**NumPy:**

1. # Create list baseball

baseball = [180, 215, 210, 210, 188, 176, 209, 200]

# Import the he package as np

import numpy as np

# Create a numpy array from baseball: np\_baseball

np\_baseball = np.array(baseball)

# Print out type of np\_baseball

print(np\_baseball)

1. # height and weight are available as regular lists

# Import numpy

import numpy as np

# Create array from height\_in with metric units: np\_height\_m

np\_height\_m = np.array(height\_in) \* 0.0254

# Create array from weight\_lb with metric units: np\_weight\_kg

np\_weight\_kg = np.array(weight\_lb) \* 0.453592

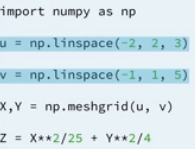
# Calculate the BMI: bmi

bmi = np.array(np\_weight\_kg) / (np\_height\_m \*\* 2)

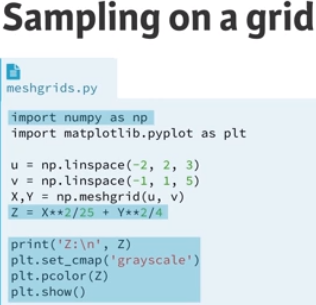
# Print out bmi

print(bmi)

**Using meshgrid():**

****

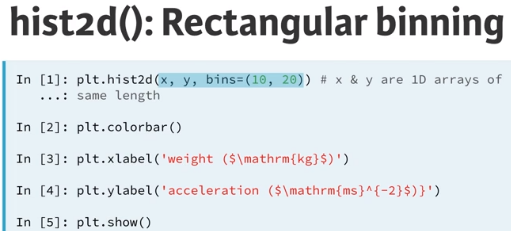
* Here, with the help of meshgrid we can make 2d arrays by combining 2 one d arrays, that are u and v.

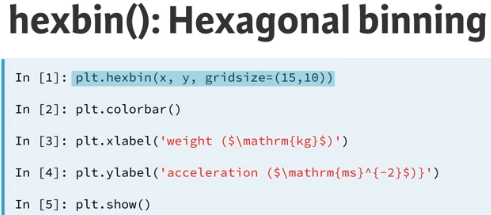


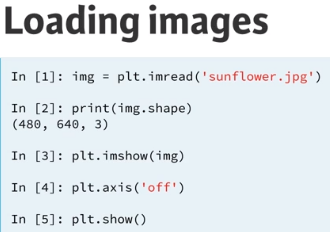
* Z line fills every entry of the 2D array.
* Print line prints it
* Following 3 lines print show array as an image
* Dark pixels will be close to 0
* Light pixels representing larger values
* When Z is printed, the value increses from 1 to 6 from left to right firs and than from top to bottom.
* We can visualize array by: plt.pcolor(Z)
* Pclor stands for pseudo color.

Color Bar:

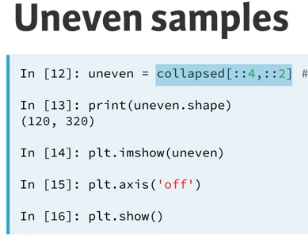
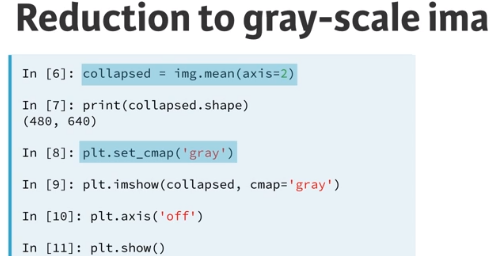
* Plt .colorbar()
* We can choose color map by:
* Plt.pcolor(Z, cmap=’grey’)
* To remove empty color we can use
* Plt.axis(‘tight’)
* Improve the spacing between the subplots with plt.tight\_layout() and display the figure.
* Contour Plots: plt.contour(Z), Filled Contour plot: plt.contourf(X,Y,Z,30)

