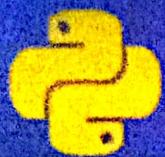


# PYTHON PARA FINANZAS QUANT



## APIs - Market Data

t[3]



Juan Pablo Pisano  
@johnGalt\_is-www

Impreso en:  
[laimprentadigital.com.ar](http://laimprentadigital.com.ar)

# Indice

Funciones .....	11
Funciones con argumentos con un valor por default .....	14
Pensando funcionalmente .....	21
Aplicaciones Prácticas .....	22
Ejercicios .....	26
Respuestas .....	29
APIs .....	39
Tipos de API y OAuth .....	41
Generalidades de una API .....	42
Rate Limits.....	45
Status Code .....	45
API AlphaVantage .....	47
HardCodeando una Función .....	53
Mas funciones de la API AlphaVantage .....	58
Series Ajustadas.....	59
Cotización actual.....	60
Cotizaciones de otros mercados.....	62
Búsqueda de Activos.....	63
FOREX .....	65
Crypto .....	68
Indicadores Técnicos .....	69
Otros indicadores menos conocidos.....	70
Medias Móviles .....	71
MACD .....	73
Ejercicios .....	74
Respuestas .....	76
FinnHub API.....	88
Perfil básico de la compañía .....	92
Ejecutivos de una compañía.....	92
Obtener Noticias .....	95
Noticias de una Empresa.....	96
Mayores eventos .....	96
News Sentiment .....	97
Pares similares .....	98
Métricas de Análisis Fundamental.....	98
Filings (SEC) .....	104
IPO Calendar.....	107
Recomendaciones de "Analistas".....	108

Target de Precio .....	110
Upgrades & DownGrades .....	111
Estimaciones .....	112
EPS Surprise .....	113
Series históricas .....	115
Ejercicios .....	129
Respuestas.....	130
API FMP - Análisis Fundamental.....	136
Paneles de precios en tiempo real .....	139
Cotizaciones intradiarias .....	142
Cotizaciones históricas.....	143
Batch Requests.....	145
Datos de Análisis Fundamental. Perfil básico .....	146
Performance por sector Histórica.....	148
Hoja de Balance .....	150
Estado de resultados.....	153
Cash Flow o Flujo de Fondos .....	154
Ratios financieros.....	156
Métricas Financieras .....	158
Valuación Empresarial.....	159
Valuación por flujo de fondos descontado.....	160
Rating por valuaciones.....	161
13F Filings .....	162
Ejercicios .....	165
Respuestas.....	166
IEX Cloud API.....	173
Peso de las Api Calls.....	173
Endpoint BASE.....	174
Libro de Precios .....	175
Trades más grandes del día (sandBox) .....	178
Volumen por Exchange (sandBox) .....	178
Top 10 Holders (SandBox) .....	179
Estimaciones de EPS (SandBox) .....	180
Estimaciones de Precio Target (Sandbox) .....	181
Collections: Listas, Sectores, Tags .....	182
IPOs, Ofertas públicas iniciales (Sandbox).....	184
Opciones (SandBox) .....	186
Social Sentiment (SandBox) .....	188
Treasuries.....	189
Otros Endpoints .....	190
Ejercicios .....	192

Respuestas.....	193
API abierta de la reserva Federal .....	197
Ejercicios .....	200
Lectura directa de CSVs .....	203
Websockets .....	205
Ejercicios .....	215
Respuestas.....	216

# Funciones

Vamos a arrancar por algo que venimos haciendo repetidamente hasta este bloque que es leer datos de mercado de archivos de excel, van a ver en este bloque como vamos a bajar datos de mercado directo de la web, pero quise arrancar así, porque la inmensa mayoría usa a fondo excel para sus análisis y me pareció un buen puente para empezar a meterse en python. Pero de ahora en más no vamos a tipear las líneas que tipeábamos cuando leímos un excel, sino que vamos a crear una función que se encargue de ello

Las funciones son un conjunto de instrucciones (líneas de código) que generalmente necesitamos reutilizar mas de una vez

Son el alma de la programación ya que va a ser muy común que necesitemos reutilizar "pedazos de código" de hecho dejé este tema para el cuarto bloque, no solo porque quería concentrar el foco en el resto de herramientas primero sino para que ahora se valore mas el uso de funciones ya que si llegaron hasta acá seguramente ya se habrán topado con esto en algún otro tutorial y además ya habrán percibido la necesidad de usar este tipo de herramientas

Ejemplo

```
def hora():
    import datetime as dt
    print(dt.datetime.now())
```

```
hora()
```

```
2020-04-25 23:35:54.439333
```

También podemos ir agregando cosas:

```
def hora():
    import datetime as dt
    time = dt.datetime.now()
    time = time.strftime("%H:%M:%S")
    print(time)

hora()
```

```
23:40:42
```

Pero ¿si quisiéramos guardar esa hora en una variable? Si se fijan la función hora() que definimos, imprime la hora en pantalla pero no podemos "capturar" el dato, solo se ejecuta la función, la imprime y chau..

