Cleaning

November 23, 2023

```
[ ]: # Import required libraries
     import pandas as pd
     import numpy as np

     # Load the original DataFrame from a CSV file
     df_original = pd.read_csv('~/Documents/Typhoons/data_typhoons.txt')

     # Define columns to drop from the original DataFrame
     columns_to_drop = ['Unnamed: 0.2', 'Unnamed: 0.1', 'Unnamed: 0']

     # Drop the specified columns from the original DataFrame
     df_columns_dropped = df_original.drop(columns_to_drop, axis=1)

     # Define the list of columns to be created based on splitting
     columns_to_be_created = [
         'Date and Time',
         'Indicator',
         'Grade',
         'Latitude of the Center',
         'Longitude of the Center',
         'Central Pressure',
         'Maximum sustained wind speed',
         'Direction of the longest radius of 50kt winds or greater',
         'Longest radius of 50kt winds or greater',
         'Shortest radius of 50kt winds or greater',
         'Direction of the longest radius of 30kt winds or greater',
         'Longest radius of 30kt winds or greater',
         'Shortest radius of 30kt winds or greater',
         'Indicator of landfall or passage'
     ]

     # Define columns to split based on spaces
     columns_to_split = df_columns_dropped.columns

     # Split the specified columns based on spaces and concatenate them with the␣
      ↪original DataFrame
     for column in columns_to_split:
```

```
    split_df = df_columns_dropped[column].str.split(expand=True)
    # Rename the new columns with the original column name and a numeric suffix
    split_df.columns = [f'{column}_{i + 1}' for i in range(split_df.shape[1])]
    # Concatenate the split DataFrame with the original DataFrame
    df_columns_dropped = pd.concat([df_columns_dropped, split_df], axis=1)
    # Drop the original column
    df_columns_dropped.drop(column, axis=1, inplace=True)

# Rename the columns to match the desired names
df_columns_dropped.rename(columns=dict(zip(df_columns_dropped.columns,␣
 ↪columns_to_be_created)), inplace=True)

# Convert columns to numeric, ignoring errors
columns_to_numeric = df_columns_dropped.columns
for column in columns_to_numeric:
    df_columns_dropped[column] = pd.to_numeric(df_columns_dropped[column],␣
 ↪errors='ignore')

# Fill missing values in the 'Maximum sustained wind speed' column with '0'
df_columns_dropped['Maximum sustained wind speed'].fillna('0', inplace=True)

# Initialize lists to store storm names and their positions
names = []
international_id = []
position = []
rows = len(df_columns_dropped) - 1
row = 0
while row < rows:
    if df_columns_dropped.iloc[row, 0] == 66666:
        names.append(df_columns_dropped.loc[row, 'Maximum sustained wind␣
 ↪speed'])
        international_id.append(df_columns_dropped.loc[row, 'Indicator'])
        position.append(row)
    row = row + 1

# Initialize indices for calculating column names
lower_index = 0
upper_index = 1
column_names = []
international_id_numbers = []
while upper_index < len(position):
    column_names = column_names + ((position[upper_index] -␣
 ↪position[lower_index]) - 1) * [names[lower_index]]
    international_id_numbers = international_id_numbers +␣
 ↪((position[upper_index] - position[lower_index]) - 1) *␣
 ↪[international_id[lower_index]]
    lower_index = lower_index + 1
    upper_index = upper_index + 1

# Drop rows based on positions and extra rows
df_rows_dropped = df_columns_dropped.drop(position)
extra_rows = list(range(68697, 68742))
df_rows_dropped = df_rows_dropped.drop(extra_rows, errors='ignore')

# Create a list of storm names for the remaining rows
storm_names = 10 * ['0'] + column_names
id_numbers = 10 * ['5101'] + international_id_numbers

# Assign the 'storm_name' column to the DataFrame
df_organized = df_rows_dropped.
 ↪assign(storm_names=storm_names,id_numbers=id_numbers)

# Rename columns
df_organized.rename(columns={'storm_names':'Storm Names','id_numbers':
 ↪'International ID'}, inplace=True)

# Define the order of columns in the final DataFrame
column_order = [
    'International ID',
    'Storm Names',
    'Date and Time',
    'Indicator',
    'Grade',
    'Latitude of the Center',