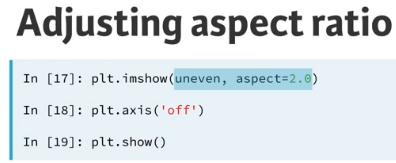
****



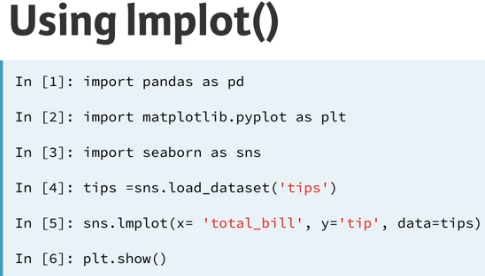


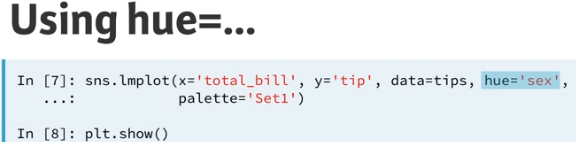
* To make rgb image to gray scale, best way is to take mean of all the value at scale.



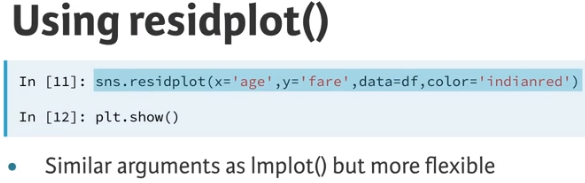


**SEABORN:**









PRACTICE CODE:

* # Generate a scatter plot of 'weight' and 'mpg' using red circles

plt.scatter(auto['weight'], auto['mpg'], label='data', color='red', marker='o')

* # Plot in blue a linear regression of order 1 between 'weight' and 'mpg'

sns.regplot(x='weight', y='mpg', data=auto, scatter=None, color = 'blue', label='order 1')

* # Plot in green a linear regression of order 2 between 'weight' and 'mpg'

sns.regplot(x='weight', y='mpg', data=auto, scatter=None, color = 'green', label='order 2',order=2)

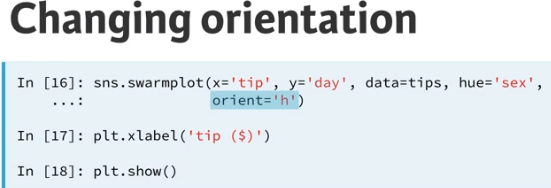
* # Add a legend and display the plot

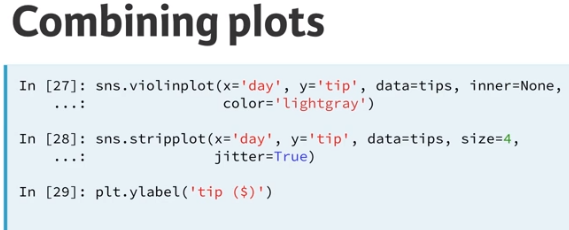
plt.legend(loc ='upper right')

plt.show()



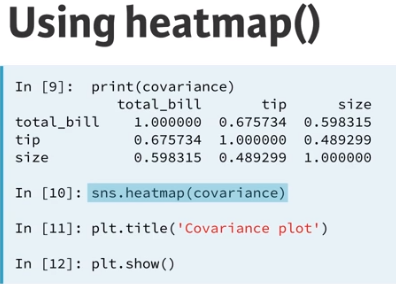


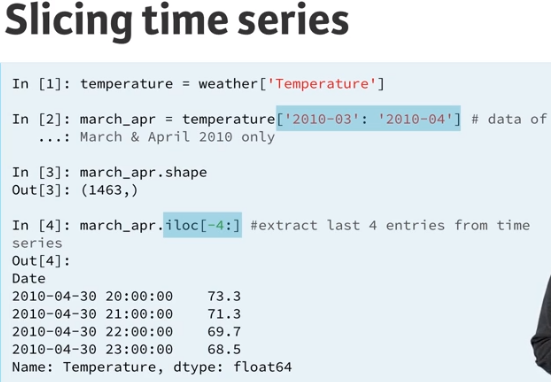




# parameters-> kind=’kde’

# parameters-> hue=’sex’











**Data Visualization - Basic:**

Very important in Data Visualization:

Explore Data

Report insights

**Matplotlib:**

Import matplotlib.pyplot as plt

Year = [1950,1970,1990,2010]

Pop = [2.519,3.629,5.263,6.972]

plt.plot(Year,Pop)

plt.show()

**Scatter Plot:**

Import matplotlib.pyplot as plt

Year = [1950,1970,1990,2010]

Pop = [2.519,3.629,5.263,6.972]

plt.scatter(Year,Pop)

plt.show()

* A correlation will become clear when you display the GDP per capita on a logarithmic scale. Add the line plt.xscale('log').

