Simulación
Tema: Simulación de Eventos Examen.

Enunciado:

Diseñe y desarrolle un modelo y/o script que permita simular el siguiente caso real:

- Obtener datos de tendencia de twitter o facebook(crawler o webscraping), para ello se puede obtener a través del API [4].

  Title: Titulo del Post/Twitter

  Word count: la cantidad de palabras del artículo,
  # of Links: los enlaces externos que contiene,
  # of comments: cantidad de comentarios,
  # Shares: compartidos.
  HashTag
  Etc.

  En base a ello, se pretende proponer y generar una predicción de cuántas veces será compartido un post/twitter utilizando regresión [2].

- Selenium utiliza el protocolo WebDriver para controlar un navegador web, como Chrome, Firefox o Safari. El navegador puede ejecutarse de forma local o remota.

Por esta razón debemos instalar el Chrome y agregar el siguiente driver para utilizar de forma remota.

```
from selenium import webdriver

DRIVER_PATH = 'chromedriver.exe'
driver = webdriver.Chrome(executable_path=DRIVER_PATH)
driver.get('https://twitter.com/LassoGuillermo')
h1 = driver.find_element_by_class_name('css-1dbjc4n')
```

- Debemos definir de que pagina vamos a obtener la información.

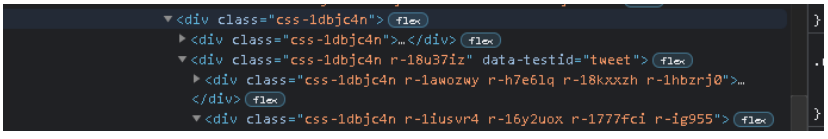
En este caso vamos a obtener los tweets del presidente con el link:

```
driver.get('https://twitter.com/LassoGuillermo')
```

- Guardamos toda la información obtenido por selenium dentro de h1

h1 tiene toda la información que se presenta dentro del navegador.
con el método que incorpora selenium de **find_element_by_class_name()** realizamos la búsqueda del siguiente nombre de class “'css-1dbjc4n”

```
h1 = driver.find_element_by_class_name('css-1dbjc4n')
```



- Generamos las listas donde vamos a guardar la información.

```
list_dataset =list()
list_contenido=list()
list_reply=list()
list_retweet=list()
list_like=list()
```

- Una vez obtenida la información debemos darnos en cuenta que se debe realizar un scroll dentro de la página para que nuestro h1 obtenga los nuevos tweets por lo cual generamos un for que realice un scroll 50 veces dentro de la página.
- Ahora que obtenemos toda la información debemos filtrar por las siguientes etiquetas.
  - Para Obtener el contenido del Tweet
    - `driver.find_elements_by_xpath("//div[@dir='auto']")`
  - Para Obtener la cantidad de comentarios
    - `driver.find_elements_by_xpath("//div[@data-testid='reply']")`
  - Para Obtener la cantidad de retweets
    - `driver.find_elements_by_xpath("//div[@data-testid='retweet']")`
  - Para Obtener la cantidad de likes
    - `driver.find_elements_by_xpath("//div[@data-testid='like']")`

Agregamos la información de cada lista para generar un dataset.

```
for i in range(1,50):
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")

    tweet_textos = driver.find_elements_by_xpath("//div[@dir='auto']")
    for aux in range(len(tweet_textos)):
        if(tweet_textos[aux].text == '·'):
            list_contenido.append(tweet_textos[aux+1].text)

    tweet_replys = driver.find_elements_by_xpath("//div[@data-testid='reply']")
    for tweet_reply in tweet_replys:
        list_reply.append(tweet_reply.text)

    tweet_retweets = driver.find_elements_by_xpath("//div[@data-testid='retweet']")
    for tweet_retweet in tweet_retweets:
        list_retweet.append(tweet_retweet.text)

    tweet_likes = driver.find_elements_by_xpath("//div[@data-testid='like']")
    for tweet_like in tweet_likes:

        list_like.append(tweet_like.text)


    time.sleep(3)

    print(i)

for j in range(len(list_contenido)) :
    list_dataset.append([list_contenido[j],list_reply[j],list_retweet[j],list_like[j]])
    print(list_contenido[j]," likes = ",list_like[j])
    print("**********************************************************************")
print(len(list_dataset))
```

