# Create the list of new indexes: new\_idx

new\_idx = [i.upper() for i in sales.index]

# Assign new\_idx to sales.index

sales.index = new\_idx

# Print the sales DataFrame

print(sales)

==

# Assign the string 'MONTHS' to sales.index.name

sales.index.name = 'MONTHS'

# Print the sales DataFrame

print(sales)

# Assign the string 'PRODUCTS' to sales.columns.name

sales.columns.name = 'PRODUCTS'

# Print the sales dataframe again

print(sales)

==

# Generate the list of months: months

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']

# Assign months to sales.index

sales.index = months

# Print the modified sales DataFrame

print(sales)

==

# Set the index to be the columns ['state', 'month']: sales

sales = sales.set\_index(['state', 'month'])

# Sort the MultiIndex: sales

sales = sales.sort\_index()

# Print the sales DataFrame

print(sales)

==

# Set the index to the column 'state': sales

sales = sales.set\_index('state')

# Print the sales DataFrame

print(sales)

# Access the data from 'NY'

print(sales.loc['NY'])

==

# Look up data for NY in month 1 in sales: NY\_month1

NY\_month1 = sales.loc[('NY', 1)]

# Look up data for CA and TX in month 2: CA\_TX\_month2

CA\_TX\_month2 = sales.loc[(['CA', 'TX'], 2),:]

# Access the inner month index and look up data for all states in month 2: all\_month2

all\_month2 = sales.loc[(slice(None), 2),:]

==

# Pivot the users DataFrame: visitors\_pivot

visitors\_pivot = users.pivot(index='weekday', columns='city', values='visitors')

# Print the pivoted DataFrame

print(visitors\_pivot)

==

# Pivot users with signups indexed by weekday and city: signups\_pivot

signups\_pivot = users.pivot(index='weekday', columns='city', values='signups')

# Print signups\_pivot

print(signups\_pivot)

# Pivot users pivoted by both signups and visitors: pivot

pivot = users.pivot(index='weekday', columns='city')

# Print the pivoted DataFrame

print(pivot)

==

# Unstack users by 'weekday': byweekday

byweekday = users.unstack(level=1)

# Print the byweekday DataFrame

print(byweekday)

# Stack byweekday by 'weekday' and print it

print(byweekday.stack(level=1))

==

# Unstack users by 'city': bycity

bycity = users.unstack(level=0)

# Print the bycity DataFrame

print(bycity)

# Stack bycity by 'city' and print it

print(bycity.stack(level=1))

# Note change of level and it returns a different ordered index that might need swapping as well as sorting

==

# Stack 'city' back into the index of bycity: newusers

newusers = bycity.stack(level='city')

# Swap the levels of the index of newusers: newusers

newusers = newusers.swaplevel(0,1)

# Print newusers and verify that the index is not sorted

print(newusers)

# Sort the index of newusers: newusers

newusers = newusers.sort\_index()

# Print newusers and verify that the index is now sorted

print(newusers)

# Verify that the new DataFrame is equal to the original

print(newusers.equals(users))

==

# Reset the index: visitors\_by\_city\_weekday

visitors\_by\_city\_weekday = visitors\_by\_city\_weekday.reset\_index()

# Print visitors\_by\_city\_weekday

print(visitors\_by\_city\_weekday)

# Melt visitors\_by\_city\_weekday: visitors

visitors = pd.melt(visitors\_by\_city\_weekday, id\_vars='weekday', value\_name='visitors')

# Print visitors

print(visitors)

==

# Melt users: skinny

skinny = pd.melt(users ,id\_vars=['weekday', 'city'],value\_vars=['visitors', 'signups'])

# Print skinny

print(skinny)

==

# Set the new index: users\_idx

users\_idx = users.set\_index(['city', 'weekday'])

# Print the users\_idx DataFrame

print(users\_idx)

# Obtain the key-value pairs: kv\_pairs

kv\_pairs = pd.melt(users\_idx, col\_level=0)

# Print the key-value pairs

print(kv\_pairs)

==

# Create the DataFrame with the appropriate pivot table: by\_city\_day

by\_city\_day = users.pivot\_table(index='weekday', columns='city')

# Print by\_city\_day

print(by\_city\_day)

==

# Use a pivot table to display the count of each column: count\_by\_weekday1

count\_by\_weekday1 = users.pivot\_table(users, index='weekday', aggfunc='count')

# Print count\_by\_weekday

print(count\_by\_weekday1)

# Replace 'aggfunc='count'' with 'aggfunc=len': count\_by\_weekday2

count\_by\_weekday2 = users.pivot\_table(users, index='weekday', aggfunc=len)

# Verify that the same result is obtained

print('==========================================')

print(count\_by\_weekday1.equals(count\_by\_weekday2))

==

# Create the DataFrame with the appropriate pivot table: signups\_and\_visitors

signups\_and\_visitors = users.pivot\_table(users, index='weekday', aggfunc=sum)

# Print signups\_and\_visitors

print(signups\_and\_visitors)

# Add in the margins: signups\_and\_visitors\_total

signups\_and\_visitors\_total = users.pivot\_table(users, index='weekday', aggfunc=sum, margins=True)

# Print signups\_and\_visitors\_total

print(signups\_and\_visitors\_total)

==

# Group titanic by 'pclass'

by\_class = titanic.groupby('pclass')

# Aggregate 'survived' column of by\_class by count

count\_by\_class = by\_class['survived'].count()

# Print count\_by\_class

print(count\_by\_class)

# Group titanic by 'embarked' and 'pclass'

by\_mult = titanic.groupby(['embarked','pclass'])

# Aggregate 'survived' column of by\_mult by count

count\_mult = by\_mult['survived'].count()

# Print count\_mult

print(count\_mult)

==

# Read life\_fname into a DataFrame: life

life = pd.read\_csv(life\_fname, index\_col='Country')

# Read regions\_fname into a DataFrame: regions

regions = pd.read\_csv(regions\_fname, index\_col='Country')

# Group life by regions['region']: life\_by\_region

life\_by\_region = life.groupby(regions['region'])

# Print the mean over the '2010' column of life\_by\_region

print(life\_by\_region['2010'].mean())

==

# Group titanic by 'pclass': by\_class

by\_class = titanic.groupby('pclass')

# Select 'age' and 'fare'

by\_class\_sub = by\_class[['age','fare']]

# Aggregate by\_class\_sub by 'max' and 'median': aggregated

aggregated = by\_class\_sub.agg(['max', 'median'])

# Print the maximum age in each class

print(aggregated.loc[:, ('age','max')])

# Print the median fare in each class

print(aggregated.loc[:, ('fare','median')])

==

# Read the CSV file into a DataFrame and sort the index: gapminder

gapminder = pd.read\_csv('gapminder.csv', index\_col=['Year', 'region', 'Country']).sort\_index()

# Group gapminder by 'Year' and 'region': by\_year\_region

by\_year\_region = gapminder.groupby(level=['Year', 'region'])

# Define the function to compute spread: spread

def spread(series):

return series.max() - series.min()

# Create the dictionary: aggregator

aggregator = {'population':'sum', 'child\_mortality':'mean', 'gdp':spread}

# Aggregate by\_year\_region using the dictionary: aggregated

aggregated = by\_year\_region.agg(aggregator)

# Print the last 6 entries of aggregated

print(aggregated.tail(6))

==

# Read file: sales

sales = pd.read\_csv('sales.csv', index\_col='Date', parse\_dates=True)

# Create a groupby object: by\_day

by\_day = sales.groupby(sales.index.strftime('%a'))

# Create sum: units\_sum

units\_sum = by\_day['Units'].sum()

# Print units\_sum

print(units\_sum)

==

# Import zscore

from scipy.stats import zscore

# Group gapminder\_2010: standardized

standardized = gapminder\_2010.groupby('region')['life', 'fertility'].transform(zscore)

# Construct a Boolean Series to identify outliers: outliers

outliers = (standardized['life'] < -3) | (standardized['fertility'] > 3)

# Filter gapminder\_2010 by the outliers: gm\_outliers

gm\_outliers = gapminder\_2010.loc[outliers]

# Print gm\_outliers

print(gm\_outliers)

# Guatemala Haiti Tajikistan Timor-Leste

==

# Create a groupby object: by\_sex\_class

by\_sex\_class = titanic.groupby(['sex', 'pclass'])

# Write a function that imputes median

def impute\_median(series):

return series.fillna(series.median())

# Impute age and assign to titanic['age']

titanic.age = by\_sex\_class['age'].transform(impute\_median)

# Print the output of titanic.tail(10)

print(titanic.tail(10))

==

# Group gapminder\_2010 by 'region': regional

regional = gapminder\_2010.groupby('region')

# Apply the disparity function on regional: reg\_disp

reg\_disp = regional.apply(disparity)

# Print the disparity of 'United States', 'United Kingdom', and 'China'

print(reg\_disp.loc[['United States', 'United Kingdom', 'China']])

==

# Create a groupby object using titanic over the 'sex' column: by\_sex

by\_sex = titanic.groupby('sex')

# Call by\_sex.apply with the function c\_deck\_survival

c\_surv\_by\_sex = by\_sex.apply(c\_deck\_survival)

# Print the survival rates

print(c\_surv\_by\_sex)

==

# Read the CSV file into a DataFrame: sales

sales = pd.read\_csv('sales.csv', index\_col='Date', parse\_dates=True)

# Group sales by 'Company': by\_company

by\_company = sales.groupby('Company')

# Compute the sum of the 'Units' of by\_company: by\_com\_sum

by\_com\_sum = by\_company['Units'].sum()

print(by\_com\_sum)

# Filter 'Units' where the sum is > 35: by\_com\_filt

by\_com\_filt = by\_company.filter(lambda g: g['Units'].sum() > 35)

print(by\_com\_filt)

==

# Create the Boolean Series: under10

under10 = pd.Series(titanic['age'] < 10).map({True:'under 10', False:'over 10'})

# Group by under10 and compute the survival rate

survived\_mean\_1 = titanic.groupby(under10)['survived'].mean()

print(survived\_mean\_1)

# Group by under10 and pclass and compute the survival rate

survived\_mean\_2 = titanic.groupby([under10, 'pclass'])['survived'].mean()

print(survived\_mean\_2)

==

# Select the 'NOC' column of medals: country\_names

country\_names = medals['NOC']

# Count the number of medals won by each country: medal\_counts

medal\_counts = country\_names.value\_counts()

# Print top 15 countries ranked by medals

print(medal\_counts.head(15))

==

# Construct the pivot table: counted

counted = medals.pivot\_table(index='NOC', values='Athlete', columns='Medal', aggfunc='count')

# Create the new column: counted['totals']

counted['totals'] = counted.sum(axis='columns')

# Sort counted by the 'totals' column

counted = counted.sort\_values(by='totals', ascending=False)

# Print the top 15 rows of counted

print(counted.head(15))

==

# Select columns: ev\_gen

ev\_gen = medals[['Event\_gender', 'Gender']]

# Drop duplicate pairs: ev\_gen\_uniques

ev\_gen\_uniques = ev\_gen.drop\_duplicates()

# Print ev\_gen\_uniques

print(ev\_gen\_uniques)

# numbers disagree???

==

# Group medals by the two columns: medals\_by\_gender

medals\_by\_gender = medals.groupby(['Event\_gender', 'Gender'])

# Create a DataFrame with a group count: medal\_count\_by\_gender

medal\_count\_by\_gender = medals\_by\_gender.count()

# Print medal\_count\_by\_gender

print(medal\_count\_by\_gender)

# right numbers

==

# Create the Boolean Series: sus

sus = (medals.Event\_gender=='W') & (medals.Gender=='Men')

# Create a DataFrame with the suspicious row: suspect

suspect = medals[sus]

# Print suspect

print(suspect)

==

# Group medals by 'NOC': country\_grouped

country\_grouped = medals.groupby('NOC')

# Compute the number of distinct sports in which each country won medals: Nsports

Nsports = country\_grouped['Sport'].nunique()

# Sort the values of Nsports in descending order

Nsports = Nsports.sort\_values(ascending=False)

# Print the top 15 rows of Nsports

print(Nsports.head(15))

==

# Create a Boolean Series that is True when 'Edition' is between 1952 and 1988: during\_cold\_war

during\_cold\_war = (medals.Edition >= 1952) & (medals.Edition <= 1988)

# Extract rows for which 'NOC' is either 'USA' or 'URS': is\_usa\_urs

is\_usa\_urs = medals.NOC.isin(['USA', 'URS'])

# Use during\_cold\_war and is\_usa\_urs to create the DataFrame: cold\_war\_medals

cold\_war\_medals = medals.loc[during\_cold\_war & is\_usa\_urs]

# Group cold\_war\_medals by 'NOC'

country\_grouped = cold\_war\_medals.groupby('NOC')

# Create Nsports

Nsports = country\_grouped.Sport.nunique().sort\_values(ascending=False)

# Print Nsports

print(Nsports)

==

# Create the pivot table: medals\_won\_by\_country

medals\_won\_by\_country = medals.pivot\_table(index='Edition', columns='NOC', values='Athlete', aggfunc='count')

# Slice medals\_won\_by\_country: cold\_war\_usa\_urs\_medals

cold\_war\_usa\_urs\_medals = medals\_won\_by\_country.loc[1952:1988, ['USA','URS']]

# Create most\_medals

most\_medals = cold\_war\_usa\_urs\_medals.idxmax(axis=1)

# Print most\_medals.value\_counts()

print(most\_medals.value\_counts())

==

# Create the DataFrame: usa

usa = medals[medals.NOC=='USA']

# Group usa by ['Edition', 'Medal'] and aggregate over 'Athlete'

usa\_medals\_by\_year = usa.pivot\_table(index=['Edition', 'Medal'],aggfunc='count')['Athlete']

# or usa.groupby(['Edition', 'Medal'])['Athlete'].count()

# Reshape usa\_medals\_by\_year by unstacking

usa\_medals\_by\_year = usa\_medals\_by\_year.unstack(level='Medal')

# Plot the DataFrame usa\_medals\_by\_year

usa\_medals\_by\_year.plot()

plt.show()

==

# Create the DataFrame: usa

usa = medals[medals.NOC == 'USA']

# Group usa by 'Edition', 'Medal', and 'Athlete'

usa\_medals\_by\_year = usa.groupby(['Edition', 'Medal'])['Athlete'].count()

# Reshape usa\_medals\_by\_year by unstacking

usa\_medals\_by\_year = usa\_medals\_by\_year.unstack(level='Medal')

# Create an area plot of usa\_medals\_by\_year

usa\_medals\_by\_year.plot.area()

plt.show()

==

# Redefine 'Medal' as an ordered categorical

medals.Medal = pd.Categorical(values=medals.Medal, categories=['Bronze', 'Silver', 'Gold'], ordered=True)

# Create the DataFrame: usa

usa = medals[medals.NOC == 'USA']

# Group usa by 'Edition', 'Medal', and 'Athlete'

usa\_medals\_by\_year = usa.groupby(['Edition', 'Medal'])['Athlete'].count()

# Reshape usa\_medals\_by\_year by unstacking

usa\_medals\_by\_year = usa\_medals\_by\_year.unstack(level='Medal')

# Create an area plot of usa\_medals\_by\_year

usa\_medals\_by\_year.plot.area()

plt.show()

==