# Put the x-axis on a logarithmic scale

plt.xscale('log')

**Histogram:**

**#bins -> by default -> 10**

Import matplotlib.pyplot as plt

Values = [0,0.6,0.8,1.4,1.6,2.2,2.5,3.5,3.9,4,6]

Plt.hist(values,bins=3)#bins = block

Plt.show()

# plt.clf()cleans it up again so you can start afresh.

# Build histogram with 5 bins

plt.hist(life\_exp,bins = 5)

# Show and clean up plot

plt.show()

plt.clf()

# Build histogram with 20 bins

plt.hist(life\_exp,bins = 20)

# Show and clean up again

plt.show()

plt.clf()

**#**

Plt.xlabel(‘year’)

Plt.ylabel(‘Population’)

Plt.title(‘Projection’)

Plt.yticks([0,2,4,6])

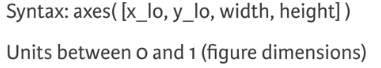
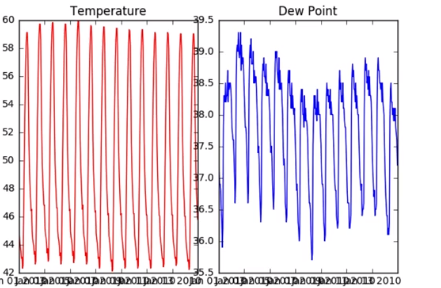
plt.xticks(tick\_val,tick\_lab)

pt.text(x,y,’Pakistan’)

**DATA VISUALIZATION – PART1:**

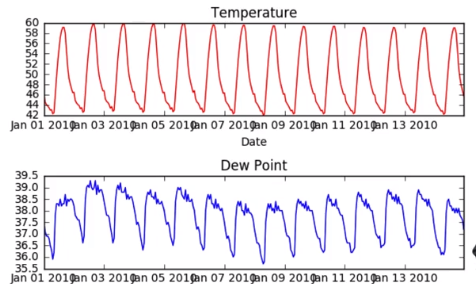
**Using Axes():**

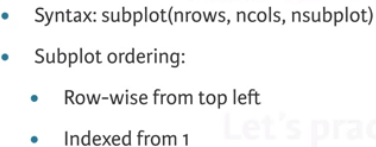
****#First axes for the left and Second one for right side.

****

#requires left corners, width and height

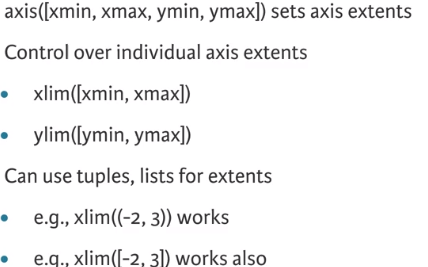
**USING SUBPLOTS:**

****

****

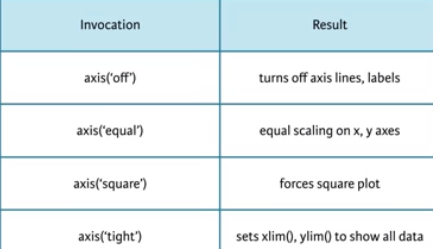
* Plt.plot(x\_axis,y\_axis,c=’blue’) #c represents color.

**Controlling Axis:**

****

* Plt.xlim((1947,1957)) #zoom
* Plt.axis((1947,1957,0,1000))

**Other Axis Options:**

****

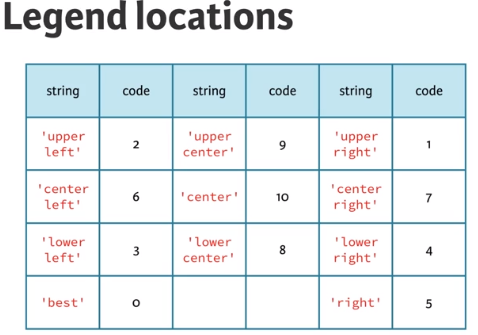
* Save figure:

plt.savefig('xlim\_and\_ylim.png')

**Legends:**

* Provide labels for overlaid points and curves

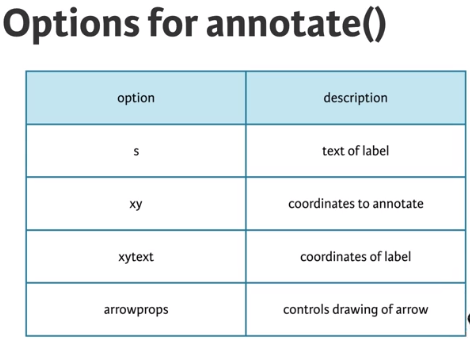




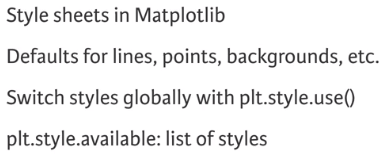
**Plot Annotations:**

* Text labels and arrows using annotate() method.
* Keyword arrowpops : dict of arrow properties, 1. Width 2. Color 3. Etc.

arrowdrops=dict(facecolor = ‘black’)



**Plot Styles:**

****

**PANDAS:**

* Tabular dataset examples
* Each row = observation
* Each column = variable
* Pandas are high level data manipulation tool
* Built on NumPy

Import pandas as Pd

Brics = Pd.DataFrame(dict)

#brics = pd.read\_csv(‘path/to/brics.csv’, index\_col = 0)

Practice Code:

# Pre-defined lists

names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']

dr = [True, False, False, False, True, True, True]

cpc = [809, 731, 588, 18, 200, 70, 45]

# Import pandas as pd

import pandas as pd

# Create dictionary my\_dict with three key:value pairs: my\_dict

my\_dict = {'country' : names,'drives\_right' : dr,'cars\_per\_cap' : cpc}

# Build a DataFrame cars from my\_dict: cars

cars = pd.DataFrame(my\_dict)

# Print cars

print(cars)

**Pandas Part 2:**

* Loc(label – based)
* Iloc(integer position – based)

|  |  |
| --- | --- |
|  | import pandas as pd |
|  | cars = pd.read\_csv('cars.csv', index\_col = 0) |
|  |  |
|  | # Print out observation for Japan |
|  | print(cars.loc['JAP']) |
|  |  |
|  | # Print out observations for Australia and Egypt |
|  | print(cars.loc[['AUS', 'EG']]) |

**Iteration in Array:**

for x in np.nditer(my\_array) :

**Loop over data frames:**

for lab, row in brics.iterrows() :

...

**For Rows in Data Frame:**

for lab, row in cars.iterrows() :

print(str(lab) + ": " + str(row["cars\_per\_cap"]))

**For Column in Data Frame:**

for lab, row in brics.iterrows() :

brics.loc[lab, "name\_length"] = len(row["country"])

**Comparision of 2 version, iterrows and apply:**

for lab, row in brics.iterrows() :

brics.loc[lab, "name\_length"] = len(row["country"])

brics["name\_length"] = brics["country"].apply(len)

**List Comprehension:**

# Create a 5 x 5 matrix using a list of lists: matrix

matrix = [[col for col in range(5)] for row in range(5)]

# Print the matrix

for row in matrix:

print(row)

**Flat Files:**

* Flat files are Text files containing records
* That is table data without structured relationships.

**Importing Flat Files Using NumPy:**

Import numpy as np

Data = np.loadtxt(filename, delimiter = “,” , skiprows = 1, usecols = [0,2] , dtype = str)

Print(Data)

#This is handy just for same data types.

**Working With Mixed Data Type:**

data = np.genfromtxt('titanic.csv', delimiter=',', names=True, dtype=None)

#names is the header

* You have just used np.genfromtxt() to import data containing mixed datatypes. There is also another function np.recfromcsv() that behaves similarly to np.genfromtxt(), except that its default dtype is None. In this exercise, you'll practice using this to achieve the same result
* Default diameter is ‘ , ’ and names is None.

