Exercise1

Numeric data or ... ?

In this exercise, and throughout this chapter, you'll be working with bicycle ride sharing data in San Francisco called ride_sharing. It contains information on the start and end stations, the trip duration, and some user information for a bike sharing service.

The user_type column contains information on whether a user is taking a free ride and takes on the following values:

- `1` for free riders.
- `2` for pay per ride.
- `3` for monthly subscribers.

In this instance, you will print the information of ride_sharing using .info() and see a firsthand example of how an incorrect data type can flaw your analysis of the dataset. The pandas package is imported as pd.

Instructions

- Print the information of ride_sharing.
- Use .describe() to print the summary statistics of the user_type column from ride_sharing.
- Convert user_type into categorical by assigning it the 'category' data type and store it in the user_type_cat column.
- Make sure you converted user_type_cat correctly by using an assert statement.

In [ ]:

```
# Print the information of ride_sharing
print(____.____())

# Print summary statistics of user_type column
print(ride_sharing['____'].____())

# Convert user_type from integer to category
ride_sharing['user_type_cat'] = ride_sharing['user_type'].____

# Write an assert statement confirming the change
assert ride_sharing['user_type_cat'].____ == '____'

# Print new summary statistics
print(ride_sharing['user_type_cat'].describe())

#_____________________#
#Solutions

# Print the information of ride_sharing
print(ride_sharing.info())

# Print summary statistics of user_type column
print(ride_sharing['user_type'].describe())

# Convert user_type from integer to category
ride_sharing['user_type_cat'] = ride_sharing['user_type'].astype('category')

# Write an assert statement confirming the change
assert ride_sharing['user_type_cat'].dtype == 'category'

# Print new summary statistics
print(ride_sharing['user_type_cat'].describe())
```

Exercise2

Summing strings and concatenating numbers

In the previous exercise, you were able to identify that category is the correct data type for user_type and convert it in order to extract relevant statistical summaries that shed light on the distribution of user_type.

Another common data type problem is importing what should be numerical values as strings, as mathematical operations such as summing and multiplication lead to string concatenation, not numerical outputs.

In this exercise, you'll be converting the string column duration to the type int. Before that however, you will need to make sure to strip "minutes" from the column in order to make sure pandas reads it as numerical. The pandas package has been imported as pd.

Instructions

- Use the .strip() method to strip duration of "minutes" and store it in the duration_trim column.
- Convert duration_trim to int and store it in the duration_time column.
- Write an assert statement that checks if duration_time's data type is now an int.
- Print the average ride duration.

In [ ]:

```
# Strip duration of minutes
ride_sharing['duration_trim'] = ride_sharing['duration'].____.____()

# Convert duration to integer
ride_sharing['duration_time'] = ____

# Write an assert statement making sure of conversion
assert ride_sharing['____'].____ == '____'

# Print formed columns and calculate average ride duration
print(ride_sharing[['duration','duration_trim','duration_time']])
print(____)

#______________________#
#Solutions

# Strip duration of minutes
ride_sharing['duration_trim'] = ride_sharing['duration'].str.strip('minutes')

# Convert duration to integer
ride_sharing['duration_time'] = ride_sharing['duration_trim'].astype('int')

# Write an assert statement making sure of conversion
assert ride_sharing['duration_time'].dtype == 'int'

# Print formed columns and calculate average ride duration
print(ride_sharing[['duration','duration_trim','duration_time']])
print(ride_sharing['duration_time'].mean())
```

Exercise3

Tire size constraints

In this lesson, you're going to build on top of the work you've been doing with the ride_sharing DataFrame. You'll be working with the tire_sizes column which contains data on each bike's tire size.

Bicycle tire sizes could be either 26″, 27″ or 29″ and are here correctly stored as a categorical value. In an effort to cut maintenance costs, the ride sharing provider decided to set the maximum tire size to be 27″.

In this exercise, you will make sure the tire_sizes column has the correct range by first converting it to an integer, then setting and testing the new upper limit of 27″ for tire sizes.

Instructions

- Convert the tire_sizes column from category to 'int'.
- Use .loc[] to set all values of tire_sizes above 27 to 27.
- Reconvert back tire_sizes to 'category' from int.
- Print the description of the tire_sizes.

In [ ]:

```
# Convert tire_sizes to integer
ride_sharing['tire_sizes'] = ____['____'].____('____')

# Set all values above 27 to 27
ride_sharing.____[____ > ____, ____] = ____

# Reconvert tire_sizes back to categorical
ride_sharing['tire_sizes'] = ____

# Print tire size description
print(ride_sharing['tire_sizes'].____())

#______________________#
#Solutions

# Convert tire_sizes to integer
ride_sharing['tire_sizes'] = ride_sharing['tire_sizes'].astype('int')

# Set all values above 27 to 27
ride_sharing.loc[ride_sharing['tire_sizes'] > 27, 'tire_sizes'] = 27

# Reconvert tire_sizes back to categorical
ride_sharing['tire_sizes'] = ride_sharing['tire_sizes'].astype('category')

# Print tire size description
print(ride_sharing['tire_sizes'].describe())
```

Exercise4

Back to the future

A new update to the data pipeline feeding into the ride_sharing DataFrame has been updated to register each ride's date. This information is stored in the ride_date column of the type object, which represents strings in pandas.

