Pandas-03 Notes

Content

- **Restructuring data**
  - `pd.melt()`
  - `pd.pivot_table()`
- **Writing to a file**
- **Basic EDA (Uber usecase)**:
  - Importance of EDA
  - Basic info of data:
    - `df.shape`
    - `df.info`
    - `df.head`
  - Handling datetime

```
0import pandas as pd
```

Restructuring data

```
!gdown 1wG8GaMi1hhfY6DsUPFjFtDYkumA-m2Xj
```

```
Downloading...
From: https://drive.google.com/uc?id=1wG8GaMi1hhfY6DsUPFjFtDYkumA-m2Xj
To: /content/weather.csv
100% 12.0k/12.0k [00:00<00:00, 16.8MB/s]
```

```
xweather = pd.read_csv('weather.csv')
weather.head()
```

	year	month	element	day1	day2	day3	day4	day5	da
0	2018	1	max	17.573016	19.796815	22.412495	17.813163	20.165825	17.0605
1	2018	1	min	22.725760	21.007865	17.730792	18.045290	20.766734	18.6566

Shape of this dataset

```
weather.shape
```

```
(22, 34)
```

Takeaway:

- **Usually** no. of rows are more than no. of columns; But in this case, **no. of columns > no. of rows**
- The **columns are** `day1, day2, day3, ...` **so on**
- `max` and `min` **value of each day** of the month is **in a separate row**

Restructuring DataFrame to columns [`year, month, day, min, max`]

- `pd.melt()`

```
pd.melt()
```

Refer: https://pandas.pydata.org/docs/reference/api/pandas.melt.html

- Pass in the **DataFrame**
- Pass in the **column names that we DON'T want to change**
- Pass in the **new column name** that will store **labels of columns to melt**
  - From `day1, day2` ... through to `day31` **into a single column** `day`
- Finally pass in **new column name** that will have **values from columns we are melting**
  - Let's call it `temp`

```
weather_melt = pd.melt(weather,
                       id_vars=['year', 'month', 'element'],
                       var_name="day",
                       value_name="temp")

# We'll store the result in a new variable

weather_melt
```

	year	month	element	day	temp
0	2018	1	max	day1	17.573016
1	2018	1	min	day1	22.725760
2	2018	2	max	day1	19.015120
3	2018	2	min	day1	18.653843
4	2018	3	max	day1	20.741115
...	...	...	...	...	...
677	2018	10	min	day31	21.691537
678	2018	11	max	day31	20.750438
679	2018	11	min	day31	18.939767
680	2018	12	max	day31	19.648924
681	2018	12	min	day31	18.775539

Notice:

- It's **using 2 rows for a single day**
  - One for `min temp` and one for `max temp`

We want to **split one single column into multiple columns**

```
year | month | day | min_temp | max_temp
```

- `pivot_table()`

Refer: https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html

- Pass in the **column names that we DON'T want to change**
- Pass in the **column name that we want to split**
  - **Labels of new columns will come from the column we want to split**
- Pass in the **column from which we will get our values**

```
weather_tidy = weather_melt.pivot(index=['year', 'month', 'day'],
                                  columns = 'element',
                                  values='temp')

weather_tidy
```

		element	max	min
year	month	day		
2018	1	day1	17.573016	22.725760
		day10	19.067288	19.931129
		day11	19.361002	22.598325
		day12	20.982134	17.715137
		day13	21.668005	17.940334
	...	...	...	...
	12	day5	21.375349	20.865535
		day6	17.992885	20.310116
		dav7	19.683359	20.531823

Takeaway:

- Notice that a multi-index df has been created.
- We can change this using `reset_index()`

```
weather_tidy = weather_tidy.reset_index()
weather_tidy
```

	element	year	month	day	max	min
0	2018	1	day1	17.573016	22.725760
1	2018	1	day10	19.067288	19.931129
2	2018	1	day11	19.361002	22.598325
3	2018	1	day12	20.982134	17.715137
4	2018	1	day13	21.668005	17.940334
...	...	...	...	...	...
336	2018	12	day5	21.375349	20.865535
337	2018	12	day6	17.992885	20.310116
338	2018	12	day7	19.683359	20.531823
339	2018	12	day8	20.477046	19.310346
340	2018	12	day9	20.210640	22.820992

341 rows × 5 columns

Writing to file

```
weather_tidy.to_csv('weather_tidy.csv', sep=",")
```

Exploratory Data Analysis - Uber Drives Data

- It include performing operations on data like:
  - **Cleaning** the data
  - Dealing with **missing values**
  - **Adding** a few **columns** that might be helpful
  - **Restructuring** the data etc..

```
!gdown 1TL2hWkMWtD1ExVgaQhWP6A2swR8F8cVB
```

```
Downloading...
From: https://drive.google.com/uc?id=1TL2hWkMWtD1ExVgaQhWP6A2swR8F8cVB
To: H:\Scaler work\dsml-course\07-08-09-Pandas\UberDrives.csv

  0%|          | 0.00/86.4k [00:00<?, ?B/s]
100%|##########| 86.4k/86.4k [00:00<00:00, 8.50MB/s]
```

```
data = pd.read_csv('UberDrives.csv')
data
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
0	1/1/2016 21:11	1/1/2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	1/2/2016 1:25	1/2/2016 1:37	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	1/2/2016 20:25	1/2/2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	1/5/2016 17:31	1/5/2016 17:45	Business	Fort Pierce	Fort Pierce	4.7	Meeting
4	1/6/2016 14:42	1/6/2016 15:49	Business	Fort Pierce	West Palm Beach	63.7	Customer Visit
...	...	...	...	...	...	...	...
1151	12/31/2016 13:24	12/31/2016 13:42	Business	Kar?chi	Unknown Location	3.9	Temporary Site
	12/31/2016	12/31/2016		Unknown	Unknown		

```
data.shape
```

```
(1156, 7)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1156 entries, 0 to 1155
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   START_DATE*  1156 non-null   object 
 1   END_DATE*    1155 non-null   object 
 2   CATEGORY*    1155 non-null   object 
 3   START*       1155 non-null   object 
 4   STOP*        1155 non-null   object 
 5   MILES*       1156 non-null   float64
 6   PURPOSE*     653 non-null    object 
dtypes: float64(1), object(6)
memory usage: 63.3+ KB
```

```
data.head()
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
0	1/1/2016 21:11	1/1/2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	1/2/2016 1:25	1/2/2016 1:37	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	1/2/2016 20:25	1/2/2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	1/5/2016	1/5/2016	[illegible]	Fort	[illegible]	[illegible]	[illegible]

```
data.tail()
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
1151	12/31/2016 13:24	12/31/2016 13:42	Business	Kar?chi	Unknown Location	3.9	Temporary Site
1152	12/31/2016 15:03	12/31/2016 15:38	Business	Unknown Location	Unknown Location	16.2	Meeting
1153	12/31/2016 21:32	12/31/2016 21:50	Business	Katunayake	Gampaha	6.4	Temporary Site
1154	12/31/2016	12/31/2016	[illegible]	[illegible]	[illegible]	[illegible]	Temporary

Dropping last row as it is NOT actual data about particular trip. It only exists for Miles

```
data.drop(1155, inplace=True)
```

```
data.tail()
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
1150	12/31/2016 1:07	12/31/2016 1:14	Business	Kar?chi	Kar?chi	0.7	Meeting
1151	12/31/2016 13:24	12/31/2016 13:42	Business	Kar?chi	Unknown Location	3.9	Temporary Site

Is there any column having missing values?

```
data.isnull().sum()
```

```
START_DATE*      0
END_DATE*        0
CATEGORY*        0
START*           0
STOP*            0
MILES*           0
PURPOSE*       502
dtype: int64
```

Are there any duplicates values?

```
data[data.duplicated()]
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
492	6/28/2016 23:34	6/28/2016 23:59	Business	Durham	Cary	9.9	Meeting

Drop Duplicates:

```
data.drop_duplicates(inplace=True)
```

Notice each column label has a `*` at the end?

Can we do something to remove this?

- List comprehension

```
data.columns = [x[:-1] for x in data.columns]
```

```
data.head()
```

	START_DATE	END_DATE	CATEGORY	START	STOP	MILES	PURPOSE
0	1/1/2016 21:11	1/1/2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain

Handling Date-Time data

- `to_datetime()`
- It takes as input:
  - Array/Scalars with values having proper date/time format
  - `dayfirst` : Indicating if the day comes first in the date format used
  - `yearfirst` : Indicates if year comes first in the date format

Refer: https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html

```
data['START_DATE'] = pd.to_datetime(data['START_DATE']) # will leave to explore how you ca

data
```

	START_DATE	END_DATE	CATEGORY	START	STOP	MILES	PURPOSE
0	2016-01-01 21:11:00	1/1/2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	2016-01-02 01:25:00	1/2/2016 1:37	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	2016-01-02 20:25:00	1/2/2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	2016-01-05 17:31:00	1/5/2016 17:45	Business	Fort Pierce	Fort Pierce	4.7	Meeting
4	2016-01-06 14:42:00	1/6/2016 15:49	Business	Fort Pierce	West Palm Beach	63.7	Customer Visit
...	...	...	...	...	...	...	...
1150	2016-12-31 01:07:00	12/31/2016 1:14	Business	Kar?chi	Kar?chi	0.7	Meeting
1151	2016-12-31 13:24:00	12/31/2016 13:42	Business	Kar?chi	Unknown Location	3.9	Temporary Site

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1154 entries, 0 to 1154
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   START_DATE  1154 non-null   datetime64[ns]
 1   END_DATE    1154 non-null   object
```

```
 2   CATEGORY    1154 non-null   object
 3   START       1154 non-null   object
 4   STOP        1154 non-null   object
 5   MILES       1154 non-null   float64
 6   PURPOSE     652 non-null    object
dtypes: datetime64[ns](1), float64(1), object(5)
memory usage: 72.1+ KB
```

```
data['END_DATE'] = pd.to_datetime(data['END_DATE'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1154 entries, 0 to 1154
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   START_DATE  1154 non-null   datetime64[ns]
 1   END_DATE    1154 non-null   datetime64[ns]
 2   CATEGORY    1154 non-null   object
 3   START       1154 non-null   object
 4   STOP        1154 non-null   object
 5   MILES       1154 non-null   float64
 6   PURPOSE     652 non-null    object
dtypes: datetime64[ns](2), float64(1), object(4)
memory usage: 72.1+ KB
```

Cross Check for rows where start timestamp and end timestamp are same ?

Because having exactly zero miles is not really possible

```
data[data['START_DATE']==data['END_DATE']]
```

	START_DATE	END_DATE	CATEGORY	START	STOP	MILES	PURPOSE
751	2016-09-06 17:49:00	2016-09-06 17:49:00	Business	Unknown Location	Unknown Location	69.1	NaN
761	2016-09-16 07:08:00	2016-09-16 07:08:00	Business	Unknown Location	Unknown Location	1.6	NaN
798	2016-10-08 15:03:00	2016-10-08 15:03:00	Business	Karachi	Karachi	3.6	NaN

```
data.drop(data.index[[751,761,798,807]],inplace=True)
data
```

	START_DATE	END_DATE	CATEGORY	START	STOP	MILES	PURPOSE
0	2016-01-01 21:11:00	2016-01-01 21:17:00	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	2016-01-02 01:25:00	2016-01-02 01:37:00	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	2016-01-02 20:25:00	2016-01-02 20:38:00	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	2016-01-05 17:31:00	2016-01-05 17:45:00	Business	Fort Pierce	Fort Pierce	4.7	Meeting
	2016-01-06	2016-01-06			West Palm		

```
ts = data['END_DATE'][0]
ts
```

```
Timestamp('2016-01-01 21:17:00')
```

Now how can we extract the year from this date ?

- Using the `year` attribute

```
ts.year
```

```
2016
```

Similarly we can also access the month and day using the `month` and `day` attributes

```
ts.month
```

```
1
```

```
ts.day
```

```
1
```

But what if we want to know the name of the month or the day of the week on that date ?

- using `month_name()` and `day_name()` methods

```
ts.month_name()
```

```
'January'
```

```
ts.day_name()
```

```
'Friday'
```

```
ts.dayofweek
```

```
4
```

```
ts.hour
```

```
21
```

- We can similarly extract minutes and seconds

Let's add a new column having only `year`

- `.dt`

```
data['END_DATE'].dt
```

```
<pandas.core.indexes.accessors.DatetimeProperties object at 0x000002210B969278>
```

- **`dt` gives properties of values in a column**

```
data['END_DATE'].dt.year
```

```
0       2016
1       2016
2       2016
3       2016
4       2016
        ...
1150    2016
1151    2016
1152    2016
1153    2016
1154    2016
Name: END_DATE, Length: 1150, dtype: int64
```

```
data['year'] = data['END_DATE'].dt.year
data.head()
```

	START_DATE	END_DATE	CATEGORY	START	STOP	MILES	PURPOSE	year
0	2016-01-01 21:11:00	2016-01-01 21:17:00	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain	2016
1	2016-01-02 01:25:00	2016-01-02 01:37:00	Business	Fort Pierce	Fort Pierce	5.0	NaN	2016
2	2016-01-02 20:25:00	2016-01-02 20:38:00	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies	2016
	2016-01-05	2016-01-05		Fort	Fort			

Lets try to answer the following questions:

- **What is the shortest journey made?**
- **What is the longest journey made?**
- **What is the average journey made?**
- **How many years of data do we have?**

```
data.describe()
```

	MILES	year
count	1150.000000	1150.0
mean	10.584609	2016.0
std	21.623241	0.0
min	0.500000	2016.0
25%	2.900000	2016.0
50%	6.000000	2016.0
75%	10.400000	2016.0
max	310.300000	2016.0

So We can answer all the questions by looking at it:

- **What is the shortest journey made?** - 0.5 miles
- **What is the longest journey made?** - 310.3 miles
- **What is the average journey made?** - mean value
- **How many years of data do we have?** - Just 1 year - 2016