Day - 1

12 January 2024 12:16

- input function is used to take input from user

  Syntax => input("Hello, your name?")

  output => Hello, your name

```
print("Hello " + input("what is your name \n") + " ! ")
```

1st => input stament will execute {"nanu"}

2nd => Hello nanu !

```
print(len(input()))
```

1st : input() => take input from user

2nd : len() => len() to calculate length of String

3rd : Print() => Print

- Variables :

```
name = input("what is your name ?")
length = len(name)
print(length)
```

variable/ function

name => Stores input

lenght => len() fn calcu the length of name variable

Print Modifiers

```
print("She said: "Hello" and then left.")

print('She said: "Hello" and then left.')

print("She said: \"Hello\" and then left.")
```

Day-2
12 January 2024 18:37

(Data types) always follows "PEDMAS" => (), **, /, +-

- Strings :- "Hello"

Code => `print("Hello"[0])`

output => H

Reasone =>

H	e	l	l	o
0	1	2	3	4

- type() :-

```
print(type(num_char))
```

↳ type() :- this function is used to find type of an variable

- type Casting :-

```
# Type casting
int_char = str(num_char)
```

↳ Explicitly name the type to Convert data into

In these case int is Converted into Char.

- float & Power :-

**note: we can not Combine str with float & int

```
(.venv) PS D:\Python> & d:/Python/.venv/Scripts/python.exe d:/Python/Day-2/mathsOperations.py
Traceback (most recent call last):
  File "d:\Python\Day-2\mathsOperations.py", line 3, in <module>
    print( "By defalut maths division is in float data type ->" + f )
           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~
TypeError: can only concatenate str (not "float") to str
(.venv) PS D:\Python> & d:/Python/.venv/Scripts/python.exe d:/Python/Day-2/mathsOperations.py
By defalut maths division is in float data type ->3.0
Traceback (most recent call last):
  File "d:\Python\Day-2\mathsOperations.py", line 4, in <module>
    print( "Power of an number ->" + 6 ** 2)
           ~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~
```

Work around:

```
f = 6/2
p = 6 ** 2
print( "By defalut maths division is in float data type ->" + str(f) )
print( "Power of an number ->" + str(p) + "\n" )
```

i.e typeCasting float & int into str

then printing

- Rounding:

a) Round(): `print(round(8/3,3))`

Specify the no. of terms after decimal to round off too.

b) Direct: float to int Conversion

`print(8//3)`

print(8 // 5)

using double // to do round off directly

- Float no. :

'{:.2f}'.format(round(billSplit,2))}")

we use it to Explicity tell, that we want 2 decimal places

So, if output is 36.6

it will get converted into 36.60

Day-3

13 January 2024 13:07

- Logical operators :

A and B
A or B
not B

- String Manupilation :-

```
combined_names = combined_names.lower()
t = combined_names.count('t')
```

↳ used to convert string into lower case

↳ Count() fn, counts the no. of times, in these Case 't', occurs in the given string

---

* Note :-

```
choice1 = input("You\'re at crossroad, choose \"left\" or \"right\" ").lower()
```

use back Slash '\' to use "" or '' quotes in the print Sentance

Day-7

14 January 2024 18:14

- Random Module's -

```
word_list = ["aardvark", "baboon", "camel"]

#TODO-1 - Randomly choose a word from the word_list and assign it to a variable called chosen_word.

chosen_word = random.choice(word_list)
```

↳ Select random value from a list data Structure in Python

Day-5

14 January 2024 17:22

→ for loop :-

```
f = ['a','b','c']
a = f


for f1 in f:
    print(f1)
```

} Simple for loop traversal

Print Output:

```
a
b
c
a
b
c
c
```

Variables:

```
-{
    f: -[
        "a",
        "b",
        "c"
    ],
    a: "c",
    f1: "c"
}
```

If we use same name as list then

```
for a in a:
    print(a)
print(a)
```

a = {a, b, c}

in for loop

a = a
a = b
a = c, after the for loop is ended its execution the the list name 'a' will have only a = 'c' value

reason as we have overwritten rest of the values might be)

Day-4

14 January 2024 10:46

→ Random Module's

```
import random
```
→ Module name

Generate random integer

```
random_int = random.randint(1,10)
print(random_int)
```
start, end (both inclusive)

```
random_float = random.random() # range is from 0.000.... to 0.999999
print(random_float)
```

Generate floating point numbers from 0.00 to 0.99

```
print(random_float * 3)
```

Note's `random.shuffle(passwordList)` ; use Shuffle from random module to randomize the list

→ List Data Structure's —

①

```
list = ["Delhi","Maharastra"]

print(list)
print(list[0]) → 1st
print(list[1]) → 2nd
print(list[-1]) → Last
print(list[-2]) → 2nd Last

list.append("M.P.")   ← add new item at the end
print(list[-1])

list.extend(["U.P.","Assam"])   → extend the orignal list

print(list)

list.insert(0,"Haryana")   → insert / replace

print(list)
```

index, new font

Print Output:

```
['Delhi', 'Maharastra']
Delhi
Maharastra
Maharastra
Delhi
M.P.
['Delhi', 'Maharastra', 'M.P.', 'U.P.', 'Assam']
['Haryana', 'Delhi', 'Maharastra', 'M.P.', 'U.P.', 'Assam']
```

Variables:

```
-{
    list: -[
        "Haryana",
        "Delhi",
        "Maharastra",
        "M.P.",
        "U.P.",
        "Assam"
    ]
}
```

②

```
abc = ['a','b','c']
letter_index = abc.index(letter)
```

It is used to find index of matching string from the

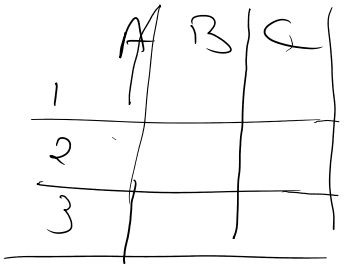
List

③ nesting in Python goes from outside to inside

```
map[number_index][letter_index] = "X"
```

B3 → map[3] [B]

A3 → map[3] [A]



Day-9

18 January 2024 13:10

- Dictionary :-

A data Structure that works on the principle of key value pair

= { "key" : value }

Note :- all types of nesting like

- list nested in Dictionary
- Dictionary nested into Dictionary
- Dictionary nested into list

Day-10

18 January 2024 13:16

```
# sub
def sub(n1, n2):
    return n1-n2

# mul
def mul(n1, n2):
    return n1*n2

# divide
def divide(n1, n2):
    return n1/n2

n1 = int(input("Whats the 1st number ? "))
n2 = int(input("Whats the 2nd number ? "))

# dictionary

operations = {
    "+" : add,
    "-" : sub,
    "*" : mul,
    "/" : divide
}
for i in operations:
    print(i)

operations_symbol = input("Pick an operation from the line above: ")

#
calculator_function = operations[operations_symbol]
ans = calculator_function(n1,n2)
print(f"{n1} {operations_symbol} {n2} = {ans}")
```

functions

Each and every key{i.e Symbols} are linked to pairs that are functions {eg: add, sub}

after selection of Symbol then this calculation_function will point to the function for the key from the Dictionary

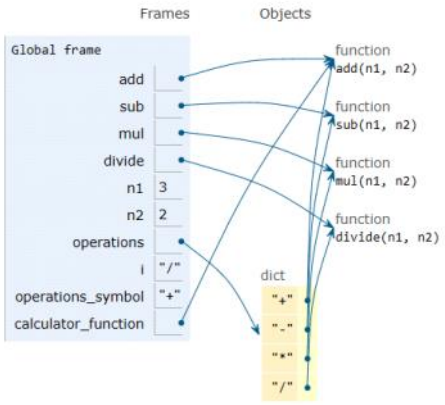
Basically, it talks into the function based on the Symbol, selected by user.

→ Python Visualizer's

Python_Visualizer

Print output (drag lower right corner to resize)

```
Whats the 1st number ? 3
Whats the 2nd number ? 2
+
-
*
/
Pick an operation from the line above: +
```



Day-12

20 January 2024 13:25

→ Scope:

*Note:-

Loop & if Statement doesn't count as creating a new Scope in Python.

But functions doo create new Scope i.e local Scope.

⇒

- Changing global variables isn't reccomended and can be done

1)

```
main.py +
1  ene =1
2
3  def a():
4    global ene
5    ene = 2
6
7  a()
8  print(ene)
```

2)

```
enemies = 1

def increase_enemies():
  print(f"enemies inside function: {enemies}")
  return enemies + 1

enemies = increase_enemies()
print(f"enemies outside function: {enemies}")
```

Day-16

21 January 2024 12:39

OOP's

↳ class starts with an Capital letter

```
car = CarBlueprint()
```

car → obj

CarBlueprint() → class

Day-17

21 January 2024 18:29

Pascal Casing

↳ for class

|||

1st Capital letter

Snake Casing

↳ for everything else

||

apple_is_type_of_fruit

```
class User:
    pass

user_1 = User()
user_1.id = "007"

print(user_1.id)
```

object class

Attribute

Attributes are attached to an object of a class

(or)

Attributes are variable that are associated with an object.

