gwpylqjd6

March 12, 2024

[2]:
```
import pandas as pd

df = pd.read_csv('AllTrailsUsaNationalParks.csv')
print(df.head())
print(df.info())
```

```
   trail_id                         name                   area_name  \
0  10020048      Harding Ice Field Trail  Kenai Fjords National Park
1  10236086  Mount Healy Overlook Trail        Denali National Park
2  10267857           Exit Glacier Trail  Kenai Fjords National Park
3  10236076         Horseshoe Lake Trail        Denali National Park
4  10236082           Triple Lakes Trail        Denali National Park

             city_name state_name   country_name  \
0               Seward     Alaska  United States
1  Denali National Park     Alaska  United States
2               Seward     Alaska  United States
3  Denali National Park     Alaska  United States
4  Denali National Park     Alaska  United States

                                  _geoloc  popularity     length  \
0  {'lat': 60.18852, 'lng': -149.63156}     24.8931  15610.598
1  {'lat': 63.73049, 'lng': -148.91968}     18.0311   6920.162
2    {'lat': 60.18879, 'lng': -149.631}     17.7821   2896.812
3    {'lat': 63.73661, 'lng': -148.915}     16.2674   3379.614
4  {'lat': 63.73319, 'lng': -148.89682}     12.5935  29772.790

   elevation_gain  …  num_reviews  \
0       1161.8976  …          423
1        507.7968  …          260
2         81.9912  …          224
3        119.7864  …          237
4       1124.7120  …          110

                                            features  \
0  ['dogs-no', 'forest', 'river', 'views', 'water…
1  ['dogs-no', 'forest', 'views', 'wild-flowers',…
2  ['dogs-no', 'partially-paved', 'views', 'wildl…
```

```
3  ['dogs-no', 'forest', 'lake', 'kids', 'views',…
4  ['dogs-no', 'lake', 'views', 'wild-flowers', '…

                                       activities  units       lat  \
0  ['birding', 'camping', 'hiking', 'nature-trips…      i  60.18852
1  ['birding', 'camping', 'hiking', 'nature-trips…      i  63.73049
2                              ['hiking', 'walking']      i  60.18879
3  ['birding', 'hiking', 'nature-trips', 'trail-r…      i  63.73661
4  ['birding', 'fishing', 'hiking', 'nature-trips…      i  63.73319

         lng summer_temp winter_temp  annual_rain  annual_snow
0 -149.63156      19.600     -14.096      1828.43     1042.097
1 -148.91968      21.996     -30.400       400.09      124.166
2 -149.63100      19.600     -14.096      1828.43     1042.097
3 -148.91500      22.096     -30.300       400.09      124.166
4 -148.89682      22.296     -30.200       400.09      124.166

[5 rows x 24 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3313 entries, 0 to 3312
Data columns (total 24 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   trail_id           3313 non-null   int64
 1   name               3313 non-null   object
 2   area_name          3313 non-null   object
 3   city_name          3313 non-null   object
 4   state_name         3313 non-null   object
 5   country_name       3313 non-null   object
 6   _geoloc            3313 non-null   object
 7   popularity         3313 non-null   float64
 8   length             3313 non-null   float64
 9   elevation_gain     3313 non-null   float64
 10  difficulty_rating  3313 non-null   int64
 11  route_type         3313 non-null   object
 12  visitor_usage      3060 non-null   float64
 13  avg_rating         3313 non-null   float64
 14  num_reviews        3313 non-null   int64
 15  features           3313 non-null   object
 16  activities         3313 non-null   object
 17  units              3313 non-null   object
 18  lat                3313 non-null   float64
 19  lng                3313 non-null   float64
 20  summer_temp        3313 non-null   float64
 21  winter_temp        3313 non-null   float64
 22  annual_rain        3313 non-null   float64
 23  annual_snow        3313 non-null   float64
dtypes: float64(11), int64(3), object(10)
```

```
memory usage: 621.3+ KB
None
```

Here's a summary of the initial observations and the next steps for data cleaning:

1. **Missing Values**: The `visitor_usage` column has missing values that need to be addressed.
2. **Data Types**: At first glance, data types seem appropriate for most columns, but we'll need to inspect certain columns like `_geoloc` which contains JSON-like strings, to see if any adjustments are needed.
3. **Duplicate Rows**: We'll check for and remove any duplicate rows.
4. **Outliers and Anomalies**: We'll need to identify and decide how to handle outliers in numerical columns such as `popularity`, `length`, `elevation_gain`, etc.
5. **Text and Categorical Data Cleaning**: For columns like `name`, `area_name`, `city_name`, `state_name`, etc., we'll ensure consistency in formatting.

