Basic

ขนาดของจุดแปรผัน = mean

Using Helper function

```
help(pd.melt)
help(pd.pivot_table)
help(pd.merge)
help(df.iloc)

//df helper need to have df first (any)
df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
help(df.loc)
help(df.join)
```

Copy dataframe

```
medals_without_stack = medals.copy()
```

Count Rows, Cols

```
df.shape[0], df.shape[1]
axis=0 is row, axis=1 is column
```

drop column

```
x = x.drop('Date', axis=1)
```

Set index and drop

```
meet_df = meet_df.set_index(meet_df.MeetID).drop('MeetID', axis=1)
```

Set index with previous index (Add index)

```
y = x.set_index(x.TimeStamp, append=True) <- assume that x has "MeetID" as its previous index
y will now have 2 indicies
```

Replace the index and removes the prev back to col

```
y = x.reset_index()
y = y.set_index(x.TimeStamp)
```

Count NaN

```
df.yourCols.isna().sum()
```

Count Not NaN

```
df.yourCols.notna().sum()
```

loc, iloc

```
df.loc['rows condition' , 'cols condition']
df.loc[(df['a'] > df['b']) & (df['c'] < df['d']), ['a', 'b', 'c']]
df.loc[df['a'] == df.a.max() , :]
medals.loc[:, medals.columns.str.startswith("Summer")]
medals.loc[ ["USA"], :]
medals.loc[["THA", "SIN", "MAS"], medals.columns.str.contains("Gold") | (medals.columns == "Country")]
medals[ (medals.loc[:, medals.columns.str.contains("Summer")].sum(axis=1) > 200) &
           (medals.loc[:, medals.columns.str.contains("Silver")].sum(axis=1) > 200) ]
```

count contains

```
df.yourCols.str.contains('xxxxxx').sum()
dfN = air_crash.loc[air_crash.Location.str.contains('Thailand').fillna(False), : ]
```

count2

```
medals_long.groupby('continent')['Country'].size()
```

isin

```
medals.loc[~medals["Country"].isin(drinks["country"]), "Country"]
(loc country name in medals that is not in drink's country)
```

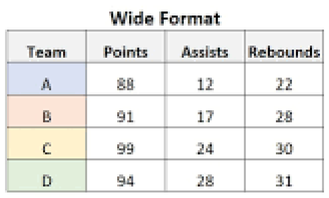
sorting

```
air_crash.sort_values(['Fatalities Percent','Aboard'], ascending=[False, False])
df.sort_index()
```

string splitting into columns

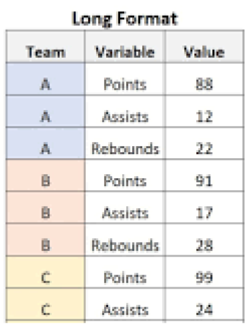
```
df['MeetAddress'] = df['MeetPath'].str.split('/').str.get(0)
df['MeetAddressRoad'] = df['MeetPath'].str.split('/').str.get(1)
```

Reshaping



Wide Format

Team	Points	Assists	Rebounds
A	88	12	22
B	91	17	28
C	99	24	30
D	94	28	31



Long Format

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24

Wide to Long

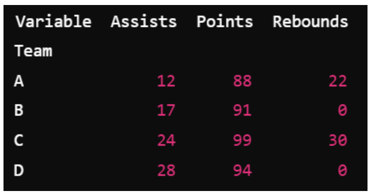
```
df_long = pd.melt(df_wide, id_vars=["Team"], var_name="Variable", value_name="Value")
```

Long to Wide

```
df_wide = df_long.pivot(index="Team", columns="Variable", values="Value")
```

Some aggfunc example

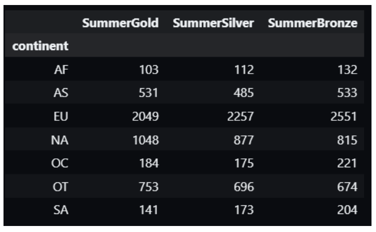
```
X = df_long.pivot_table(index="Team", columns="Variable", values="Value", aggfunc="sum", fill_value=0)
```