⇒ Initialization of class

```
class Car:
    def __init__(self):
    #initialise attributes
```

Constructor

→ Functions defined in a class are called method

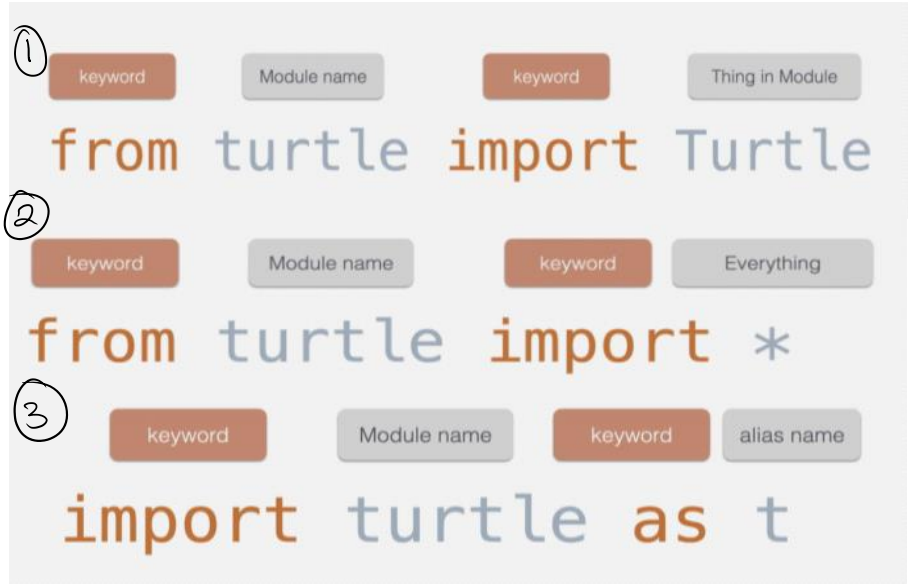
→ Functions defined in a class are called method and method in Python always requires "self" parameter.

Day-18

21 January 2024 22:07

→ Modules :-

* Import modules :



→ Tupple :-

like list data type in Python only

difference being Tupple is immutable

Eg :- my_tupple = (1, 2, 4)

tupple to list conversion

↳ list (my-tupple)

Day-20

23 January 2024 10:27

Note:-

Day-21

23 January 2024 22:21

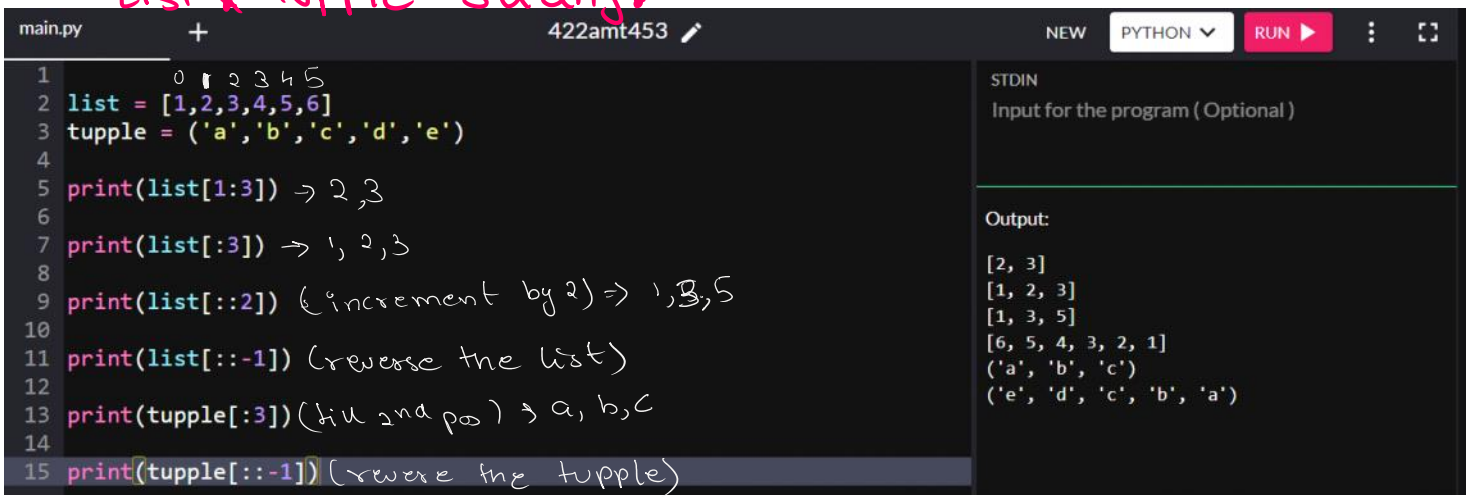
(Day 20 Q21)

→ Inheritance :-

```
class A():
    def __init__():
```

```
class B(A):
    def __init__(self):
        super().__init__()
```

List & Tupple Slicing :-



→ File Handling :

① Opening :-

```
file = open("Day-24\my.txt")
contents = file.read()
print(contents)
file.close()
```

method ①

```
with open("Day-24\my.txt") as file:
    contents = file.read()
    print(contents)
```

method ②

diff blw m① & m② is that, in m① we need to close file manually

③ write :

```
with open("Day-24\my.txt",mode="a") as file:
    file.write("\nNew text")
```

append mode : add text to existing txt present in the file.

```
with open("Day-24\ New_file.txt",mode="w") as file:
    file.write("\nNew @-@ text")
```

write : overwrite txt if present in the file

If file doesn't exist and mode = "w" then it will create file of that specified name

Day-25 Pandas

30 January 2024 13:33

→ Pandas(CSV):

Data Types:

Frame → Complete table

Series → row

	A	B	C	D
	day	temp	condition	
	Monday	12	Sunny	
	Tuesday	14	Rain	
	Wednesday	15	Rain	
	Thursday	14	Cloudy	
	Friday	21	Sunny	
	Saturday	22	Sunny	
	Sunday	24	Sunny	

→ Frame

Series

⇒ View all the programs in the Day-25

Day-26

30 January 2024 22:46

→ List Comprehension – speciality of Python

For Loop

```
numbers = [1, 2, 3]
new_list = []
for n in numbers:
    add_1 = n + 1
    new_list.append(add_1)
```

List Comprehension

```
numbers = [1, 2, 3]
new_list =[n + 1 for n in numbers]
```

Flow of list comprehension –

[→] eg: [i for i in num if i > 5]

1st 2nd 3rd

→ Dictionary Comprehension:

⇒ Dictionary Comprehension

```
new_dict ={new_key:new_value for (key,value) in dict.items()}
```

⇒ Conditional Dictionary Comprehension

```
new_dict ={new_key:new_value for (key, value) in dict.items() if test}
```

Day-27

31 January 2024 22:05

→ Tkinter (GUI) :-

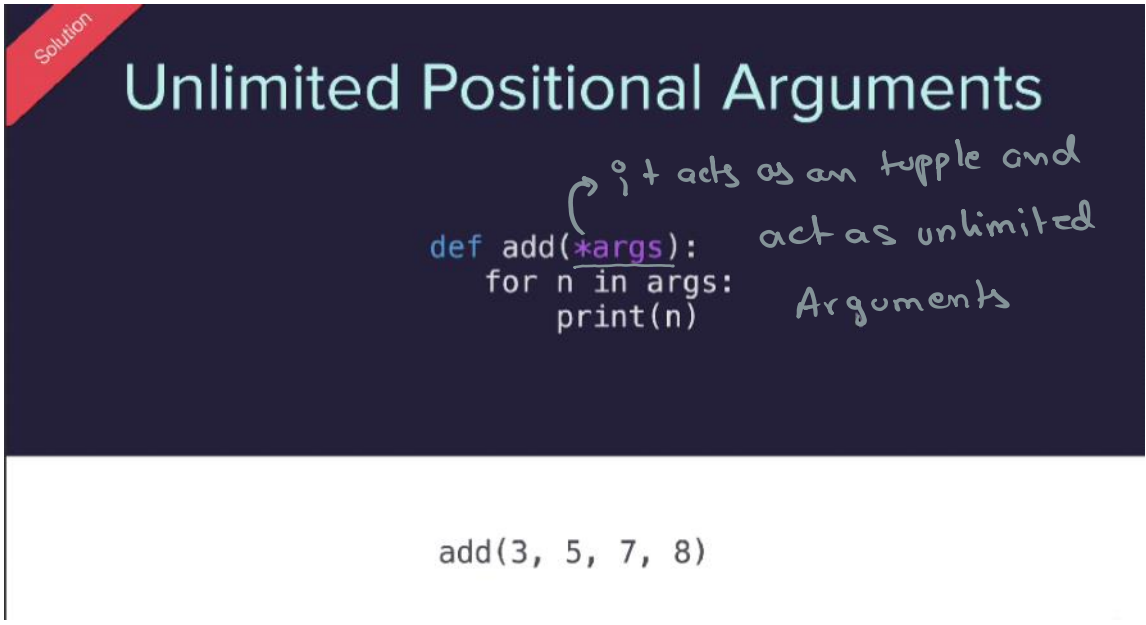
Sample Code

```
import tkinter

window = tkinter.Tk()
window.title("1st GUI Program")
window.minsize(width=500, height=300)

# Lable
my_lable = tkinter.Label(text="Lable Here", font={"Arial", 2, "bold"})
my_lable.pack(side="left", expand=1)


window.mainloop()
```

→ Python Advance Arguments :-

① Pos Argument (notation : *name)