[3]:
```python
# Inspecting the distribution of 'visitor_usage' to decide on handling missing values
visitor_usage_median = df['visitor_usage'].median()
visitor_usage_mean = df['visitor_usage'].mean()

visitor_usage_median, visitor_usage_mean
```

[3]:
```
(2.0, 1.877124183006536)
```

[4]:
```python
# Fill missing values in 'visitor_usage' with the median
df['visitor_usage'].fillna(visitor_usage_median, inplace=True)

# Drop duplicate rows, if any
initial_row_count = len(df)
df.drop_duplicates(inplace=True)
duplicates_removed = initial_row_count - len(df)

# Convert '_geoloc' to separate lat and lng columns if needed (inspect first)
geoloc_example = df['_geoloc'].iloc[0]

# Report back the number of duplicates removed and inspect the _geoloc column format
duplicates_removed, geoloc_example
```

[4]:
```
(0, "{'lat': 60.18852, 'lng': -149.63156}")
```

The _geoloc column contains JSON-like strings specifying latitude (lat) and longitude (lng) values. Since the latitude (lat) and longitude (lng) are already provided as separate columns, there's no need to extract data from the _geoloc column, but we might consider removing it to clean up the dataset and avoid redundancy.

Next, let's inspect numerical columns for outliers. We'll focus on key numerical columns such as popularity, length, elevation_gain, and difficulty_rating. Identifying outliers typically involves calculating the interquartile range (IQR) and then determining which values fall outside of this range. However, due to the diverse nature of these metrics (e.g., trail lengths can vary significantly by design), we'll need to be careful with how we define and handle outliers.

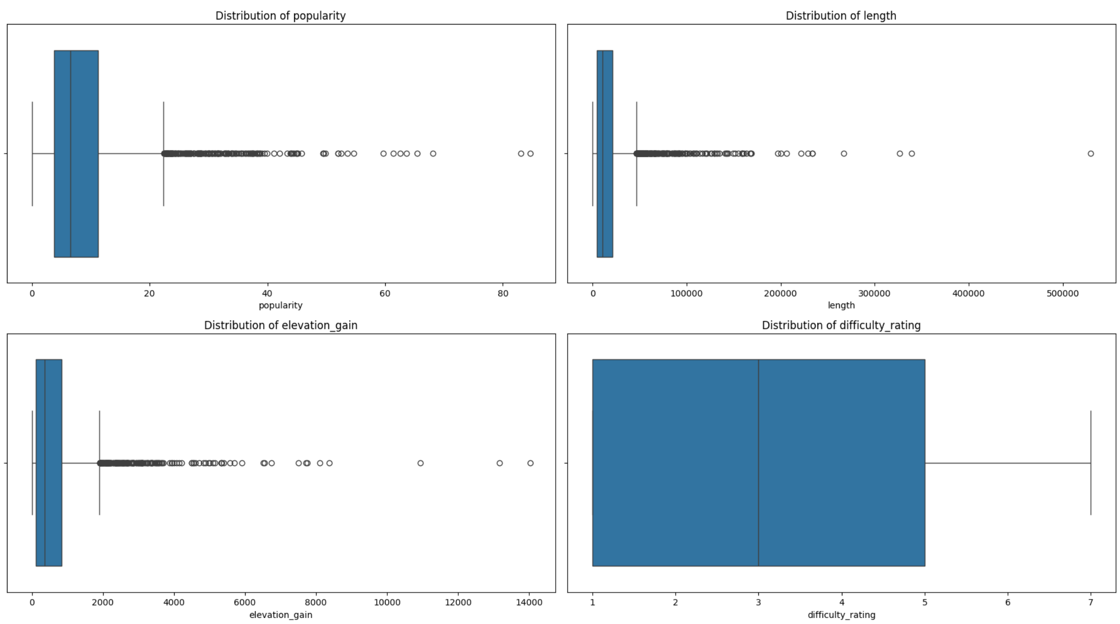
[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Setting up the matplotlib figure
plt.figure(figsize=(18, 10))

