Basic

**Copy dataframe**

medals\_without\_stack = medals.copy()

**Count Rows, Cols**

df.shape[0], df.shape[1]

**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() , :]

**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"]

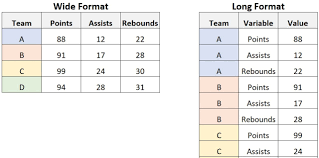
(loc country name in medals that is not in drink’s country)

**sorting**

air\_crash.sort\_values(['Fatalities Percent','Aboard'], ascending=[False, False])

df.sort\_index()

Reshaping



**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\_table(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)

A screenshot of a game

Description automatically generated

**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

A screenshot of a black screen

Description automatically generated

**Multiple groupby**

p = medals\_long.groupby(['continent', 'Season', 'Medal'])['Count'].sum().unstack(level='Medal').fillna(0)

p

A screenshot of a computer screen

Description automatically generated

**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

Stack & Unstack Examples

A black background with white text and pink numbers

Description automatically generatedDefault dataframe: df\_multi

df\_stacked = df\_multi.stack() df\_unstacked = df\_multi.unstack()

A screenshot of a computer

Description automatically generatedA screenshot of a black screen

Description automatically generated

df\_partial\_stacked = df\_multi.stack(level="Quarter") df\_partial\_unstacked = df\_multi.unstack(level="Team")

A screenshot of a computer

Description automatically generatedA black screen with white text and pink numbers

Description automatically generated

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

Default dataframe: df df.unstack(level=0)<- move “team” to cols

A black background with white and pink numbers

Description automatically generatedA black background with white text and pink numbers

Description automatically generated

df\_unstack\_level1 = df.unstack(level=1)<- move “Quarter” to cols

A black background with white text and pink numbers

Description automatically generated

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.Date) <- 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()

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.resample('min').size() <- e.g. resample every minute

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

Seaborn

displot

sns.displot(usa\_player.shots, kind='hist',height=3)

A graph with blue bars

Description automatically generatedkind = ‘hist’ -> Histrogram

sns.displot(players.passes, kind='kde', height=3)

kind = ‘kde’ -> Probability density Function

A blue line graph with numbers

Description automatically generated

**displot among multiple datas**

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

A diagram of a line graph

Description automatically generated

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)

A graph of blue dots

Description automatically generated

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

A graph with colored dots

Description automatically generated

**Multiple relplot**

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

A screenshot of a graph

Description automatically generated

Catplot

A group of graphs on a graph paper

Description automatically generatedA graph of different colored squares

Description automatically generatedsns.catplot(data, x='team', y='passes', height=3, kind='box', hue='team')