Eg -

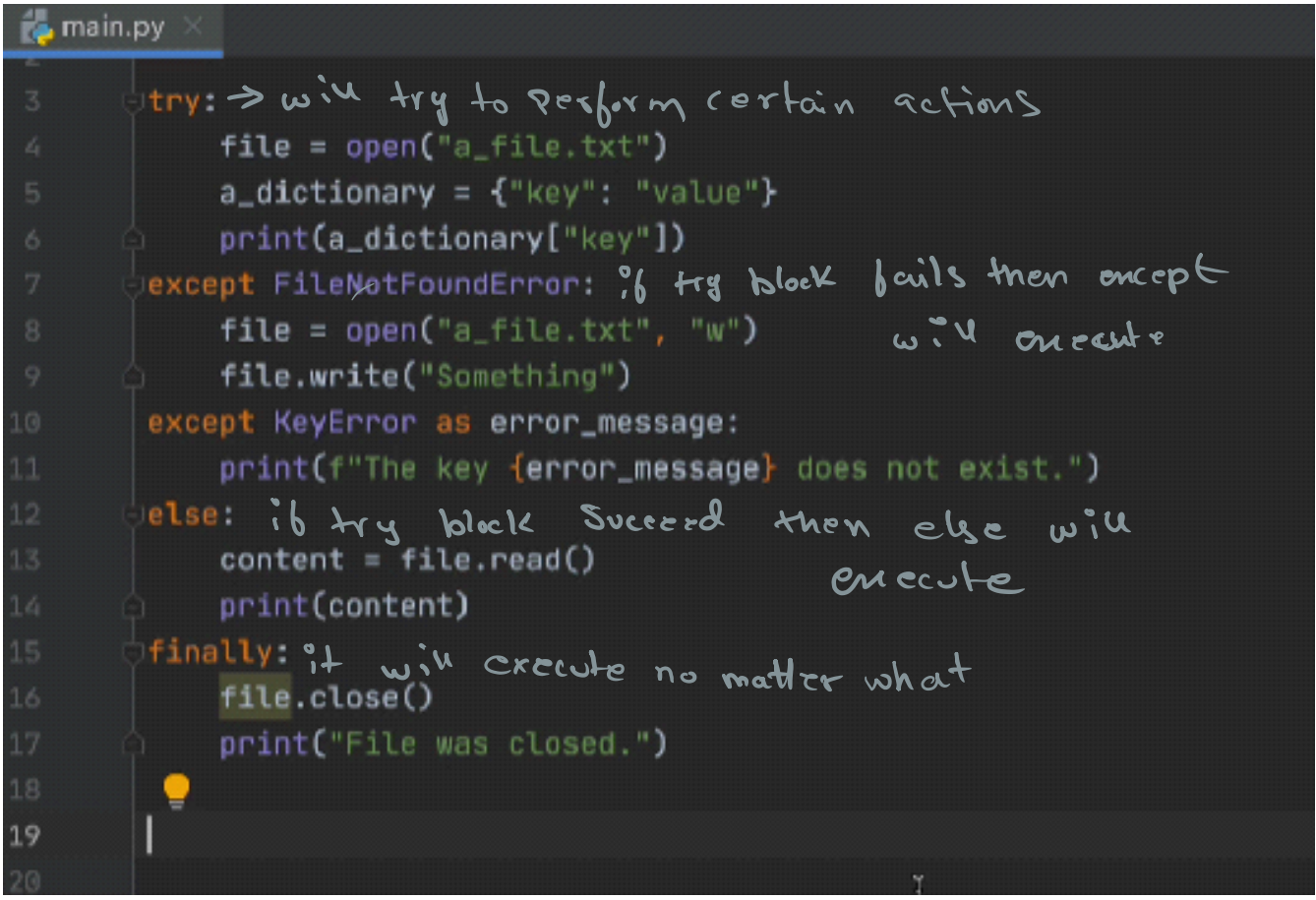


② Keyword Arguments : (**name)

Day-30

02 February 2024 18:12

→ Error Handling | Exception Handeling :—



Dy-37

09 February 2024 21:39

HTTP Requests

GET	requests.get(), get data from
POST	requests.post(), transfer data using api
PUT	requests.put(), update data
DELETE	requests.delete(), delete data

Day-45

29 February 2024 21:40

Python Day-45 bs4_tutorial

Beautifulsoup : Difference between .find() and .select()

Asked 7 years, 8 months ago Modified 3 years, 8 months ago Viewed 106k times

77

When you use ***BeautifulSoup*** to scrape a certain part of a website, you can use

- `soup.find()` and `soup.findAll()` or
- `soup.select()`.

Is there a difference between the `.find()` and the `.select()` methods? (e.g. In performance or flexibility, etc.) Or are they the same?

python python-3.x beautifulsoup

Share Improve this question Follow

edited Jun 11, 2020 at 15:36
Max
3,971 ● 2 ● 10 ● 26

asked Jun 25, 2016 at 12:09
Dieter
2,559 ● 1 ● 23 ● 42

22 `select()` accepts CSS selectors, `find()` does not – Andrea Corbellini Jun 25, 2016 at 12:13

See crummy.com/software/BeautifulSoup/bs4/doc/#find and crummy.com/software/BeautifulSoup/bs4/doc/#css-selectors – Andrea Corbellini Jun 25, 2016 at 12:15

Day-54-

20 March 2024 23:02

link: /post/<post_id>

```python
@app.route('/post/<int:post_id>')
def show_post(post_id):
    # TODO: Retrieve a BlogPost from the database based on the post_id
    requested_post = db.get_or_404(BlogPost,post_id)
    return render_template("post.html", post=requested_post)
```

will call this function

```
<div class="post-preview">
  <a href="{{ url_for('show_post', post_id=post.id) }}">
    <h2 class="post-title">{{ post.title }}</h2>
    <h3 class="post-subtitle">{{ post.subtitle }}</h3>
  </a>
```

```python
@app.route('/<name>')
def greet(name):
    return f"Hello {name}!"
```

same for it to work

/ or /

```
<li>
    <a href="{{ url_for('delete', id=book.id) }}">Delete</a>
    {{book.title}} - {{book.author}} - {{book.rating}}/10
    <a href="{{ url_for('edit', id=book.id) }}">Edit Rating</a>
</li>
```

link: /delete?id=<>

```python
@app.route("/delete")
def delete():
    book_id = request.args.get('id')

    # DELETE A RECORD BY ID
    book_to_delete = db.get_or_404(Book, book_id)
    # Alternative way to select the book to delete.
    # book_to_delete = db.session.execute(db.select(Book).where(Book.id == book_id)).scalar()
    db.session.delete(book_to_delete)
    db.session.commit()
    return redirect(url_for('home'))
```

Day-60 (Forms)

21 March 2024 22:11

① Normal HTML/CSS form

```python
@app.route('/login',methods=['POST'])
def receive_data():
    name = request.form['username']
    password = request.form['password']
    return f"<h1>Name: {name}, Password:{password}</h1>"
```

to get data entered in the form

id/name value

```html
<form action="{{ url_for('receive_data') }}" method="post">
    <label>Name</label>
    <input type="text" placeholder="name" name="username">
    <label>Password</label>
    <input type="text" placeholder="password" name="password">
    <button type="submit">Ok</button>
</form>
```

(text area) => for more writing space

=> To retrive data from form Submitted then use request.method="Post"

```python
@app.route('/register', methods=["GET","POST"])
def register():
    if request.method == "POST":
        new_user = User(
            email=request.form.get('email'),
            name=request.form.get('name'),
            password=request.form.get('password')
        )
        db.session.add(new_user)
        db.session.commit()
        return render_template("secrets.html", name=request.form.get('name'))
    return render_template("register.html")
```

```html
{% extends "base.html" %} {% block content %}

<!--TODO: Display the user's name in the h1 title-->
<div class="container">
  <h1 class="title">Welcome, {{name}}!</h1>
  <!--TODO: Make a GET request to the /download path-->
  <a href="#">Download Your File</a>
</div>
{% endblock %}
```

② WTF forms:-

RuntimeError: A secret key is required to use CSRF.

Do include key.

```python
app = Flask(__name__)
app.config['SECRET_KEY'] = 'Apple' # Can be any thing otherwise gives error
Bootstrap5(app)                    # RuntimeError: A secret key is required to use CSRF.
```

```python
from flask import Flask, render_template
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Email
from flask_bootstrap import Bootstrap5
```

```python
from wtforms.validators import DataRequired, Email
from flask_bootstrap import Bootstrap5


class LoginForm(FlaskForm):
    email = StringField(label='Email', validators=[DataRequired(), Email()])
    password = PasswordField(label='Password', validators=[DataRequired()]) # label will appear on website
    submit = SubmitField(label='Log - In')
```

"Post" → when Submit button of the form is pressed

→ to retrive data entered by user in the form

```python
@app.route("/login",methods = ["GET","POST"])
def login():
    login_form = LoginForm()
    if login_form.validate_on_submit():
        if login_form.email.data == "admin@email.com" and login_form.password.data == "12345678":
            return render_template("success.html")
        else:
            return render_template("denied.html")
    return render_template('login.html', form=login_form)
```

```jinja
{% from 'Bootstrap5/form.html' import render_form %}

{% block title %} Login {% endblock %}

{%block content%}
    <div class="container">
        <h1>Login</h1>
        {{render_form(form)}}
    </div>
{% endblock %}
```

(3) CKEditor + WTF form

↳ Provides extra options for end user to formate the data entered by them in the form.

```python
from flask import Flask, render_template, redirect, url_for
from flask_bootstrap import Bootstrap5
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import Integer, String, Text
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired, URL
from flask_ckeditor import CKEditor, CKEditorField
from datetime import date


app = Flask(__name__)
app.config['SECRET_KEY'] = '8BYkEfBA6O6donzWlSihBXox7C0sKR6b'
ckeditor = CKEditor(app)
Bootstrap5(app)
```

```python
class BlogForm(FlaskForm):
    title = StringField(label="Title",validators=[ DataRequired() ])
    subtitle = StringField(label="Sub-Title",validators=[ DataRequired() ])
    auth_name = StringField(label="Author's Name",validators=[ DataRequired() ]
    url = StringField(label="URL",validators=[ DataRequired(), URL() ])
    body = CKEditorField(label="Body",validators=[ DataRequired() ])
    submit = SubmitField(label= 'Submit')
```

```python
@app.route("/new-post", methods=["GET", "POST"])
def add_new_post():
    form = BlogForm()
    if form.validate_on_submit():
        new_post = BlogPost(
            title=form.title.data,
            subtitle=form.subtitle.data,
            body=form.body.data,
            img_url=form.url.data,
            author=form.auth_name.data,
            date=date.today().strftime("%B %d, %Y")
        )
        db.session.add(new_post)
        db.session.commit()
        return redirect(url_for("get_all_posts"))

    return render_template("make-post.html",form=form)
```

```
{% from "bootstrap5/form.html" import render_form %}
{% block content %}
{% include "header.html" %}

        <!-- TODO:-Add CKEditor and render the WTF form here -->
        {{ ckeditor.load() }}
        {{ ckeditor.config(name='body') }}
        {{ render_form(form, novalidate=True, button_map={"submit": "primary"}) }}
```

SQL

21 March 2024 22:50

~~Day-63~~ for SQLite different CRUD operations.