# Selecting columns to check for outliers
columns_to_plot = ['popularity', 'length', 'elevation_gain',␣
↪'difficulty_rating']

for i, column in enumerate(columns_to_plot, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[column])
    plt.title(f'Distribution of {column}')

plt.tight_layout()
plt.show()
```



The boxplot visualizations provide insights into the distribution of the `popularity`, `length`, `elevation_gain`, and `difficulty_rating` columns:

- **Popularity**: Most values are concentrated in a lower range, but there are several outliers indicating trails with significantly higher popularity scores.
- **Length**: The `length` of trails varies widely, with many outliers indicating exceptionally long trails. This is expected given the nature of hiking trails, which can range from short walks to

extensive multi-day hikes.

- **Elevation Gain**: Similar to `length`, `elevation_gain` shows a wide range of values with outliers indicating trails with significant elevation changes.
- **Difficulty Rating**: This appears to be a discrete variable with a more limited range of values. There are no visible outliers in this case, which makes sense as this is likely a categorical rating system.

Given the nature of the data, outright removal of outliers may not be appropriate for columns like `length` and `elevation_gain`, as these outliers could represent valid, albeit extreme, data points (e.g., exceptionally long or challenging trails). For `popularity`, careful consideration is also needed, as high popularity scores might be of interest for analysis.

Instead of removing outliers, a more nuanced approach might involve:

- Ensuring that outliers do not result from data entry errors.
- Considering whether to analyze certain outliers separately or to apply transformations for certain analyses.

Next steps in the cleaning process might focus on ensuring consistency in categorical and textual data, and considering whether any transformations or adjustments are necessary for the numerical data for specific analyses, rather than removing outliers wholesale.

Let's proceed to ensure text and categorical columns are cleaned for consistency, such as trimming whitespace and converting text to a uniform case where applicable. Since we've addressed missing values, duplicates, and assessed outliers, this step will help in preparing the dataset for analysis by ensuring consistent categorical data.

[6]:
```python
# Clean text and categorical data: trimming whitespace and converting to␣
↪lowercase for consistency
text_columns = ['name', 'area_name', 'city_name', 'state_name', 'country_name',␣
↪'route_type']

for col in text_columns:
    df[col] = df[col].str.strip().str.lower()

# Check if the cleaning was successful for one example column
df['area_name'].head()
```

[6]:
```
0    kenai fjords national park
1          denali national park
2    kenai fjords national park
3          denali national park
4          denali national park
Name: area_name, dtype: object
```

[7]:
```python
# One-hot encoding of 'route_type' column
df_encoded = pd.get_dummies(df, columns=['route_type'])

# Standardize/Normalize numerical columns example (using 'length' as an example)
from sklearn.preprocessing import StandardScaler
```

```python
# Create a StandardScaler object
scaler = StandardScaler()

# Fit and transform the 'length' column
# Note: For demonstration, we'll just transform this column without applying it
 ↪back to avoid altering the original data's scale
length_scaled = scaler.fit_transform(df[['length']])

# Check the encoding result and example scaled data
df_encoded.head(), length_scaled[:5]
```

[7]:
```
(   trail_id                        name                 area_name  \
 0  10020048     harding ice field trail  kenai fjords national park
 1  10236086  mount healy overlook trail       denali national park
 2  10267857          exit glacier trail  kenai fjords national park
 3  10236076        horseshoe lake trail       denali national park
 4  10236082          triple lakes trail       denali national park

                city_name state_name   country_name  \
 0                 seward     alaska  united states
 1  denali national park     alaska  united states
 2                 seward     alaska  united states
 3  denali national park     alaska  united states
 4  denali national park     alaska  united states