    'Longitude of the Center',
    'Central Pressure',
    'Maximum sustained wind speed',
    'Direction of the longest radius of 50kt winds or greater',
    'Longest radius of 50kt winds or greater',
    'Shortest radius of 50kt winds or greater',
    'Direction of the longest radius of 30kt winds or greater',
    'Longest radius of 30kt winds or greater'
]

# Reorder columns
df_ordered = df_organized[column_order]

# Replace '0' with NaN
df_ordered.replace({'0': np.nan}, inplace=True)

# Map numerical grade values to their corresponding descriptions
map_grade = {
    2: 'Tropical Depression (TD)',
    3: 'Tropical Storm (TS)',
    4: 'Severe Tropical Storm (STS)',
    5: 'Typhoon (TY)',
    6: 'Extra-tropical Cyclone (L)',
    7: 'Just entering into the responsible area of RSMC Tokyo-Typhoon Center',
    8: 'Not used',
    9: 'Tropical Cyclone of TS intensity or higher',
}

# Replace numerical grade values with their descriptions
df_ordered['Grade'] = df_ordered['Grade'].replace(map_grade)

# Drop the 'Indicator' column
df_categorized = df_ordered.drop(columns='Indicator')

# Define a custom function to parse date strings
def parse_custom_datetime(input_string):
    try:
        # Extract year, month, day, and hour from the input string
        year = input_string[:2]
        month = input_string[2:4]
        day = input_string[4:6]
        hour = input_string[6:8]

        # Format the components into the desired format
        formatted_datetime = f"{year}-{month}-{day} {hour}:00:00"
        return formatted_datetime
    except IndexError:
        print("Error: Input string does not have the expected length.")
        return None

# Parse date strings using the custom function
i = 0
parsed = []
while i < len(df_categorized):
    input_string = str(df_categorized.iloc[i, 2])
    parsed_datetime = parse_custom_datetime(input_string)
    parsed.append(parsed_datetime)
    i = i + 1

# Add the parsed date as a new column
df_date_and_time = df_ordered.assign(Date_Time_Parsed=parsed)

# Drop the original 'Date and Time' column
df_date_and_time = df_date_and_time.drop(columns='Date and Time')

# Define a custom function to adjust two-digit years
def parse_two_digit_year(x):
    try:
        # Split the date components
        parts = x.split('-')

        # Convert the year part to an integer
        year = int(parts[0])

        # Adjust the year to be in the 20th century if necessary
        if year < 50:
            year += 2000
        else:
            year += 1900

        # Reconstruct the date with the adjusted year
        return f"{year}-{parts[1]}-{parts[2]}"
    except IndexError:
        print("Error: Input string does not have the expected length.")
        return None

# Apply the custom function to the 'Date' column
df_date_and_time['Parsed_Date_and_Time'] = df_date_and_time['Date_Time_Parsed'].
 ↪apply(parse_two_digit_year)

# Drop the 'Date_Time_Parsed' column
df_date_time_dropped = df_date_and_time.drop(columns='Date_Time_Parsed')

# Drop the original 'Date' column and rename the new 'Parsed_Date' column
df_date_time_dropped.rename(columns={'Parsed_Date_and_Time': 'Date and Time'},␣
 ↪inplace=True)

# Define the new order of columns
new_column_order = [
    'International ID',
    'Storm Names',
    'Date and Time',
    'Grade',
    'Latitude of the Center',
    'Longitude of the Center',
    'Central Pressure',
    'Maximum sustained wind speed',
    'Direction of the longest radius of 50kt winds or greater',
    'Longest radius of 50kt winds or greater',
    'Shortest radius of 50kt winds or greater',
    'Direction of the longest radius of 30kt winds or greater',
    'Longest radius of 30kt winds or greater'
]
```

```
# Reorder columns
df_new_order = df_date_time_dropped[new_column_order]

# Convert the 'Date' column to datetime format
df_new_order['Date and Time'] = pd.to_datetime(df_new_order['Date and Time'],␣
 ↪errors='coerce')

df_new_order.to_csv('Cleaned.csv')
```

```
/tmp/ipykernel_21439/64201400.py:219: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_new_order['Date and Time'] = pd.to_datetime(df_new_order['Date and Time'],
errors='coerce')
```

[ ]:

```

```