RESTFUL API

21 March 2024 23:03

* Day - 66 for Complete guide

Day-68

22 March 2024 22:45

-> flash Message :-

Simple Flashing

So here is a full example:

```
from flask import Flask, flash, redirect, render_template, \
     request, url_for

app = Flask(__name__)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        if request.form['username'] != 'admin' or \
                request.form['password'] != 'secret':
            error = 'Invalid credentials'
        else:
            flash('You were successfully logged in')
            return redirect(url_for('index'))
    return render_template('login.html', error=error)
```

And here is the `layout.html` template which does the magic:

```
<!doctype html>
<title>My Application</title>
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <ul class=flashes>
    {% for message in messages %}
      <li>{{ message }}</li>
    {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
{% block body %}{% endblock %}
```

```python
@app.route('/login', methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form.get('email')
        password = request.form.get('password')

        # Find user by email entered.
        result = db.session.execute(db.select(User).where(User.email == email))
        user = result.scalar()
        if not user:
            flash("That email does not exist, please try again.")
            return redirect(url_for('login'))

        elif not check_password_hash(user.password, password):
            flash('Password incorrect, please try again.')
            return redirect(url_for('login'))
        # Check stored password hash against entered password hashed.
        else:
            login_user(user)
            return redirect(url_for('secrets'))

    return render_template("login.html")
```

```
{% extends "base.html" %}
{% block content %}

<div class="box">
    <h1>Login</h1>
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            {% for message in messages %}
            <p>{{ message }}</p>
            {% endfor %}
        {% endif %}
    {% endwith %}
    <form method="post">
        <input type="text" name="email" placeholder="Email" required="required"/>
        <input type="password" name="password" placeholder="Password" required="required"/>
        <button type="submit" class="btn btn-primary btn-block btn-large">Let me in.</button>
    </form>
</div>

{% endblock %}
```

Login

22 March 2024 22:50

```
from flask_login import LoginManager
login_manager = LoginManager()
```

```
login_manager.init_app(app)
```

```
@login_manager.user_loader
def load_user(user_id):
    return User.get(user_id)
```

Extra used in Day-69 Project

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    # Here we use a class of some kind to represent and validate our
    # client-side form data. For example, WTForms is a library that will
    # handle this for us, and we use a custom LoginForm to validate.
    form = LoginForm()
    if form.validate_on_submit():
        # Login and validate the user.
        # user should be an instance of your `User` class
        login_user(user)

        flask.flash('Logged in successfully.')

        next = flask.request.args.get('next')
        # url_has_allowed_host_and_scheme should check if the url is safe
        # for redirects, meaning it matches the request host.
        # See Django's url_has_allowed_host_and_scheme for an example.
        if not url_has_allowed_host_and_scheme(next, request.host):
            return flask.abort(400)

        return flask.redirect(next or flask.url_for('index'))
    return flask.render_template('login.html', form=form)
```

```
{% if current_user.is_authenticated %}
  Hi {{ current_user.name }}!
{% endif %}
```

Views that require your users to be logged in can be decorated with the

```
@app.route("/settings")
@login_required
def settings():
    pass
```

When the user is ready to log out:

```
@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(somewhere)
```

Day-72

Then import pandas into your notebook and read the .csv file.

```
import pandas as pd
df = pd.read_csv('salaries_by_college_major.csv')
```

```
[9] df.head()
```

To Print 1st 5 entries

	Undergraduate Major	Starting Median Salary	Mid-Career Median Salary	Mid-Career 10th Percentile Salary	Mid-Career 90th Percentile Salary	Group
0	Accounting	46000.0	77100.0	42200.0	152000.0	Business
1	Aerospace Engineering	57700.0	101000.0	64300.0	161000.0	STEM
2	Agriculture	42600.0	71900.0	36300.0	150000.0	Business
3	Anthropology	36800.0	61500.0	33800.0	138000.0	HASS
4	Architecture	41600.0	76800.0	50600.0	136000.0	Business

df.tail() is used to print last 5 entries

df.shape is used to find no. of rows & cols present.

df.columns is used to find names of all the columns

df.isna() used to find NaN or void entries

```
clean_df = df.dropna()
clean_df.tail()
```

used to drop all NaN values

df['col_name'].max() used to find max possible value in the col

df['col_name'].idmax() used to find index of max value

df['col_name'].loc[index] will return col_name value at the specified index

df['col1'].subtract['col2'] col1-col2

df.insert('index','col_name',col_val) will insert col_val under col_name at index in the existing Pandas dataframe

df[['col1']['col2']] to get the simplified view of col1 & col2 only.

df.sort_values('col_name',ascending= false) to sort values according to the col_name specified values

df.groupby('col_name').____ used to grp similar values together for col_name

Count() total of each option present in the col_name

mean() mean " " " " " " "

Sum() Sum " " " " " " "

value_count() quick way to find members of each categories

```
pd.options.display.float_format = '{:,.2f}'.format
```

used to format pandas data frame to show only 2 digits after decimal place

Day-73

```
df = pd.read_csv('QueryResults.csv', names=['DATE', 'TAG', 'POSTS'],
header=0)
```

Setting the header row to 0 allows us to substitute our own column names.

pd.to_datetime(df.DATE) used to convert into date & time format

- Pivot

	Age	Actor	Power
0	Young	Jack	100
1	Young	Arnold	80
2	Young	Keanu	25
3	Young	Sylvester	50
4	Old	Jack	99
5	Old	Arnold	75
6	Old	Keanu	5

→

Actor	Arnold	Jack	Keanu	Sylvester
Age				
Old	75.0	99.0	5.0	NaN
Young	80.0	100.0	25.0	50.0

```
pivoted_df = test_df.pivot(index='Age', columns='Actor', values='Power')
pivoted_df
```

- NaN values replace

instead of dropping the row that contains NaN values

```
reshaped_df.fillna(0, inplace=True)
```

The `inplace` argument means that we are updating reshaped_df. Without this argument we would have to write something like this:

```
reshaped_df = reshaped_df.fillna(0)
```

df.isna().values.any() returns a simple T/F is NaN values present in the entire table.

Day-74

`df['col'].nunique()` - used to find no. of unique entries present in a column

`df.sort_values('col')` used to sort dataframe on the basis of the values of 'col'.

★ Aggrigator function

`.agg( )` specify the operation which we need to perform

Eg. `.agg({'col_name': pd.Series.nunique})`

Operation to be performed

★ Merge

`merge_df = pd.merge('col1','col2', on='id')`

```
[51] merged_df = pd.merge(set_theme_count, themes, on='id')
     merged_df[:3]
```

	id	set_count	name	parent_id
0	158	753	Star Wars	NaN
1	501	656	Gear	NaN
2	494	398	Friends	NaN

Day-75

`df.describe()` get to get whole bunch of descriptive stats.

```
df_tesla.describe()
```

	TSLA_WEB_SEARCH	TSLA_USD_CLOSE
count	124.000000	124.000000
mean	8.725806	50.962145
std	5.870332	65.908389
min	2.000000	3.896000
25%	3.750000	7.352500
50%	8.000000	44.653000

```
df_tesla.describe()
```

	TSLA_WEB_SEARCH	TSLA_USD_CLOSE
count	124.000000	124.000000
mean	8.725806	50.962145
std	5.870332	65.908389
min	2.000000	3.896000
25%	3.750000	7.352500
50%	8.000000	44.653000
75%	12.000000	58.991999
max	31.000000	498.320007

Str to Datetime

```
df.Month = pd.to_datetime(df.Month)
df.time = pd.to_datetime(df.time)
```

→ Data Resampling

used to Convert data freq eg- daily to Monthly

```
df- = df.resample('_', 'col-name').____
```

Month
Year
T-minute

last() last available at the end of time span.

mean() avg through out the time span

→ Rolling Avg -

```
roll-df = df['col'].rolling(window=n).mean()
```

df.Sample(n) Print any n random rows

→ Drop

```
df.drop(['col_name'], axis=____, inplace=True)
```

Same → 0
Same → 1
→ 'index' used to delete row
→ 'columns' " " " columns

→ Duplicate Entries

`df.duplicated()` return T/F values if duplicates are pre

`df.drop_duplicates(subset=['c1','c2','c3'])`

Drop duplicates, if values repeats/same in the specified subset Columns.

`df.info()` give info abot data names of columns, data type of entries

→ Data Manupilation

String Modification

`df.col_name.astype(str).str.replace(';',':')`

org to be

Convert data to str type

org value → to be replaced with value

Day-76

- stack() it is used to stack after we have split columns with multiple data options.

Day-78

data.info()

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5391 entries, 0 to 5390
Data columns (total 6 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Rank                   5391 non-null   int64
 1   Release_Date           5391 non-null   object
 2   Movie_Title            5391 non-null   object
 3   USD_Production_Budget  5391 non-null   object
 4   USD_Worldwide_Gross    5391 non-null   object
 5   USD_Domestic_Gross     5391 non-null   object
dtypes: int64(1), object(5)
memory usage: 252.8+ KB
```

Convert object into Sutable data type

pd.to_numeric(str) used to convert str to int

```
chars_to_remove = [',', '$']
columns_to_clean = ['USD_Production_Budget',
                    'USD_Worldwide_Gross',
                    'USD_Domestic_Gross']

for col in columns_to_clean:
    for char in chars_to_remove:
        # Replace each character with an empty string
        data[col] = data[col].astype(str).str.replace(char, "")
    # Convert column to a numeric data type
    data[col] = pd.to_numeric(data[col])
```

```
data.Release_Date = pd.to_datetime(data.Release_Date)
```

