P09

June 6, 2019

1 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 demás 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 Dólar Americano en relación con el Nuevo Peso Mexicano(MXP), hago el énfasis en el Nuevo Peso ya que antes del 1ř 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 administración. Para mitigar los costos excesivos de los productos, se acordó la eliminación de los últimos 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 registros desde el 1ř de enero de 1993 al 7 de abril de 2019, por la cantidad de datos que manejo decidí descargar la información en tres archivos, cada uno representa una década, 1990's, 2000's y 2010's. Los datos fueron obtenidos del sitio web mx.investing.com

1.1 Objetivo

```
- Pronosticar el precio del dólar para el 8 de abril de 2019
```

1.2 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 [1]: import pandas as pd
        df90s = pd.read_csv('old/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('old/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('old/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 transponemos esta matriz de datos, reemplazamos el nombre de las columnas con la fecha del valor, quitamos las fechas del DataFrame y guardamos todos en un .csv para respaldar la información procesada. Imprimimos los tipos de dato antes de la transposición para asegurarnos de que todo quedo en el tipo de dato correcto.

```
In [2]: historico = pd.concat([df90s,df00s,df10s], ignore_index=True)

        print(historico.dtypes)

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

        historico.to_csv('old/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 [5]: 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=1)
        plt.title('Variación del precio del dolar', fontsize = 20)
        plt.xlabel('Días cotizados', fontsize = 20)
        plt.ylabel('Precio del dolar', fontsize = 20)

Out[5]: Text(0, 0.5, 'Precio del dolar')
```

1.3 Pronóstico de Holt-Winters

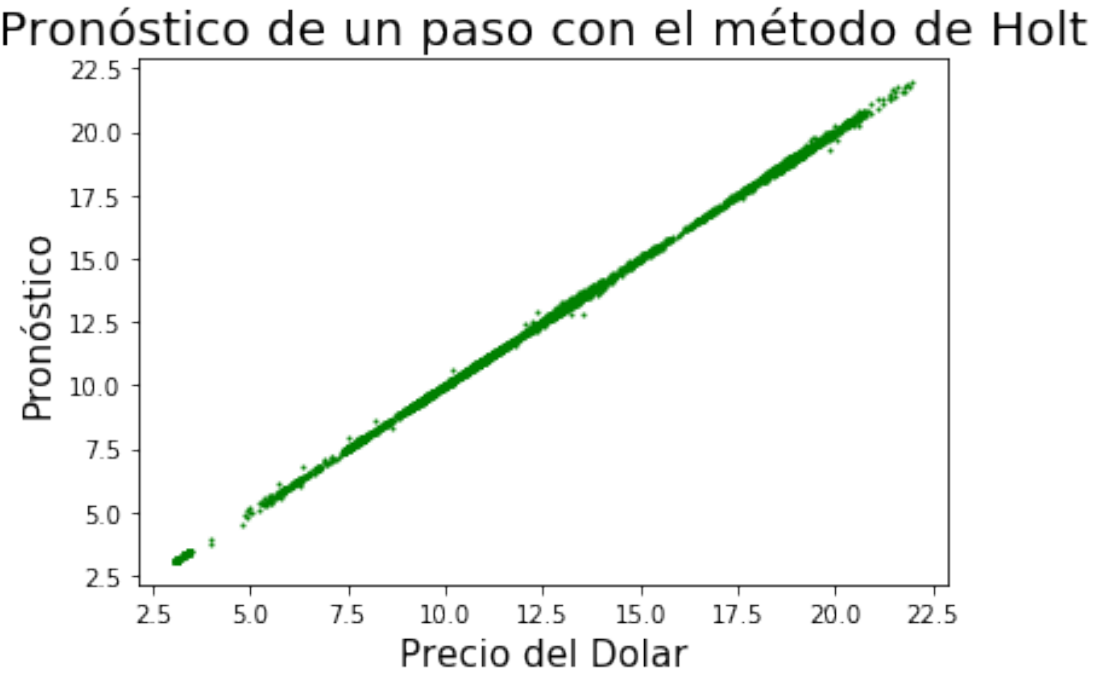
El modelo Holt-Winters incorpora un conjunto de procedimientos que conforman el núcleo de la familia de series temporales de suavizamiento exponencial. A diferencia de muchas otras técnicas, el modelo Holt-Winters puede adaptarse fácilmente a cambios y tendencias, así como a patrones

estacionales. En comparación con otras técnicas, como ARIMA, el tiempo necesario para calcular el pronóstico es considerablemente más rápido. Más allá de sus características técnicas, su aplicación en entornos de negocio es muy común. De hecho, Holt-Winters se utiliza habitualmente por muchas compañías para pronosticar la demanda a corto plazo cuando los datos de venta contienen tendencias y patrones estacionales de un modo subyacente.

Ahora se aplica el pronostico de un paso de Holt

```
In [7]: from statsmodels.tsa.api import Holt

        historico = historico[['Cierre','Apertura','Máximo','Mínimo']]

        d = historico
        print(d.columns)
        lbls = historico.columns
        x = range(len(lbls))
        pronosticos = []
        for i in range(len(d)):
            y = asarray(d.loc[i,:])
            f = Holt(asarray(y)).fit(smoothing_level = 0.1)
            pronosticos.append(f.forecast(1))
        plt.title('Pronóstico de un paso con el método de Holt', fontsize = 20)
        plt.xlabel('Precio del Dolar', fontsize = 15)
        plt.ylabel('Pronóstico', fontsize = 15)
        plt.scatter(historico.Cierre, pronosticos, c = 'g', s = 1)
        plt.show()
```

```
Index(['Cierre', 'Apertura', 'Máximo', 'Mínimo'], dtype='object')
```



Parece que la información está en una línea recta de pendiente igual a uno, por la cantidad de los datos parece que el pronostico acierta al valor real, y además están fuertemente correlacionados.

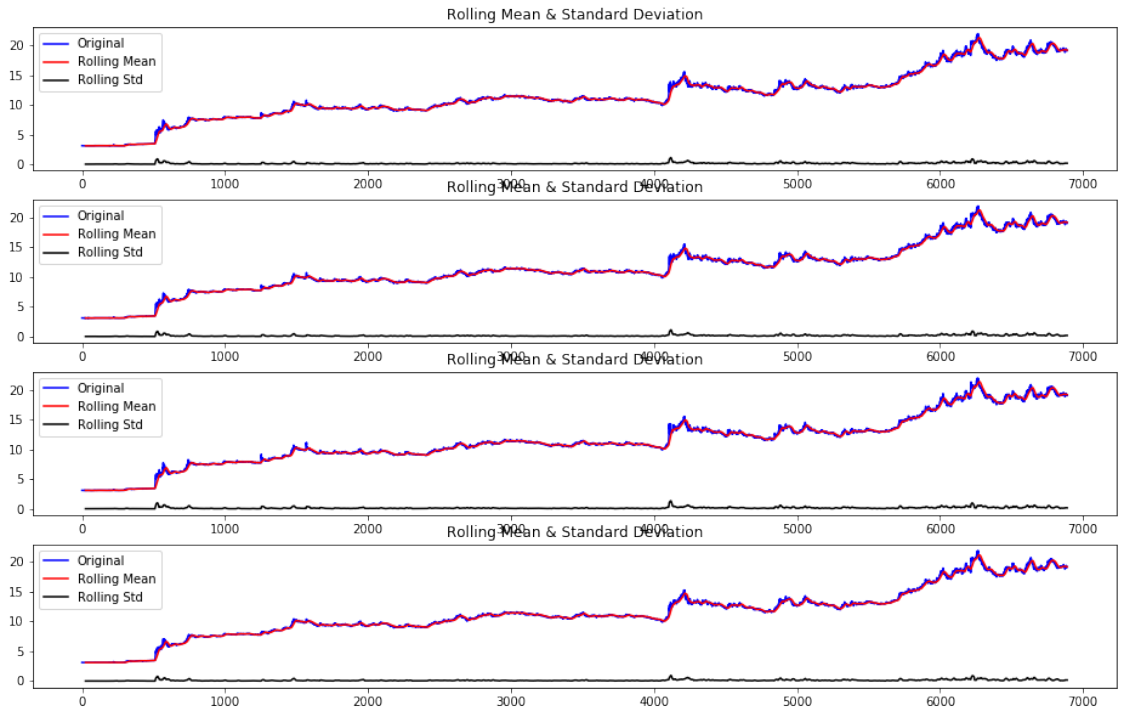
1.4 Prueba de Dickey-Fuller aumentada

La prueba de Dickey-Fuller aumentada (ADF) es una prueba de raíz unitaria para una muestra de una serie de tiempo. Es una versión aumentada de la prueba Dickey-Fuller para un conjunto más amplio y más complejo de modelos de series de tiempo. La estadística Dickey-Fuller Aumentada (ADF), utilizada en la prueba, es un número negativo. Cuanto más negativo es, más fuerte es el rechazo de la hipótesis nula de que existe una raíz unitaria para un cierto nivel de confianza.

Ahora se aplica la prueba de Dickey-Fuller aumentada

```
In [8]: from statsmodels.tsa.stattools import adfuller

        # rutina de https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes
        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','
            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

        plt.rcParams["figure.figsize"] = [16, 10]
        f = plt.figure()
        i = 1
        lvls = historico.columns
        r = len(lvls)
        print(r)
        t = ''
        w = 26
        for c in lvls:
            #print(historico[c])
            t += test_stationarity(historico[c], w, r, i)
            i += 1
        plt.plot()
        print(t)
```

```
4


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 (1%)        -3.431
Critical Value (5%)        -2.862
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 (1%)        -3.431
Critical Value (5%)        -2.862
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 (1%)        -3.431
Critical Value (5%)        -2.862
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 (1%)        -3.431
Critical Value (5%)        -2.862
Critical Value (10%)        -2.567
```

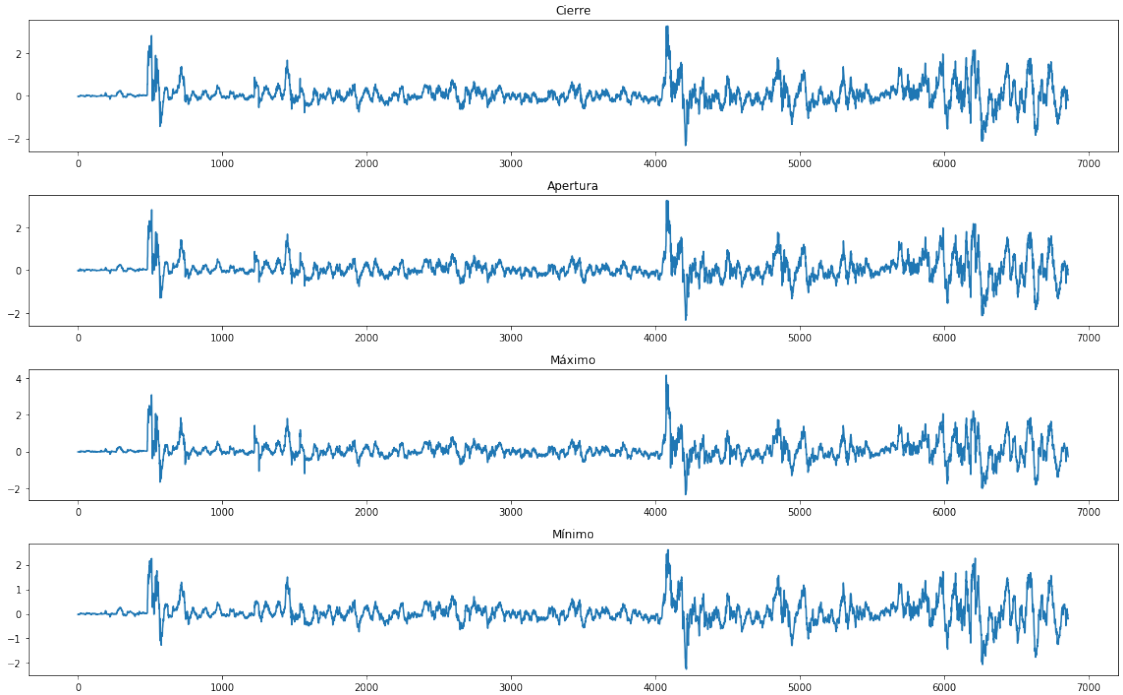


Se intenta restar de cada valor el valor correspondiente un mes antes (hace 30 días)

```
In [9]: from pandas import Series
        def difference(dataset, interval=1):
            diff = list()
            for i in range(interval, len(dataset)):
                value = dataset[i] - dataset[i - interval]
                diff.append(value)
            return Series(diff)

        f = plt.figure()
        i = 1
        for c in ['Cierre','Apertura','Máximo','Mínimo']:
            ts = difference(asarray(d[c]), interval = 30)
            ts = ts.dropna()
            plt.subplot(4, 1, i)
            plt.title(c)
            plt.plot(ts)
            i += 1
            result = adfuller(ts)
            print('{:s}\nADF Stat\t{:.3f}\np-value\t{:.3f}\n'.format(c, result[0], result[1]))
            for key, value in result[4].items():
                print('\t%s: %.3f' % (key, value))
        plt.tight_layout()
```

```
Cierre
```

```
ADF Stat        -9.539
p-value         0.000

        1%: -3.431
        5%: -2.862
        10%: -2.567
Apertura
ADF Stat        -9.549
p-value         0.000

        1%: -3.431
        5%: -2.862
        10%: -2.567
Máximo
ADF Stat        -9.324
p-value         0.000

        1%: -3.431
        5%: -2.862
        10%: -2.567
Mínimo
ADF Stat        -9.413
p-value         0.000

        1%: -3.431
        5%: -2.862
        10%: -2.567
```



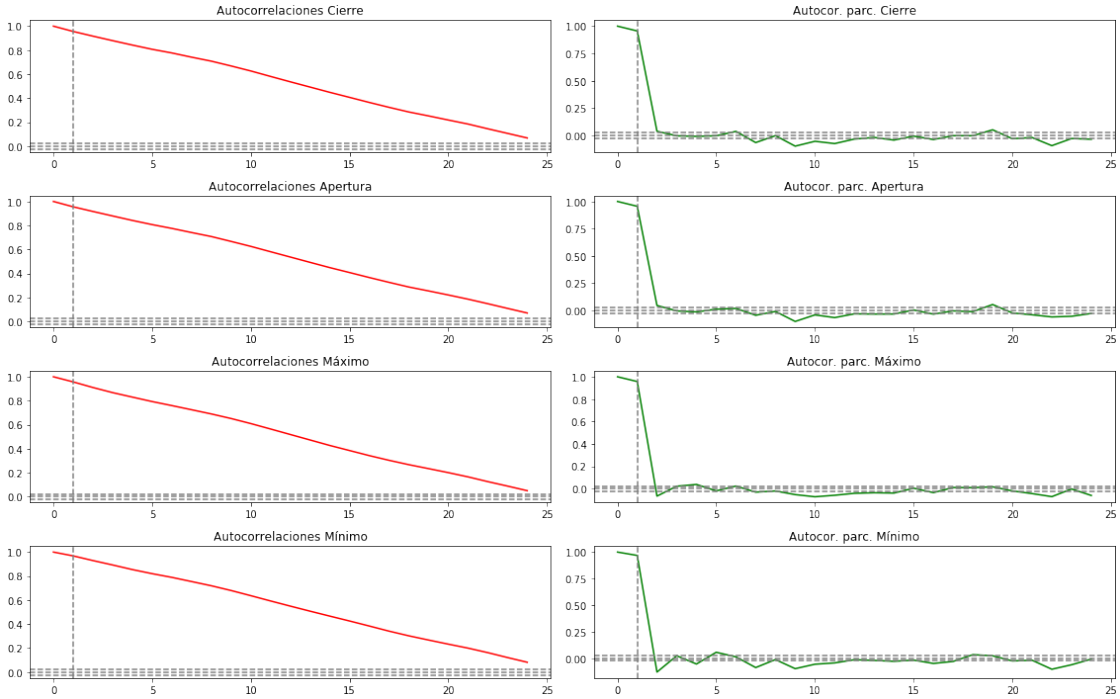
Según la prueba estadística los cuatro factores ya quedaron listos para poder pronosticar. Todos tienen el valor de p=0, entonces todos tienen valor q=1.

