Advanaced pandas

In [1]:
```
import pandas as pd
import numpy as np
```

Maps¶

A map is a term, borrowed from mathematics, for a function that takes one set of values and "maps" them to another set of values. In data science we often have a need for creating new representations from existing data, or for transforming data from the format it is in now to the format that we want it to be in later. Maps are what handle this work, making them extremely important for getting your work done!

There are two mapping methods that you will use often.

1. map()
2. apply()

In [6]:
```
autos = pd.read_csv('titanic.csv', encoding = "Latin-1")
```

In [8]:
```
df = autos
```

Add a column

In [13]:
```
df['New_col'] = np.random.randint(10)
```

In [15]:
```
df.head()
```

Out[15]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [17]:
```
df['New_col'] = "Hello world"
```

In [20]:
```
df.drop("new_col", axis=1, inplace = True)
```

In [21]:
```
df.head()
```

Out[21]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [23]:
```
df.shape
```

Out[23]:
```
(891, 13)
```

Lambda function

Syntax

lambda arguments : expression

In [24]:
```
def disk_area(radius):
    return 3.14 * radius * radius
```

In [25]:
```
x = lambda a : a + 10
```

In [26]:
```
print(x(5))
```

```
15
```

In [17]:
```
x = lambda a : a + 10
print(x(5))
```

```
15
```

In [18]:
```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

In [27]:
```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
13
```

In [28]:
```
x = lambda a, b, c, d : a + b * c + d
print(x(4, 5, 8, 2))
```

```
46
```

Map function map()

In [29]:
```
age_mean = df.Age.mean()
```

In [30]:
```
age_mean
```

Out[30]:
```
29.69911764705882
```

In [34]:
```
df['diff_age'] = df.Age.map(lambda age: age - age_mean)
```

Using .map allows the same functions to be used for each row

In [35]:
```
df.head()
```

Out[35]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [15]:
```
df['diff_age'] = df.Age.map(lambda age: np.abs(age - age_mean))
```

In [16]:
```
df.head()
```

Out[16]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

Apply function apply()

apply() is the equivalent method if we want to transform a whole DataFrame by calling a custom method on each row.

In [36]:
```
def get_data(row):
    if row.Age > 18 and row.Sex == 'male':
        return 'adult male'
    elif row.Age <= 18 and row.Sex == 'male':
        return 'minor male'
    elif row.Age > 18 and row.Sex == 'female':
        return 'adult female'
    else:
        return 'minor male'
```

In [37]:
```
df['stage_of_life'] = df.apply(get_data, axis='columns')
```

In [38]:
```
df.head()
```

Out[38]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Eml
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [39]:
```
df.stage_of_life.value_counts()
```

Out[39]:
```
adult male      382
minor male      316
adult female    193
Name: stage_of_life, dtype: int64
```

In [40]:
```
df.Age.apply(lambda x : x)
```

Out[40]:
```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       ...
886    27.0
887    19.0
888     NaN
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

Now it's your turn

Grouping and Sorting

As a data analysts, your job is to find pattern from the data

Maps allow us to transform data in a DataFrame or Series one value at a time for an entire column. However, often we want to group our data, and then do something specific to the group the data is in.

As you'll learn, we do this with the groupby()

1 = survived 0 = died

In [41]:
```
df.groupby('Survived')
```

Out[41]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000026F78EA5C70>
```

In [42]:
```
df.groupby('Survived').Age.mean()
```

Out[42]:
```
Survived
0    30.626179
1    28.343690
Name: Age, dtype: float64
```

In [43]:
```
df.groupby('Survived').Fare.mean()
```

Out[43]:
```
Survived
0    22.117887
```

```
1    48.395408
Name: Fare, dtype: float64
```

In [44]:
```
df.groupby(['Sex', 'Survived']).Age.mean()
```

Out[44]:
```
Sex     Survived
female  0           25.046875
        1           28.847716
male    0           31.618056
        1           27.276022
Name: Age, dtype: float64
```

In [45]:
```
df.groupby(['Sex', 'Survived']).Fare.mean()
```

Out[45]:
```
Sex     Survived
female  0           23.024385
        1           51.938573
male    0           21.960993
        1           40.821484
Name: Fare, dtype: float64
```

In [46]:
```
df.Survived.value_counts()
```

Out[46]:
```
0    549
1    342
Name: Survived, dtype: int64
```

In [47]:
```
df.Survived.value_counts(normalize=True)
```

Out[47]:
```
0    0.616162
1    0.383838
Name: Survived, dtype: float64
```

In [48]:
```
df.groupby(['Survived']).count()
```

Out[48]:

Survived	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	New_co
0	549	549	549	549	424	549	549	549	549	68	549	54
1	342	342	342	342	290	342	342	342	342	136	340	34

In [49]:
```
df.groupby(['Sex', 'Survived']).count()
```

Out[49]:

Sex	Survived	PassengerId	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	New
female	**0**	81	81	81	64	81	81	81	81	6	81	
	1	233	233	233	197	233	233	233	233	91	231	
male	**0**	468	468	468	360	468	468	468	468	62	468	

Sex	Survived	PassengerId	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	New
	1	109	109	109	93	109	109	109	109	45	109	

In [50]:
```
df.groupby(['Survived']).Age.agg([len, min, max])
```

Out[50]:

Survived	len	min	max
0	549	1.00	74.0
1	342	0.42	80.0

In [51]:
```
df.groupby(['Survived', 'Sex']).Age.agg([len, min, max])
```

Out[51]:

Survived	Sex	len	min	max
0	female	81	2.00	57.0
	male	468	1.00	74.0
1	female	233	0.75	63.0
	male	109	0.42	80.0

Reset Index

In [52]:
```
aa = df.groupby(['Survived', 'Sex']).Age.agg([len, min, max]).reset_index()
```

In [53]:
```
aa
```

Out[53]:

	Survived	Sex	len	min	max
0	0	female	81	2.00	57.0
1	0	male	468	1.00	74.0
2	1	female	233	0.75	63.0
3	1	male	109	0.42	80.0

Sort values

In [54]:
```
aa.sort_values(['min'], ascending=True)
```

Out[54]:

	Survived	Sex	len	min	max
3	1	male	109	0.42	80.0
2	1	female	233	0.75	63.0
1	0	male	468	1.00	74.0
0	0	female	81	2.00	57.0

In [55]:
```
aa.sort_values(['min'], ascending=False)
```

Out[55]:

	Survived	Sex	len	min	max
0	0	female	81	2.00	57.0
1	0	male	468	1.00	74.0
2	1	female	233	0.75	63.0
3	1	male	109	0.42	80.0

Missing data

In [56]:
```
df.isnull().sum()
```

Out[56]:
```
PassengerId        0
Survived           0
Pclass             0
Name               0
Sex                0
Age              177
SibSp              0
Parch              0
Ticket             0
Fare               0
Cabin            687
Embarked           2
New_col            0
diff_age         177
stage_of_life      0
dtype: int64
```

In [57]:
```
df.isnull().sum(axis =0)
```

Out[57]:
```
PassengerId        0
Survived           0
Pclass             0
Name               0
Sex                0
Age              177
SibSp              0
Parch              0
Ticket             0
Fare               0
Cabin            687
Embarked           2
```

```
New_col            0
diff_age         177
stage_of_life      0
dtype: int64
```

In [58]:
```
df.isnull().sum(axis =1)
```

Out[58]:
```
0      1
1      0
2      1
3      0
4      1
      ..
886    1
887    0
888    3
889    0
890    1
Length: 891, dtype: int64
```

Let's analyse the data

In [ ]:
```

```