Data Cleaning Missing or Null Values

How to deal with missing or null values is a pretty complex topic in data analysis, statistics, and programming. We're going to just go into some of the basics here and introduce a few tools Pandas gives us to address missing data.

When we say a value is missing or is null, we do not mean that the value is an empty string, such as `' '`, or a zero value, such as `0` or `0.0`. Missing or null means there is no value present at all. For the survey data we are looking at now, it means the person taking the survey provided no answer. When we are representing our data with a series or a dataframe, we can't simply skip missing values, they must be encoded in some way. A series with a missing value at the fifth index can't just be one value shorter. It throws off everything. Instead, we need to plug in a special value that represents a missing value so that when we perform operations on the series, nothing goes awry. For example, if we ask for the mean value of a series of values that contains some missing values, we would not want to sum all the values and then divide them by the length of the series. The length would include the missing values and throw off the mean value.

Unfortunately, depending on what packages you are using with Python, the exact way a missing value is described or represented varies. Missing values are also called null values. Python or Pandas might represent or refer to missing values with: * `None`, * `np.NaN` for missing floats (NaN is an abbreviation for `not a number`) * `pd.NaT` for missing time data

The Pandas methods we are going to use refer to missing values as 'na', which is short for 'not available'. These methods will typically detect multiple types of missing data.

Finding Missing Values

The method `.isna()` can be used with either a dataframe or individual series. It returns a boolean object where all missing values are mapped to True. All other values are mapped to False.

```
# using .isna() with a dataframe
df.isna().head(10)
```

	Timestamp	musicartist	height	city	30min	travel	likepizza	deepdish	sport	spell	hangout	talk	year	quote
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False

	Timestamp	musicartist	height	city	30min	travel	likepizza	deepdish	sport	spell	hangout	talk	year	quote
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	True	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	True	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False	False	False

```
# using .isna() with a series
df.loc[:, '30min'].isna().head(10)
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8     True
9    False
Name: 30min, dtype: bool
```

If you want to count values that are not missing, you can use `.notna()`.

```
# using .notna() with a series
df.loc[:, '30min'].notna().head(10)
0     True
1     True
2     True
3     True
4     True
5     True
6     True
7     True
8    False
9     True
Name: 30min, dtype: bool
```

Both `.sum()` and `.count()` can be used with a series of booleans. * `.sum()` will treat each `True` as a `1` and add them all together. * `.count()` will only count non-missing values (if you want to count missing and non-missing values, use `len()` instead).

```
column_name = '30min'

# counts missing values by summing booleans (True = 1, False = 0)
missing_count = df.loc[:, column_name].isna().sum()

# counts not missing values using .count()
not_missing_count1 = df.loc[:, column_name].count()

# counts not missing values using .notna().sum() (True = 1, False = 0)
# same result as previous
not_missing_count2 = df.loc[:, column_name].notna().sum()

# count rows in series (including missing)
total_count = len(df.loc[:, column_name])

# print summary
print(f'{column_name}\n-missing values: {missing_count}' \
      f'\n-not missing values: {not_missing_count1}' \
      f'\n-total count: {total_count}')
30min
-missing values: 15
-not missing values: 189
-total count: 204
```

Dropping Missing Values

One way to deal with missing values in datasets is to remove any entry that has a missing value. This approach makes cleaning your data easier, you simply remove problematic entries, but it can also negatively affect the quality of your analysis. If the missing data is not missing at random, then your analysis may produce results that are biased in one way or another.

So please remember, while dropping entries with missing values is common, and it is certainly easy, it can also create issues that can be subtle and misleading.

If you decide you do want to remove entries with missing data, Pandas makes it very easy. Removing data from a dataframe is sometimes referred to as 'dropping' data, so Pandas has a specific method called `.dropna()` that can be used to remove either rows or columns with missing values.

```
# dropna() applied to the entire dataframe, drops rows
```

```
df_all_remove_rows = df.dropna(axis='index') # removes rows with missing values

# check the result
df_all_remove_rows.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 174 entries, 0 to 202
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Timestamp    174 non-null    object
 1   musicartist  174 non-null    object
 2   height       174 non-null    object
 3   city         174 non-null    object
 4   30min        174 non-null    object
 5   travel       174 non-null    object
 6   likepizza    174 non-null    float64
 7   deepdish     174 non-null    object
 8   sport        174 non-null    object
 9   spell        174 non-null    object
 10  hangout      174 non-null    object
 11  talk         174 non-null    object
 12  year         174 non-null    object
 13  quote        174 non-null    object
dtypes: float64(1), object(13)
memory usage: 20.4+ KB
# dropna() applied to the entire dataframe, drops columns with missing values
df_all_removed = df.dropna(axis='columns') # removes columns with missing values

# check the result
df_all_removed.info() # only a single column has no missing values
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Timestamp  204 non-null    object
dtypes: object(1)
memory usage: 1.7+ KB
# dropna applied to a single series
series_30removed = df.loc[:, '30min'].dropna()

# check the result
print(f'length before drop: {len(df.loc[:, "30min"])}')
print(f'length after drop: {len(series_30removed)}')
 length before drop: 204
length after drop: 189
```

Replacing Missing Values

Another approach to dealing with missing values is to replace them.
The `.fillna()` method will replace any missing values with a specified replacement value. In this example, we replace any missing values with the mean of the series.

```
# calculate the mean replacement value by first dropping missing values
mean_replacement_value = df.loc[:, 'likepizza'].dropna().mean()

# use the calculated mean to replace the missing values (assigned to a new
series)
df.loc[:, 'likepizza_w_replace'] = df.loc[:,
'likepizza'].fillna(mean_replacement_value)

# check the result
df.loc[:, ['likepizza', 'likepizza_w_replace']].info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 2 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   likepizza           203 non-null    float64
 1   likepizza_w_replace  204 non-null    float64
dtypes: float64(2)
memory usage: 3.3 KB
```