Reporte de Práctica 9: Pronósticos con statsmodels

En esta práctica trabajamos con valores distintos a los de las demás debido a que no se concretó la actualización de las demas prácticas y por ende, no tenemos un conjunto de datos validos para pronosticar. Así que siguiendo el ejemplo de la Dra. Elisa, busque otro conjunto de datos con suficientes registros para poder realizar un pronostico decente.

En esta ocasión el conjunto de datos que decidí utilizar es el histórico del valor del Dolar Americano en relación al Nuevo Peso Mexicano(MXP), hago el enfasís en el Nuevo Peso ya que antes del 1° de Enero de 1993 el Peso Mexicano(MXN) tenia mil veces el valor que representaba, es decir, 1 MXP = 1000 MXN, debido a una devaluación acordada por la administración ejecutiva de México entre los años 1988 y 1994. El país atravesó una inflación muy alta heredada de los malos manejos de las anteriores administraciones. Para mitigar los costos excesivos de los productos, se acordó la eliminacion de los ultimos tres ceros de cualquier número que representara dinero. Virtualmente los productos pasaron de costar millones a costar miles.

Debido a que quiero hacer un pronostico acertado, busque todos los resgitros desde el 1° de Enero de 1993 al 8 de Abril de 2019, por la cantidad de datos que manejo decidí descargar la información en tres archivos, cada uno representa una decada, 1990's, 2000's y 2010's. Los datos fueron obtenidos del sitio web mx.investing.com (https://mx.investing.com/currencies/usd-mxn-historical-data)

Objetivo

Pronosticar el precio del dolar para el 9 de Abril de 2019

Preparación de los datos

Una vez cargados los datos en nuestro repositorio, procedemos a importarlos a python3 y los ingresamos a un dataframe de pandas, reemplazamos los puntos de las fechas por espacios vacios para poder formatear la entrada de la fecha. Luego ordenamos las fechas de forma creciente.

In [247]:
```
import pandas as pd
df90s = pd.read_csv('Pronosticos/DHUSD_MXN1990.csv')
df90s['Fecha'] = df90s['Fecha'].apply(lambda x: x.replace('.',''))
df90s['Fecha'] = pd.to_datetime(df90s['Fecha'], format='%d%m%Y')
df90s['% var.'] = df90s['% var.'].str.rstrip('%').astype('float') / 100.0
df90s = df90s.sort_values(['Fecha'])
print("90s = ",len(df90s))

#print(df90s)


df00s = pd.read_csv('Pronosticos/DHUSD_MXN2000.csv')
df00s['Fecha'] = df00s['Fecha'].apply(lambda x: x.replace('.',''))
df00s['Fecha'] = pd.to_datetime(df00s['Fecha'], format='%d%m%Y')
df00s['% var.'] = df00s['% var.'].str.rstrip('%').astype('float') / 100.0
df00s = df00s.sort_values(['Fecha'])
print("00s = ",len(df00s))

#print(df00s)


df10s = pd.read_csv('Pronosticos/DHUSD_MXN2010.csv')
df10s['Fecha'] = df10s['Fecha'].apply(lambda x: x.replace('.',''))
df10s['Fecha'] = pd.to_datetime(df10s['Fecha'], format='%d%m%Y')
df10s['% var.'] = df10s['% var.'].str.rstrip('%').astype('float') / 100.0
df10s = df10s.sort_values(['Fecha'])
print("10s = ",len(df10s))


#print(df10s)
```

```
('90s = ', 1821)
('00s = ', 2606)
('10s = ', 2463)
```

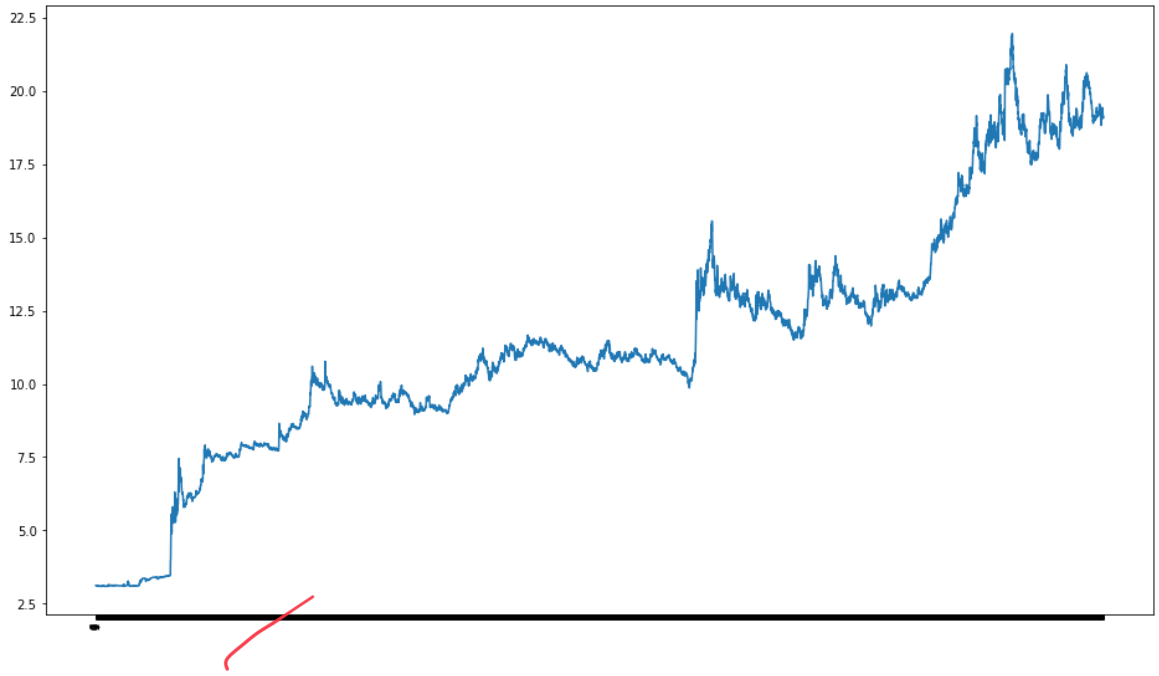
Una vez ordenados los datos como los necesitamos, los fusionamos en un solo dataframe, y trasnponemos esta matriz de datos, reemplazamos el nombre de las columnas con la fecha del valor, quitamos las dechas del dataframe y guardamos todos en un .csv para respaldar la información procesada. Imprimimos los tipos de dato antes de la transposicion para asegurarnos de que todo quedo en el tipo de dato correcto.

In [248]:
```
historico = pd.concat([df90s,df00s,df10s], ignore_index=True)

print(historico.dtypes)


#historico = historico.transpose()
#historico.columns = historico.iloc[0]
#historico = historico[1:]

print(len(historico),len(historico.columns))

historico.to_csv('historicoUSD_MXN.csv', index=False)
```

```
Fecha       datetime64[ns]
Cierre             float64
Apertura           float64
Máximo             float64
Mínimo             float64
% var.             float64
dtype: object
(6890, 6)
```

Una vez ordenada la información como la necesitamos, empezamos a observar el comportamiento de los datos graficando.

In [249]:
```
import matplotlib.pyplot as plt
from numpy import asarray
ejemplo = historico['Cierre']
#print(ejemplo)
x = range(len(ejemplo))
plt.plot(x, asarray(ejemplo))
plt.xticks(x, "04/01/1993", rotation='vertical',fontsize=8)
plt.show()
```



Ahora aplicaremos el pronostico de un paso de Holt

In [180]:
```
from statsmodels.tsa.api import Holt


#historico = historico.transpose()
#historico.columns = historico.iloc[0]
#historico = historico[1:]

#print(historico)
ibis = historico['Cierre']
#print(ibis)
x = range(len(ibis))
#print(x)
pronosticos = []
for i in range(len(historico)):
    y = asarray(historico['Cierre'].loc[:i])
    f = Holt(asarray(y)).fit(smoothing_level = 0.1)
    pronosticos.append(f.forecast(1)[0])
plt.title('Pronóstico de un paso con el método de Holt', fontsize = 20)
plt.xlabel('CF 1 op verdadero', fontsize = 15)
plt.ylabel('Pronóstico', fontsize = 15)
plt.scatter(d.CF1ra, pronosticos, c = 'g', s = 50)
plt.show()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-180-910304bee12a> in <module>()
     13 for i in range(len(historico)):
     14     y = asarray(historico['Cierre'].loc[i:])
```

```
---> 15     f = Holt(asarray(y)).fit(smoothing_level = 0.1)
     16     pronosticos.append(f.forecast(1))
     17 plt.title('Pronóstico de un paso con el método de Holt', fontsize = 20)

/home/samataroukami/.local/lib/python2.7/site-packages/statsmodels/tsa/holtwinters.pyc in fit
(self, smoothing_level, smoothing_slope, damping_slope, optimized)
    887         return super(Holt, self).fit(smoothing_level=smoothing_level,
    888                                      smoothing_slope=smoothing_slope, damping_slope=d
amping_slope,
--> 889                                      optimized=optimized)

/home/samataroukami/.local/lib/python2.7/site-packages/statsmodels/tsa/holtwinters.pyc in fit
(self, smoothing_level, smoothing_slope, smoothing_seasonal, damping_slope, optimized, use_bo
xcox, remove_bias, use_basinhopping)
    584                 # solution to parameters
    585                 res = minimize(func, p[xi], args=(
--> 586                     xi, p, y, l, b, s, m, self.nobs, max_seen), bounds=bounds[xi])

    587             p[xi] = res.x
    588             [alpha, beta, gamma, l0, b0, phi] = p[:6]

/home/samataroukami/.local/lib/python2.7/site-packages/scipy/optimize/_minimize.pyc in minimi
ze(fun, x0, args, method, jac, hess, hessp, bounds, constraints, tol, callback, options)
    599     elif meth == 'l-bfgs-b':
    600         return _minimize_lbfgsb(fun, x0, args, jac, bounds,
--> 601                                 callback=callback, **options)
    602     elif meth == 'tnc':
    603         return _minimize_tnc(fun, x0, args, jac, bounds, callback=callback,

/home/samataroukami/.local/lib/python2.7/site-packages/scipy/optimize/lbfgsb.pyc in _minimize
_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps, maxfun, maxiter, iprint, c
allback, maxls, **unknown_options)
    333             # until the completion of the current minimization iteration.
    334             # Overwrite f and g:
--> 335             f, g = func_and_grad(x)
    336         elif task_str.startswith(b'NEW_X'):
    337             # new iteration

/home/samataroukami/.local/lib/python2.7/site-packages/scipy/optimize/lbfgsb.pyc in func_and_
grad(x)
    278     if jac is None:
    279         def func_and_grad(x):
--> 280             f = fun(x, *args)
    281             g = _approx_fprime_helper(x, fun, epsilon, args=args, f0=f)
    282             return f, g

/home/samataroukami/.local/lib/python2.7/site-packages/scipy/optimize/optimize.pyc in functio
n_wrapper(*wrapper_args)
    298     def function_wrapper(*wrapper_args):
    299         ncalls[0] += 1
--> 300         return function(*(wrapper_args + args))
    301
    302     return ncalls, function_wrapper

/home/samataroukami/.local/lib/python2.7/site-packages/statsmodels/tsa/holtwinters.pyc in _ho
lt_add_dam(x, xi, p, y, l, b, s, m, n, max_seen)
     85     for i in range(1, n):
     86         l[i] = (y_alpha[i - 1]) + (alphac * (l[i - 1] + phi * b[i - 1]))
---> 87         b[i] = (beta * (l[i] - l[i - 1])) + (betac * phi * b[i - 1])
     88     return sqeuclidean(l + phi * b, y)
     89

KeyboardInterrupt:
```

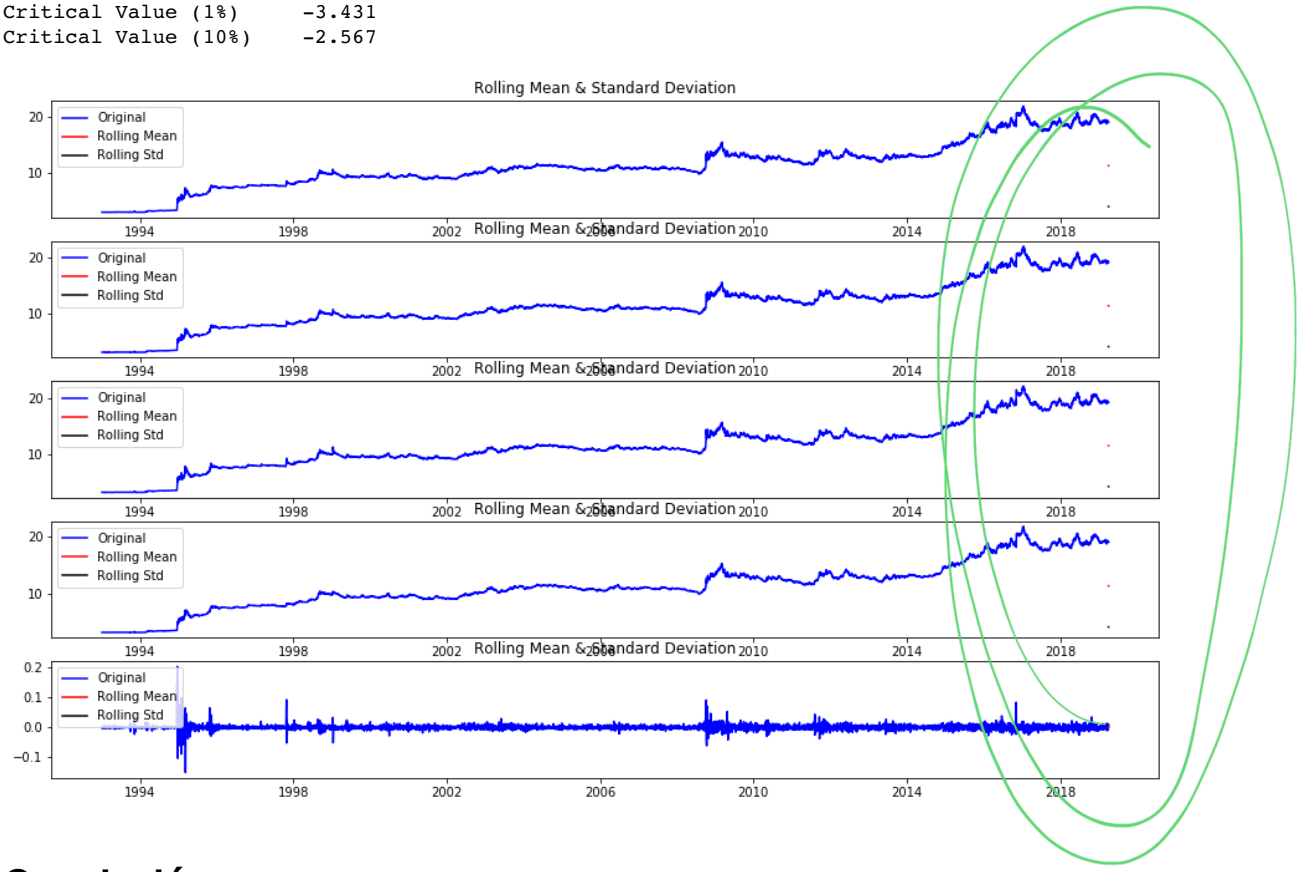
Como tarda mucho en procesar la información pasaremos a las demás pruebas

In [250]:
```
historico = historico.transpose()
historico.columns = historico.iloc[0]
historico = historico[1:]
historico = historico.transpose()
```

In [271]:
```
from statsmodels.tsa.stattools import adfuller

# rutina de https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
def test_stationarity(ts, w, r, i):
    rolmean = ts.rolling(w).mean()
    rolstd = ts.rolling(w).std()
    plt.subplot(r, 1, i)
    orig = plt.plot(ts, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Standard Deviation')
    dftest = adfuller(ts, autolag='BIC')
    #dftest = adfuller(ts)
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of O
bservations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value ({:s})'.format(key)] = value
    return '\n\nResults of Dickey-Fuller Test:\n' + '\n'.join(['{:s}\t{:.3f}'.format(k, v) for (k
, v) in dfoutput.items()])




plt.rcParams["figure.figsize"] = [16, 10]
f = plt.figure()
i = 1
lvls = historico.columns
r = len(lvls)
t = ''
w = 6889
for c in lvls:
    #print(historico[c])
    t += test_stationarity(historico[c], w, r, i)
    i += 1
plt.plot()
print(t)
```

```
Results of Dickey-Fuller Test:
Test Statistic  -1.068
p-value 0.728
#Lags Used      1.000
Number of Observations Used     6888.000
Critical Value (5%)     -2.862
Critical Value (1%)     -3.431
Critical Value (10%)    -2.567

Results of Dickey-Fuller Test:
Test Statistic  -1.070
p-value 0.727
#Lags Used      1.000
Number of Observations Used     6888.000
Critical Value (5%)     -2.862
Critical Value (1%)     -3.431
Critical Value (10%)    -2.567

Results of Dickey-Fuller Test:
Test Statistic  -1.101
p-value 0.715
#Lags Used      3.000
Number of Observations Used     6886.000
Critical Value (5%)     -2.862
```

```
Critical Value (1%)     -3.431
Critical Value (10%)    -2.567

Results of Dickey-Fuller Test:
Test Statistic  -1.013
p-value 0.749
#Lags Used      8.000
Number of Observations Used     6881.000
Critical Value (5%)     -2.862
Critical Value (1%)     -3.431
Critical Value (10%)    -2.567

Results of Dickey-Fuller Test:
Test Statistic  -19.643
p-value 0.000
#Lags Used      12.000
Number of Observations Used     6877.000
Critical Value (5%)     -2.862
Critical Value (1%)     -3.431
Critical Value (10%)    -2.567
```



Conclusión

Demasiados datos para trabajar, falta obtener p y q para pronosticar. --08 de Abril 2019-- Luis Angel Gutierrez Rodriguez 1484412 (tel:1484412)