Variable	Assists	Points	Rebounds
Team			
A	12	88	22
B	17	91	0
C	24	99	30
D	28	94	0

doing mean, sum in some interested data groupby

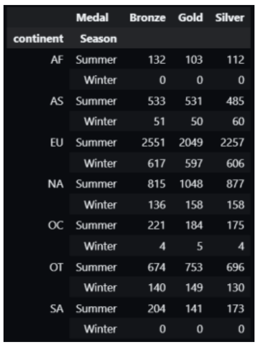
```
average_gold_per_country = summer_gold_data.groupby('continent')['Count'].mean()
average_gold_per_country

u  = medals_without_index.groupby('continent')[['SummerGold', 'SummerSilver', 'SummerBronze']].sum()
u
```



	SummerGold	SummerSilver	SummerBronze
continent			
AF	103	112	132
AS	531	485	533
EU	2049	2257	2551
NA	1048	877	815
OC	184	175	221
OT	753	696	674
SA	141	173	204

Multiple groupby

```
p = medals_long.groupby(['continent', 'Season', 'Medal'])['Count'].sum().unstack(level='Medal').fillna(0)
p
```



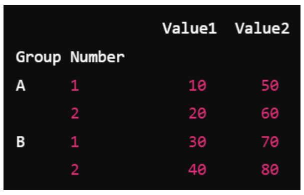
	Medal	Bronze	Gold	Silver
continent	Season			
AF	Summer	132	103	112
	Winter	0	0	0
AS	Summer	533	531	485
	Winter	51	50	60
EU	Summer	2551	2049	2257
	Winter	617	597	606
NA	Summer	815	1048	877
	Winter	136	158	158
OC	Summer	221	184	175
	Winter	4	5	4
OT	Summer	674	753	696
	Winter	140	149	130
SA	Summer	204	141	173
	Winter	0	0	0

Multiple index creation from pd tuples

```
x = pd.MultiIndex.from_tuples([("Summer", "SummerGame"),
                               ("Summer", "SummerGold"),
                               ("Summer", "SummerSilver"),
                               ("Summer", "SummerBronze"),
                               ("Winter", "WinterGame"),
                               ("Winter", "WinterGold"),
                               ("Winter", "WinterSilver"),
                               ("Winter", "WinterBronze")])
medals.columns = x
a
```

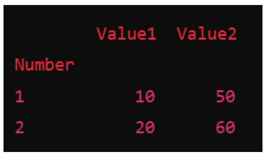
or

```
df = pd.DataFrame({
"Group": ["A", "A", "B", "B"],
"Number": [1, 2, 1, 2],
"Value1": [10, 20, 30, 40],
"Value2": [50, 60, 70, 80]
})
df = df.set_index(["Group", "Number"])
```



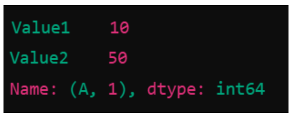
```
              Value1  Value2
Group Number
A     1           10      50
      2           20      60
B     1           30      70
      2           40      80
```

```
df.loc[‘A’]
```



```
        Value1  Value2
Number
1           10      50
2           20      60
```

```
df.loc[("A", 1)]
```

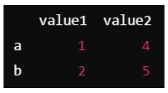


```
Value1    10
Value2    50
Name: (A, 1), dtype: int64
```

joining

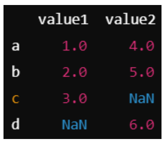
```
df1 = pd.DataFrame({'value1': [1, 2, 3]}, index=['a', 'b', 'c'])
df2 = pd.DataFrame({'value2': [4, 5, 6]}, index=['a', 'b', 'd'])
result = df1.join(df2, how='???')
```

inner -> Keeps only the rows that are common in both DataFrames.



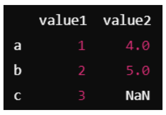
```
   value1  value2
a       1       4
b       2       5
```

outer -> Keeps all, but fill missing value with NaN



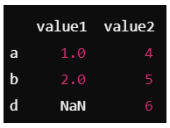
```
   value1  value2
a     1.0     4.0
b     2.0     5.0
c     3.0     NaN
d     NaN     6.0
```

left -> Keeps all rows from the left DataFrame, fill missing value with NaN



```
   value1  value2
a       1     4.0
b       2     5.0
c       3     NaN
```

right -> Keeps all rows from the right DataFrame, fill missing value with NaN



```
   value1  value2
a     1.0       4
b     2.0       5
d     NaN       6
```

Concat

```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
result = pd.concat([df1, df2]) <- Concat along the rows
```