Resultado

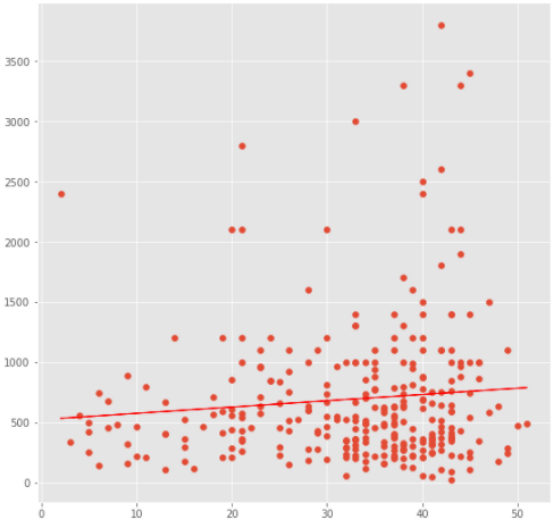
	contenido	reply	retweet	like
0	Nuestro proyecto de Ley Orgánica de Libre Expr...	268	597	2 mil
1	Hoy envié a la Asamblea Nacional el proyecto d...	498	1000	5,1 mil
2	Organización gubernamental de Ecuador	169	558	4,5 mil
3	Hoy tuve la oportunidad de dialogar con \n@leo...	581	880	4,3 mil
4	Fue un honor recibir a representantes de \n@US...	207	639	2,9 mil
..	...	...	...	...
295	Los subsidios SON PARA LOS POBRES. No son para...	81	571	2,1 mil
296	En 6 días tenemos una responsabilidad históric...	425	2100	5,3 mil
297	Las 4 claves de mi Plan de Gobierno: fortalece...	221	549	1,7 mil
298	¡Estamos en el Distrito 1 de Guayas con \n@Jua...	175	354	1,2 mil
299	Esta noche \n@soyfdelrincon\n entrevista en Co...	501	1000	3,6 mil

[300 rows x 4 columns]

Regresión Lineal

- Para la generación de la regresión agregamos una nueva columna que contenga la información de la cantidad de palabras que tiene cada tweet y la cantidad de retweets.

```
# Asignamos nuestra variable de entrada X para entrenamiento y las etiquetas Y.
dataX =filtered_data[["cantidad_palabras"]]
X_train = np.array(dataX)
y_train = filtered_data['retweet'].values

# Creamos el objeto de Regresión Linear
regr = linear_model.LinearRegression()

# Entrenamos nuestro modelo
regr.fit(X_train, y_train)

# Hacemos las predicciones que en definitiva una línea (en este caso, al ser 2D)
y_pred = regr.predict(X_train)


# Este es el valor donde corta el eje Y (en X=0)
print('Independent term: \n', regr.intercept_)
# Error Cuadrado Medio
print("Mean squared error: %.2f" % mean_squared_error(y_train, y_pred))
# Puntaje de Varianza. El mejor puntaje es un 1.0
print('Variance score: %.2f' % r2_score(y_train, y_pred))
```



Predicción

- Para realizar la predicción ingresamos la cantidad de palabras que tendría una publicación para obtener como salida la cantidad de retweets que tendría dicha publicación.

Prediccion

In [557]:
```
# Quiero predecir cuántos "Shares" voy a obtener por un artículo con 50 palabras,
# según nuestro modelo, hacemos:
y_Dosmil = regr.predict([[50]])
print('Con una publicacion de 50 palabras obtendremos un total de compartidos de :',int(y_Dosmil))
```

```
Con una publicacion de 50 palabras obtendremos un total de compartidos de : 117
```

Posteriormente se debe seguir un procesos de vacunación en los recintos electorales que se describe a continuación (**Tomar el proceso para el proyecto final**):

