IMPORTACION DE LIBRERIAS

In [1]:
```
import pandas as pd
import numpy as np
from datetime import timedelta, datetime
import altair as alt
import matplotlib.pyplot as plt
import scipy.integrate as spi
from scipy.optimize import minimize
from scipy.integrate import solve_ivp
```

Simulación del modelo SIR para COVID-19

In [2]:
```
I0=10
R0=0
S0 = 100000
t = 365
y0 = S0,I0,R0
```

In [3]:
```
def filter_country(df,country,start_date):
    country_df = df[df['Country/Region'] == country]
    return country_df.iloc[0].loc[start_date:]

def load_data(path_confirmed,path_recovered,country,date):
    df_confirmed = filter_country(pd.read_csv(path_confirmed),country,date)
    df_recovered = filter_country(pd.read_csv(path_recovered),country,date)
    return df_confirmed,df_recovered
```

In [4]:
```
data_confirmed,data_recovered=load_data('time_series_covid19_confirmed_global.csv','time_series
```

In [5]:
```
def loss_confirmed_recovered(point, data, recovered):
    size = len(data)
    beta, gamma = point
    def SIR(t, y):
        S = y[0]
        I = y[1]
        R = y[2]
        return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
    solution = solve_ivp(SIR, [0, size], [S0,I0,R0], t_eval=np.arange(0, size, 1), vectorized=T
    l1 = np.sqrt(np.mean((solution.y[1] - data)**2))
    l2 = np.sqrt(np.mean((solution.y[2] - recovered)**2))
    alpha = 0.1
    return alpha * l1 + (1 - alpha) * l2
```

In [6]:
```
def loss_confirmed(point, data):
    size = len(data)
    beta, gamma = point
    def SIR(t, y):
        S = y[0]
        I = y[1]
        R = y[2]
        return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
    solution = solve_ivp(SIR, [0, size], [S0,I0,R0], t_eval=np.arange(0, size, 1), vectorized=T
    return np.sqrt(np.mean((solution.y[1] - data)**2))
```

In [7]:
```
optimal_cr = minimize(
            loss_confirmed_recovered,
            [0.001, 0.001],
            args=(data_confirmed,data_recovered),
            method='L-BFGS-B',
            bounds=[(0.00000001, 0.4), (0.00000001, 0.4)])

optimal_c = minimize(
            loss_confirmed,
```

```
              [0.001, 0.001],
              args=(data_confirmed),
              method='L-BFGS-B',
              bounds=[(0.00000001, 0.4), (0.00000001, 0.4)])
```

In [8]:

```
beta_cr,gamma_cr = optimal_cr.x
beta_c,gamma_c = optimal_c.x
print("valor gamma: {}".format(gamma_cr))
print("valor beta: {}".format(beta_cr))

print("SEGUNDA FUNCION")
print("valor gamma: {}".format(gamma_c))
print("valor beta: {}".format(beta_c))
```

```
valor gamma: 0.021119722828508655
valor beta: 1.2416130843313147e-06
SEGUNDA FUNCION
valor gamma: 4.531191648057099e-06
valor beta: 4.531191648057099e-06
```

In [9]:

```
R_0= beta_cr/gamma_cr
print("Número de reproducción R_0: {}".format(R_0))

R_0= beta_c/gamma_c
print("Número de reproducción R_0: {}".format(R_0))
```

```
Número de reproducción R_0: 5.878926984095226e-05
Número de reproducción R_0: 1.0
```

In [10]:

```
def extend_index(index, new_size):
        values = index.values
        current = datetime.strptime(index[-1], '%m/%d/%y')
        while len(values) < new_size:
            current = current + timedelta(days=1)
            values = np.append(values, datetime.strftime(current, '%m/%d/%y'))
        return values

def predict(beta, gamma, data):
        predict_range = t
        new_index = extend_index(data.index, predict_range)
        size = len(new_index)
        def SIR(t, y):
            S = y[0]
            I = y[1]
            R = y[2]
            return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
        extended_actual = np.concatenate((data.values, [None] * (size - len(data.values))))
        return new_index, extended_actual, solve_ivp(SIR, [0, size], [S0,I0,R0], t_eval=np.arar
```

In [11]:

```
# Procedemos a realizar las predicciones para nuestros datos, teniendo en cuenta el beta y gamm
#A continuación realizamos dichas predicciones para la primera y segunda función

new_index, extended_actual, prediction_cr = predict(beta_cr, gamma_cr, data_confirmed)
new_index, extended_actual, prediction_c = predict(beta_c, gamma_c, data_confirmed)
```

- Después de realizar las predicciones para el modelo SIR procedemos a armar un dataframe para poder visualizar nuestro modelo terminado. A la final tendremos dos datasets ya que el uno es de la primera función, mientras que el otro es de la segunda función

In [12]:

```
df_cr = pd.DataFrame(
            {'date':[i for i in range(0,len(new_index))],
            'suceptible': prediction_cr.y[0],
            'infected': prediction_cr.y[1],
            'recovered': prediction_cr.y[2]})
df_c = pd.DataFrame(
            {'date':[i for i in range(0,len(new_index))],
            'suceptible': prediction_c.y[0],
```

```
                  'infected': prediction_c.y[1],
                  'recovered': prediction_c.y[2]})
```

In [13]:
```
df_cr.head(5)
```

Out[13]:

	date	suceptible	infected	recovered
0	0	100000.000000	10.000000	0.000000
1	1	99998.692160	11.085376	0.222464
2	2	99997.242286	12.288623	0.469091
3	3	99995.635191	13.622343	0.742466
4	4	99993.853835	15.100676	1.045489

In [14]:
```
df_c.head(5)
```

Out[14]:

	date	suceptible	infected	recovered
0	0	100000.000000	10.000000	0.000000
1	1	99994.268098	15.731844	0.000057
2	2	99985.248471	24.751382	0.000148
3	3	99971.080854	38.918857	0.000289
4	4	99948.767098	61.232389	0.000512

- Con este conjunto de datos procedemos a graficarlos, para poder ver la diferencia que existe al usar la primera función y la segunda función

In [15]:
```
alt.Chart(df_cr.melt('date')).mark_line().encode(
    x='date',
    y=alt.Y('value'),
    color='variable'
).properties(
    title='Modelo SIR para Ecuador con casos confirmados y recuperados'
).interactive()
```

Out[15]:

In [16]:
```
alt.Chart(df_c.melt('date')).mark_line().encode(
    x='date',
    y=alt.Y('value'),
    color='variable'
).properties(
    title='Modelo SIR para Ecuador solo casos confirmados'
).interactive()
```

Out[16]: