Missing Data

Data Science & Machine Learning Team

03/31/2021

Jingjie Zhang, PhD

jingjie.zhang@hms.com

Agenda

- Why Missing Data Occurs?
  - Missing Data Mechanisms
  - Formats of Missing Data
- Handling Missing Data in Python/Pandas
- Handling Missing Data Strategies
  - Dropping Cases/Columns
  - Mean/Median/Mode Imputation
  - Predicting Missing Cases Using Machine Learning
  - Multiple Imputation Is For Inference, Not For Prediction
- Examples in Python

Why Missing Data Occurs?

Three general “missing mechanisms” moving from the simplest to the general:

- Missing Completely at Random (MCAR)
  - If the probability of missing data is the same for all units.
  - Deletion missing data does not bias your inference.
  - Roll a dice; lottery number, ICD10 - CMs …
- Missing at Random (MAR)
  - If the probability of missing data depends on a set of observed responses.
  - Most common, missing values can be excluded (treated as NAs) or imputed.
  - CPT depends on ICD10-CM.
- Missing Not at Random (MNAR)
  - If the mechanism of missing data does not meet MCAR or MAR.
  - The only way to obtain unbiased estimates is to model the missing data process.
  - Covid-19 Symptoms

Formats of Missing Data

The presence of missing data:

- Null
  - Absence of everything; missing; empty
  - In HMS EDW_AR_FL, missing data format.
- Data-specific convention
  - Blank "" or " " or any invisible characters.
  - -9999 or -1
  - Boolean mask: True/False; 0/1
  - "?" In HMS EDW_CTS_FL, missing data format.
- Global convention
  - Nan (numpy nan type – np.nan, IEEE floating-point specification) or None (python object)
  - Most common

Pandas default
Missing data formats

Working with Missing Data in Numpy/Pandas array- None

- None can only present in arrays with data type “object”

```
array1 = np.array([1, 2, None, 3, ])
array1
```

```
array([1, 2, None, 3], dtype=object)
```

- Operations on python “object” type is much slower than operations on arrays with native types

```
for dtype in ['object', 'int']:
    print("dtype =", dtype)
    %timeit np.arange(1E6, dtype = dtype).sum()
    print()
```

```
dtype = object
56 ms ± 2.81 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

dtype = int
2.11 ms ± 196 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

- Cannot perform aggregation (sum/min/max/avg) across array with Nonetype

```
array1.sum()
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

Working with Missing Data in Numpy/Pandas array- NAN

- NaN (Not a Number): a special floating-point value recognized by all systems

```
array2 = np.array([1, 2, np.nan, 3])
array2.dtype
```

```
dtype('float64')
```

- Result of arithmetic with NaN will be another NaN

```
print(1 + np.nan, 1*np.nan, array2.sum(), array2.min())
```

```
nan nan nan nan
```

- Nan is a floating-point value; there is no equivalent NaN for integers, strings, or other types.

None & NaN in Pandas

- They are **interchangeable**

```
df = pd.Series([1, np.nan, 2, None])
df
```

```
0    1.0
1    NaN
2    2.0
3    NaN
dtype: float64
```

- Up-casting conventions in Pandas when NA values are introduced:

Typeclass	Conversion When Storing NAs	NA Sentinel Value
`object`	No change	`None` or `np.nan`
`float`	No change	`np.nan`
`int`	cast to `float`	`np.nan`
`boolean`	No change	`<NA>`

```
for dtype in ['object', 'float', 'int', 'boolean']:
    df = pd.Series([0, 1], dtype = dtype)
    df[0] = np.nan
    print("Original Series dtype = ", dtype)
    print(df, '\n')
```

```
Original Series dtype =  object
0    NaN
1      1
dtype: object

Original Series dtype =  float
0    NaN
1    1.0
dtype: float64

Original Series dtype =  int
0    NaN
1    1.0
dtype: float64

Original Series dtype =  boolean
0    <NA>
1    True
dtype: boolean
```

Handling Missing Data in Pandas

- Detecting null values - isnull() / isna() / notnull() / notna()

```
df
```

```
0     1.0
1     NaN
2     2.0
3     NaN
dtype: float64
```

```
df.isnull()
```

```
0    False
1     True
2    False
3     True
dtype: bool
```

```
df.notna()
```

```
0     True
1    False
2     True
3    False
dtype: bool
```

```
df[df.notna()]
```

```
0     1.0
2     2.0
dtype: float64
```

- Dropping null values – dropna()

	A	B	C
0	NaN	1.0	1
1	2.0	2.0	2
2	3.0	NaN	3

```
df.dropna(axis = 1)
```

	C
0	1
1	2
2	3

```
df.dropna(axis = 0)
```

	A	B	C
1	2.0	2.0	2

- Filling null values – fillna()

```
df
```

```
0     1.0
1     NaN
2     2.0
3     NaN
4     3.0
dtype: float64
```

```
df.fillna(0)
```

```
0     1.0
1     0.0
2     2.0
3     0.0
4     3.0
dtype: float64
```

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

```
0     1.0
1     2.0
2     2.0
3     3.0
4     3.0
dtype: float64
```

Next Observation Carried Backward(NOCB)

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

```
0     1.0
1     1.0
2     2.0
3     2.0
4     3.0
dtype: float64
```

Last Observation Carried Forward(LOCF)

Handling Strategies - Deletion

- Listwsie deletion (aka complete case analysis)
  - Simply drop all rows/samples containing missing values.
  - Pros: easy to implement
  - Cons: loss of data; increase the standard error and widen the confidence interval.
- Drop columns/fields/variables
  - Simply drop columns with majority of data are missing values
  - Be cautious using this approach

Handling Strategies - Deletion

Listwise Deletion

Gender	Age	Weight
F	20	130
~~M~~	~~NaN~~	~~150~~
F	30	132
M	40	160
~~F~~	~~43~~	~~NaN~~
F	50	150

```
df.dropna(axis = 0)
```

Drop Column

Gender	Age	Weight
F	20	130
M	NaN	150
F	30	132
M	40	160
F	43	NaN
F	50	150

```
df.dropna(axis = 1)
```

Handling Strategies – Mean/Median/Mode Imputation

Replace missing values with the variable mean, median or most frequent (mode) value.

