James Cross - Assignment 02 Task 1

Pandas Data Structures

Data can be read by NumPy in multiple different formats, in this example it is a csv file.

This is a way to display the dimensions of the data, displayed as a tuple. A tuple is immutable, meaning it can not be changed.

Data can be read by NumPy in multiple different formats, in this example it is a csv file.
`python data.shape`

`dtype` is a way to provide the data type. Because it is a tuple, it will give the data type of each along with the name given by the csv. Numpy arrays can only be a single data type, but the way this is more of a structured array which allows multiple types.

```
data.dtype
```

This will measure the time it takes to execute finding the maximum value of the fourth entry of each row in data, in this case the ‘mag’ value. `python %%timeit max([row[3] for row in data])`

This creates a dictionary ({key: value}) of the name of the data along with the value

```
array_dict = {
    col: np.array([row[i] for row in data])
    for i, col in enumerate(data.dtype.names)
}
array_dict
```

Calculate the time to calculate the same as the previous nax of mag. This results in a quicker result on average.

```
%%timeit
array_dict['mag'].max()
```

This creates an array from the dict created earlier of just the maximum mag value entry.

```
np.array([
    value[array_dict['mag'].argmax()]
    for key, value in array_dict.items()
])
```

Create a series named "place"

```
import pandas as pd

place = pd.Series(array_dict['place'], name='place')
place
```

Give the name of the series, 'place'

```
place.name
```

Give the data type of place, 'O' (for object)

```
place.dtype
```

The shape is still (5,) because the series was made on the original data strcture, but with the 'place' name

```
place.shape
```

Output the values of place series

```
place.values
```

Another way to get the number of values in the series.

```
place_index = place.index
place_index
```

This gets the array of the number of values

```
place_index.values
```

… and also the data type, which was also provided by place.values

```
place_index.dtype
```

… and the shape of the series, showing it is still (5,)

```
place_index.shape
```

Check if the values are unique

```
place_index.is_unique
```

It is possible to do math with arrays of equal size

```
np.array([1, 1, 1]) + np.array([-1, 0, 1])
```

---

since `y` is iven a custom index, `x[0]` has nothing to add to, and neither does `y[5]` so they result in `NaN`

```
numbers = np.linspace(0, 10, num=5) # makes numpy array([0, 2.5, 5, 7.5, 10])
x = pd.Series(numbers) # index is [0, 1, 2, 3, 4]
y = pd.Series(numbers, index=pd.Index([1, 2, 3, 4, 5]))
x + y
```

Create a dataframe

```
df = pd.DataFrame(array_dict)
df
```

---

list the data types of each data entry, which are a mix of objects, floats, and ints

```
df.dtypes
```

---

give the values of the entries. this only contains the values, not the names of them

```
df.values
```

---

give the names of the data points which correspond to `df.values`

```
df.columns
```

---

find the rows and columns of the dataframe

```
df.shape
```

---

dataframes can be added together, but the data type determines if they are appended or *actually* added (or subtracted, multiplied, etc)

```
df + df
```

Creating DataFrames

I'm going to merge a lot of these code snippets together and add comments to make it a bit more concise than above.

```
np.random.seed(0) # set a seed for reproducibility
# the numbers will still be random, but they will be the same random numbers each time
pd.Series(np.random.rand(5), name='random')
pd.Series(np.linspace(0, 10, num=5)).to_frame()

np.random.seed(0) # set seed so result is reproducible
pd.DataFrame(
    {
        'random': np.random.rand(5),
        'text': ['hot', 'warm', 'cool', 'cold', None],
        'truth': [np.random.choice([True, False]) for _ in range(5)]
    },
    index=pd.date_range(
        end=dt.date(2019, 4, 21),   #ending date
        freq='1D',                  #move one day at a time(backwards)
```

```python
        periods=5,                  #5 days total
        name='date'                 #name the field 'date'
    )
)

pd.DataFrame([
    {'mag': 5.2, 'place': 'California'},
    {'mag': 1.2, 'place': 'Alaska'},
    {'mag': 0.2, 'place': 'California'},
])          # a df can also be made using dictionaries
```

A dataframe can be made from many different types of inputs:

- dictionaries
- tuples
- numpy arrays

terminal commands can be ran in jupyter

```python
df = pd.read_csv(
    'https://github.com/stefmolin/'
    'Hands-On-Data-Analysis-with-Pandas-2nd-edition'
    '/blob/master/ch_02/data/earthquakes.csv?raw=True'
)

df.to_csv('output.csv', index=False)
# index is row numbers which is a bit redundant with ways that pandas lets us navigate dfs
```

Making Dataframes from API

```python
import datetime as dt
import pandas as pd
import requests

yesterday = dt.date.today() - dt.timedelta(days=1)
api = 'https://earthquake.usgs.gov/fdsnws/event/1/query'
payload = {
    'format': 'geojson',
    'starttime': yesterday - dt.timedelta(days=30),
    'endtime': yesterday
}
response = requests.get(api, params=payload)

# let's make sure the request was OK
response.status_code
```

```
earthquake_json = response.json()
earthquake_json.keys()
# APIs give data in JSON format, which varies from source, so looking at the keys helps to f

earthquake_json['metadata']
# Includes variables like the time it was pulled (in epoch) and access urls

earthquake_json['features'][0]
# First entry in the API, pulling 'features' which contains the data we care most about

earthquake_properties_data = [
    quake['properties'] for quake in earthquake_json['features']
]
df = pd.DataFrame(earthquake_properties_data)
df.head()
# JSON is cool and all, but DFs help make it easier to read.
# df.head just picks the first 5, though we have access to all of the data. For large data s
```

Inspecting Dataframe

```
import numpy as np
import pandas as pd

df = pd.read_csv('data/earthquakes.csv')
# check if the df has contents
df.empty
# check the shape of the df
df.shape
# gather the column names of the df
df.columns
# show the first 5 rows of the df
df.head()
# show the last 2 rows of the df
df.tail(2)
# show the data types of each column
df.dtypes
# give a bit more info on the columns
df.info()
# stats on the int columns of the df
df.describe()
# can sort it by percentiles
df.describe(percentiles=[0.05, 0.95])
# get the same/similar statistics on object datatypes
df.describe(include=np.object)
```

```python
# or include all data types
df.describe(include='all')
# df.[column_name].describe gives describe output for that column
df.felt.describe()
# ... which works the same for other functions. find unique values in 'felt'
df.alert.unique()
# count the number of each unique values in alert column
df.alert.value_counts()
```