**Importing Flat Files Using Pandas:**

Import pandas as pd

Data = pd.read\_csv(filename)

#We can check the first 5 rows of the data frame by

Data.head()

#We can also convert data frame to numpy array by

Data\_Array = Data.values

# Read the first 5 rows of the file into a DataFrame: data

data = pd.read\_csv(file,nrows = 5,header = None)

**Loading a pickled file:**

* Serialize means is converting the object into a sequence of bytes, or a bytestream.

# Import pickle package

import pickle

# Open pickle file and load data: d

with open('data.pkl', 'rb') as file:

d = pickle.load(file)

# Print d

print(d)

# Print datatype of d

print(type(d))

# Listing sheets in Excel files:

# Import pandas

import pandas as pd

# Assign spreadsheet filename: file

file = 'battledeath.xlsx'

# Load spreadsheet: xl

xls = pd.ExcelFile(pd.ExcelFile(file))

print(xls.sheet\_names) # Print sheet names

# Importing sheets from Excel files:

# Load a sheet into a DataFrame by name: df1

df1 = xls.parse('2004')

# Print the head of the DataFrame df1

print(df1.head())

# Load a sheet into a DataFrame by index: df2

df2 = xls.parse(0)

# Print the head of the DataFrame df2

print(df2.head())

# SAS and Stata Files:

# SAS = Statical Analysis System

# Stata = ‘Stats’ + ‘data’

# Used for analysis and business intelligence

# Importing SAS Files:

# Import pandas as pd

# From sas7bdat import SAS7BDAT

# Importing SAS Files:

# Import pandas as pd

# Data = pd.read\_stata(‘urb.dta’)

# Importing HDF5 Files:

# HDF5 = Hierarchical Data Version 5

# Standard for storing large quantities of numerical data

# Import h5py

# Data = h5py.File(filename,’r’)

# Structure of HDf5 file consist of 3 groups:

# Meta: contain meta data for the file,

# Quality: contain information about data quality

# Strain: contain data from interferometer, i.e. measurement

# Importing MATLAB Files:

# Scipy.io.loadmat(fileName) – read .mat files

# Scipy.io.savemat() – write .mat files

# Import scipy.io

# Introduction To Relational Data Base:

# SQLALchemy

# from sqlalchemy import create\_engine

# engine = create\_engine(“sqlite:///abc.sqlite”)#1 STRING PARAMETER -> “ typeOfTheDB : ///NameOfDB”

# To get table names:

# Table\_Names = engine.table\_names()

# SQL:

# SELECT 'SQL is cool!'

# AS result;

# If you only want to return a certain number of results, you can use the LIMIT keyword to limit the number of rows returned:

# SELECT \*

# FROM people

# LIMIT 10;

# SELECT DISTINCT:

# Often your results will include many duplicate values. If you want to select all the unique values from a column, you can use the DISTINCT keyword.

# SELECT DISTINCT language

# FROM films;

# Learning COUNT:

# What if you want to count the number of employees in your employees table? The COUNT statement lets you do this by returning the number of rows in one or more columns.

# SELECT COUNT(\*)

# FROM people;

# #COUNT + DISTINCT:

# SELECT COUNT(DISTINCT birthdate)

# FROM people;

# = equal

# <> not equal

# < less than

# > greater than

# <= less than or equal to

# >= greater than or equal to

# SELECT title, release\_year

# FROM films

# WHERE (release\_year >= 1990 AND release\_year < 2000)

# AND (language = 'French' OR language = 'Spanish')

# AND gross > 2000000;

# SELECT title

# FROM films

# WHERE release\_year

# BETWEEN 1994 AND 2000;

# WHERE IN:

# SELECT name

# FROM kids

# WHERE age IN (2, 4, 6, 8, 10);

# IS NULL and NOT NULL:

# SELECT title

# FROM films

# WHERE language IS NULL;

# LIKE and NOT LIKE:

# SELECT name

# FROM people

# WHERE name LIKE 'B%';

# Aggregate Functions:

# SELECT AVG(budget)

# FROM films;

# SELECT MAX(budget)

# FROM films;

# SELECT SUM(budget)

# FROM films;

# SELECT MAX(gross)

# FROM films

# WHERE release\_year BETWEEN 2000 AND 2012;

# SELECT MAX(budget) AS max\_budget,

# MAX(duration) AS max\_duration

# FROM films;

# Group BY:

# SELECT sex, count(\*)

# FROM employees

# GROUP BY sex;

# By default ORDER BY will sort in ascending order. If you want to sort the results in descending order, you can use the DESC keyword.

# In SQL, aggregate functions can't be used in WHERE clauses.

# In this case, we use HAVING

* SELECT release\_year
* FROM films
* GROUP BY release\_year
* HAVING COUNT(title) > 10;

# Introduction To Relational Data Base:

# tables: information about all tables in your current database

# columns: information about all columns in all of the tables in your current database

# CREATE New Table:

# CREATE TABLE weather)

# clouds DT(text),

# temp numeric,

# weather char(5)

# );

# To add columns you can use the following SQL query:

# ALTER TABLE table\_name

# ADD COLUMN column\_name data\_type;

# DISTINCT:

# SELECT COUNT(DISTINCT org)

# FROM uni\_Prof;

# INSERT DISTINCT:

# INSERT INTO org

# SELECT DISTINCT orgs,

# org\_sector

# FROM uni\_Prof

# INSERT INTO:

# INSERT INTO table\_name(col\_a,col\_b)

# VALUES ("valu\_A","valu\_B");

# RENAME:

# ALTER TABLE T1

# RENAME colA TO colB;

# DROP:

# ALTER TABLE T2

# DROP COLUMN colA;

# -- Delete the university\_professors table

# DROP TABLE university\_professors;

# Integrity Constraints:

# 1. Attribute Constraints (e.g: DT on columns)

# 2. Key Constrains (e.g: Primary Keys)

# 3. Referential Integrity Constrainsts (enforced through Foreign Keys)

# if temp is in int and wind\_speed is in text

# SELECT temp \* CAST(wind\_speed AS integer) AS wind\_chill

# FROM weather;

# 

# 

# 

# 

# Convert types USING a function

# If you don't want to reserve too much space for a certain varchar column, you can truncate the values before converting its type.

# For this, you can use the following syntax:

# ALTER TABLE table\_name

# ALTER COLUMN column\_name

# TYPE varchar(x)

# USING SUBSTRING(column\_name FROM 1 FOR x)

# NOT NULL:

# CREATE TABLE stud(

# lastname varchar(64) not null

# );

# ALTER TABLE table\_name

# ALTER COLUMN column\_name

# SET NOT NULL; #(To remove null, we can use DROP instead of SET)

# UNIQUE:

# We can use UNIQUE keyword to make sure that this column only contain distinct values

# CREATE TABLE A(

# columnName UNIQUE

# );

# ALTER TABLE B

# ADD CONSTRAINT some\_name UNIQUE(columnName)

# 

# 

# Surrogate Keys:

# They are artificial PK

# esa table jaha dhang sy PK na bnrhi ho ya 2 column ko mila k bnani prh rhi ho tw waha hum khud naya column bnalety or numbering krdety

# Add a surrogate Key with serial Data Type:

# ALTER TABLE cars

# ADD COLUMN id serial PRIMARY KEY; #serial type

# 

# 

# -- Rename the university\_shortname column

# ALTER TABLE professors

# RENAME COLUMN university\_shortname TO university\_id;

# -- Add a foreign key on professors referencing universities

# ALTER TABLE professors

# ADD CONSTRAINT professors\_fkey FOREIGN KEY (university\_id) REFERENCES universities (id);

# 

# 

# JOIN TABLE LINKED BY A FOREIGN KEY:

# -- Select all professors working for universities in the city of Zurich

# SELECT professors.lastname, universities.id, universities.university\_city

# FROM professors

# JOIN universities

# ON professors.university\_id = universities.id

# WHERE universities.university\_city = 'Zurich';

# 

# ## Add a professor\_id column with integer data type to affiliations, and declare it to be a foreign key that references the id column in professors.