- Pros: use the whole dataset
- Cons: reduce variance and the correlation between variables.

Gender	Age
F	20
M	NaN
F	25
M	30
F	40
F	40

Mean

Gender	Age
F	20
M	31
F	25
M	30
F	40
F	40

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy = 'mean')
```

Median

Gender	Age
F	20
M	30
F	25
M	30
F	40
F	40

```
SimpleImputer(strategy = 'median')
```

Mode

Gender	Age
F	20
M	40
F	25
M	30
F	40
F	40

```
SimpleImputer(strategy = 'most_frequent')
```

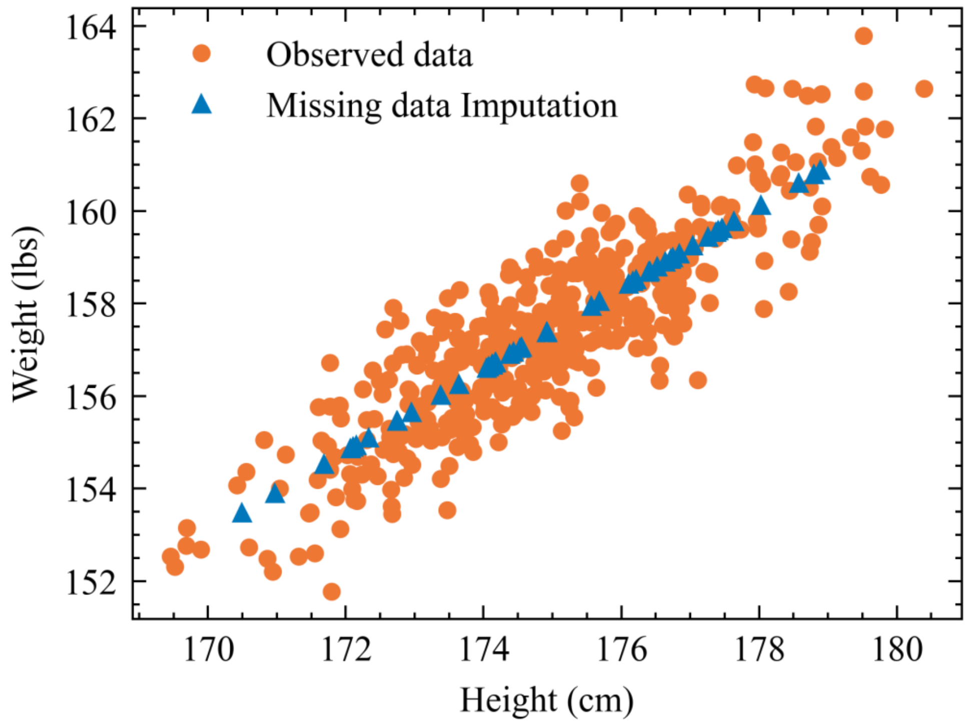
Handling Strategies – Machine Learning Model Inference

Predictive/Statistical models to infer the values of missing data. There are many options for such predictive model – Linear regression / Random Forest / KNN / Neural Networks…

- Pros: use information from the observed data; can be effective with cross-validation.
- Cons: over-estimate correlation.

- Linear Regression

Height (cm)	Weight (lbs)
170	150
171	NaN
175	155
180	162
177	NaN
178	160



- Depends on the model performance
- Follow the assumed relationship of the model

Handling Strategies – Multiple Imputation

Multiple imputation: missing values are filled multiple times to create “complete” datasets.

- Cons: Having multiple values reduces bias.
- Pros: Highly technical and difficult to implement.

- Multiple Imputation by Chained Equations (MICE)

Gender	Age	Income
F	33	NaN
NaN	18	12,000
M	NaN	13,542

Mean impute

Gender	Age	Income
F	33	12.771
F	18	12,000
M	25.5	13,542

Age back to NaN

Gender	Age	Income
F	33	12.771
F	18	12,000
M	NaN	13,542

Predictive Model using Gender&Weight to infer Age

Gender	Age	Income
F	33	12.771
F	18	12,000
M	35.3	13,542

Income back to NaN

Gender	Age	Income
F	33	NaN
F	18	12,000
M	35.3	13,542

Gender	Age	Income
F	33	13,103
F	18	12,000
M	35.3	13,542

Continue to Gender ….
Can do many iterations

Python Examples

- Notebook with these examples on Github

Questions?

Future Topics

Have requests?

Let me know!

Introduction to Data Science Course Outline

Andrew Wheeler, PhD, andrew.wheeler@hms.com

- Lesson 01: Data Science 101
- Lesson 02: Machine Learning 101
- Lesson 03: Evaluating Predictions
- Lesson 04: Intro Data Transformation in Python
- Lesson 05: Data Visualization 101
- Lesson 06: Feature Engineering
- Lesson 07: Missing Data
- Lesson 08: Big Data and Parallel Computing Intro
- Lesson 09: Dimension Reduction and Unsupervised Learning
- Lesson 10: High Cardinality (Many Categories)
- Lesson 11: Intro to Forecasting
- Lesson 12: Conducting Experiments

Missing Data

Data Science & Machine Learning Team

03/31/2021

Jingjie Zhang, PhD

jingjie.zhang@hms.com