Subsetting Data

```python
import pandas as pd

df = pd.read_csv('data/earthquakes.csv')

# Access the 'mag' column of the df
df.mag
df['mag']

# Access both 'mag' and 'title'
df[['mag', 'title']]

# Access title, time and all columns starting with 'mag' using list comps
df[
    ['title', 'time']
    + [col for col in df.columns if col.startswith('mag')]
]

# list all columns starting with 'mag'
[col for col in df.columns if col.startswith('mag')]

# add 'title' and 'time' columns to 'mag'-starting columns
['title', 'time'] + [col for col in df.columns if col.startswith('mag')]

# slice the df to show index 100-102
df[100:103]
# show the title and time columns of the slice
df[['title', 'time']][100:103]

# compare the title and time of the two. this is true because its comparing the same entries
df[100:103][['title', 'time']].equals(
    df[['title', 'time']][100:103]
)
```

```python
# attempt to change values to lowercase (non-persistant)
df[110:113]['title'] = df[110:113]['title'].str.lower()
df[110:113]['title']

# change the values to lowercase (persistant) using suggested method
df.loc[110:112, 'title'] = df.loc[110:112, 'title'].str.lower()
df.loc[110:112, 'title']

# other method of indexing. first shows everything under title
# second shows slices row 10-14, column title and mag
df.loc[:,'title']
df.loc[10:15, ['title', 'mag']]

# get rows 10-14 and columns 19 and 8 (title and mag)
df.iloc[10:15, [19, 8]]
# get rows 10-14 and columns 6 through 9
df.iloc[10:15, 6:10]

# compare the output of using loc and iloc to see if results are equal
df.iloc[10:15, 6:10].equals(
    df.loc[10:14, 'gap':'magType']
)

# get the value in column 'mag' for the row with label 10 (label search)
df.at[10, 'mag']
# get the value in column at index 8 for the index 10 (index search)
df.iat[10, 8]

# returns whether the result has a mag greater than 2. searches everything
df.mag > 2

# get the rows where the mag is greater than 7.0
df[df.mag >= 7.0]

# get the specific columns where mag is greater than 7.0
df.loc[
    df.mag >= 7.0,
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]

# get the specific columns where tsnumai is 1 and alert is red
df.loc[
    (df.tsunami == 1) & (df.alert == 'red'),
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]
```

```python
# get he same columns, but using OR instead of AND
df.loc[
    (df.tsunami == 1) | (df.alert == 'red'),
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]

# get Alaska earthquakes with alerts not being null
df.loc[
    (df.place.str.contains('Alaska')) & (df.alert.notnull()),
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]

# use regex to get quakes in california with a mag > 3.8
df.loc[
    (df.place.str.contains(r'CA|California$')) & (df.mag > 3.8),
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]

# return rows with a mag between 6.5 and 7.5
df.loc[
    df.mag.between(6.5, 7.5),
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]

# return rows with a magType that is in the list ['mw', 'mwb']
df.loc[
    df.magType.isin(['mw', 'mwb']),
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]

# returns index with the highest and lowest mag value
[df.mag.idxmin(), df.mag.idxmax()]

# returns the columns of the highest and lowest mag value
df.loc[
    [df.mag.idxmin(), df.mag.idxmax()],
    ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']
]

# filters df to show mag and magType, returns first 5 rows
df.filter(items=['mag', 'magType']).head()

# filters to show columns with 'mag', returns first 5 rows
df.filter(like='mag').head()

# filters columns starting with 't' and returns first 5 rows
```

```
df.filter(regex=r'^t').head()

# sets 'place' as the index. filters for 'Japan'. filters for mentioned columns. returns fi
df.set_index('place').filter(like='Japan', axis=0).filter(items=['mag', 'magType', 'title'])

# set 'place' as index. filters titles for containing 'Japan'.
# returns first 5 rows
df.set_index('place').title.filter(like='Japan').head()
```

Adding and Removing Data

```
import pandas as pd

# read the csv and only get the columns mentioned
df = pd.read_csv(
    'data/earthquakes.csv',
    usecols=['time', 'title', 'place', 'magType', 'mag', 'alert', 'tsunami']
)

# make a new column 'source' where all entries get 'USGS API'
df['source'] = 'USGS API'
df.head()

# make new column. boolean, if mag is < 0 then bool is true
df['mag_negative'] = df.mag < 0
df.head()

# in place, get info after the column (typically state or country).
# show only uniques
df.place.str.extract(r', (.*$)')[0].sort_values().unique()

# create new columns for quakes in california and alaska
# pick a random sample (with seed)
df.assign(
    in_ca=df.parsed_place.str.endswith('California'),
    in_alaska=df.parsed_place.str.endswith('Alaska')
).sample(5, random_state=0)

# new column for neither in cali or alaska
df.assign(
    in_ca=df.parsed_place == 'California',
    in_alaska=df.parsed_place == 'Alaska',
    neither=lambda x: ~x.in_ca & ~x.in_alaska
).sample(5, random_state=0)
```

```python
# create subsets of tsnumai and no_tsunami quakes. return shape of each
tsunami = df[df.tsunami == 1]
no_tsunami = df[df.tsunami == 0]
tsunami.shape, no_tsunami.shape

# delete the source column
del df['source']
df.columns

# try to delete source column, unless there is a KeyError
try:
    del df['source']
except KeyError:
    # handle the error here
    print('not there anymore')

# remove mag_negative column, but creates a list of its values
mag_negative = df.pop('mag_negative')
df.columns

# it can still be filtered
df[mag_negative].head()

# remove the first two rules and display the first 2 rows after
df.drop([0, 1]).head(2)

# create list of cols not in list and removes from df
cols_to_drop = [
    col for col in df.columns
    if col not in ['alert', 'mag', 'title', 'time', 'tsunami']
]
df.drop(columns=cols_to_drop).head()

# compares two methods of dropping the columns to see if equal (true)
df.drop(columns=cols_to_drop).equals(
    df.drop(cols_to_drop, axis=1)
)

# removes cols from the list in place so original df is changed
df.drop(columns=cols_to_drop, inplace=True)
df.head()
```