verify changes have been effected
print(airlines['dest_region'].unique())
print(airlines['dest_size'].unique())
```

Exercise3

Remapping categories

To better understand survey respondents from airlines, you want to find out if there is a relationship between certain responses and the day of the week and wait time at the gate.

The airlines DataFrame contains the day and wait_min columns, which are categorical and numerical respectively. The day column contains the exact day a flight took place, and wait_min contains the amount of minutes it took travelers to wait at the gate. To make your analysis easier, you want to create two new categorical variables:

- `wait_type` : `'short'` for 0-60 min, 'medium' for 60-180 and long for 180+
- `day_week` : `'weekday'` if day is in the weekday, 'weekend' if day is in the weekend.

The pandas and numpy packages have been imported as pd and np. Let's create some new categorical data!

Instructions

- Create the ranges and labels for the wait_type column mentioned in the description.
- Create the wait_type column by from wait_min by using pd.cut(), while inputting label_ranges and label_names in the correct arguments.
- Create the mapping dictionary mapping weekdays to 'weekday' and weekend days to 'weekend'.
- Create the day_week column by using .replace().

In [ ]:
```
# Create ranges for categories
label_ranges = [0, 60, ____, np.inf]
label_names = ['short', ____, ____]

# Create wait_type column
airlines['wait_type'] = pd.____(____, bins = ____,
                                labels = ____)

# Create mappings and replace
mappings = {'Monday':'weekday', 'Tuesday':'____', 'Wednesday': '____',
            'Thursday': '____', '____': '____',
            'Saturday': 'weekend', '____': '____'}

airlines['day_week'] = airlines['day'].____(mappings)

#_____________________#
#Solutions

# Create ranges for categories
label_ranges = [0, 60, 180, np.inf]
label_names = ['short', 'medium', 'long']

# Create wait_type column
airlines['wait_type'] = pd.cut(airlines['wait_min'], bins = label_ranges,
                               labels = label_names)

# Create mappings and replace
mappings = {'Monday':'weekday', 'Tuesday':'weekday', 'Wednesday': 'weekday',
            'Thursday': 'weekday', 'Friday': 'weekday',
            'Saturday': 'weekend', 'Sunday': 'weekend'}

airlines['day_week'] = airlines['day'].replace(mappings)
```

Exercise4

Removing titles and taking names

While collecting survey respondent metadata in the airlines DataFrame, the full name of respondents was saved in the full_name column. However upon closer inspection, you found that a lot of the different names are prefixed by honorifics such as "Dr.", "Mr.", "Ms." and "Miss".

Your ultimate objective is to create two new columns named first_name and last_name, containing the first and last names of respondents respectively. Before doing so however, you need to remove honorifics.

The airlines DataFrame is in your environment, alongside pandas as pd.

Instructions

- Remove "Dr.", "Mr.", "Miss" and "Ms." from full_name by replacing them with an empty string "" in that order.
- Run the assert statement using .str.contains() that tests whether full_name still contains any of the honorifics.

In [ ]:
```
# Replace "Dr." with empty string ""
airlines['full_name'] = airlines['full_name'].____.____("____","")

# Replace "Mr." with empty string ""
airlines['full_name'] = ____

# Replace "Miss" with empty string ""
____

# Replace "Ms." with empty string ""
____

# Assert that full_name has no honorifics
assert airlines['full_name'].str.contains('Ms.|Mr.|Miss|Dr.').any() == False


#_____________________#
#Solutions

# Replace "Dr." with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Dr.", "")

# Replace "Mr." with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Mr.", "")

# Replace "Miss" with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Miss", "")

# Replace "Ms." with empty string ""
airlines['full_name'] = airlines['full_name'].str.replace("Ms.", "")

# Assert that full_name has no honorifics
assert airlines['full_name'].str.contains('Ms.|Mr.|Miss|Dr.').any() == False
```

Exercise5

Keeping it descriptive

To further understand travelers' experiences in the San Francisco Airport, the quality assurance department sent out a qualitative questionnaire to all travelers who gave the airport the worst score on all possible categories. The objective behind this questionnaire is to identify common patterns in what travelers are saying about the airport.

Their response is stored in the survey_response column. Upon a closer look, you realized a few of the answers gave the shortest possible character amount without much substance. In this exercise, you will isolate the responses with a character count higher than 40 , and make sure your new DataFrame contains responses with 40 characters or more using an assert statement.

The airlines DataFrame is in your environment, and pandas is imported as pd.

Instructions

- Using the airlines DataFrame, store the length of each instance in the survey_response column in resp_length by using .str.len().
- Isolate the rows of airlines with resp_length higher than 40.
- Assert that the smallest survey_response length in airlines_survey is now bigger than 40.

In [ ]:
```
# Store length of each row in survey_response column
resp_length = ____

# Find rows in airlines where resp_length > 40
airlines_survey = airlines[____ > ____]

# Assert minimum survey_response length is > 40
assert ____.str.len().____ > _____

# Print new survey_response column
print(airlines_survey['survey_response'])

#_____________________#
#Solutions

# Store length of each row in survey_response column
resp_length = airlines['survey_response'].str.len()

# Find rows in airlines where resp_length > 40
airlines_survey = airlines[resp_length > 40]

# Assert minimum survey_response length is > 40
assert airlines_survey['survey_response'].str.len().min() > 40

# Print new survey_response column
print(airlines_survey['survey_response'])
```