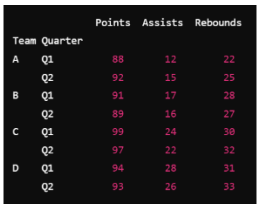
```
   A  B
0  1  3
1  2  4
0  5  7
1  6  8
```

```
pd.concat([df1, df2], axis=1) <- along cols
```

```
   A  B  A  B
0  1  3  5  7
1  2  4  6  8
```

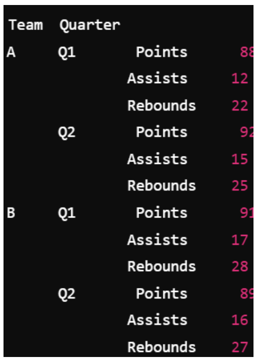
Stack & Unstack Examples

```
Default dataframe: df_multi
```



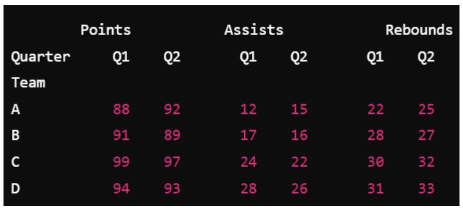
```
              Points  Assists  Rebounds
Team Quarter
A    Q1           88       12        22
     Q2           92       15        25
B    Q1           91       17        28
     Q2           89       16        27
C    Q1           99       24        30
     Q2           97       22        32
D    Q1           94       28        31
     Q2           93       26        33
```

```
df_stacked = df_multi.stack()
```



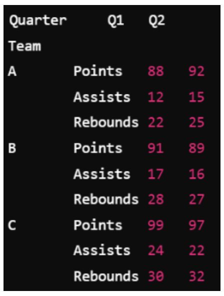
```
Team  Quarter
A     Q1        Points    8
                Assists   12
                Rebounds  22
      Q2        Points    9
                Assists   15
                Rebounds  25
B     Q1        Points    9
                Assists   17
                Rebounds  28
      Q2        Points    8
                Assists   16
                Rebounds  27
```

```
df_unstacked = df_multi.unstack()
```



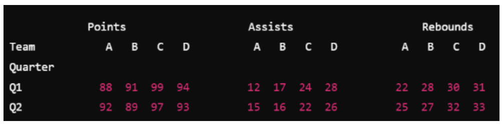
```
         Points    Assists    Rebounds
Quarter  Q1   Q2   Q1   Q2    Q1   Q2
Team
A        88   92   12   15    22   25
B        91   89   17   16    28   27
C        99   97   24   22    30   32
D        94   93   28   26    31   33
```

```
df_partial_stacked = df_multi.stack(level="Quarter")
```



```
Quarter          Q1   Q2
Team
A     Points     88   92
      Assists    12   15
      Rebounds   22   25
B     Points     91   89
      Assists    17   16
      Rebounds   28   27
C     Points     99   97
      Assists    24   22
      Rebounds   30   32
```

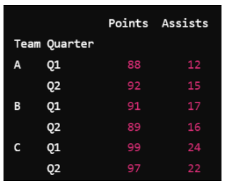
```
df_partial_unstacked = df_multi.unstack(level="Team")
```



```
         Points          Assists         Rebounds
Team     A   B   C   D   A   B   C   D   A   B   C   D
Quarter
Q1       88  91  99  94  12  17  24  28  22  28  30  31
Q2       92  89  97  93  15  16  22  26  25  27  32  33
```

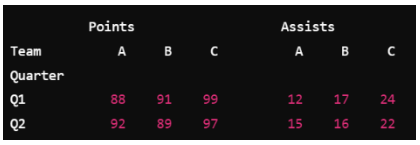
`Unstack levels level=0 -> outer, level=1 -> inner`

```
Default dataframe: df
```



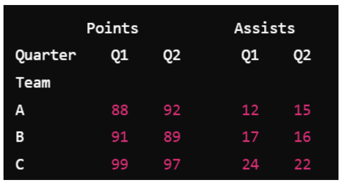
```
              Points  Assists
Team Quarter
A    Q1           88       12
     Q2           92       15
B    Q1           91       17
     Q2           89       16
C    Q1           99       24
     Q2           97       22
```

```
df.unstack(level=0)<- move "team" to cols
```



```
         Points        Assists
Team     A   B   C     A   B   C
Quarter
Q1       88  91  99    12  17  24
Q2       92  89  97    15  16  22
```

```
df_unstack_level1 = df.unstack(level=1)<- move "Quarter" to cols
```



```
         Points    Assists
Quarter  Q1   Q2   Q1   Q2
Team
A        88   92   12   15
B        91   89   17   16
C        99   97   24   22
```