- Solo se va a tener en cuanta uno de los recintos electorales (investigar datos de cuantas personas asisten a votar).
- Tomar los resultados de la regresión para la vacuna según la llegada.
- Se tiene una promedio que el 80% de personas realizaran el proceso de vacunación dentro del Ecuador.
- Dentro del procesos se tiene que alrededor del 5% - 10% no podrán vacunarse.
- Las personas solo tiene un recinto electoral para realizar el proceso.
- Las personas realizan la primera vacuna y 30 días después la segunda vacuna.
- La persona se acerca a la mesa y hacen fila en caso de ser necesario para recibir la vacuna.
- Realiza la vacunación en un tiempo aleatorio entre 5 a 10 minutos.
- Debe esperar 20 minutos dentro del establecimiento para verificar que no tenga problemas de salud.
- La persona recibe su certificado de vacunación y la fecha de la próxima vacuna entre 2 – 3 minutos.
- La persona sale del recinto electoral.
- Regresan para la próxima fecha y se repite el ciclo.

El proceso de simulación desarrollado deberá considerar los siguientes aspectos:

- Se debe establecer un modelo basado en modelos matemáticos para la predicción del numero de veces que se compartirá o la tendencia del presidente basada en redes sociales.
- El programa deberá generar gráficas que indiquen la ecuación matemática de las tendencias .
- Deben calcularse las siguientes métricas del sistema de simulación de eventos discretos :
  - Total de de personas que realizaron el proceso de vacunación.
  - Grafico del porcentaje de personas que no recibieron la vacuna.
  - El tiempo promedio de espera.

- Generamos nuestras variables globales
  - En este caso nos interesa saber la cantidad de mesas que tendra el recinto electoral.
  - Definimos la espera obligatoria de 20 minutos después de ser vacunado.
  - La cantidad de tiempo que labora ese recinto electoral.

```
# Cantidad de mesas-casetas de vacunacion dentro del recinto.
CANTIDAD_MESA = 20
# Espera Obligatoria.
ESPERA_OBLIGATORIA = 20
# Tiempo de la simulacion 8H = 480 M
TIEMPO_LABORAL = 480
#Generamos una fila
FILA_INGRESO=20
```

- Definamos los diccionarios de datos en los cuales vamos a guardar la información de cada persona para diseñar un gráfico de cómo es la tendencia dentro de la simulación.
    - **Diccionario_Personas_Tiempo_Vacunacion** es el tiempo de vacunación que le tomó a una persona hasta ingresar a la mesa y ser vacunado sin contar con la espera obligatoria y el certificado.
    - **Diccionario_Tiempo_Espera** es el tiempo en general consta del tiempo que le tomo ingresar a la mesa, vacunación , espera obligatoria y la entrega del certificado.
    - **Diccionario_Personas_No_Vacunadas** es la información de las personas que no se vacunaron por tener algún problema previo.

```
# Dicciccionario con los tiempo de vacunación por persona
DICCIONARIO_PERSONA_TIEMPO_VACUNACION = {}
# Diccionario con los tiempos de espera por persona.
DICCIONARIO_TIEMPO_DE_ESPERA = {}
# Diccionario con los Personas No vacunadas.
DICCIONARIO_PERSONAS_NO_VACUNADAS = {}
```

Definimos las funciones para guardar la información dentro de cada diccionario.

- Dentro de esos debemos obtener la id de la persona
- Tiempo que le toma a una persona realizar la vacunación o el tiempo de espera hasta llegar al certificado.

In [9]:
```
# Obtener el tiempo total que le toma a una persona hasta ser vacunada.
def tiempo_hasta_vacunacion(persona, tiempoPersona):
     # Dicciccionario con los tiempo de vacunación por persona
    global DICCIONARIO_PERSONA_TIEMPO_VACUNACION
    DICCIONARIO_PERSONA_TIEMPO_VACUNACION[int(persona[persona.find(" ")+1:len(persona)])] = tiempoPersona
```

In [10]:
```
# Obtener el tiempo total de espera de una persona
def tiempo_espera_vacunacion(persona, tiempoEspera):
    # Diccionario el tiempo de espera por persona.
    global DICCIONARIO_TIEMPO_DE_ESPERA
    DICCIONARIO_TIEMPO_DE_ESPERA[int(persona[persona.find(" ")+1:len(persona)])] = tiempoEspera
```