→ Multi Conditional Statment

bit-wise operator

```
data.loc[(data.x==0) & (data.y!=0)]
```

or

```
data.query('data.x==0 and data.y!=0')
```

→ Floor Division

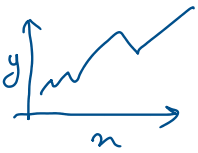
```
5.0 / 2
# output: 2.5
5.0 // 2
# output: 2.0
```

1999 // 2 → 199

Day-73

```
import matplotlib.pyplot as plt
```

plt.plot(x,y) it will plot a simple graph ; y



→ Styling the chart

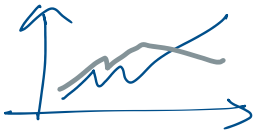
plt. ____

- `.figure()` - allows us to resize our chart
- `.xticks()` - configures our x-axis
- `.yticks()` - configures our y-axis
- `.xlabel()` - add text to the x-axis
- `.ylabel()` - add text to the y-axis
- `.ylim()` - allows us to set a lower and upper bound

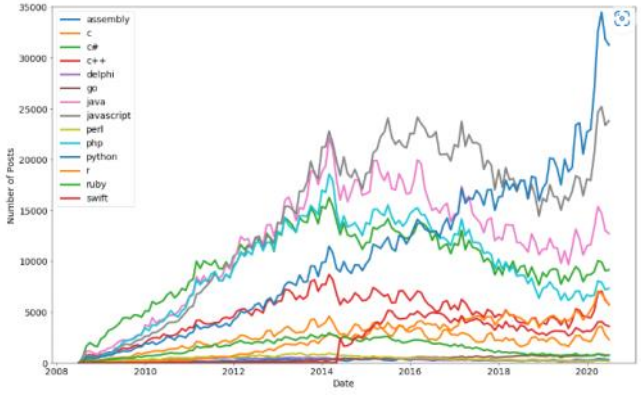
→ Multiline graph plot

(-) plt.plot(x,y)

(-) plt.plot(x,y')



```
plt.figure(figsize=(16,10))
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Number of Posts', fontsize=14)
plt.ylim(0, 35000)

for column in reshaped_df.columns:
    plt.plot(reshaped_df.index, reshaped_df[column],
             linewidth=3, label=reshaped_df[column].name)

plt.legend(fontsize=16)
```



n_df = df.rolling(window=int) . ____ used to perform rolling window calculations

sum()

n_df = df.rolling(window: [illegible]

sum()

mean()

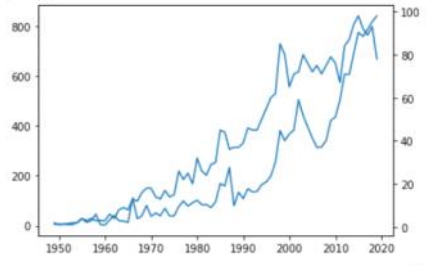
-> Two Axis

ax1 = plt.gca() #get current axis

ax2 = ax1.twinx()

```
ax1 = plt.gca() # get the axis
ax2 = ax1.twinx() # create another axis that shares the same x-axis

ax1.plot(sets_by_year.index[:-2], sets_by_year.set_num[:-2])
ax2.plot(themes_by_year.index[:-2], themes_by_year.nr_themes[:-2])
```

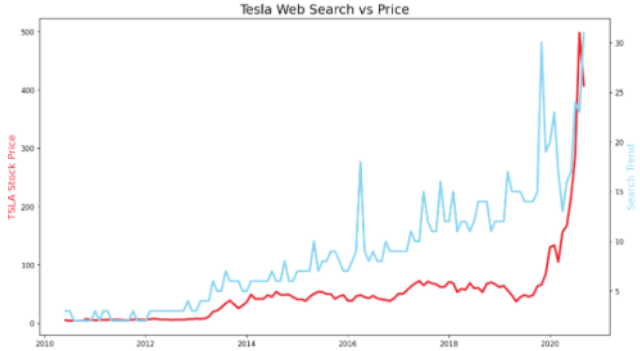
```
[<matplotlib.lines.Line2D at 0x7f974e2c4da0>]
```



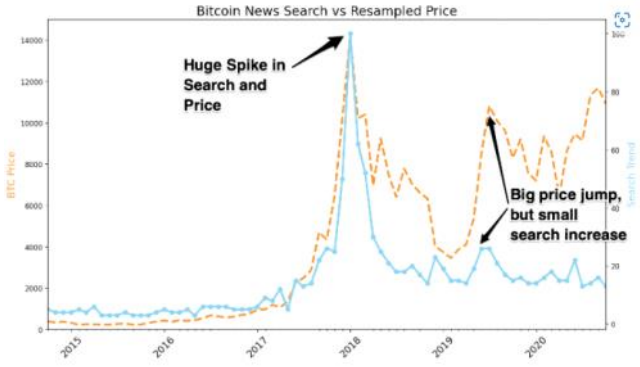
ax.set_xlim([ ])

used to specify the limits for the axis.

```
# increases size and resolution
plt.figure(figsize=(14,8), dpi=120)
plt.title('Tesla Web Search vs Price', fontsize=18)

ax1 = plt.gca()
ax2 = ax1.twinx()

# Also, increase fontsize and linewidth for larger charts
ax1.set_ylabel('TSLA Stock Price', color='#E6232E', fontsize=14)
ax2.set_ylabel('Search Trend', color='skyblue', fontsize=14)

ax1.plot(df_tesla.MONTH, df_tesla.TSLA_USD_CLOSE, color='#E6232E',
linewidth=3)
ax2.plot(df_tesla.MONTH, df_tesla.TSLA_WEB_SEARCH, color='skyblue',
linewidth=3)

# Displays chart explicitly
plt.show()
```



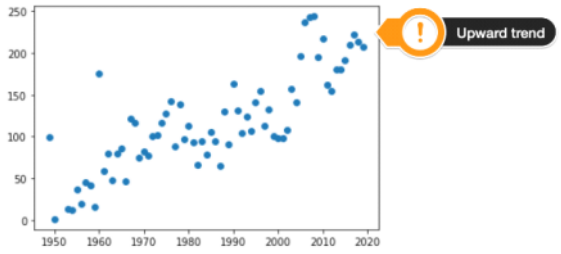

```
# Experiment with the linestyle and markers
ax1.plot(df_btc_monthly.index, df_btc_monthly.CLOSE,
         color='#F08F2E', linewidth=3, linestyle='--')
ax2.plot(df_btc_monthly.index, df_btc_search.BTC_NEWS_SEARCH,
         color='skyblue', linewidth=3, marker='o')
```



-> Scatter plot

```
[44] plt.scatter(parts_per_set.index[:-2], parts_per_set.num_parts[:-2])
```

```
<matplotlib.collections.PathCollection at 0x7fbe7e60be80>
```

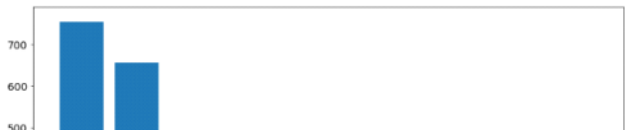


-> Bar Graph

plt.bar(x= —, y= —)

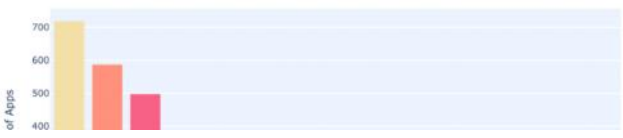
```
plt.figure(figsize=(14,8))
plt.xticks(fontsize=14, rotation=45)
plt.yticks(fontsize=14)
plt.ylabel('Nr of Sets', fontsize=14)
plt.xlabel('Theme Name', fontsize=14)

plt.bar(merged_df.name[:10], merged_df.set_count[:10])
```

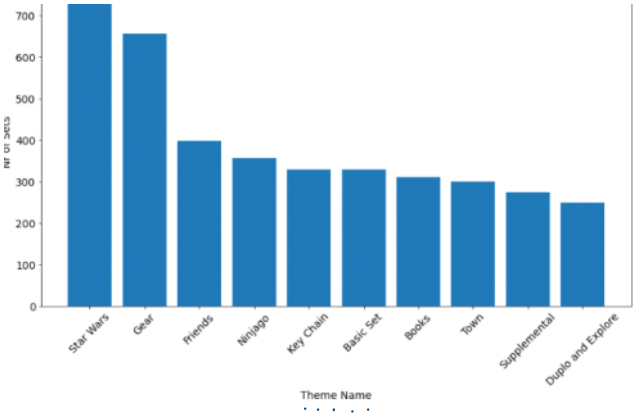
-> overall chart size
-> x-axis customization
y-axis

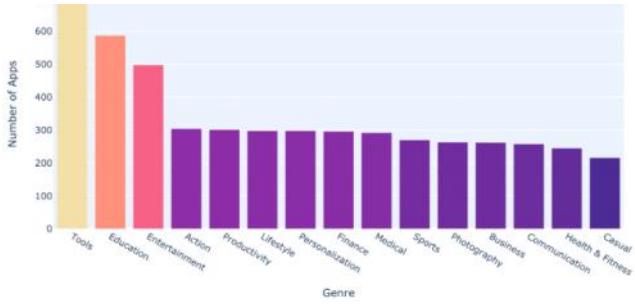



```
bar = px.bar(x = num_genres.index[:15], # index = category name
             y = num_genres.values[:15], # count
             title='Top Genres',
             hover_name=num_genres.index[:15],
             color=num_genres.values[:15],
             color_continuous_scale='Agsunset')

bar.update_layout(xaxis_title='Genre',
yaxis_title='Number of Apps',
coloraxis_showscale=False)

bar.show()
```