Categorical

Create and Assign Categorical

```
x = pd.Categorical(df.day, categories=['Thur', 'Fri', 'Sat', 'Sun'], ordered=True)
df['day_cat'] = x
df.sort_values('day_cat') <- U can sort it!
```

Add and Remove from Categories

```
df.day_cat.cat.add_categories('Wed')
df.day_cat.cat.remove_categories('Thur') <- every Thur will became NaN after this!
```

Reorder Categories

```
df.day_cat.cat.reorder_categories(['Wed','Thur','Fri','Sat','Sun'])
df.day_cat.cat.as_unordered <- cancel the sort
```

Datetime

Convert to timestamp

```
df['Timestamp'] = pd_todatetime(df.yourDateColumn) <- replace w your date column!
```

Convert to timestamp and set as an index

```
df['Timestamp'] = pd.to_datetime(df.Time)
df.set_index('Timestamp', inplace=True) <- set as index
df.drop('Time', axis='columns', inplace=True) <- drop old time column
df.sort_index(inplace=True)
df
```

loc with timestamp

```
df.loc['2016-10-30 7:00': '2016-10-30 9:00']
```

Calculate time difference between indicies

```
time_difference = df.index[1] - df.index[0]
```

Count size in period

```
daliy_pandinwhai = df.resample('D').size().to_period() <- Count in 1 Day
daliy_pandinwhai
```

Count size in that day

```
df.loc['2016-10-30'].shape[0]
```

Find max empty gap

```
df['time_diff'] = df.index.to_series().diff() <- convert timestamp index to series
max_gap = df['time_diff'].max()
max_gap <- in case u wanna print it

end = df['time_diff'].idxmax() <- get the timestamp out
start = end - max_gap
print(f"{start} to {end}")
```

Show the differences of data between indicies

```
df['time_difference'] = df.index.to_series().diff()
print(df)
```

Count Empty Periods

```
hourly_counts = df.resample('h').size() <- Will count hourly empty period
num_empty_periods = len(hourly_counts[hourly_counts == 0]) <- len of empty period
num_empty_periods
```

Find max period and max value from some periods

```
minutely_counts = df.Magnitude.resample('5min').mean() <- e.g. find max mean every 5 minutes of Magnitude
max_count_period = minutely_counts.idxmax()
max_count_value = minutely_counts.max()
```

find average, mean, sum of value rolling window

```
daily_max_magnitude = df.resample('D')['Magnitude'].max() <- max magnitude everyday
rolling_avg_3_days = daily_max_magnitude.rolling(window=3).mean() <- mean of magnitude every 3 days
rolling_avg_3_days
```

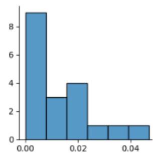
Resample with multiple indicies

```
P = y.groupby('MeetID').resample('10YE', level='TimeStamp').size()
```

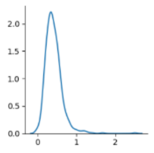
Seaborn

displot

```
sns.displot(usa_player.shots, kind='hist',height=3)
kind = 'hist' -> Histrogram
```

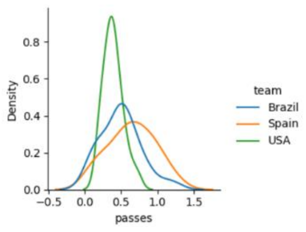


```
sns.displot(players.passes, kind='kde', rug=False , height=3) <- Rug is for Rug U know it
kind = 'kde' -> Probability density Function
```



displot among multiple datas

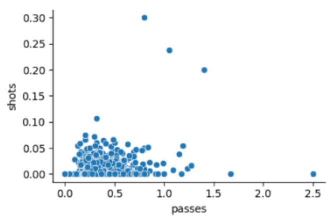
```
sns.displot(data, x='passes', kind='kde', height=3, hue='team')
```



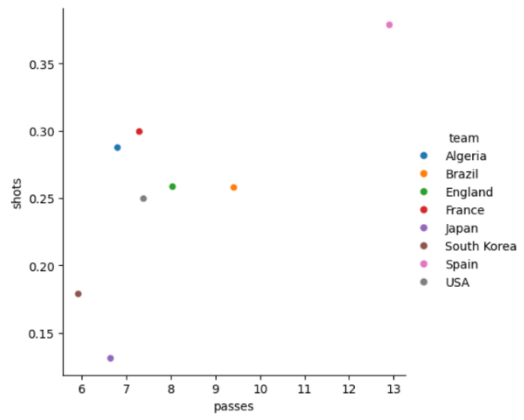
Relplot

Assume filtered_data is your filtered interested data (e.g. only usa team)
Relplot defaults as 'scatter'

```
sns.relplot(players, x='passes', y='shots', height=3, aspect=1.5)
```

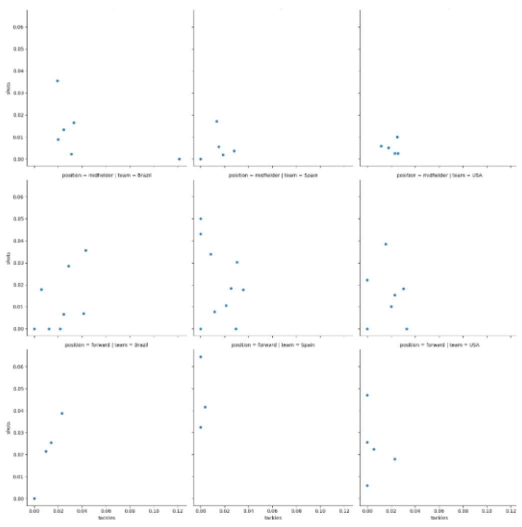


```
data_aggregated = data2.groupby('team').agg({'passes': 'sum', 'shots': 'sum'})
sns.relplot(data_aggregated, x='passes', y='shots', hue='team', kind='scatter') <- kind defaults as scatter
```



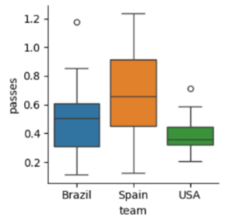
Multiple relplot

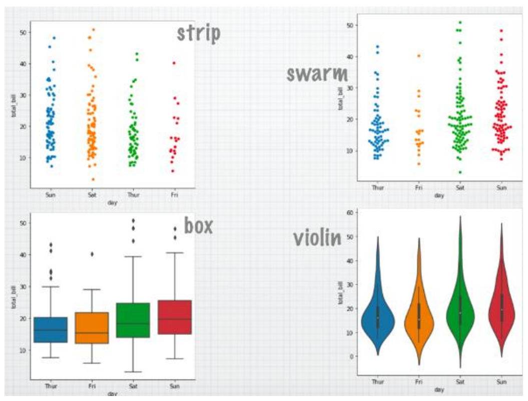
```
sns.relplot(data, x='tackles', y='shots', row='position', col='team')
```



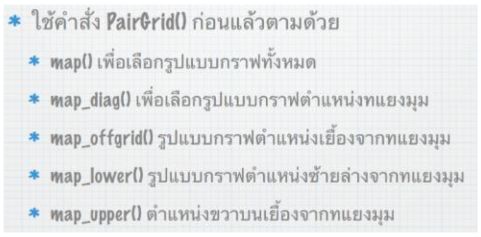
Catplot

```
sns.catplot(data, x='team', y='passes', height=3, kind='box', hue='team')
```

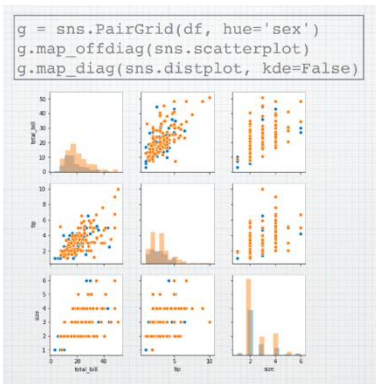




PairGrid



* ใช้คำสั่ง **PairGrid()** ก่อนแล้วตามด้วย
  * **map()** เพื่อเลือกรูปแบบกราฟทั้งหมด
  * **map_diag()** เพื่อเลือกรูปแบบกราฟตำแหน่งทแยงมุม
  * **map_offgrid()** รูปแบบกราฟตำแหน่งเยื้องจากทแยงมุม
  * **map_lower()** รูปแบบกราฟตำแหน่งซ้ายล่างจากทแยงมุม
  * **map_upper()** ตำแหน่งขวาบนเยื้องจากทแยงมุม



```
g = sns.PairGrid(df, hue='sex')
g.map_offdiag(sns.scatterplot)
g.map_diag(sns.distplot, kde=False)
```