Data_cleaning_New

February 22, 2024

Ricardo Kaleb Flores Alfonso

1 Data Cleaning: Handling missing Values

[ ]:
```
# We're going to be looking at how to deal with missing values :D
# To get started, Download the required Dataset which I'm going to use in this
 ↪tutorial
```

[ ]:
```
# Import the libs we will use
import pandas as pd
import numpy as np
```

[ ]:
```
# read in all our data

# Detailed NFL Play-by-Play Data 2009-2017
nfl_data = pd.read_csv("NFL Play by Play 2009-2017.csv")

nfl_data.head(5)

# With low_memory=True (default behavior):
# - Pandas will read the file in chunks, analyzing each chunk to determine the
 ↪best data types for each column.
#   This can be memory-efficient for large files because it doesn't load the
 ↪entire file into memory at once.
# - However, because it reads the data in chunks, Pandas may not accurately
 ↪infer the data types for each column,
#   especially if the data types change within the file or if there's missing
 ↪data.
# - This approach is suitable for conserving memory but may result in slower
 ↪performance and potential errors in data type inference.

# With low_memory=False:
# - Pandas will read the entire file into memory at once, allowing it to
 ↪analyze the entire dataset before inferring data types.
# - This can be faster and more accurate for inferring data types, especially
 ↪with mixed data types or complex datasets.
```

```
# - However, it requires more memory, so it's only suitable for files that can
↪comfortably fit into memory without causing memory errors.
```

```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_2460\2845440133.py:4: DtypeWarning: Columns
(25,51) have mixed types. Specify dtype option on import or set
low_memory=False.
  nfl_data = pd.read_csv("NFL Play by Play 2009-2017.csv")
```

[ ]:

```
         Date      GameID  Drive  qtr  down   time  TimeUnder  TimeSecs  \
0  2009-09-10  2009091000      1    1   NaN  15:00         15    3600.0
1  2009-09-10  2009091000      1    1   1.0  14:53         15    3593.0
2  2009-09-10  2009091000      1    1   2.0  14:16         15    3556.0
3  2009-09-10  2009091000      1    1   3.0  13:35         14    3515.0
4  2009-09-10  2009091000      1    1   4.0  13:27         14    3507.0

   PlayTimeDiff SideofField  …    yacEPA  Home_WP_pre  Away_WP_pre  \
0           0.0         TEN  …       NaN     0.485675     0.514325
1           7.0         PIT  …  1.146076     0.546433     0.453567
2          37.0         PIT  …       NaN     0.551088     0.448912
3          41.0         PIT  … -5.031425     0.510793     0.489207
4           8.0         PIT  …       NaN     0.461217     0.538783

   Home_WP_post  Away_WP_post  Win_Prob       WPA    airWPA    yacWPA  Season
0      0.546433      0.453567  0.485675  0.060758       NaN       NaN    2009
1      0.551088      0.448912  0.546433  0.004655 -0.032244  0.036899    2009
2      0.510793      0.489207  0.551088 -0.040295       NaN       NaN    2009
3      0.461217      0.538783  0.510793 -0.049576  0.106663 -0.156239    2009
4      0.558929      0.441071  0.461217  0.097712       NaN       NaN    2009

[5 rows x 102 columns]
```

[ ]:

```
# getting a sample
sample= nfl_data.sample(4)

sample
# The .sample() method in Pandas is used to get a random sample of rows from a
↪DataFrame.
# In your code, nfl_data.sample(4) will return a DataFrame containing 4
↪randomly selected rows from the nfl_data DataFrame.
# This method is often used for exploratory data analysis or to quickly inspect
↪a small subset of the data
# without having to go through the entire dataset. It's useful for
↪understanding the structure of the data,
# identifying potential patterns or anomalies, and making preliminary
↪assessments.
```

[ ]:
```
                Date      GameID  Drive  qtr  down   time  TimeUnder  TimeSecs  \
148748  2012-10-14  2012101403     16    3   1.0  13:09         14    1689.0
406759  2017-12-31  2017123110      1    1   1.0  14:11         15    3551.0
12702   2009-10-11  2009101109      4    1   1.0  05:01          6    3001.0
387589  2017-11-12  2017111208     13    2   1.0  00:14          1    1814.0

        PlayTimeDiff SideofField  …    yacEPA  Home_WP_pre  Away_WP_pre  \
148748           8.0         CLE  …       NaN     0.238102     0.761898
406759          26.0          TB  …       NaN     0.546739     0.453261
12702           36.0         ARI  …  1.066975     0.789929     0.210071
387589           0.0         HOU  …       NaN     0.558718     0.441282

        Home_WP_post  Away_WP_post  Win_Prob       WPA    airWPA    yacWPA  \
148748      0.223664      0.776336  0.238102 -0.014439       NaN       NaN
406759      0.577553      0.422447  0.546739  0.030814       NaN       NaN
12702       0.812573      0.187427  0.789929  0.022643 -0.002511  0.025154
387589      0.582077      0.417923  0.441282 -0.023359       NaN       NaN

        Season
148748    2012
406759    2017
12702     2009
387589    2017

[4 rows x 102 columns]
```