1.5 ARIMA

Un modelo autorregresivo integrado de promedio móvil o ARIMA (acrónimo del inglés autoregressive integrated moving average) es un modelo estadístico que utiliza variaciones y regresiones de datos estadísticos con el fin de encontrar patrones para una predicción hacia el futuro. Se trata de un modelo dinámico de series temporales, es decir, las estimaciones futuras vienen explicadas por los datos del pasado y no por variables independientes.

Se aplica un ARIMA sobre los datos.

```
In [10]: from statsmodels.tsa.stattools import acf, pacf
         from math import sqrt

         f = plt.figure()
         i = 1
         n = len(d)
         q = {'Cierre': 1, 'Apertura': 1, 'Máximo': 1, 'Mínimo': 1}
         p = {'Cierre': 1, 'Apertura': 1, 'Máximo': 1, 'Mínimo': 1}
         for c in ['Cierre','Apertura','Máximo','Mínimo']:
             ts = difference(asarray(d[c]), interval = 26).dropna()
             # manera del primer tutorial
             plt.subplot(4, 2, i)
             i += 1
             plt.plot(acf(ts, nlags = 24), 'r')
             plt.axhline(y = 0, linestyle='--', color = 'gray')
             plt.axvline(x = q[c], linestyle = '--', color='gray')
             plt.axhline(y = -1.96 / sqrt(n), linestyle = '--', color='gray')
             plt.axhline(y = 1.96 / sqrt(n), linestyle = '--', color='gray')
             plt.title('Autocorrelaciones ' + c)
             plt.subplot(4, 2, i)
             i += 1
             plt.plot(pacf(ts, nlags = 24, method='ols'), 'g')
             plt.axhline(y = 0, linestyle = '--', color = 'gray')
             plt.axvline(x = p[c], linestyle = '--', color='gray')
             plt.axhline(y = -1.96 / sqrt(n),linestyle = '--', color='gray')
             plt.axhline(y = 1.96 / sqrt(n),linestyle = '--', color='gray')
             plt.title('Autocor. parc. ' + c)
         plt.tight_layout()
```



Entonces, siguiendo el procedimiento del primer tutorial, buscando la coordenada x del primer cruce con el intervalo de confianza superior, se obtiene q = {'Cierre': 1, 'Apertura': 1, 'Máximo': 1, 'Mínimo': 1} y p = {'Cierre': 1, 'Apertura': 1, 'Máximo': 1, 'Mínimo': 1}

1.6 Pronóstico

Aplicando la información obtenida podemos pronosticar el valor del dólar el 8 de abril de 2019.