Generamos la principal que tendrá la espera obligatoria , cant mesas, tiempo de vacunación.

```
class Principal(object):

    def __init__(self, environment,esperaObligatoria, cantMesas, tiempoVacunacion):

        # Guardamos como variable el entorno de ejecucion
        self.env = environment

        # Variable para el tiempo de espera obligatorio.
        self.esperaObligatoria = esperaObligatoria

        #Creamos el recurso que representa las mesas dentro del recinto
        self.mesas = simpy.Resource(environment, cantMesas)

        # Variable para el tiempo de vacunacion
        self.tiempoVacunacion = tiempoVacunacion
```

- La principal contará con la función de espera obligatoria ,el tiempo de vacunación ,certificado.

```python
#Funcion de Espera Obligatoria
def esperarObligatoria(self):
    yield self.env.timeout(self.esperaObligatoria)

#Funcion random del tiempo que le toma realizar la vacunacion.
def vacunar(self, persona):
    tiempo_randomico=random.randint(5, 10)
    yield self.env.timeout(tiempo_randomico)

#Funcion de vacunacion por persona
def certificado(self):
    tiempo_randomico=random.randint(2, 3)
    yield self.env.timeout(tiempo_randomico)
```

Cada una de las funciones utiliza un método random el cual está entre los valores que se establecen

- Espera Obligatoria es siempre de 20 Minutos.
- Vacunar es un número randómico entre 5 a 10 minutos
- Certificado es entre 2 a 3 minutos.

Simulacion

```python
def recinto_borja(env, personaId, gene_vacunacion):
    # Usamos el reloj de la simulacion (env.now()) para indicar la llegada cada persona.
    print('Inicio de la persona [%s] llega al recinto Borja  %.2f.' % (personaId, env.now))

    # Guardamos El Tiempo de inicio para calcular el tiempo que nos tomara la vacunacion por persona.
    inicioEspera = env.now

    #ALeatorio de personas que ingresan a vacunarse

    # De la cola sacamos un total de 17 a 35 personas que  no cumplen o tienen problemas para ser vacunada
    control=random.randint(5, 10)
    #print(control)
    if random.randint(1,100) <= control :
        #print("Guardar Vacuna De :",personaId[personaId.find(" ")+1:len(personaId)],"presenta problemas De Salud Tiempo")
        DICCIONARIO_PERSONAS_NO_VACUNADAS[personaId[personaId.find(" ")+1:len(personaId)]] = (personaId, env.now)
        print('La persona [%s] no puede vacunarse '%(personaId))

    else:

        # Especificamos el uso de una mesa disponible con request().
        with gene_vacunacion.mesas.request() as mesa:
            #global VACUNADOS
            # Iniciamos el tiempo apenas ingresa.
            inicio = env.now

            # Agregramos a la persona en una mesa disponible
            yield mesa
            print('La persona [%s] pasa a la mesa de vacunacion : %.2f.' % (personaId, env.now))


            # Realizamos la vacunacion
            yield env.process(gene_vacunacion.vacunar(personaId))


            # Mandamos a guardar la informacion del tiempo que le tomo la vacunacion a la persona.
            tiempo_hasta_vacunacion(personaId, env.now-inicio)


            print('La persona [%s] fue vacunada a las %.2f.' % (personaId, env.now))



            #Realizamos el check de salud con una espera de 20m
            yield env.process(gene_vacunacion.esperarObligatoria())


            print('La persona [%s] pasa a recibir su certificado a las %.2f.' % (personaId, env.now))
            #Realizamos la generacion del certificado.
            yield env.process(gene_vacunacion.certificado())

            # La persona despues de ser vacuna sale del centro de vacunación
            print('Salida de la persona [%s] sale del recinto a las %.2f.' % (personaId, env.now))
            tiempo_espera_vacunacion(personaId, env.now-inicioEspera)
            print("Tiempo Total De",personaId,"=",env.now-inicioEspera)
```

- Dentro de esta función realizamos el ingreso de las personas y calculamos un random que nos permita definir la cantidad de personas que no se van a vacunar y estos se encuentran dentro de la fila.

```
def recinto_borja(env, personaId, gene_vacunacion):
    # Usamos el reloj de la simulacion (env.now()) para indicar la llegada cada persona.
    print('Inicio de la persona [%s] llega al recinto Borja  %.2f.' % (personaId, env.now))

    # Guardamos EL Tiempo de inicio para calcular el tiempo que nos tomara la vacunacion por persona.
    inicioEspera = env.now

    #Aleatorio de personas que ingresan a vacunarse

    # De la cola sacamos un total de 17 a 35 personas que  no cumplen o tienen problemas para ser vacunada
    control=random.randint(5, 10)
    #print(control)
    if random.randint(1,100) <= control :
        #print("Guardar Vacuna De :",personaId[personaId.find(" ")+1:len(personaId)],"presenta problemas De Salud Tiempo")
        DICCIONARIO_PERSONAS_NO_VACUNADAS[personaId[personaId.find(" ")+1:len(personaId)]] = (personaId, env.now)
        print('La persona [%s] no puede vacunarse '%(personaId))
```

- Si no está dentro de los valores entonces ingresa a la simulación de la vacunación.

```
  # Especificamos el uso de una mesa disponible con request().
  with gene_vacunacion.mesas.request() as mesa:
      #global VACUNADOS
      # Iniciamos el tiempo apenas ingresa.
      inicio = env.now

      # Agregramos a la persona en una mesa disponible
      yield mesa
      print('La persona [%s] pasa a la mesa de vacunacion : %.2f.' % (personaId, env.now))


      # Realizamos la vacunacion
      yield env.process(gene_vacunacion.vacunar(personaId))


      # Mandamos a guardar la informacion del tiempo que le tomo la vacunacion a la persona.
      tiempo_hasta_vacunacion(personaId, env.now-inicio)


      print('La persona [%s] fue vacunada a las %.2f.' % (personaId, env.now))



      #Realizamos el check de salud con una espera de 20m
      yield env.process(gene_vacunacion.esperarObligatoria())


      print('La persona [%s] pasa a recibir su certificado a las %.2f.' % (personaId, env.now))
      #Realizamos la generacion del certificado.
      yield env.process(gene_vacunacion.certificado())

      # La persona despues de ser vacuna sale del centro de vacunación
      print('Salida de la persona [%s] sale del recinto a las %.2f.' % (personaId, env.now))
      tiempo_espera_vacunacion(personaId, env.now-inicioEspera)
      print("Tiempo Total De",personaId,"=",env.now-inicioEspera)
```

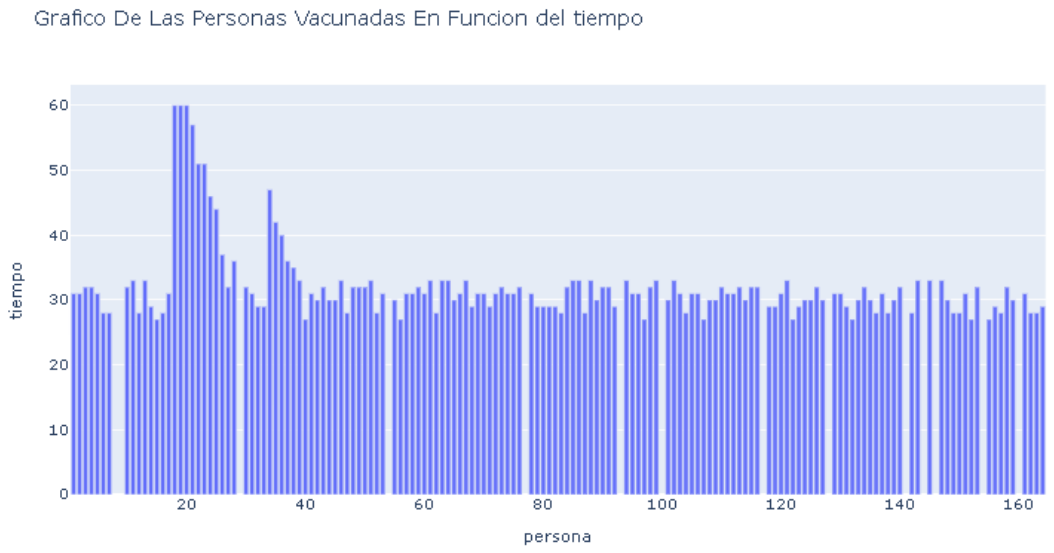
Para generar la fila de espera para el ingreso utilizamos un for con un rango que definamos.

```
def ejecutar_simulacion(env,esperaobligatoria, controlmesa,tiempoVacunacion):
    gene_vacunacion=Principal(env,esperaobligatoria, controlmesa,tiempoVacunacion)
    global PERSONAS

    # Creamos fila de  20 persona que llegan y esperan afuera hasta que abran las puertas.
    for i in range(FILA_INGRESO):
        env.process(recinto_borja(env, '%d'%(i+1), gene_vacunacion))
        PERSONAS =PERSONAS+1


    # Ejecutamos la simulacion
    while True:
        #Intervalo Llegada
        yield env.timeout(random.randint(1,5))
        i+=1
        PERSONAS=PERSONAS+1
        #print("Total:",PERSONAS)
        env.process(recinto_borja(env,'%d'%(i+1), gene_vacunacion))


print('Simulacion Vacunacion Ecuador')

# Inicializamos la semilla aleatoria
random.seed(77)

# Creamos el entorno de simulacion
env=simpy.Environment()
env.process(ejecutar_simulacion(env,ESPERA_OBLIGATORIA,CANTIDAD_MESA,TIEMPO_VACUNACION))

# Ejecutamos el proceso durante el tiempo de simulacion
env.run(until = TIEMPO_LABORAL)
```

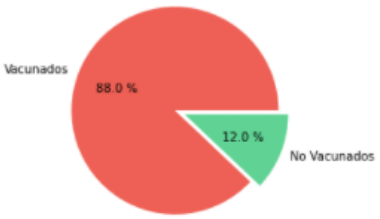
Resultados:

```
La persona [163] pasa a recibir su certificado a las 471.00.
Salida de la persona [162] sale del recinto a las 471.00.
Tiempo Total De 162 = 28
Inicio de la persona [172] llega al recinto Borja  473.00.
La persona [170] fue vacunada a las 473.00.
La persona [172] pasa a la mesa de vacunacion : 473.00.
Salida de la persona [163] sale del recinto a las 473.00.
Tiempo Total De 163 = 28
Inicio de la persona [173] llega al recinto Borja  474.00.
La persona [173] pasa a la mesa de vacunacion : 474.00.
Inicio de la persona [174] llega al recinto Borja  475.00.
La persona [169] fue vacunada a las 475.00.
La persona [171] fue vacunada a las 475.00.
La persona [174] pasa a la mesa de vacunacion : 475.00.
La persona [164] pasa a recibir su certificado a las 476.00.
Salida de la persona [164] sale del recinto a las 478.00.
Tiempo Total De 164 = 29
Inicio de la persona [175] llega al recinto Borja  479.00.
La persona [165] pasa a recibir su certificado a las 479.00.
La persona [175] pasa a la mesa de vacunacion : 479.00.
```

In [45]:
```
import pandas as pd
df = pd.DataFrame([[key, DICCIONARIO_TIEMPO_DE_ESPERA[key]] for key in DICCIONARIO_TIEMPO_DE_ESPERA.keys()], columns=['persona', 'tie
import plotly.express as px
fig = px.bar(df, x='persona', y="tiempo",title="Grafico De Las Personas Vacunadas En Funcion del tiempo")
fig.show()
```



In [41]:
```
import matplotlib.pyplot as plt

gen = [PERSONAS,PERSONAS-len(DICCIONARIO_TIEMPO_DE_ESPERA)]
colores = ["#EE6055","#60D394"]
nombres = ["Vacunados","No Vacunados"]
desfase = (0, 0.1)
plt.pie(gen, labels=nombres, autopct="%0.1f %%", colors=colores,explode=desfase)
plt.show()
```



In [42]:
```
import pandas as pd
df = pd.DataFrame([[key, DICCIONARIO_TIEMPO_DE_ESPERA[key]] for key in DICCIONARIO_TIEMPO_DE_ESPERA.keys()], columns=['persona', 'tie
```