A bug was discovered which was relaying rides taken today as taken next year. To fix this, you will find all instances of the ride_date column that occur anytime in the future, and set the maximum possible value of this column to today's date. Before doing so, you would need to convert ride_date to a datetime object.

The datetime package has been imported as dt, alongside all the packages you've been using till now.

Instructions

- Convert ride_date to a datetime object using to_datetime(), then convert the datetime object into a date and store it in ride_dt column.
- Create the variable today, which stores today's date by using the dt.date.today() function.
- For all instances of ride_dt in the future, set them to today's date.
- Print the maximum date in the ride_dt column.

In [ ]:

```
# Convert ride_date to date
ride_sharing['ride_dt'] = pd.____(____['____']).dt.date

# Save today's date
today = ____

# Set all in the future to today's date
ride_sharing.____[____['____'] > ____, '____'] = ____

# Print maximum of ride_dt column
print(ride_sharing['ride_dt'].____())

#______________________#
#Solutions

# Convert ride_date to date
ride_sharing['ride_dt'] = pd.to_datetime(ride_sharing['ride_date']).dt.date

# Save today's date
today = dt.date.today()

# Set all in the future to today's date
ride_sharing.loc[ride_sharing['ride_dt'] > today, 'ride_dt'] = today

# Print maximum of ride_dt column
print(ride_sharing['ride_dt'].max())
```

Exercise5

Finding duplicates

A new update to the data pipeline feeding into ride_sharing has added the ride_id column, which represents a unique identifier for each ride.

The update however coincided with radically shorter average ride duration times and irregular user birth dates set in the future. Most importantly, the number of rides taken has increased by 20% overnight, leading you to think there might be both complete and incomplete duplicates in the ride_sharing DataFrame.

In this exercise, you will confirm this suspicion by finding those duplicates. A sample of ride_sharing is in your environment, as well as all the packages you've been working with thus far.

Instructions

- Find duplicated rows of ride_id in the ride_sharing DataFrame while setting keep to False.
- Subset ride_sharing on duplicates and sort by ride_id and assign the results to duplicated_rides.
- Print the ride_id, duration and user_birth_year columns of duplicated_rides in that order.

In [ ]:
```
# Find duplicates
duplicates = ____.____(____, ____)

# Sort your duplicated rides
duplicated_rides = ride_sharing[____].____('____')

# Print relevant columns of duplicated_rides
print(duplicated_rides[['____','____','____']])

#_____________________#
#Solutions

# Find duplicates
duplicates = ride_sharing.duplicated(subset = 'ride_id', keep = False)

# Sort your duplicated rides
duplicated_rides = ride_sharing[duplicates].sort_values('ride_id')

# Print relevant columns
print(duplicated_rides[['ride_id','duration','user_birth_year']])
```

Exercise6

Treating duplicates

In the last exercise, you were able to verify that the new update feeding into ride_sharing contains a bug generating both complete and incomplete duplicated rows for some values of the ride_id column, with occasional discrepant values for the user_birth_year and duration columns.

In this exercise, you will be treating those duplicated rows by first dropping complete duplicates, and then merging the incomplete duplicate rows into one while keeping the average duration, and the minimum user_birth_year for each set of incomplete duplicate rows.

Instructions

- Drop complete duplicates in ride_sharing and store the results in ride_dup.
- Create the statistics dictionary which holds minimum aggregation for user_birth_year and mean aggregation for duration.
- Drop incomplete duplicates by grouping by ride_id and applying the aggregation in statistics.
- Find duplicates again and run the assert statement to verify de-duplication.

In [ ]:
```
# Drop complete duplicates from ride_sharing
ride_dup = ____.____()

# Create statistics dictionary for aggregation function
statistics = {'user_birth_year': ____, 'duration': ____}

# Group by ride_id and compute new statistics
ride_unique = ride_dup.____('____').____(____).reset_index()

# Find duplicated values again
duplicates = ride_unique.____(subset = 'ride_id', keep = False)
duplicated_rides = ride_unique[duplicates == True]

# Assert duplicates are processed
assert duplicated_rides.shape[0] == 0

#_____________________#
#Solutions

# Drop complete duplicates from ride_sharing
ride_dup = ride_sharing.drop_duplicates()

# Create statistics dictionary for aggregation function
statistics = {'user_birth_year': 'min', 'duration': 'mean'}

# Group by ride_id and compute new statistics
ride_unique = ride_dup.groupby('ride_id').agg(statistics).reset_index()

# Find duplicated values again
duplicates = ride_unique.duplicated(subset = 'ride_id', keep = False)
duplicated_rides = ride_unique[duplicates == True]

# Assert duplicates are processed
assert duplicated_rides.shape[0] == 0
```