Dealing with Missing Data - Lab

Introduction

In this lab, we'll work through strategies for data cleaning and dealing with missing values (`NaN` s).

Objectives

In this lab you will:

- Identify missing values in a dataframe using built-in methods
- Explain why missing values are a problem in data science

Dataset

In this lab, we'll continue working with the *Titanic Survivors* dataset, which can be found in `'titanic.csv'`.

Before we can get going, we'll need to import the usual libraries. In the cell below, import:

- `pandas` as `pd`
- `numpy` as `np`
- `matplotlib.pyplot` as `plt`
- set `%matplotlib inline`

```
# Import necessary libraries below
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Now, let's get started by reading in the data from the `'titanic.csv'` file and storing it the DataFrame `df`. Subsequently, be sure to preview the data.

```
# Use pandas to load the csv file
df = pd.read_csv('titanic.csv')
df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	Unnamed: 0	PassengerId	Survived	Pclass	Name	Sex	Age
0	0	1	0	3	Braund, Mr. Owen Harris	male	22.0

	Unnamed: 0	PassengerId	Survived	Pclass	Name	Sex	Age
1	1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	2	3	1	3	Heikkinen, Miss. Laina	female	26.0
3	3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
4	4	5	0	3	Allen, Mr. William Henry	male	35.0

Find missing values in a DataFrame

Before we can deal with missing values, we first need to find them. There are several easy ways to detect them. We will start by answering very general questions, such as "does this DataFrame contain any null values?", and then narrowing our focus each time the answer to a question is "yes".

We'll start by checking to see if the DataFrame contains **any** missing values (NaNs) at all.

Hint: If you do this correctly, it will require method chaining, and will return a boolean value for each column.

```
df.isna().any()
```

```
Unnamed: 0     False
PassengerId    False
Survived       False
Pclass         False
Name           False
Sex            False
Age             True
SibSp          False
Parch          False
Ticket         False
Fare           False
Cabin           True
Embarked        True
dtype: bool
```

Now we know which columns contain missing values, but not how many.

In the cell below, chain a different method with `isna()` to check how many total missing values are in each column.

Expected Output:

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
df.isna().sum()
```

```
Unnamed: 0       0
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
```

```
Age           177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin         687
Embarked        2
dtype: int64
```

Now that we know how many missing values exist in each column, we can make some decisions about how to deal with them.

We'll deal with each column individually, and employ a different strategy for each.

Dropping the column

The first column we'll deal with is the `Cabin` column. We'll begin by examining this column more closely.

In the cell below:

- Determine what percentage of rows in this column contain missing values
- Print out the number of unique values in this column

```
print('Percentage of Null Cabin Values:', len(df[df.Cabin.isna()])/ len(df))
print('Number of Unique Cabin Values:', df.Cabin.nunique())
```

```
Percentage of Null Cabin Values: 0.7710437710437711
Number of Unique Cabin Values: 147
```

With this many missing values, it's probably best for us to just drop this column completely.

In the cell below:

- Drop the `Cabin` column in place from the `df` DataFrame
- Then, check the remaining number of null values in the dataset by using the code you wrote previously

```
df = df.drop('Cabin', axis = 1)
df.isna().sum()
```

```
Unnamed: 0       0
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Embarked         2
dtype: int64
```

Computing placeholder values

Recall that another common strategy for dealing with missing values is to replace them with the mean or median for that column. We'll begin by investigating the current version of the `'Age'` column.

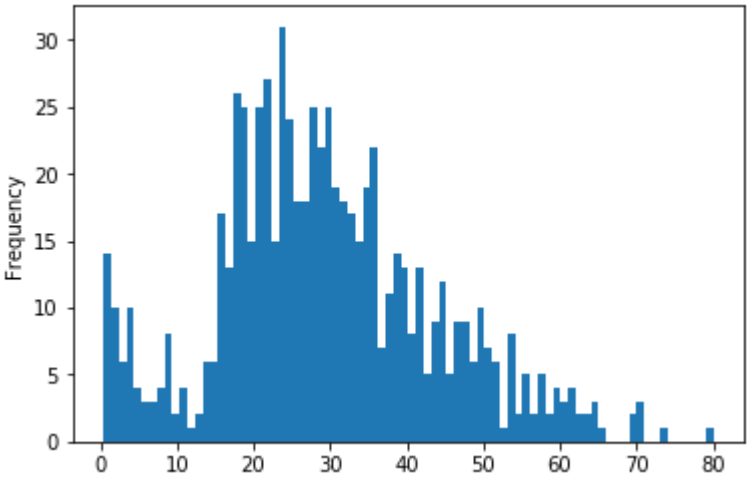
In the cell below:

- Plot a histogram of values in the `'Age'` column with 80 bins (1 for each year)
- Print out the mean and median for the column

```
age_mean = df['Age'].mean()
age_median = df['Age'].median()
df['Age'].plot(kind='hist', bins=80)

print("Mean Value for Age column: {}".format(age_mean))
print("Median Value for Age column: {}".format(age_median))
```

```
Mean Value for Age column: 29.69911764705882
Median Value for Age column: 28.0
```



From the visualization above, we can see the data has a slightly positive skew.

In the cell below, replace all missing values in the `'Age'` column with the median of the column. **Do not hard code this value -- use the methods from pandas or numpy to make this easier!** Do this replacement in place on the DataFrame.

```
df['Age'] = df['Age'].fillna(value=df['Age'].median)
```

Now that we've replaced the values in the `'Age'` column, let's confirm that they've been replaced.

In the cell below, check how many null values remain in the dataset.

```
df.isna().sum()
```

```
Unnamed: 0     0
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       2
dtype: int64
```

Great! Now we need to deal with the two pesky missing values in the `'Embarked'` column.

Dropping rows that contain missing values

Perhaps the most common solution to dealing with missing values is to simply drop any rows that contain them. Of course, this is only a good idea if the number dropped does not constitute a significant portion of our dataset. Often, you'll need to make the overall determination to see if dropping the values is an acceptable loss, or if it is a better idea to just drop an offending column (e.g. the `'Cabin'` column) or to impute placeholder values instead.

In the cell below, use the appropriate built-in DataFrame method to drop the rows containing missing values. Do this in place on the DataFrame.

```
df = df.dropna()
df.isna().sum()
```

```
Unnamed: 0     0
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

Great! We've dealt with all the ***obvious*** missing values, but we should also take some time to make sure that there aren't symbols or numbers included that are meant to denote a missing value.

Missing values with placeholders

A common thing to see when working with datasets is missing values denoted with a preassigned code or symbol. Let's check to ensure that each categorical column contains only what we expect.

In the cell below, return the unique values in the `'Embarked'`, `'Sex'`, `'Pclass'`, and `'Survived'` columns to ensure that there are no values in there that we don't understand or can't account for.

```
for col in ['Embarked','Sex', 'Pclass','Survived']:
    print('Values for {}:\n{}\n\n'.format(col, df[col].unique()))
```

```
Values for Embarked:
['S' 'C' 'Q']


Values for Sex:
['male' 'female']


Values for Pclass:
['3' '1' '2' '?']


Values for Survived:
[0 1]
```

It looks like the `'Pclass'` column contains some missing values denoted by a placeholder!

In the cell below, investigate how many placeholder values this column contains. Then, deal with these missing values using whichever strategy you believe is most appropriate in this case.

```
df.Pclass.value_counts(normalize=True)
```

```
3    0.527559
1    0.224972
2    0.193476
?    0.053993
Name: Pclass, dtype: float64
```

```
# Observation: account for 5% of the data
# Method: randomly select a class acccording to current distribution
rel_prob = [.53, .22, .19]
prob = [i/sum(rel_prob) for i in rel_prob]
def impute_pclass(value):
    if value == '?':
        return np.random.choice(['3','1','2'], p=prob)
    else:
        return value
```

```
df.Pclass = df.Pclass.map(lambda x: impute_pclass(x))
df.Pclass.value_counts(normalize=True)
```

```
3    0.547807
1    0.242970
2    0.209224
Name: Pclass, dtype: float64
```

Question: What is the benefit of treating missing values as a separate valid category? What is the benefit of removing or replacing them? What are the drawbacks of each? Finally, which strategy did you choose? Explain your choice below.

Write your answer below this line:

```
# Sample response:

# By treating missing values as a separate category, information is preserved.
# Perhaps there is a reason that this information is missing.
# By removing or replacing missing information, we can more easily conduct mathemati
# I chose to randomly replace for now. I could have just as easily removed the data.
# Concerns include that I imputed the wrong value (indeed it was a random guess).
# The strategy for dealing with missing data will depend on our desired application,
# but regardless of the approach taken, the ramifications of how missing data are ha
# For example, imputing the median of our age reduces variance
# and assumes that a new value would be close to the center of the distribution
# (albeit this assumption is statistically likely).
```

Now, let's do a final check to ensure that there are no more missing values remaining in this dataset.

In the cell below, reuse the code you wrote at the beginning of the notebook to check how many null values our dataset now contains.

```
df.isna().sum()
```

```
Unnamed: 0     0
PassengerId    0
Survived       0
Pclass         0
Name           0
```

```
Sex           0
Age           0
SibSp         0
Parch         0
Ticket        0
Fare          0
Embarked      0
dtype: int64
```

Great! Those all seem in line with our expectations. We can confidently say that this dataset contains no pesky missing values that will mess up our analysis later on!

Summary

In this lab, we learned:

- How to detect missing values in our dataset
- How to deal with missing values by dropping rows
- How to deal with missing values by imputing mean/median values
- Strategies for detecting missing values encoded with a placeholder

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

- **Jupyter Notebook** 100.0%