                                   _geoloc  popularity     length  \
 0  {'lat': 60.18852, 'lng': -149.63156}     24.8931  15610.598
 1  {'lat': 63.73049, 'lng': -148.91968}     18.0311   6920.162
 2    {'lat': 60.18879, 'lng': -149.631}     17.7821   2896.812
 3    {'lat': 63.73661, 'lng': -148.915}     16.2674   3379.614
 4  {'lat': 63.73319, 'lng': -148.89682}     12.5935  29772.790

    elevation_gain  …  units       lat        lng  summer_temp winter_temp  \
 0       1161.8976  …      i  60.18852 -149.63156       19.600     -14.096
 1        507.7968  …      i  63.73049 -148.91968       21.996     -30.400
 2         81.9912  …      i  60.18879 -149.63100       19.600     -14.096
 3        119.7864  …      i  63.73661 -148.91500       22.096     -30.300
 4       1124.7120  …      i  63.73319 -148.89682       22.296     -30.200

   annual_rain annual_snow  route_type_loop  route_type_out and back  \
 0     1828.43    1042.097                0                        1
 1      400.09     124.166                0                        1
 2     1828.43    1042.097                0                        1
 3      400.09     124.166                1                        0
 4      400.09     124.166                0                        1
```

```
   route_type_point to point
0                          0
1                          0
2                          0
3                          0
4                          0

[5 rows x 26 columns],
array([[-0.08105001],
       [-0.42193794],
       [-0.57975642],
       [-0.5608182 ],
       [ 0.47447106]]))
```

Let's proceed with encoding the `route_type` column using one-hot encoding as an example of how to prepare categorical data for analysis or machine learning. This will transform `route_type` into multiple binary columns, each representing a possible category. Additionally, we'll briefly discuss the standardization of numerical data, which is key for certain analyses and machine learning models.

The `route_type` column has been successfully encoded into multiple binary columns, each representing a category of route type (i.e., `loop`, `out and back`, `point to point`). This encoding transforms the categorical data into a format suitable for analysis or machine learning models that require numerical input.

Additionally, the `length` column has been standardized as an example of normalizing numerical data. The output shows the first five scaled values of the `length` column, which have been adjusted to have a mean of 0 and a standard deviation of 1. This process is crucial for certain machine learning algorithms that are sensitive to the scale of the input features.

[8]:
```
# 'features' column is a string representation of a list, we'll parse it and␣
↪create indicator columns for a few example features
import ast  # To safely evaluate strings containing Python literals

# A simplified function to create indicator variables for example features
def extract_features(row, feature_list):
    # Safely evaluate the string to a list
    features = ast.literal_eval(row)
    return [feature in features for feature in feature_list]

# Example features to extract
example_features = ['dogs-no', 'forest', 'river']

# Apply the function to the 'features' column
features_df = pd.DataFrame(df['features'].apply(lambda x: extract_features(x,␣
↪example_features)).tolist(), columns=example_features)

# Concatenate the new features to the original dataframe
```

```
df_cleaned = pd.concat([df, features_df], axis=1)

# Show the result for the new columns
df_cleaned.head()[['features'] + example_features]
```

[8]:

	features	dogs-no	forest	river
0	['dogs-no', 'forest', 'river', 'views', 'water…	True	True	True
1	['dogs-no', 'forest', 'views', 'wild-flowers',…	True	True	False
2	['dogs-no', 'partially-paved', 'views', 'wildl…	True	False	False
3	['dogs-no', 'forest', 'lake', 'kids', 'views',…	True	True	False
4	['dogs-no', 'lake', 'views', 'wild-flowers', '…	True	False	False

The process successfully parsed the `features` column, which contains lists of features as strings, and created binary indicators for a few example features: `dogs-no`, `forest`, and `river`. This resulted in new columns where each row indicates whether a particular feature is present (`True`) or not (`False`) for each trail.

This transformation enhances the dataset by making it easier to analyze the presence or absence of specific features across trails. For instance, you can now easily filter trails that do not allow dogs, are within forests, or are alongside rivers.

[10]:
```
# Replace 'new_file.csv' with the path where you want to save the new file
df_cleaned.to_csv('Final_Alltrails.csv', index=False)
```

[12]:
```
from google.colab import files
files.download('Final_Alltrails.csv')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[ ]: