Python para Análisis de Datos

Módulo 05

Pandas

Missing values

Tener un dataset con missing values es un caso muy común a la hora de trabajar con datos. Puede suceder que algunos datos se hayan perdido, por lo que no estén disponibles, o que no se hayan recolectado correctamente. Incluso puede suceder que la información jamás haya existido.

Existen varias técnicas a la hora de trabajar con missing values y pandas ofrece funcionalidades para manipularlos.

Missing values

Para representar missing values pandas usa principalmente el valor `np.nan` *(NaN: Not a Number).* Este es un valor especial dentro de los números flotantes.

Otro valor especial dentro de los números flotantes es `np.inf` (que se usa para representar el infinito, no missing values).

El otro valor que se usa para representar missing values es `None`, que es un tipo de dato de Python con un único valor.

```
type(np.nan), type(np.inf), type(None)
```

```
(float, float, NoneType)
```

Missing values

Como `np.nan` es un flotante, cuando se encuentra en una columna de enteros, toda la columna es promovida al tipo de dato flotante.

```
df = pd.DataFrame({"A": [1,3,5,4,2], "B": [8,2,np.nan,1,5]})
df
```

	A	B
0	1	8.0
1	3	2.0
2	5	NaN
3	4	1.0
4	2	5.0

Missing values

A la hora de hacer cálculos entre `NaN` y otros valores, el resultado siempre se promueve a `NaN`.

Sin embargo, algunos métodos permiten controlar cómo se trata los missing values. Los métodos asociados a los operadores permite definir un valor a reemplazar para `NaN` antes de realizar la operación.

```
df['A'] + df['B']
```

```
0    9.0
1    5.0
2    NaN
3    5.0
4    7.0
dtype: float64
```

```
df["A"].add(df["B"])
```

```
0    9.0
1    5.0
2    NaN
3    5.0
4    7.0
dtype: float64
```

```
df["A"].add(df["B"], fill_value=0)
```

```
0    9.0
1    5.0
2    5.0
3    5.0
4    7.0
dtype: float64
```

Missing values

A su vez, las funciones de agregación también permiten controlar cómo se trata `NaN`.

Por defecto son ignorados, pero se puede controlar con el parámetro `skipna`.

```
df.mean()
```

```
A    3.0
B    4.0
dtype: float64
```

```
df.mean(skipna=False)
```

```
A    3.0
B    NaN
dtype: float64
```

isna

Para detectar missing values está el método `isna` (o su alias, `isnull`).

Este método mapea los datos del dataframe o serie a booleanos. Sólo mapea `np.nan` y `None` a `True` y cualquier otro valor a `False`.

Alternativamente, está el método `notna` (y su alias, `notnull`) para hacer el mapeo inverso.

```
df.isna()
```

	A	B
0	False	False
1	False	False
2	False	True
3	False	False
4	False	False

```
df.notna()
```

	A	B
0	True	True
1	True	True
2	True	False
3	True	True
4	True	True

isna

Teniendo en cuenta que `True` se trata como 1 y `False` como 0, podemos realizar estadísticas sobre la cantidad de missing values.

```
# Cantidad de NaN por columna
# en el dataset del titanic
data.isna().sum()
```

```
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
```

```
# columnas que presentan NaN
data.isna().any()
```

```
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
```

dropna

Una forma de tratar los `NaN` es simplemente descartarlos.

El método `dropna` elimina las filas que contengan al menos un `NaN`. Se puede limitar las columnas a considerar con el parámetro `subset` o eliminar las filas con todas las columnas `NaN` con `how`.

```
data.dropna().info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 183 entries, 1 to 889
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  183 non-null    int64  
 1   Survived     183 non-null    int64  
 2   Pclass       183 non-null    int64  
 3   Name         183 non-null    object 
 4   Sex          183 non-null    object 
 5   Age          183 non-null    float64
 6   SibSp        183 non-null    int64  
 7   Parch        183 non-null    int64  
 8   Ticket       183 non-null    object 
 9   Fare         183 non-null    float64
 10  Cabin        183 non-null    object 
 11  Embarked     183 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

fillna

Para no perder tanta información, también es posible reemplazar los `NaN` por un valor que consideremos apropiado con el método `fillna`.

Por ejemplo, reemplazar los `NaN` en la columna *Age* por el promedio de edad.

```
data["Age"].fillna(data["Age"].mean())
```

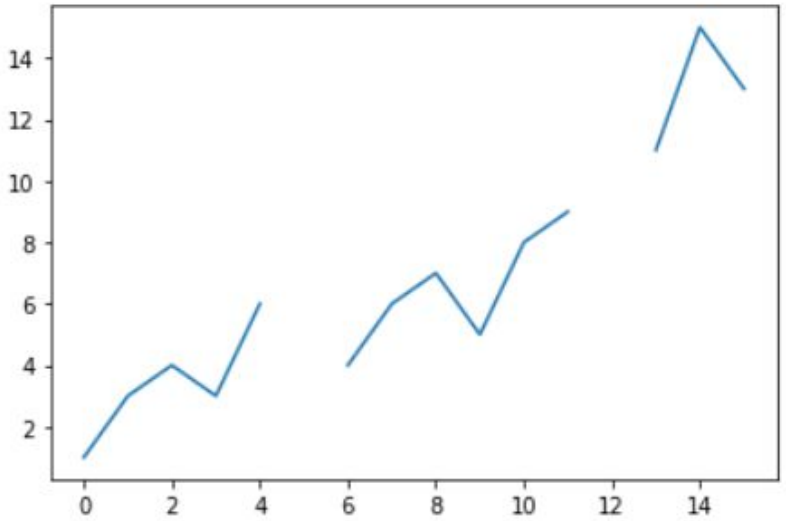
```
0      22.000000
1      38.000000
2      26.000000
3      35.000000
4      35.000000
         ...
886    27.000000
887    19.000000
888    29.699118
889    26.000000
890    32.000000
Name: Age, Length: 891, dtype: float64
```

interpolate

Cuando tenemos datos ordenados (por ejemplo una serie de tiempo) se pueden interpolar los datos faltantes con el método `interpolate`.

```
s = pd.Series([1,3,4,3,6,np.nan,4,6,7,5,8,9,np.nan,11,15,13])
s.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f801ac20a00>
```

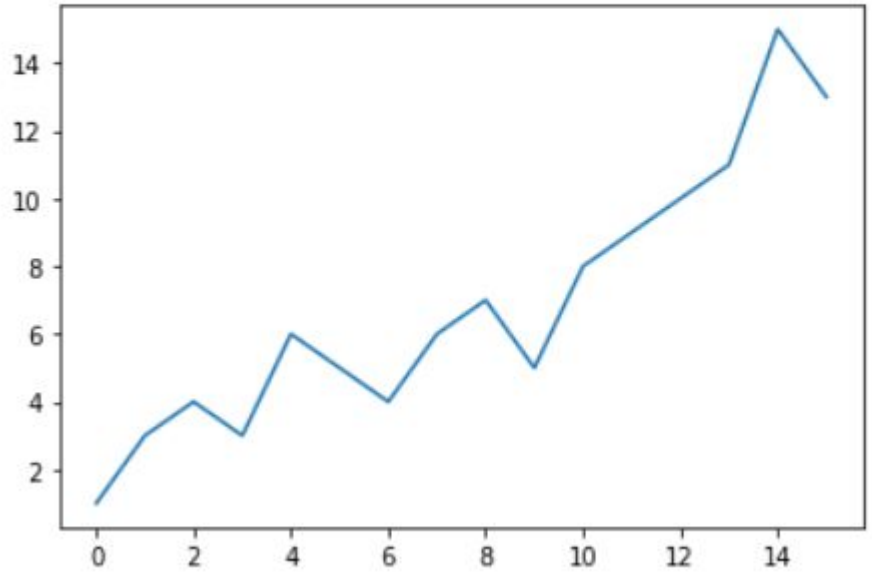


interpolate

Por defecto hace una interpolación lineal, pero se pueden especificar otros métodos.

```
s.interpolate().plot()
```

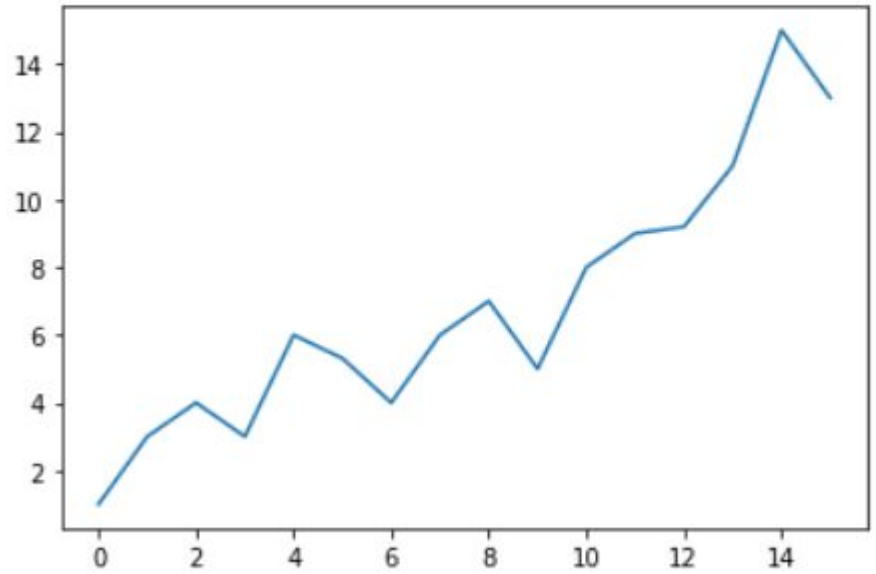
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f801abf77c0>
```



interpolate

```
s.interpolate(method="quadratic").plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f801ab18280>
```



¡Muchas gracias!

¡Sigamos trabajando!