Top Genres







→ Data Visualization & Style Helper

⇒ Ticks for time Axis

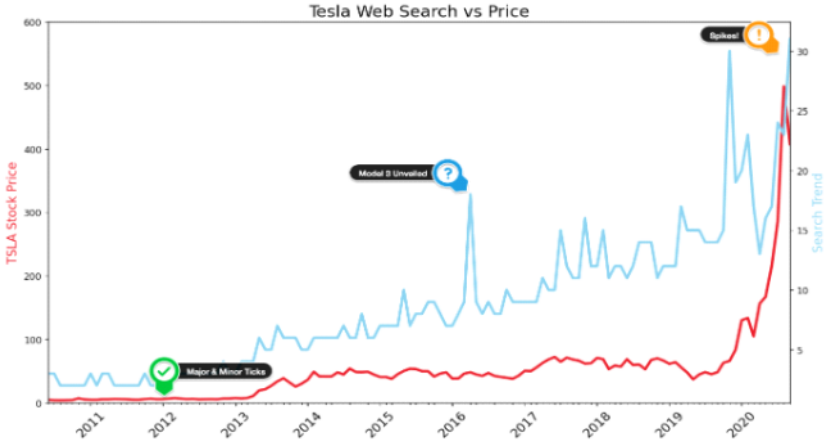
import matplotlib.dates as mdates

y = mdates.YearLocator()

m = mdates.MonthLocator()

y_fmt = mdates.DateFormatter('%y')

1. ax1.xaxis.set_major_locator(years)
2. ax1.xaxis.set_major_formatter(years_fmt)
3. ax1.xaxis.set_minor_locator(months)



→ Pie chart

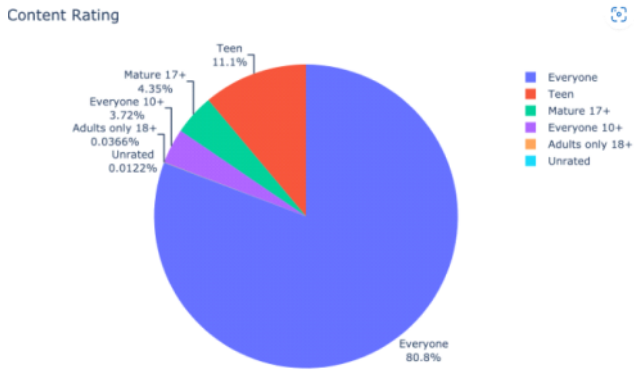
```
import plotly.express as px
```

```
fig = px.pie(labels= df.index, values = df.values)
fig.show()
```

labels= df.index → Size of all values

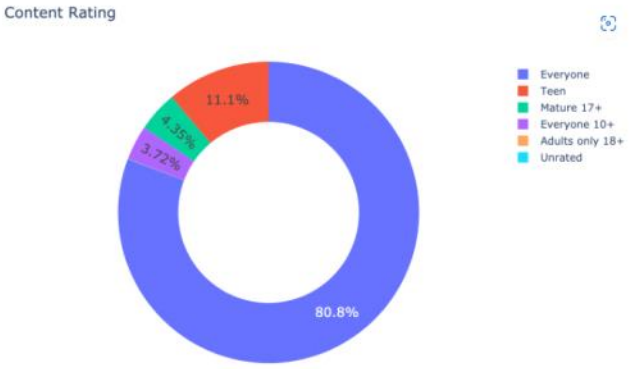
values = df.values → Size of individual value

```
fig = px.pie(labels=ratings.index,
values=ratings.values,
title="Content Rating",
names=ratings.index,
)
fig.update_traces(textposition='outside', textinfo='percent+label')

fig.show()
```



Donut

```
fig = px.pie(labels=ratings.index,
values=ratings.values,
title="Content Rating",
names=ratings.index,
hole=0.6,
)
fig.update_traces(textposition='inside', textfont_size=15,
textinfo='percent')

fig.show()
```



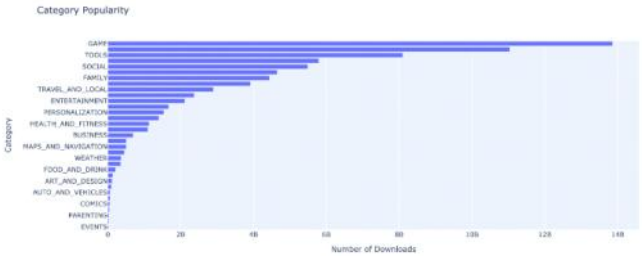
-> Bar Graph

Px. bar (x = ,

y =)

```
h_bar = px.bar(x = category_installs.Installs,
               y = category_installs.index,
               orientation='h',
               title='Category Popularity')

h_bar.update_layout(xaxis_title='Number of Downloads',
yaxis_title='Category')
h_bar.show()
```

customization




```
cat_cntry_bar = px.bar(x=merged_df.cat_prize,
                       y=merged_df.birth_country_current,
                       color=merged_df.category,
                       orientation='h',
                       title='Top 20 Countries by Number of Prizes and
Category')

cat_cntry_bar.update_layout(xaxis_title='Number of Prizes',
                            yaxis_title='Country')
cat_cntry_bar.show()
```

df-free_vs_paid :

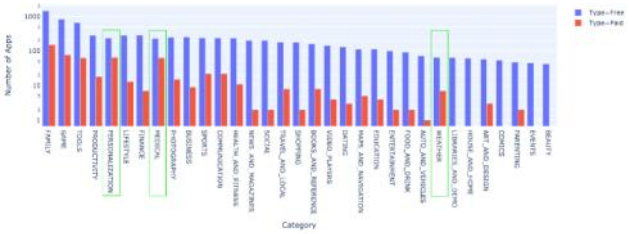
Categories Type App

```
g_bar = px.bar(df_free_vs_paid,
               x='Category',
               y='App',
               title='Free vs Paid Apps by Category',
               color='Type',
```

df-free_vs_paid

Categories	Type	App
GAME	FREE	834
.	.	.
.	.	.
.	.	.
GAME	PAID	93

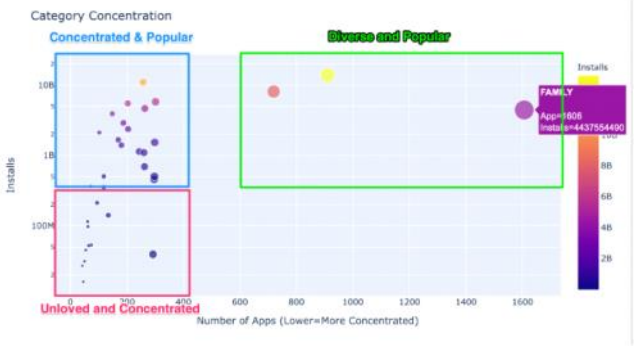
```
            x='Category',
            y='App',
            title='Free vs Paid Apps by Category',
            color='Type',
            barmode='group')

g_bar.update_layout(xaxis_title='Category',
                    yaxis_title='Number of Apps',
                    xaxis={'categoryorder':'total descending'},
                    yaxis=dict(type='log'))

g_bar.show()
```



→ Scatter plot

px.scatter (data,
x= ,
y=)

```
scatter = px.scatter(cat_merged_df, # data
                     x='App', # column name
                     y='Installs',
                     title='Category Concentration',
                     size='App',
                     hover_name=cat_merged_df.index,
                     color='Installs')

scatter.update_layout(xaxis_title="Number of Apps (Lower=More Concentrated)",
                      yaxis_title="Installs",
                      yaxis=dict(type='log'))

scatter.show()
```



→ Box Plot

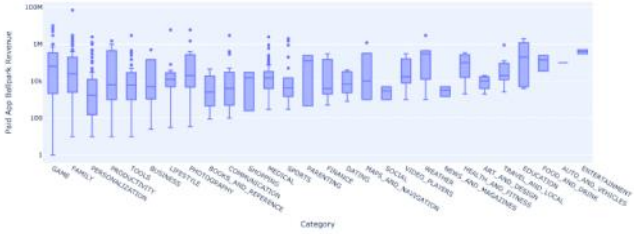
px.box (data, x= , y=)

```
box = px.box(df_apps_clean,
             y='Installs',
             x='Type',
             color='Type',
             notched=True,
             points='all',
             title='How Many Downloads are Paid Apps Giving Up?')

box.update_layout(yaxis=dict(type='log'))

box.show()
```

→ defines the data type that is to be plotted on the axis


```
df_paid_apps = df_apps_clean[df_apps_clean['Type'] == 'Paid']
box = px.box(df_paid_apps,
             x='Category',
             y='Revenue_Estimate',
             title='How Much Can Paid Apps Earn?')

box.update_layout(xaxis_title='Category',
                  yaxis_title='Paid App Ballpark Revenue',
                  xaxis={'categoryorder':'min ascending'},
                  yaxis=dict(type='log'))


box.show()
```

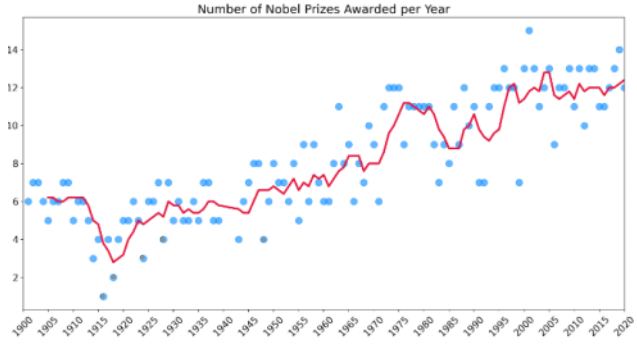
→ Sort categories on the basis of no. of downloads



→ Scatter + Graph

```
plt.figure(figsize=(16,8), dpi=200)
plt.title('Number of Nobel Prizes Awarded per Year', fontsize=18)
plt.yticks(fontsize=14)
plt.xticks(ticks=np.arange(1900, 2021, step=5),
           fontsize=14,
           rotation=45)

ax = plt.gca() # get current axis
ax.set_xlim(1900, 2020)

ax.scatter(x=prize_per_year.index,
           y=prize_per_year.values,
           c='dodgerblue',
           alpha=0.7,
           s=100,)

ax.plot(prize_per_year.index,
        moving_average.values,
        c='crimson',
        linewidth=3,)

plt.show()
```

