pandas Dataframes - Slicing and Filtering

lesson_2_2_2

We will use the same dataframe as last lesson.

Import packages

```
import pandas as pd
```

Creating a Basic Dataframe From JSON

```
# define the data as a list
data = [
    ("Dexter","Johnsons","dog","shiba inu","red
sesame",1.5,35,"m",False,"both",True),
    ("Alfred","Johnsons","cat","mix","tuxedo",4,12,"m",True,"indoor",True),
    ("Petra","Smith","cat","ragdoll","calico",None,10,"f",False,"both",True),
    ("Ava","Smith","dog","mix","blk/wht",12,32,"f",True,"both",False),
    ("Schroder","Brown","cat","mix","orange",13,15,"m",False,"indoor",True),
    ("Blackbeard","Brown","bird","parrot","multi",5,3,"f",False,"indoor",),
]

# define the labels
labels =
["name","owner","type","breed","color","age","weight","gender","health
issues","indoor/outboor","vaccinated"]

# create dataframe
vet_records = pd.DataFrame.from_records(data, columns=labels)
```

A Note of Caution

Changes and updates to a dataframe is only permanent if saved to the dataframe. So for example we might say `vet_records = ...` to permanently change the dataframe `vet_records`. In many cases keeping a reference dataframe is a good practice. For example, `vet_records_dogs = vet_records[vet_records.type=="dog"]` instead of `vet_records = vet_records[vet_records.type=="dog"]`. This will leave you with a dataframe to reference that contains the unaldulterated data.

Grouping and Counting Data

Using counting and grouping can help you get a better grasp of the data.

```
# How many types of pets do we have?
vet_records.type.count()
```

```
vet_records.groupby('type').count()
```

```
vet_records.type.value_counts()
```

Slicing (Filtering) Data

Slicing data, that is, picking parts of teh data you want to use for a specific purpose is easy with pandas once you have the conpcets down.

Here we slice the data to get only the weight column.

```
# Create a pandas series from the dataframe
weight = vet_records['weight']
```

```
weight
```

Notice that vet_records was not changed

```
vet_records.head()
```

While `weight` does show us all the weights for the animals in the dataframe, unless we are interested in straight weight values for some calculation, it is not very useful data. A list of numbers by themselves is usually not data that can be used.

So, instead let's get all the dog weights.

```
# Collect the dog weights only using a boolean filter
dog_weight = vet_records.weight[vet_records.type=='dog']
```

```
dog_weight
```

While this still only is a list of values, at least by the variable name we know these are the weights of all the dogs in the sample.

A better way might be to just slice all the dog data.

```
dogs = vet_records[vet_records.type=='dog']
```

```
dogs
```

Using `loc` and `iloc`

- `loc` allows you to use column names to slice data
- `iloc` requires the use of index numbers. Example: `.iloc[row, column]`. Remember: python indexes starting at 0.

```
# get the pet name and owner for the 2nd record in the dataframe
vet_records.loc[1,["name", "owner"]]
```

```
# get the pet name and owner for all pets in the dataframe
vet_records.loc[:,["name", "owner"]]
```

```
# get all the names of the pets using iloc
vet_records.iloc[:,0]
```

```
# get the name Petra
vet_records.iloc[2,0]
```

```
# get the color and age of the 3rd and 4th pet, notice these are not
contiguous
vet_records.iloc[[2,3],[4,5]]
```

`.isin` can be used to gather data about a list of items

Collect the data for Dexter and Blackbeard

```
vet_records[vet_records.name.isin(['Dexter','Blackbeard'])]
```

~ can be used as a *not* logical operator.

Here we ask for all pets ***not*** named Dexter or Blackbeard

```
vet_records[~vet_records.name.isin(['Dexter','Blackbeard'])]
```

Boolean Masks

There are times when a boolean mask will be useful to you. They are similar to filtereing by booleans, but involve using `mask` file. The `mask` name is what I choose to call them they can be named anything you like.

Create a mask for male pets.

```
mask = vet_records.gender=='m'
```

Notice this is a series of `True` and `False` where if the gender column as "m", then it was True.

```
mask
```

Applying this series as a mask results in only returning the male pets. You can also use `~` to get the female pets.

```
vet_records[mask]
```

Finally check to see that vet_records was not altered.

```
vet_records
```

None and NaN

`.isna` will create a boolean dataframe `True` where the value is `NaN` or `None`.

It is advisable to deal with NaN and None values before doing ny calculations. A NaN and None cell are ignored during calculations.

```
vet_records.isna()
```

```
vet_records_example = vet_records_example.fillna(0)
```

```
vet_records_example
```

Use `fillna` With a Values Dictionary

```
values = {"age": 12, "vaccinated": False}
```

```
vet_records.fillna(value=values)
```

Notice that `vet_records` was not changed. It would need to set equal to another variable or itself to save the changes.

```
vet_records
```

```
vet_records_na = vet_records.fillna(value=values)
```

```
vet_records_na
```