# -- Add a professor\_id column

# ALTER TABLE affiliations

# ADD COLUMN professor\_id integer REFERENCES professors (id);

# Updating Rows:

# UPDATE table\_a

# SET column\_to\_update = table\_b.column\_to\_update\_from

# FROM table\_b

# WHERE condition1 AND condition2 AND ...;

# 

# 

# 

# CASCADE poora delete krdeta agar foreign key ko chera jae.

# -- Identify the correct constraint name

# SELECT constraint\_name, table\_name, constraint\_type

# FROM information\_schema.table\_constraints

# WHERE constraint\_type = 'FOREIGN KEY';

# -- Drop the right foreign key constraint

# ALTER TABLE affiliations

# DROP CONSTRAINT affiliations\_organization\_id\_fkey;

# -- Add a new foreign key constraint from affiliations to organizations which cascades deletion

# ALTER TABLE affiliations

# ADD CONSTRAINT affiliations\_organization\_id\_fkey FOREIGN KEY (organization\_id) REFERENCES organizations (id) ON DELETE CASCADE;

# -- Delete an organization

# DELETE FROM organizations

# WHERE id = 'CUREM';

# -- Check that no more affiliations with this organization exist

# SELECT \* FROM organizations

# WHERE id = 'CUREM';

# 

SELECT c1.name AS city, c2.name AS country

FROM cities AS c1

INNER JOIN countries AS c2

ON c1.country\_code = c2.code;

# -- 6. Select fields

# SELECT c.code, name, region, e.year, fertility\_rate, unemployment\_rate

# -- 1. From countries (alias as c)

# FROM countries AS c

# -- 2. Join to populations (as p)

# INNER JOIN populations AS p

# -- 3. Match on country code

# ON c.code = p.country\_code

# -- 4. Join to economies (as e)

# INNER JOIN economies AS e

# -- 5. Match on country code and year

# ON c.code = e.code

# AND c.year = e.year;

# INNER JOIN WITH USING:

# SELECT \*

# FROM countries

# INNER JOIN economies

# USING(code)

# -- 4. Select fields

# SELECT c.name AS country,c.continent,l.name AS language,l.official

# -- 1. From countries (alias as c)

# FROM countries AS c

# -- 2. Join to languages (as l)

# INNER JOIN languages AS l

# -- 3. Match using code

# USING(code);

# SELF JOIN:

# 

# CASE, WHEN AND THEN:

# 

# INTO:

SELECT name, continent, code, surface\_area,

CASE WHEN surface\_area > 2000000

THEN 'large'

WHEN surface\_area > 350000

THEN 'medium'

ELSE 'small' END

AS geosize\_group

INTO countries\_plus

FROM countries;

# 

# RIGHT JOIN:

# 

# FULL JOIN:

# 

# CROSS JOIN:

# 

# UNION: INTERSECT:

# 

# EXCEPT:

# 

# SEMI-JOIN:

# 

# ANTI JOIN:

# 

# 

# 

# 

# Workflow of SQL Queries:

# Import packages and functions

# Create database engine

# Connect to the engine

# Query the database

# Save query result to a Data Frame

# Close the connection.

# First SQL Query:

# from sqlalchemy import create\_engine 1b. import pandas as pd

# engine = create\_engine(‘ sqlite : /// abc.sqlite ’)

# con = engine.connect()

# rs = con.execute( “ SELECT \* FROM Orders ”)

# df = pd.DataFrame(rs.fetchall()) #fetchall for all rows (Set the Data Frame column names by : df.columns = rs.keys() )

# con.close()

# 

# Ordering your SQL records with ORDER BY:

You can also order your SQL query results. For example, if you wanted to get all records from the Customer table of the Chinook database and order them in increasing order by the column SupportRepId, you could do so with the following query:

"SELECT \* FROM Customer ORDER BY SupportRepId"