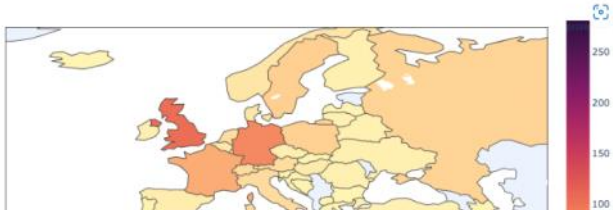
0 → transperant
1 → opaque

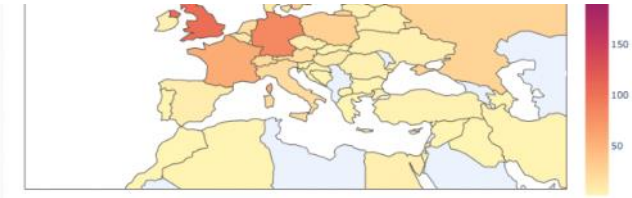


→ Choropleth Map

px.choropleth (data,
locations='ISO',
color = ' ',)

```
world_map = px.choropleth(df_countries,
                          locations='ISO',
                          color='prize',
                          hover_name='birth_country_current',
color_continuous_scale=px.colors.sequential.matter)

world_map.update_layout(coloraxis_showscale=True,)

world_map.show()
```

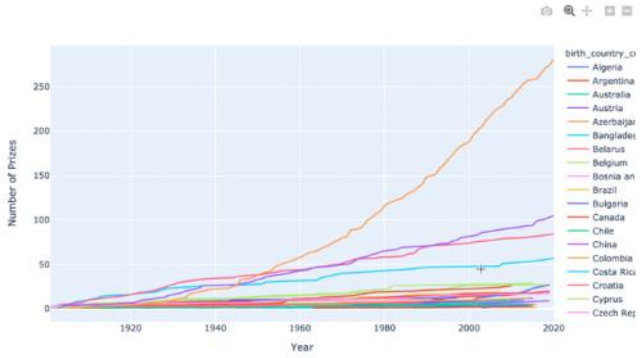




→ Line Graph

px.line(data, x= , y=)

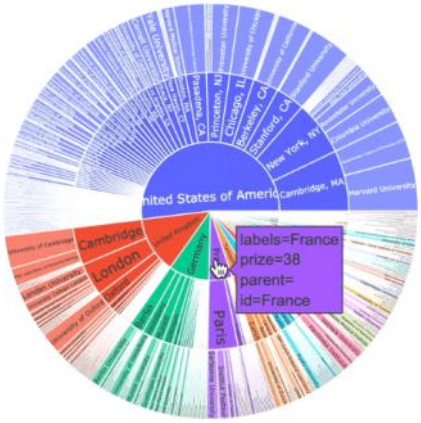
```
l_chart = px.line(cumulative_prizes,
                  x='year',
                  y='prize',
                  color='birth_country_current',
                  hover_name='birth_country_current')

l_chart.update_layout(xaxis_title='Year',
                      yaxis_title='Number of Prizes')

l_chart.show()
```



→ Sunburst Chart

px. Sunburst(data , Path= ['c1','c2'], values=' ')

```
burst = px.sunburst(country_city_org,
                    path=['organization_country', 'organization_city',
'organization_name'],
                    values='prize',
                    title='Where do Discoveries Take Place?',
                   )

burst.update_layout(xaxis_title='Number of Prizes',
                    yaxis_title='City',
                    coloraxis_showscale=False)

burst.show()
```

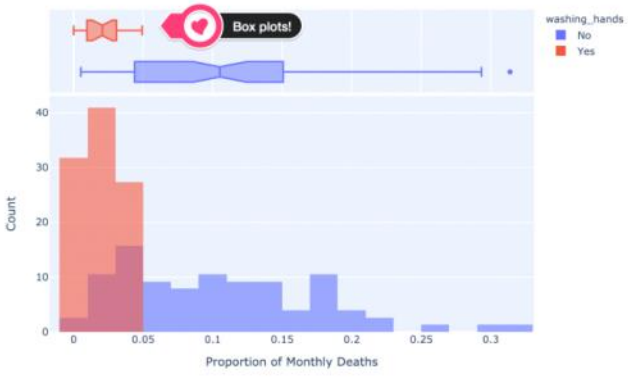


> Histogram

px.histogram(data, x=' ', nbins=)

```
hist = px.histogram(df_monthly,
                    x='pct_deaths',
                    color='washing_hands',
                    nbins=30,
                    opacity=0.6,
                    barmode='overlay',
                    histnorm='percent',
                    marginal='box',)

hist.update_layout(xaxis_title='Proportion of Monthly Deaths',
                   yaxis_title='Count',)

hist.show()
```

I quite like how in plotly we can display our box plot from earlier at the top.



NumPy

06 April 2024 17:42

Day-77

```
import numpy as np
```

1-D arr

```
arr = np.array([1,2,3])
```

arr.shap return shape of arr

arr.ndim " no. of dimensions of arr

2-D array

```
arr = np.array([[1,2,3],
                [4,5,6]])
```

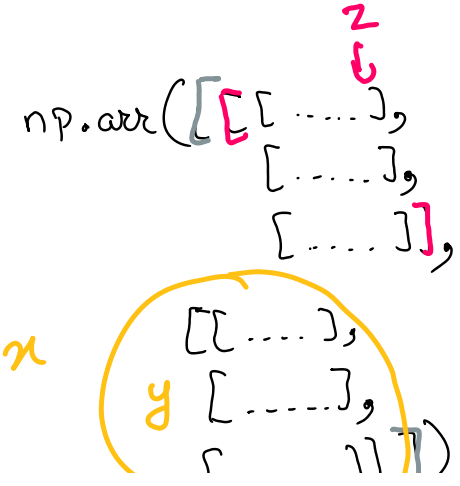
arr.shape[0] rows no.

arr.shape[1] col

arr[r,c] r/c = : for accessing all the values present

arr[0,:] -> 1,2,3

→ n-D Array




```
mystery_array = np.array([[[0, 1, 2, 3],
                           [4, 5, 6, 7]],

                          [[7, 86, 6, 98],
                           [5, 1, 0, 4]],

                          [[5, 36, 32, 48],
                           [97, 0, 27, 18]]])
```

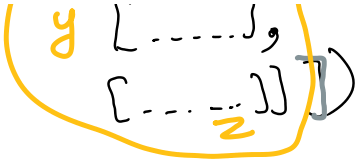
```
# Axis 0: 3rd element. Axis 1: 2nd Element. Axis 3: 4th Element
mystery_array[2, 1, 3]
```

```
18
```

```
# Retrieve all the elements on the 3rd axis that are at
# position 2 on the first axis and position 1 on the second axis.
mystery_array[2, 1, :]
```

```
array([97,  0, 27, 18])
```

```
# All the first elements on axis number 3
mystery_array[:, :, 0]
```



```
# All the first elements on axis number 3
mystery_array[:, :, 0]

array([[ 0,  4],
       [ 7,  5],
       [ 5, 97]])
```

arr[x, y, z] x → data enclosed in [[ ]]

y & z normal row & col

z

→ Arrange used generate array of evenly spaced values

or np.arrange(x, y) x - start
y - end

or np[start:end] Sub array of specified range

or np[-3:] last 3 elements

or np[4:] except 1st 4

or np[::2] every 2nd value

np.flip(arr)
or
arr[::-1] reverse the order of arr.

np.nonzero(arr) returns a tupple of non-zero values.

(or)

np.random.random((3,3,3))

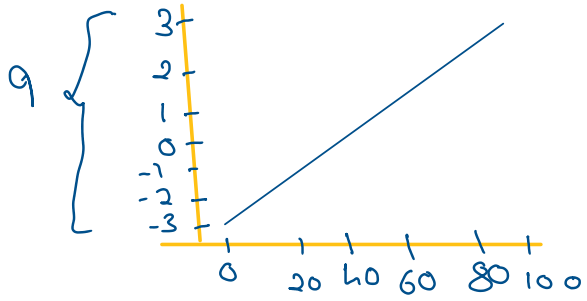
dimen row col

y = np.linespace(start=-3, stop=3, num=9)

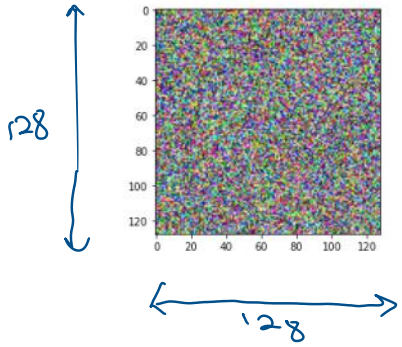
plt.plot(x,y)

plt.show()

x = np.lineplot(0,100, num=9)




```
noise = np.random.random((128,128,3))
print(noise.shape)
plt.imshow(noise)
```



imshow( ) → will display img ; RGB values

Linear Algebra with Vectors

NumPy is designed to do math (and do it well!). This means that NumPy will treat vectors, matrices and tensors in a way that a mathematician would expect. For example, if you had two vectors:

```
v1 = np.array([4, 5, 2, 7])
v2 = np.array([2, 1, 3, 3])
```

And you add them together

```
v1 + v2
```

The result will be a ndarray where all the elements have been added together.

```
array([ 6, 6, 5, 10])
```

In contrast, if we had two Python Lists

```
list1 = [4, 5, 2, 7]
list2 = [2, 1, 3, 3]
```

adding them together would just concatenate the lists.

```
list1 + list2
# output: [4, 5, 2, 7, 2, 1, 3, 3]
```

Multiplying the two vectors together also results in an element by element operation:

```
v1 * v2
```

