Raúl Correa Ocañas A01722401 29 - 02 - 2024

Data Cleaning: Handling missing Values

In [ ]:
```
# 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 tuto
```

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

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

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

# Detailed NFL Play-by-Play Data 2009-2017
nfl_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/2024 [TC2004B.101]/N

nfl_data.head(5)

# With low_memory=True (default behavior):
# - Pandas will read the file in chunks, analyzing each chunk to determine the best
#   This can be memory-efficient for large files because it doesn't load the entire
# - However, because it reads the data in chunks, Pandas may not accurately infer t
#   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 perfor

# With low_memory=False:
# - Pandas will read the entire file into memory at once, allowing it to analyze th
# - This can be faster and more accurate for inferring data types, especially with
# - However, it requires more memory, so it's only suitable for files that can comf
```

Out[ ]:

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	Sideo
0	2009-09-10	2009091000	1	1	NaN	15:00	15	3600.0	0.0	
1	2009-09-10	2009091000	1	1	1.0	14:53	15	3593.0	7.0	
2	2009-09-10	2009091000	1	1	2.0	14:16	15	3556.0	37.0	
3	2009-09-10	2009091000	1	1	3.0	13:35	14	3515.0	41.0	
4	2009-09-10	2009091000	1	1	4.0	13:27	14	3507.0	8.0	

5 rows × 102 columns

In [ ]:
```
# 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 Data
# In your code, nfl_data.sample(4) will return a DataFrame containing 4 randomly se
# This method is often used for exploratory data analysis or to quickly inspect a s
# without having to go through the entire dataset. It's useful for understanding th
# identifying potential patterns or anomalies, and making preliminary assessments.
```

Out[ ]:

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff
2094	2009-09-13	2009091310	3	1	1.0	01:24	2	2784.0	0.0
44295	2010-01-03	2010010315	5	1	1.0	03:43	4	2923.0	12.0
340458	2016-11-06	2016110600	13	2	1.0	00:22	1	1822.0	5.0
239454	2014-10-12	2014101208	6	2	2.0	10:31	11	2431.0	4.0

4 rows × 102 columns

In [ ]:
```
# This code calculates the number of missing data points (NaN or null values) per c


# 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
# where each cell is True if the corresponding cell in nfl_data is null and False o
```

```
# 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 colu
# The result is stored in the missing_values_count Series, where the index represen
# 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 te
# This is achieved using the [0:10] slicing operation, which selects the first ten
# The result is a new Series containing the counts of missing values for the first
```

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

In [ ]:
```
# That seems like a lot! It might be helpful to see what percentage of the values i
# 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 column
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()
```

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

Out[ ]:
```
24.87214126835169
```

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

- Se importan las bibliotecas necesarias: pandas para el manejo de datos, numpy para operaciones numéricas y google.colab.drive para montar Google Drive y acceder a los archivos.
- Se monta Google Drive para acceder al conjunto de datos.

- Se lee el conjunto de datos de juego por juego de la NFL de un archivo CSV.
- Se muestra una muestra de cuatro filas del conjunto de datos.
- Se calcula el número de valores faltantes por columna en el conjunto de datos y se muestra el recuento de valores faltantes para las primeras diez columnas.
- Se calcula el porcentaje de datos que faltan en todo el conjunto de datos

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.

- Si un valor está ausente porque no existe en realidad, se ilustra con ejemplos como la altura del hijo mayor de alguien que no tiene hijos o el equipo penalizado cuando no hubo una penalización. En estos casos, no tiene sentido intentar adivinar qué valor podría haber sido.
- Si un valor está ausente porque no se registró, se da el ejemplo de la temperatura en cierta hora o la cantidad de segundos restantes en el juego al momento de la jugada. En tales situaciones, se puede intentar adivinar el valor faltante basándose en los otros valores de esa columna y fila.
- Se señala que, al realizar un análisis de datos cuidadoso, es importante examinar cada columna individualmente para determinar la mejor estrategia para llenar los valores faltantes.

Drop missing values

In [ ]:
```
# 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
```

Out[ ]:

	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

In [ ]:
```
# This will tell us the total number of non null observations present including the
# Once number of entries isn't equal to number of non null observations, we can beg

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
```

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

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

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

Out[ ]:

	0	1	2
1	2.0	3.0	5
2	2.0	3.0	5

In [ ]:
```
# 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 (N
# 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.
```

Out[ ]:

	2
0	2
1	5
2	5
3	6

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

Out[ ]:

	2
0	2
1	5
2	5
3	6

In [ ]:
```
# 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
# Rows with at least 3 non-null values are retained, while rows with fewer than 3 n
# The dropna() function is used to perform this operation.
df.dropna(axis='rows', thresh=3)
```

Out[ ]:

	0	1	2
1	2.0	3.0	5
2	2.0	3.0	5

1. Se crea un DataFrame llamado `df` con valores proporcionados y muestra el DataFrame.

2. Utiliza métodos como `info()`, `isnull().sum()`, `dropna()`, y `dropna(axis='rows', thresh=3)` para tratar y eliminar los valores faltantes en el DataFrame `df`.

Filling missing values

In [ ]:
```
df
```

Out[ ]:

	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

In [ ]:
```
# 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)
```

Out[ ]:

	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

In [ ]:
```
# 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 colu
# Here, axis=0 refers to columns, indicating that the mean is calculated for each c
# The parameter 'skipna=True' indicates that any missing values (NaN) should be ski
# If skipna=True, NaN values are excluded from the calculation and do not contribut
# The mean values are returned as a Series with the column names as index labels.

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

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

In [ ]:
```
# 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 co
# This operation modifies the original DataFrame 'df' in place.

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

Out[ ]:

	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

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

# 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 w
# along the specified axis. When applied to a DataFrame, missing values are filled
# within the same column. This effectively propagates the last valid observation ba

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

Out[ ]:

	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

In [ ]:
```
# We can specify a forward-fill use previous valid observation to fill the actual g
df.fillna(method='ffill')
```

Out[ ]:

	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

1. Con el mismo `df`, se prueban distintos metodos de Data Amputation.
   - Llenar los `NA`s con otro valor
   - Tomar el promedio (siendo continuo)

- backfill (valor previo)
- frontfill (valor siguiente).