# The Pandas way to query:

# 4 line of code for the query can be written in 1 line using pandas:

# Df = pd.read\_sql\_query( “SELECT \* FROM Orders” , engine)

# Joining Tables:

# INNER JOIN in Python using Pandas:

# 

# Importing files from Web:

# The urllib package:

# Use for fetching data across the web.

# urlopen() – accept url’s instead of file names.

# From urllib.request import urlretrive

# url = ‘https ://www.google.com’

# urlretrive(url , FileNameJsimy SaveKarnaHai.csv)

# 

# # Packages the request, send the request and catch the response: r

# r = requests.get(url)

# Beautiful Soup:

# From bs4 import BeutifulSoup

# 

# 

# 

# Tidy Data:

# Columns represents separate variables

# Rows represents individual observations

# Observational units from tables

# Melting:

# 

# Pivot Data – Un-melting data:

# Opposite of melting, in melting we turned columns into rows

# Pivoting : turn unique values into separate columns

# 

# 

# Splitting a column with .str()

# # Melt tb: tb\_melt

# tb\_melt = pd.melt(frame=tb, id\_vars=['country', 'year'])

# # Create the 'gender' column

# tb\_melt['gender'] = tb\_melt.variable.str[0]

# # Create the 'age\_group' column

# tb\_melt['age\_group'] = tb\_melt.variable.str[1:]

# # Print the head of tb\_melt

# print(tb\_melt.head())

# Splitting a column with .split() and .get():

# # Melt ebola: ebola\_melt

# ebola\_melt = pd.melt(ebola, id\_vars=['Date', 'Day'], var\_name='type\_country', value\_name='counts')

# # Create the 'str\_split' column

# ebola\_melt['str\_split'] = ebola\_melt.type\_country.str.split('\_')

# # Create the 'type' column

# ebola\_melt['type'] = ebola\_melt.str\_split.str.get(0)

# # Create the 'country' column

# ebola\_melt['country'] = ebola\_melt.str\_split.str.get(1)

# # Print the head of ebola\_melt

# print(ebola\_melt.head())

# Concatenating Data:

# 

# 

# # Concatenate ebola\_melt and status\_country column-wise: ebola\_tidy

# ebola\_tidy = pd.concat([ebola\_melt,status\_country],axis = 1)

# Find and Concatenate:

# Import glob

# Csv\_files = glob.glob(‘ \*.csv ’)

# List\_data = []

# For fileName in csv\_file:

# Data = pd.read\_csv( fileName)

# List\_data.append(Data)

# Pd.concat(list\_Data)

# We cannot concatenate the data frames when the ordering of states are not the same.

# 

# Converting Data Types in frames:

# Df[ ‘ sex ’ ] = df[ ‘ sex’ ].astype(‘ category’ )

# 

# If we don’t define the parameter errors, python will give an error, as it doesn’t know how to convert dash (-) into numeric value, NAN.

# Using Regular Expression:

# 

# When using a regular expression to extract multiple numbers (or multiple pattern matches, to be exact), you can use the re.findall() function. Dan did not discuss this in the video, but it is straightforward to use: You pass in a pattern and a string to re.findall(), and it will return a list of the matches.

# matches = re.findall('\d+', 'the recipe calls for 10 strawberries and 1 banana') #10 and 1 are output.

# pattern3 = bool(re.match(pattern='[A-Z]\w\*', string='Australia'))

# 

# Drop Duplictes:

# df= df.drop\_duplicates()

# Drop Missing values:

# tips\_dropped = tips\_nan.dropna()

# Fill missing value:

# df.ColumnName . fillna(-x-)

# We can detect early warnings and errors

# We can write assert statement to verify that there is no missing values.

# assert 1==1 (this will output nothing bcz cond is true)

# assert 1==2 (this will give you an error)

# assert google.notnull().all()

# assert (ebola >= 0).all().all()

# The first .all() method will return a True or False for each column, while the second .all() method will return a single True or False.

# Principles of tidy data:

# Rows form observations

# Columns form variables

# Tidying data will make data cleaning easy

# Melting turns column into rows

# Pivot will take unique values from a column and create new columns

# # Convert the year column to numeric

# gapminder.year = pd.to\_numeric(gapminder['year'], errors='coerce')

# #Use A-Za-z to match the set of lower and upper case letters, \. to match periods, and \s to match whitespace between words.

# pattern = '^[A-Za-z\.\s]\*$'

# # Invert the mask: mask\_inverse

# mask\_inverse = ~mask

# Group gapminder by 'year' and aggregate 'life\_expectancy' by the mean. To do this:

# Use the .groupby() method on gapminder with 'year' as the argument. Then select 'life\_expectancy' and chain the .mean() method to it.

# gapminder\_agg = gapminder.groupby('year')['life\_expectancy'].mean()