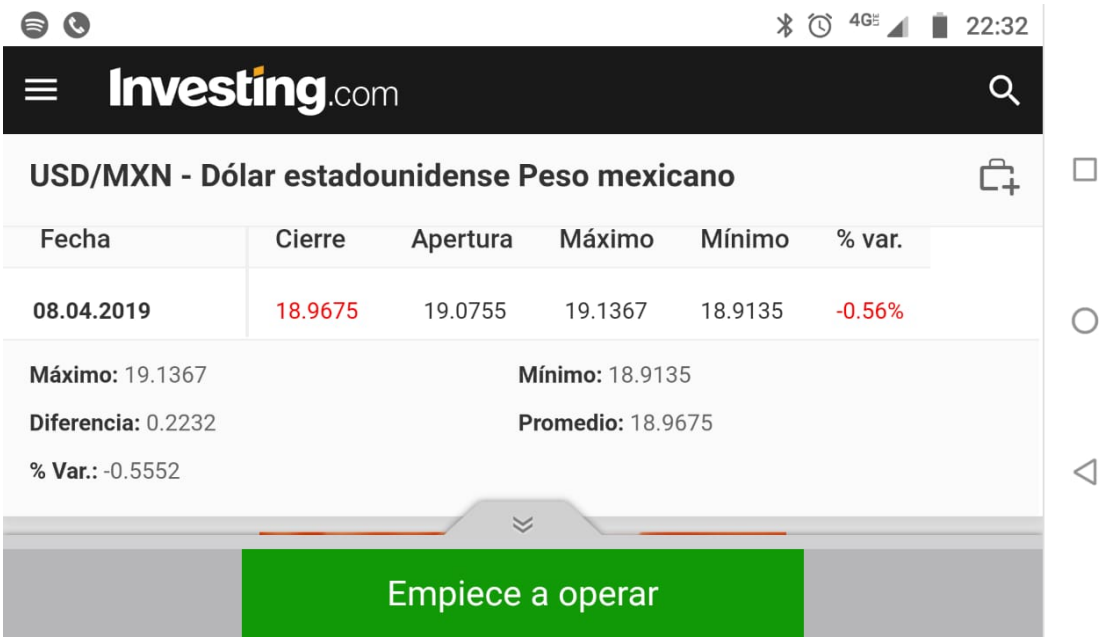
```
In [11]: from statsmodels.tsa.arima_model import ARIMA
         import pandas as pd
         import ssl

         # https://machinelearningmastery.com/time-series-forecast-study-python-monthly-sales-
         def difference(dataset, interval=1):
             diff = list()
             for i in range(interval, len(dataset)):
                 value = dataset[i] - dataset[i - interval]
                 diff.append(value)
             return pd.Series(diff)

         if getattr(ssl, '_create_unverified_context', None):
             ssl._create_default_https_context = ssl._create_unverified_context

         guardar = d.iloc[-1:, :]
         d = d.iloc[:-1, :]
         n = len(d)
```

```
        q = {'Cierre': 1, 'Apertura': 1, 'Máximo': 1, 'Mínimo': 1}
        p = {'Cierre': 1, 'Apertura': 1, 'Máximo': 1, 'Mínimo': 1}
        prons = []
        for c in ['Cierre','Apertura','Máximo','Mínimo']:
            ts = difference(asarray(d[c]), interval = 30).dropna()
            try:
                m = ARIMA(ts, order = (p[c], 1, q[c]))
                f = m.fit(trend = 'nc', disp = 0)
                print('{:s}\tpron.\t'.format(c), int(round(f.forecast()[0][0] + d[c].iloc[n -
                print('{:s}\treal\t'.format(c), guardar[c].iloc[0])
            except:
                print(c, "no se pudo pronosticar con esos parámetros")
            print('...')
```

```
Cierre        pron.         19
Cierre        real         19.1021
...
Apertura        pron.         19
Apertura        real         19.0733
...
Máximo        pron.         19
Máximo        real         19.1108
...
Mínimo        pron.         19
Mínimo        real         19.0716
...
```

```
In [3]: from IPython.display import Image
        Image(filename='Dolar8Abril2019.jpeg')
```

Out[3]:



Podemos observar en esta imagen que el modelo de pronostico tuvo un GAP por factor correspondiente a la siguiente tabla.

Parámetro	Modelo real	Realidad	GAP
Cierre	19.1021	18.9675	-0.7%
Apertura	19.0733	19.0755	+0.01%
Máximo	19.1108	19.1367	+0.13%
Mínimo	19.0716	18.9135	-0.83%

2 Conclusión

La cantidad información obtenida nos permitió hacer un pronostico acertado. Al aplicar correctamente los métodos de pronósticos, llegamos a valores muy cercanos a la realidad.

--05 de junio 2019-- Luis Angel Gutiérrez Rodríguez 1484412