Clean and prepare data in a Pandas DataFrame

Pandas provides you with several fast, flexible, and intuitive ways to clean and prepare your data. By the end of this tutorial, you'll have learned all you need to know to get started with:

Working with missing data using methods such as .fillna()

Working with duplicate data using methods such as the .remove_duplicates() method

Cleaning string data using the .str accessor.

In [2]:

```
# Loading a Sample Pandas DataFrame
import pandas as pd
import numpy as np
df = pd.DataFrame.from_dict({
    'Name': ['Nik', 'Kate', 'Evan', 'Kyra', np.NaN],
    'Age': [33, 32, 40, 57, np.NaN],
    'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN]
})
print(df)
```

```
   Name   Age  Location
0   Nik  33.0   Toronto
1  Kate  32.0    London
2  Evan  40.0  New York
3  Kyra  57.0       NaN
4   NaN   NaN       NaN
```

The first method is head() - which returns the first 5 rows of the dataset.

In [29]:

```
df.head()
```

Out[29]:

	Name	Location	Sales
1	Lana; Courtney	LONDON	243.0
2	Abel; Shakti	New york	654.0
3	Vasu; Imogene	NaN	NaN
4	Aravind; Shelly	toronto	345.0
5	Tranter; Melvyn	Madrid	NaN

Understanding the Pandas isnull Method

Pandas comes with an incredibly helpful method, .isnull(), that identifies whether a value is missing or not.

The method returns a boolean value, either True or False.

We can apply the method either to an entire DataFrame or to a single column.

The method will broadcast correctly to either the Series or the DataFrame, depending on what it's applied to.

Let's take a quick look:

In [3]:

```
# Exploring the .isnull() method
print(df.isnull())
```

```
    Name    Age  Location
0  False  False     False
1  False  False     False
2  False  False     False
3  False  False      True
4   True   True      True
```

Counting Missing Values in a Pandas DataFrame

One of the first steps you'll want to take is to understand how many missing values you actually have in your DataFrame.

One way to do this is to use a chained version the .isnull() method and the .sum() method:

In [4]:

```
print(df.isnull().sum())
```

```
Name        1
Age         1
Location    2
dtype: int64
```

Dropping Missing Data in a Pandas DataFrame

When working with missing data, it's often good to do one of two things:

   either drop the records or find ways to fill the data.

In [5]:

```
# Exploring the Pandas .dropna() method
df.dropna(
    axis=0,          # Whether to drop rows or columns
    how='any',       # Whether to drop records if 'all' or 'any' records are miss
ing
    thresh=None,     # How many columns/rows must be missing to drop
    subset=None,     # Which rows/columns to consider
    inplace=False    # Whether to drop in place (i.e., without needing to re-assi
gn)
)
```

Out[5]:

	Name	Age	Location
0	Nik	33.0	Toronto
1	Kate	32.0	London
2	Evan	40.0	New York

In [6]:

```
#By default, Pandas will drop records where any value is missing.
#Because of this, it also removed the fourth row, where only one value was missi
ng.
df.dropna()
```

Out[6]:

	Name	Age	Location
0	Nik	33.0	Toronto
1	Kate	32.0	London
2	Evan	40.0	New York

Filling Missing Data in a Pandas DataFrame

Removing missing data also removes any associated data from those records.

Because of this, it can be helpful to fill in missing values.

You can do this using the .fillna() method.

In [7]:

```
# Using .fillna() to Fill Missing Data
df = df.fillna(0)
print(df)
```

```
    Name    Age  Location
0    Nik   33.0   Toronto
1   Kate   32.0    London
2   Evan   40.0  New York
3   Kyra   57.0         0
4      0    0.0         0
```

In [8]:

```
# Filling Columns with Different Values
df = df.fillna({'Name': 'Someone', 'Age': 25, 'Location': 'USA'})
print(df)

# Returns:
#        Name    Age  Location
# 0       Nik   33.0   Toronto
# 1      Kate   32.0    London
# 2      Evan   40.0  New York
# 3      Kyra   57.0       USA
# 4   Someone   25.0       USA
```

```
    Name    Age  Location
0    Nik   33.0   Toronto
1   Kate   32.0    London
2   Evan   40.0  New York
3   Kyra   57.0         0
4      0    0.0         0
```

In [9]:

```
# Imputing a Missing Value
df['Age'] = df['Age'].fillna(df['Age'].mean())
print(df)

# Returns:
#     Name    Age  Location
# 0    Nik   33.0   Toronto
# 1   Kate   32.0    London
# 2   Evan   40.0  New York
# 3   Kyra   57.0       NaN
# 4    NaN   40.5       NaN
```

```
    Name    Age  Location
0    Nik   33.0   Toronto
1   Kate   32.0    London
2   Evan   40.0  New York
3   Kyra   57.0         0
4      0    0.0         0
```

Working with Duplicate Data in Pandas

Duplicate data can be introduced into a dataset for a number of reasons.

Sometimes this data can be valid, while other times it can present serious problems in your data's integrity.

Because of this, it's important to understand how to find and deal with duplicate data.

In [10]:

```
# Loading a Sample Pandas DataFrame
import pandas as pd
df = pd.DataFrame.from_dict({
    'Name': ['Nik', 'Kate', 'Evan', 'Kyra', 'Nik', 'Kate'],
    'Age': [33, 32, 40, 57, 33, 32],
    'Location': ['Toronto', 'London', 'New York', 'Atlanta', 'Toronto', 'Pari
s'],
    'Date Modified': ['2022-01-01', '2022-02-24', '2022-08-12', '2022-09-12', '2
022-01-01', '2022-12-09']
})

print(df)
```

```
   Name  Age  Location Date Modified
0   Nik   33   Toronto    2022-01-01
1  Kate   32    London    2022-02-24
2  Evan   40  New York    2022-08-12
3  Kyra   57   Atlanta    2022-09-12
4   Nik   33   Toronto    2022-01-01
5  Kate   32     Paris    2022-12-09
```

Identifying Duplicate Records in a Pandas DataFrame

Pandas provides a helpful method, .duplicated(), which allows you to identify duplicate records in a dataset.

The method returns boolean values when duplicate records exist.

In [11]:

```
# Identifying Duplicate Records in a Pandas DataFrame
print(df.duplicated())
```

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

In [15]:

```
#Removing Duplicate Data in a Pandas DataFrame
df.drop_duplicates()
```

Out[15]:

	Name	Age	Location	Date Modified
0	Nik	33	Toronto	2022-01-01
1	Kate	32	London	2022-02-24
2	Evan	40	New York	2022-08-12
3	Kyra	57	Atlanta	2022-09-12
5	Kate	32	Paris	2022-12-09

Cleaning Strings in Pandas

In [16]:

```
# Loading a Sample Pandas DataFrame
import pandas as pd
df = pd.DataFrame.from_dict({
    'Name': ['Tranter, Melvyn', 'Lana, Courtney', 'Abel, Shakti', 'Vasu, Imogen
e', 'Aravind, Shelly'],
    'Region': ['Region A', 'Region A', 'Region B', 'Region C', 'Region D'],
    'Location': ['TORONTO', 'LONDON', 'New york', 'ATLANTA', 'toronto'],
    'Favorite Color': ['   green  ', 'red', '  yellow', 'blue', 'purple  ']
})

print(df)
```

```
              Name    Region  Location Favorite Color
0  Tranter, Melvyn  Region A   TORONTO        green
1   Lana, Courtney  Region A    LONDON            red
2     Abel, Shakti  Region B  New york         yellow
3    Vasu, Imogene  Region C   ATLANTA           blue
4  Aravind, Shelly  Region D   toronto       purple
```

We can see that our DataFrame has some messy string data! For example, some columns contain multiple data points (first and last name), others have redundant data (the word ‘Region’), have messy capitalization (location), and have added whitespace (favorite colors).

Trimming White Space in Pandas Strings

Let’s start off by removing whitespace from text in Pandas. We can see that the column 'Favorite Color' has extra whitespace on either end of the color. Python comes with a number of methods to strip whitespace from the front of a string, the back of a string, or either end. Because the whitespace exists on either end of the string, we will make use of the .strip() method.

In [17]:

```
# Trimming Whitespace from a Pandas Column
df['Favorite Color'] = df['Favorite Color'].str.strip()
print(df)
```

```
              Name    Region  Location Favorite Color
0  Tranter, Melvyn  Region A   TORONTO          green
1   Lana, Courtney  Region A    LONDON            red
2     Abel, Shakti  Region B  New york         yellow
3    Vasu, Imogene  Region C   ATLANTA           blue
4  Aravind, Shelly  Region D   toronto         purple
```

Splitting Strings into Columns in Pandas

The 'Name' column contains both the person's last and first names. In many cases, you may want to split this column into two – one for each the first and last name. This approach will work a little differently, as we will want to assign two columns, rather than just one.

In [18]:

```
# Applying .split on a column
print(df['Name'].str.split(','))
```

```
0    [Tranter,  Melvyn]
1     [Lana,  Courtney]
2       [Abel,  Shakti]
3      [Vasu,  Imogene]
4    [Aravind,  Shelly]
Name: Name, dtype: object
```

We can see that this returned a list of strings. What we want to do, however, is assign this to multiple columns. In order to do this, we need to pass in the expand=True argument, in order to instruct Pandas to split the values into separate items. From there, we can assign the values into two columns:

In [19]:

```
# Splitting a Column into Two Columns
df[['Last Name', 'First Name']] = df['Name'].str.split(',', expand=True)

print(df)
```

```
              Name    Region  Location Favorite Color Last Name Firs
t Name
0  Tranter, Melvyn  Region A   TORONTO          green   Tranter
Melvyn
1   Lana, Courtney  Region A    LONDON            red      Lana   Co
urtney
2     Abel, Shakti  Region B  New york         yellow      Abel
Shakti
3    Vasu, Imogene  Region C   ATLANTA           blue      Vasu    I
mogene
4  Aravind, Shelly  Region D   toronto         purple   Aravind
Shelly
```

Replacing Text in Strings in Pandas

In the 'Region' column, the word “Region” is redundant. In this example, you’ll learn how to replace some text in a column. In particular, you’ll learn how to remove a given substring in a larger string. For this, we can use the aptly-named .replace() method. The method takes a string we want to replace and a string that we want to substitute with. Because we want to remove a substring, we’ll simply pass in an empty string to substitute with.

In [20]:

```
# Replacing a Substring in Pandas
df['Region'] = df['Region'].str.replace('Region ', '')
print(df)
```

```
              Name Region  Location Favorite Color Last Name First N
ame
0  Tranter, Melvyn      A   TORONTO          green   Tranter     Mel
vyn
1   Lana, Courtney      A    LONDON            red      Lana   Court
ney
2     Abel, Shakti      B  New york         yellow      Abel     Sha
kti
3    Vasu, Imogene      C   ATLANTA           blue      Vasu    Imog
ene
4  Aravind, Shelly      D   toronto         purple   Aravind     She
lly
```

Changing String Case in Pandas

In this section, we’ll learn how to fix the odd and inconsistent casing that exists in the 'Location' column. Pandas provides access to a number of methods that allow us to change cases of strings:

.upper() will convert a string to all upper case

.lower() will convert a string to all lower case

.title() will convert a string to title case

In this case, we want our locations to be in title case, so we can apply to .str.title() method to the string:

In [22]:

```
# Changing Text to Title Case in Pandas
df['Location'] = df['Location'].str.title()
print(df)
```

```
              Name Region  Location Favorite Color Last Name First N
ame
0  Tranter, Melvyn      A   Toronto          green   Tranter     Mel
vyn
1   Lana, Courtney      A    London            red      Lana   Court
ney
2     Abel, Shakti      B  New York         yellow      Abel     Sha
kti
3    Vasu, Imogene      C   Atlanta           blue      Vasu    Imog
ene
4  Aravind, Shelly      D   Toronto         purple   Aravind     She
lly
```

Exercise

In [26]:

```
# Loading a DataFrame
import pandas as pd
import numpy as np

df = pd.DataFrame.from_dict({
    'Name': ['Tranter; Melvyn', 'Lana; Courtney', 'Abel; Shakti', 'Vasu; Imogen
e', 'Aravind; Shelly', 'Tranter; Melvyn'],
    'Location': ['TORONTO', 'LONDON', 'New york', np.NaN, 'toronto', 'Madrid'],
    'Sales': [123, 243, 654, np.NaN, 345, np.NaN]
})
print(df)
```

```
              Name  Location  Sales
0  Tranter; Melvyn   TORONTO  123.0
1   Lana; Courtney    LONDON  243.0
2     Abel; Shakti  New york  654.0
3    Vasu; Imogene       NaN    NaN
4  Aravind; Shelly   toronto  345.0
5  Tranter; Melvyn    Madrid    NaN
```

Question 1: Create a First Name and a Last Name column. Note that there is a semi-colon between names.

In [25]:

```
df[['Last Name', 'First Name']] = df['Name'].str.split(';', expand=True)
print(df)
```

```
              Name  Location  Sales Last Name First Name
0  Tranter; Melvyn   TORONTO  123.0   Tranter     Melvyn
1   Lana; Courtney    LONDON  243.0      Lana   Courtney
2     Abel; Shakti  New york  654.0      Abel     Shakti
3    Vasu; Imogene       NaN    NaN      Vasu    Imogene
4  Aravind; Shelly   toronto  345.0   Aravind     Shelly
5  Tranter; Melvyn    Madrid    NaN   Tranter     Melvyn
```

Question 2: Drop any duplicate records based only on the Name column, keeping the last record.

In [27]:

```
df = df.drop_duplicates(subset='Name', keep='last')
print(df)
```

```
              Name  Location  Sales
1   Lana; Courtney    LONDON  243.0
2     Abel; Shakti  New york  654.0
3    Vasu; Imogene       NaN    NaN
4  Aravind; Shelly   toronto  345.0
5  Tranter; Melvyn    Madrid    NaN
```

Question 3: Calculate the percentage of missing records in each column.

In [28]:

```
print(df.isnull().sum() / len(df))
```

```
Name        0.0
Location    0.2
Sales       0.4
dtype: float64
```