assignment3

November 11, 2024

```
[ ]: version = "REPLACE_PACKAGE_VERSION"
```

1 SIADS 515 Week 3 Homework (HW3)

The notebook for this assignment is a little different. There are bugs and problems in the existing code that you need to fix in order to get the assert statements to pass. There are no hidden grader cells in this assignment. If you run the first cell under each question title, you'll either get errors or it will produce the wrong answer. Fix these to get the assert statements to pass. Since there are no hidden assert statements, there's no real benefit to submitting before you've finished your work.

Think of this assignment in the following way: it's your first day on the job and you've been given a notebook that was authored by someone who is no longer with your company. You've been asked to fix it. There are errors in it, and some of it was not completed by the original author. You're lucky, though, as there are assertions sprinkled throughout the notebook to help guide you along the way.

Top-level goal of notebook: Read a CSV file into a pandas DataFrame and add specific columns to it. These columns are added by applying functions to specific columns. The columns to add include: 1. A datetime column that converts "Garmin time" to standard (unix epoch) time. Note that Garmin doesn't use standard epoch offsets for their timestamps. Rather than using the number of seconds that have elapsed since midnight on January 1, 1970, they use the number of seconds from midnight on December 31, 1989.

2. A conversion of "semicircles" of latitude and longitude to two different formats: degrees, minutes, seconds 3-tuples and fractional degrees. For example, a latitude of 504719750 semicircles corresponds to a 3-tuple of degrees, minutes and seconds of (42, 18, 18.43) and 42.305121 degrees.
3. A "normalized speed" column that consists of the values for speed modified to remove outliers by replacing them with upper and lower bounds as well as normalized to z-values (i.e. by subtracting the mean from each value and dividing the result by the standard deviation).

In addition, you will need to complete a function that looks at the difference between sequential rows to determine whether the cyclist is slowing down or not.

Your task for this assignment is to debug this notebook to produce the desired results as shown in the assertions below.

1.1 Question 1

[81]:
```
import pandas as pd
import numpy as np

# Load the data
ride = pd.read_csv('assets/ride_final2.csv')

# Convert Garmin time to datetime
def garmin_time_to_datetime(series):
    """Convert Garmin FIT time by adding the number of
    seconds from January 1, 1970 to December 31, 1989.
    """
    garmin_epoch = pd.Timestamp('1989-12-31', tz = 'UTC')
    unix_epoch = pd.Timestamp('1970-01-01', tz = 'UTC')
    offset = (garmin_epoch - unix_epoch).total_seconds()

    return pd.to_datetime(series + offset, unit='s', utc=True)
```

[82]:
```
# raise NotImplementedError()
```

[83]:
```
# This is a read-only grader cell

ride['Timestamp_datetime'] = ride.Timestamp.map(garmin_time_to_datetime)

assert ride.Timestamp_datetime[0] == pd.to_datetime('2018-05-23T14:02:25',␣
 ↪utc=True), \
    "First datetime is not correct"
```

1.2 Question 2

[84]:
```
def semicircles_to_degrees(semicircles):
    '''
    Convert semicircles to degrees
    '''
    max_32_bit_int = 2**31
    return semicircles * (180/max_32_bit_int)


def degrees_to_dms(degrees_fraction):
    ''' Convert degrees to degree, minute, second 3-tuples'''
    degrees = int(degrees_fraction)
    minutes_fraction = (degrees_fraction - degrees) * 60
    minutes = int(minutes_fraction)
    seconds = round((minutes_fraction - minutes) * 60, 5)
    return (degrees, abs(minutes), abs(seconds))
```

```
def dms_to_degrees(d,m,s):
    ''' Convert degrees, minutes, seconds to fractional degrees'''
    return d+m/60+s/3600
```

[85]:
```
# Apply the conversions to the ride DataFrame
ride['Latitude_degrees'] = ride['Latitude'].map(semicircles_to_degrees)
ride['Longitude_degrees'] = ride['Longitude'].map(semicircles_to_degrees)
ride['Latitude_dms'] = ride['Latitude_degrees'].map(degrees_to_dms)
ride['Longitude_dms'] = ride['Longitude_degrees'].map(degrees_to_dms)

# Run the provided assertions
dms = degrees_to_dms(42.2833333)
assert dms[0] >= -180, "dms[0] must be greater than or equal to -180"
assert dms[0] <= 180, "dms[0] must be less than or equal to 180"
assert dms[1] >= 0, "dms[1] must be greater than or equal to 0"
assert dms[1] < 60, "dms[1] must be less than 60"
assert dms[2] >= 0, "dms[2] must be greater than or equal to 0"
assert dms[2] < 60, "dms[2] must be less than 60"
assert dms == (42, 16, 59.99988), "dms value is not correct"
assert dms_to_degrees(dms[0], dms[1], dms[2]) == 42.2833333, "dms_to_degrees()␣
 ↪conversion is not correct"

# Debugging: Checking the original Latitude and Longitude values in semicircles␣
 ↪for the last row
last_row = ride.iloc[213]

# Checking the converted Latitude and Longitude in degrees
converted_latitude = semicircles_to_degrees(last_row.Latitude)
converted_longitude = semicircles_to_degrees(last_row.Longitude)


# Validate the expected converted values
assert round(converted_latitude, 6) == 42.280569, \
    "Last row of ride does not have the correct Latitude_degrees value"
assert round(converted_longitude, 6) == -83.739442, \
    "Last row of ride does not have the correct Longitude_degrees value"
```

[86]:
```
# This is a read-only grader cell

dms = degrees_to_dms(42.2833333)
assert dms[0] >= -180, "dms[0] must be greater than or equal to -180"
assert dms[0] <= 180, "dms[0] must be less than or equal to 180"
assert dms[1] >= 0, "dms[1] must be greater than or equal to 0"
assert dms[1] < 60, "dms[1] must be less than 60"
assert dms[2] >= 0, "dms[2] must be greater than or equal to 0"
assert dms[2] < 60, "dms[2] must be less than 60"
```

```
assert dms == (42, 16, 59.99988), "dms value is not correct"
assert dms_to_degrees(dms[0], dms[1], dms[2]) == 42.2833333, "dms_to_degrees()␣
 ↪conversion is not correct"

ride['Latitude_degrees'] = ride['Latitude'].map(semicircles_to_degrees)
ride['Longitude_degrees'] = ride['Longitude'].map(semicircles_to_degrees)
ride['Latitude_dms'] = ride['Latitude_degrees'].map(degrees_to_dms)
ride['Longitude_dms'] = ride['Longitude_degrees'].map(degrees_to_dms)

last_row = ride.iloc[213]
assert round(last_row.Latitude_degrees,6) == 42.280569, \
    "Last row of ride does not have the correct Latitude_degrees value"
assert round(last_row.Longitude_degrees,6) == -83.739442, \
    "Last row of ride does not have the correct Longitude_degrees value"
```

1.3 Question 3

[87]:
```
import pandas as pd
import numpy as np

def normalize(df, pd_series_name, nsd=2):
    '''
    Take all values that are outside some bound (mean +- 2 sd by default)
    and convert them to the appropriate bound.
    Scale the results to z-scores before returning them
    '''
    df = df.copy()
    pd_series = df[pd_series_name].astype(float)

    # Find upper and lower bound for outliers
    avg = np.mean(pd_series)
    sd  = np.std(pd_series)

    # Calculate the bounds
    lower_bound = avg - nsd * sd
    upper_bound = avg + nsd * sd

    # Clip the outliers
    pd_series_clipped = pd_series.clip(lower = lower_bound, upper = upper_bound)

    # Normalize to z-scores
    normalized_series = (pd_series_clipped - avg) / sd
    normalized_series = normalized_series.round(4) # Ensure four decimal places

    return normalized_series
```

[88]:
```
# YOUR CODE HERE
#raise NotImplementedError()
```

[89]:
```
# This is a read-only grader cell

ride['Speed_normalized'] = normalize(ride,'Speed')

assert round(ride.iloc[0].Speed_normalized,4) == -1.7737, \
    "First row of ride does not have the correct value for Speed_normalized"
assert ride.iloc[213].Speed_normalized == -2.0, \
    "Last row of ride does not have the correct value for Speed_normalized"
```

1.4 Question 4

[90]:
```
def proportion_slowing(df,series_name):
    ''' Calculate the proportion of rows that represent a slower speed than the␣
 ↪previous row'''
    speed_diff = df[series_name].diff()
    slowing_down = speed_diff < 0
    proportion = slowing_down.mean()
    return proportion

proportion_slowing_down = proportion_slowing(ride, 'Speed')
```

[91]:
```
# YOUR CODE HERE
# raise NotImplementedError()
```

[92]:
```
# This is a read-only grader cell

assert round(proportion_slowing(ride,'Speed_normalized'),6) == 0.514019, \
    "proportion_slowing() does not return the correct value for the full ride␣
 ↪dataset"
assert round(proportion_slowing(ride[:10],'Speed_normalized'),6) == 0.4, \
    "proportion_slowing() does not return the correct value for the first 10␣
 ↪rows of the ride dataset"
assert round(proportion_slowing(ride[10:],'Speed_normalized'),6) == 0.519608, \
    "proportion_slowing() does not return the correct value for the last rows␣
 ↪of the ride dataset"
```

[ ]:
```

```