df = pd.DataFrame({‘A’: [1,2,3], ‘B’: [4,5,6], ‘C’: [7,8,9]})

add column: df[‘D’] = [10, 11, 12]

turn a column to string: df["A"] = df["A"].astype(str)

drop: df.drop('Country', axis = 1); df.drop(columns = [‘A’, ‘B’])

sort: df.sort\_values('Country'); df.sort\_values('Country', ascending = False)

df.sort\_values('Num Lakes', ascending = False).iloc[0]['Country'] find the country with most # lakes

select: rows: df[1:]; df[1:3]; df.iloc[0];

columns: df[‘A’]; df.loc[:, ‘A’ : ‘B’];

rows & columns: df.loc[1:3, ‘A’: ‘B’]; df.loc[:, ‘A’: ‘B’]

filter: df[(df['Population']> 120) & (df['Lakes']> 2)]

shape: df.shape

columns: df.columns

statistic: df.describe(); df[‘A’].count() / .sum() / .min() / .median() / .mean() / .var() / .std() / .nunique() / .quantile([0.25, 0.75, 0.99]) / .mode()[0]

fill N/A: df.dropna(); df[‘A’].fillna(df[‘A’].mean(), inplace = True);

df.groupby('sex')['age'].apply(lambda x:x.fillna(x.median())) fill na based on median for each group

concat: union / append row: pd.concat([df\_a, df\_b]); append column: pd.concat([df\_a, df\_b], axis = 1)

rename: rename(columns = {‘A’: ‘a’, ‘B’: ‘b’, ‘C’: ‘c’})

time diff between two dates: df['time\_diff'] = (df['next\_visit\_date'] - df['visit\_date']).dt.days

group by: df.groupby('A') ['C'].median().reset\_index()

median() / nunique() / count() … size()

df.groupby(['A', ‘B’]) ['C'].median().reset\_index()

aggregated\_df = df.groupby('City').agg(

average\_age = ('Age', 'mean'),

standard\_deviation = ('Age', 'std'),

total\_age = ('Age', 'sum'),

count\_age = ('Age', 'nunique')

).reset\_index()

aggregated\_df = df.groupby('City').agg(

average\_age = ('Age', lambda x: x.mean()),

standard\_deviation = ('Age', lambda x: x.std()),

total\_age = ('Age', lambda x: x.sum()),

count\_age = ('Age', lambda x: x.count()),

custom\_func = ('Age', lambda x: x.max() - x.min()) # can create custom functions

).reset\_index()

df.groupby('group').agg(

count\_approve = ('id', lambda x: x[df.loc[x.index, 'state'] == 'approve'].nunique()),

sum\_approve = ('value', lambda x: x[df.loc[x.index, 'state'] == 'approve'].sum()),

count\_reject = ('id', lambda x: x[df.loc[x.index, 'state'] == 'reject'].nunique()),

sum\_reject = ('value', lambda x: x[df.loc[x.index, 'state'] == 'reject'].sum())

).reset\_index()

merge / join: pd.merge(df, df\_2, how = 'inner', on = ‘id’)

pd.merge(df, df\_2, how = 'inner', left\_on = ‘id\_1’, right\_on = 'id\_2')

df.merge(df\_2, how = 'inner', left\_on = ‘id\_1', right\_on = 'id\_2')

lambda: df['A'] = df['A'].apply(lambda x: x + 10)

df['Age category'] = df['Age'].apply(lambda x: 'Child' if x < 18 else 'Adult' if x < 65 else 'Senior')

df['Age category 2'] = ['Child' if x < 18 else 'Adult' if x < 65 else 'Senior' for x in df['Age']]

def age\_group(x):

if x < 18:

return 'Child'

elif x < 65:

return 'Adult'

else:

return 'Senior'

df['Age category 3'] = df['Age'].apply(lambda x: age\_group(x))

dictionary: dictionary = {'spain' : 'madrid','usa' : 'vegas'}

dictionary.keys()

dictionary.values()

dictionary['france'] = "paris" add new entries

for key, value in dictionary.items():

print(key," : ",value)

map dataframe columns to dictionary:

df['Category'].map(category\_mapping).fillna('Unknown')

category\_mapping is a dictionary mapping categories to values

LAG (prev):

df = df.sort\_values(by=['Category', 'Year']) need to sort first

df['Sales\_lag'] = df.groupby('Category')['Sales'].shift(1)

df['YoY\_growth'] = (df['Sales'] - df['Sales\_lag']) / df['Sales\_lag']

df['YoY\_growth'].fillna(0, inplace = True)

LEAD (next):

df['Sales\_lead'] = df.groupby('Category')['Sales'].shift(-1)

rank: dense rank (not skip after tie): df['rank'] = df['salary'].rank(method = 'dense', ascending = False)

rank: regular rank (skip after tie): df['rank'] = df['salary'].rank(ascending = False)

row\_number: df['row\_number'] = df['salary'].rank(method='first', ascending=False)

aggregation window function:

df.groupby(‘A’)[‘B’].transform('max') max / min / mean / median / sum / nunique / count

df['max\_by\_dept'] = df.groupby('department')['salary'].transform('max')

df['avg\_by\_dept'] = df.groupby('department')['salary'].transform('mean')

df['median\_by\_dept'] = df.groupby('department')['salary'].transform('median')

df['sum\_by\_dept'] = df.groupby('department')['salary'].transform('sum')

df['count\_by\_dept'] = df.groupby('department')['name'].transform('nunique')

rolling window:

df['7\_day\_avg'] = df['value'].rolling(window = 7, min\_periods = 1).mean()