#output: [8,5,6,21]

→ Matrix Multiplication

np.matmul(m1,m2)

/or/

m1 @ m2

→ Image

```
from scipy import misc # contains an image of a racoon!
from PIL import Image # for reading image files
```

plt.imshow(img) used to display image

type(img) numpy.ndarray

type(img) numpy.ndarray

img/255 Converts value in sRGB

plt.imshow(img_gray, cmap='gray') → used to specify that image is gray

---

Custom image

file = 'my_img.jpg'
my_img = Image.open(file)] open img like

img_arr = np.array(my_img)] Convert img into numpy array

plt.show(img_arr)] plot the img

Day - 78

Seaborn

07 April 2024 10:56

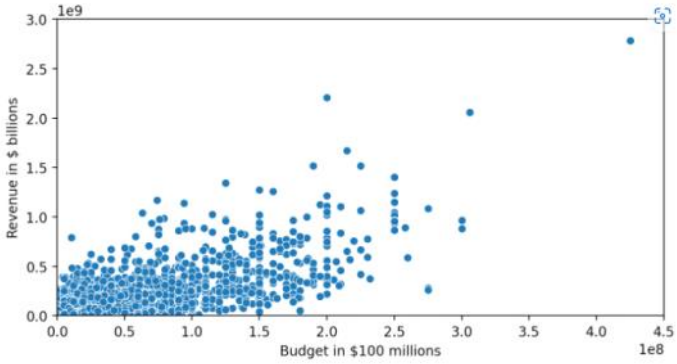
Day-78

import Seaborn as sns

-> Scatter Plot

sns.scatterplot(data = data,
x = X,
y = Y)

```
plt.figure(figsize=(8,4), dpi=200)

ax = sns.scatterplot(data=data_clean,
                     x='USD_Production_Budget',
                     y='USD_Worldwide_Gross')

ax.set(ylim=(0, 3000000000),
       xlim=(0, 450000000),
       ylabel='Revenue in $ billions',
       xlabel='Budget in $100 millions')

plt.show()
```



-> Bubble ~~Plot~~ Chart

```
plt.figure(figsize=(8,4), dpi=200)

# set styling on a single chart
with sns.axes_style('darkgrid'):
  ax = sns.scatterplot(data=data_clean,
                       x='USD_Production_Budget',
                       y='USD_Worldwide_Gross',
```

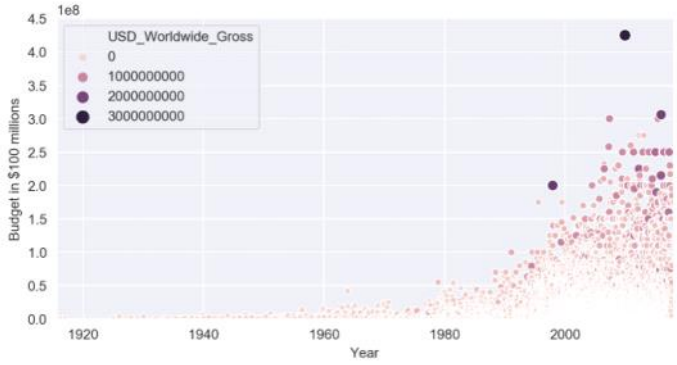
-> used to color grid

-> color of bubble

```
ax = sns.scatterplot(data=data_clean,
                     x='USD_Production_Budget',
                     y='USD_Worldwide_Gross',
                     hue='USD_Worldwide_Gross',
                     size='USD_Worldwide_Gross')

ax.set(ylim=(0, 3000000000),
       xlim=(0, 450000000),
       ylabel='Revenue in $ billions',
       xlabel='Budget in $100 millions')
```

→ color of bubble

→ determines bubble size

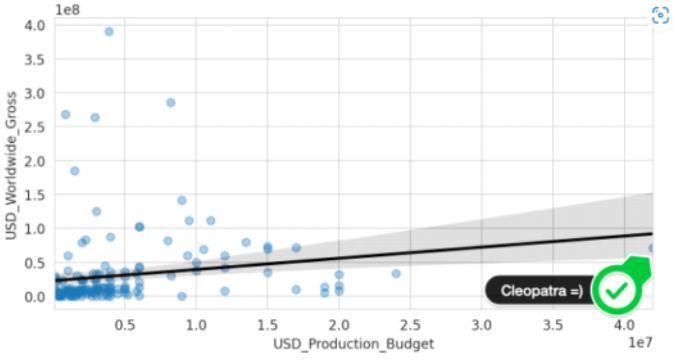


→ Linear Regression

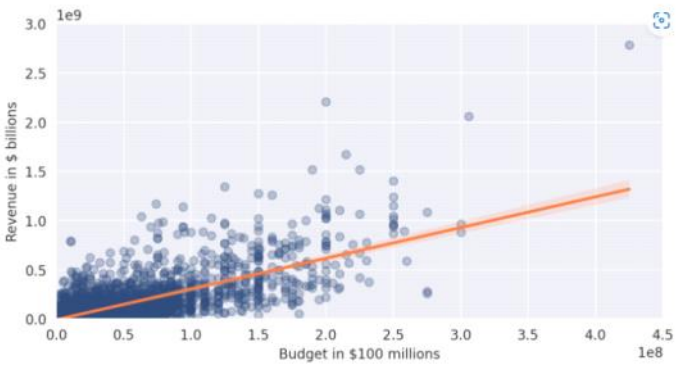
```
sns.regplot(data=data,
            x=X,
            y=Y)
```

```
plt.figure(figsize=(8,4), dpi=200)
with sns.axes_style("whitegrid"):
  sns.regplot(data=old_films,
              x='USD_Production_Budget',
              y='USD_Worldwide_Gross',
              scatter_kws = {'alpha': 0.4},
              line_kws = {'color': 'black'})
```

→ adjust transparency




```
plt.figure(figsize=(8,4), dpi=200)
with sns.axes_style('darkgrid'):
  ax = sns.regplot(data=new_films,
                   x='USD_Production_Budget',
                   y='USD_Worldwide_Gross',
                   color='#2f4b7c',
                   scatter_kws = {'alpha': 0.3},
                   line_kws = {'color': '#ff7c43'})

  ax.set(ylim=(0, 3000000000),
         xlim=(0, 450000000),
         ylabel='Revenue in $ billions',
         xlabel='Budget in $100 millions')
```

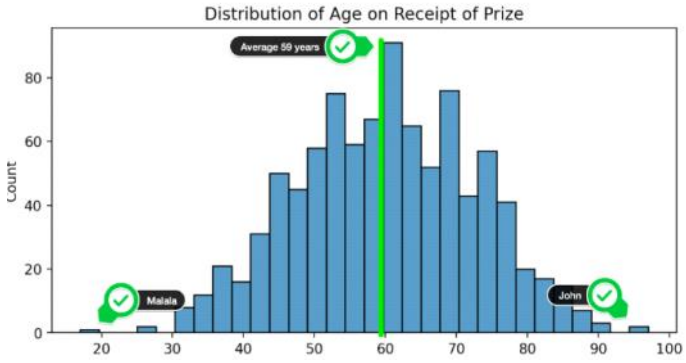


Day. 79

→ Histogram

Sns.histplot (data , x= , bins =)

```
plt.figure(figsize=(8, 4), dpi=200)
sns.histplot(data=df_data,
             x=df_data.winning_age,
             bins=30)
plt.xlabel('Age')
plt.title('Distribution of Age on Receipt of Prize')
plt.show()
```

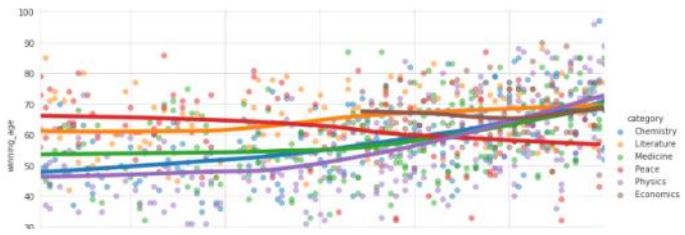


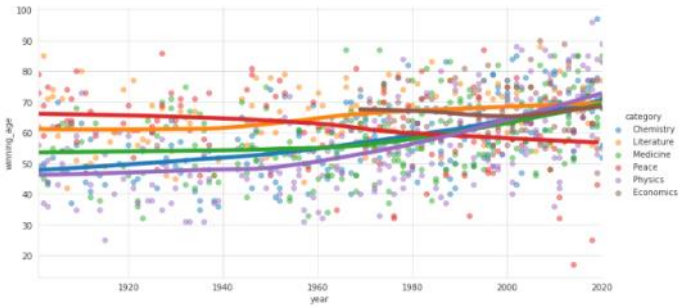
→ Lmplot

Scatter plot + Regression

```
with sns.axes_style("whitegrid"):
    sns.lmplot(data=df_data,
               x='year',
               y='winning_age',
               hue='category',
               lowess=True,
               aspect=2,
               scatter_kws={'alpha': 0.5},
               line_kws={'linewidth': 5})

plt.show()
```

→ row = 'category' ; then we would have got Seprate values for all the Categories.




100
90
80
70
60
50
40
30
20
winning_age
1920
1940
1960
1980
2000
2020
year
category
Chemistry
Literature
Medicine
Peace
Physics
Economics

Day - 78

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()

x = pd.DataFrame(data, columns=['col1'])

y = pd.DataFrame(data, columns=['col2'])
```

Find Best Line

```
reg.fit(x,y)
```

```
# Theta zero
regression.intercept_

array([-8650768.00661027])

# Theta one
regression.coef_

array([[3.12259592]])
```

Model Fitting Test -> Root Mean Square Values

```
reg.score(x,y)
```

make Predictions

$y = \theta_0 + \theta_1 x$

```
y = reg.intercept_[0] +
        reg.coef_[0,0] * x
```

y = reg.intercept_ +

reg.coef_[0,0] * x