[ ]:
```python
# This code calculates the number of missing data points (NaN or null values)␣
 ↪per column in a DataFrame.


# get the number of missing data point per column

# It first uses the .isnull() method to create a DataFrame of the same shape as␣
 ↪nfl_data,
# where each cell is True if the corresponding cell in nfl_data is null and␣
 ↪False otherwise.
# Then, the .sum() method is applied to sum up these boolean values for each␣
 ↪column,
# effectively counting the number of True values (missing/null values) in each␣
 ↪column.
# The result is stored in the missing_values_count Series, where the index␣
 ↪represents the column names,
# and the values represent the number of missing values in each column.

missing_values_count = nfl_data.isnull().sum()

# look at the number of missing points in the first ten columns
```

```
missing_values_count[0:10]




# The next line slices the missing_values_count Series to only include the␣
↪first ten rows.
# This is achieved using the [0:10] slicing operation, which selects the first␣
↪ten elements of the Series based on their index (column names).
# The result is a new Series containing the counts of missing values for the␣
↪first ten columns of the DataFrame.
```

[ ]:
```
Date                0
GameID              0
Drive               0
qtr                 0
down            61154
time              224
TimeUnder           0
TimeSecs          224
PlayTimeDiff      444
SideofField       528
dtype: int64
```

[ ]:
```
# That seems like a lot! It might be helpful to see what percentage of the␣
↪values in our dataset
# were missing to give us a better sense of the scale of this problem

# how many total missing values do we have?

# Calculate the total number of cells in the 'nfl_data' dataset
# by taking the product of its shape which represents the number of rows and␣
↪columns
total_cells = np.product(nfl_data.shape)

# Calculate the total number of missing values in the 'nfl_data' dataset
# by summing up all the missing values count across different columns
total_missing = missing_values_count.sum()
```

[ ]:
```
# percent of data that is missing
(total_missing/total_cells) * 100
```

[ ]:
```
24.87214126835169
```

[ ]:
```
# Almost a quarter of the cells in this dataset are empty!
```

1.1 Resumen

En esta sección se importaron las librerias necesarias, se importó el dataset y se realizó un pequeño analisis de los datos que se tienen. Esto se consiguio a traves de una muestra aleatoria y los primeros 5 datos, se observaron datos faltantes, por lo que se usó el metodo .isnull().sum() para obtener cuantos datos faltaban por columna. Se observó una alta cantidad de datos faltantes en las primeras 10 columnas, asi que se sumaron todos los valores nulos para descubrir cual es la cantidad de datos que faltan del total. Esta cantidad fue de 24.8% lo cual es una alta cantidad de datos.

2 Figure out why the data is missing

Is this value missing because it was not recorded or because it does not exist?

If a value is missing because it does not exist Example: the height of the oldest child of someone who doesn't have any children PenalizedTeam: falta el campo porque si no hubo penalización It doesn't make sense to try and guess what it might be

If a value is missing because it was not recorded Example: temperature a certain hour TimeSecs: cantidad de segundos que quedan en el juego cuando se realizó la jugada. You can try to guess what it might have been based on the other values in that column and row

If you're doing very careful data analysis, this is the point at which you'd look at each column individually to figure out the best strategy for filling those missing values.

3 Drop missing values

[ ]:
```python
# Create a DataFrame 'df' with the provided data:
# - The first row contains values 1, NaN (missing value), and 2
# - The second row contains values 2, 3, and 5
# - The third row contains values 2, 3, and 5 (same as the second row)
# - The fourth row contains NaN (missing value), 4, and 6
# The DataFrame 'df' is then displayed, showing the structure and values.

df = pd.DataFrame([[1,      np.nan, 2],
                   [2,      3,      5],
                    [2,      3,      5],
                   [np.nan, 4,      6]])

df
```

[ ]:

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	2.0	3.0	5
3	NaN	4.0	6

[ ]:
```python
# This will tell us the total number of non null observations present including
↪the total number of entries.
```

```
# Once number of entries isn't equal to number of non null observations, we can␣
↪begin to suspect missing values.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       3 non-null      float64
 1   1       3 non-null      float64
 2   2       4 non-null      int64
dtypes: float64(2), int64(1)
memory usage: 224.0 bytes
```

[ ]:
```
# This will tell us the total number of NaN in or data.
df.isnull().sum()
```

[ ]:
```
0    1
1    1
2    0
dtype: int64
```

[ ]:
```
# remove all the rows that contain a missing value
df.dropna()
```

[ ]:

	0	1	2
1	2.0	3.0	5
2	2.0	3.0	5

[ ]:
```
# remove columns the rows that contain a missing value
# axis{0 or 'index', 1 or 'columns'}
df.dropna(axis=1)



# Remove columns from the DataFrame 'df' that contain at least one missing␣
↪value (NaN)
# The parameter 'axis=1' specifies that we are operating along columns.
# In pandas, axis=0 refers to rows, and axis=1 refers to columns.
# Therefore, 'axis=1' indicates that we want to drop columns.
# The dropna() function is used to perform this operation.
```

[ ]:

	2
0	2
1	5
2	5
3	6

[ ]:
```
df.dropna(axis='columns')
```

[ ]:

	2
0	2
1	5
2	5
3	6

[ ]:
```
# remove all the rows with at least
# thresh          Require that many non-NA values.
# Remove rows from the DataFrame 'df' that contain less than 3 non-null values.
# The parameter 'axis='rows'' specifies that we are operating along rows.
# Here, 'rows' is used to indicate the same as axis=0, where axis=0 refers to␣
 ↪rows.
# The parameter 'thresh=3' specifies the threshold for non-null values required␣
 ↪to keep the row.
# Rows with at least 3 non-null values are retained, while rows with fewer than␣
 ↪3 non-null values are dropped.
# The dropna() function is used to perform this operation.
df.dropna(axis='rows', thresh=3)
```

[ ]:

	0	1	2
1	2.0	3.0	5
2	2.0	3.0	5

3.1 Resumen

En esta sección se hizo el mismo analisis que en la sección anterior para descubrir cuantos valores faltantes hay. En este caso se usó el metodo .dropna() para eliminar los valores faltantes. Se observó que se pueden eliminar las columnas que tengan al menos un valor faltante o las filas. El metodo tiene como predeterminado eliminar las filas.

4 Filling missing values

[ ]:
```
df
```

[ ]:

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	2.0	3.0	5
3	NaN	4.0	6

[ ]:
```
# One option we have is to specify what we want the NaN values to be replaced␣
 ↪with.
# Here, I'm saying that I would like to replace all the NaN values with 0.
df.fillna(0)
```

```
[ ]:      0    1  2
     0  1.0  0.0  2
     1  2.0  3.0  5
     2  2.0  3.0  5
     3  0.0  4.0  6
```

```
[ ]: # calculate the mean
     # skip the Na values while finding the mean

     # Calculate the mean (average) of each column in the DataFrame 'df'.
     # The parameter 'axis=0' specifies that the calculation is performed along the␣
      ↪columns.
     # Here, axis=0 refers to columns, indicating that the mean is calculated for␣
      ↪each column.
     # The parameter 'skipna=True' indicates that any missing values (NaN) should be␣
      ↪skipped during the calculation.
     # If skipna=True, NaN values are excluded from the calculation and do not␣
      ↪contribute to the mean.
     # The mean values are returned as a Series with the column names as index␣
      ↪labels.

     df.mean(axis = 0, skipna = True)
```

```
[ ]: 0    1.666667
     1    3.333333
     2    4.500000
     dtype: float64
```

```
[ ]: # use the mean to fill the gaps in that column

     # Fill missing values in column 1 of the DataFrame 'df' with the value 3.3.
     # df[1] selects the column with index 1 (second column) from the DataFrame.
     # The fillna() function is then used to replace any missing values (NaN) in␣
      ↪that column with the value 3.3.
     # This operation modifies the original DataFrame 'df' in place.

     df[1]=df[1].fillna(3.3)
     df
```

```
[ ]:      0    1  2
     0  1.0  3.3  2
     1  2.0  3.0  5
     2  2.0  3.0  5
     3  NaN  4.0  6
```

```
[ ]: # We can specify a backfill use next valid observation to fill the actual gap.
```

```python
# Fill missing values in the DataFrame 'df' using backward fill (bfill) method.
# The method parameter is set to 'bfill', which means missing values are␣
↪replaced with the next valid observation
# along the specified axis. When applied to a DataFrame, missing values are␣
↪filled with the value from the next row
# within the same column. This effectively propagates the last valid␣
↪observation backward along the specified axis.

df.fillna(method='bfill')
```

[ ]:

	0	1	2
0	1.0	3.3	2
1	2.0	3.0	5
2	2.0	3.0	5
3	NaN	4.0	6

[ ]:

```python
# We can specify a forward-fill use previous valid observation to fill the␣
↪actual gap
df.fillna(method='ffill')
```

[ ]:

	0	1	2
0	1.0	3.3	2
1	2.0	3.0	5
2	2.0	3.0	5
3	2.0	4.0	6

4.1 Resumen

En esta sección se usó .fillna() para rellenar los datos faltantes, es posible cambiar los datos por un valor especifico arbitrario, o calcular el promedio y rellenar los datos faltantes con el promedio. De igual manera se puede usar el metodo .ffill() para rellenar los datos faltantes con el valor anterior, o .bfill() para rellenar los datos faltantes con el valor siguiente.