En general las funciones mas allá de ejecutar un script, como en el caso que pusimos recién devuelven un "return", es por eso que la palabra "return" es una palabra reservada de python y no la podemos usar como nombre de variable

Entonces, veamos como sería si queremos una función que no solo imprima la hora sino que "me la devuelva" para que se la pueda asignar a una variable

```
def hora():
    import datetime as dt
    time = dt.datetime.now()
    time = time.strftime("%H:%M:%S")
    print(time)
return time

horaActual = hora()
print(horaActual)
```

23:42:50

23:42:50

Si se fijan ahora, esto me imprime dos veces la hora, la primera vez es cuando ejecuta la función, y la segunda cuando le pido que me imprima lo que contiene la variable "horaActual"

Por lo general las funciones no imprimen cosas en pantalla sino que devuelven valores que se los asignamos a una variable, si luego queremos imprimirlas los mandamos con un print pero no se suele hacer eso dentro de una función, quedaría mas parecida la función a algo así:

```
def hora():
    import datetime as dt
    time = dt.datetime.now()
    time = time.strftime("%H:%M:%S")
    return time

horaActual = hora()
print(horaActual)
```

23:45:22

Si se preguntan ¿Se puede programar sin usar funciones predefinidas?

La respuesta es obvio que se puede, pero van a ver que son super cómodas y útiles por mas que programen pequeños scripts

Muchos habrán notado que usando entornos como Jupyter, cuando leen datos de un recurso externo, ya sea un excel o una API o lo que fuera, esos datos quedan "Grabados" en la variable a la que se los asignaron y así la pueden reutilizar cuantas veces quieran, de hecho queda grabado y después de hacer muchas pruebas mas, ejecutando otros scripts dentro del mismo kernel, incluso después de horas y decenas de pruebas sigue quedando ahí grabado el dato

Bien, correcto, pero este tipo de entornos si bien esta muy bien para probar cosas o para aprender, en la vida real, no se usan para proyectos grandes, y los entornos que usan no tienen ese tipo de memoria, por lo cual para poder reutilizar código que tiparon antes, se usan funciones

Bueno, vamos al grano, veamos una función concreta, vamos a arrancar por lo básico que vimos en los bloques anteriores que es leer un archivo de excel con datos de mercado

Si recuerdan, teníamos los excels ordenados de reciente a antiguo y por lo general los necesitábamos a antiguos a recientes así que arranquemos armando una función para este proceso repetitivo

Las instrucciones eran:

```
import pandas as pd
data = pd.read_excel("AAPL.xlsx")
data = data.sort_values('timestamp', ascending=True)
data.set_index('timestamp', inplace=True)

data.head()
```

	open	high	low	close	adjusted_close	volume
timestamp						
2000-03-06	126.00	129.13	125.00	125.69	3.8960	1880000
2000-03-07	126.44	127.44	121.12	122.87	3.8086	2437600
2000-03-08	122.87	123.94	118.56	122.00	3.7816	2421700
2000-03-09	120.87	125.00	118.25	122.25	3.7894	2470700
2000-03-10	121.69	127.94	121.00	125.75	3.8979	2219700

Bien, ahora pensemos, que pasa si ahora quiero leer los datos de otro excel similar como el SPY que tengo en la misma carpeta?

En ese caso, la primera linea "import pandas as pd" ya no la voy a tippear mas porque ya en la memoria tengo esa librería cargada y no necesito andar cargándola a cada rato, de hecho es una mala práctica hacer esto porque consumimos recursos innecesarios, pero las otras 3 líneas si las tengo que tippear de nuevo y solo cambiar "AAPL.xlsx" por "SPY.xlsx", de hecho el ".xlsx" tambien queda igual, solo le tengo que cambiar el ticker

Entonces, como puedo "automatizar" esas 3 líneas de código? Definiendo una función que ejecute siempre esas líneas

```
def leerExcel(nombreArchivo):
    data = pd.read_excel(nombreArchivo)
    data = data.sort_values('timestamp', ascending=True)
    data.set_index('timestamp', inplace=True)
    return data

AAPL = leerExcel('AAPL.xlsx')
SPY = leerExcel('SPY.xlsx')

AAPL.head()
```

	open	high	low	close	adjusted_close	volume
timestamp						
2000-03-06	126.00	129.13	126.00	126.60	3.8000	1880000
2000-03-07	126.44	127.44	121.12	122.07	3.8086	2437600
2000-03-08	122.87	123.94	118.66	122.00	3.7816	2421700
2000-03-09	120.87	125.00	118.26	122.26	3.7894	2470700
2000-03-10	121.69	127.94	121.00	125.75	3.8979	2219700

Ahora Imaginen que yo en el directorio tengo miles de excels con esa misma estructura de precios de acciones diferentes, ¿no se empleza a notar la utilidad de esto?

## Funciones con argumentos con un valor por default

Vamos un poco mas, vamos a armar la función para que aparte del ticker le pueda pasar el nombre de alguna carpeta y que no haga falta pasarselo el ".xlsx" ya que todos los excels tienen esa misma extensión

```
def leerExcel(ticker, carpeta=''):
    if carpeta == '':
        ruta = ticker+'.xlsx'
    else:
        ruta = carpeta + '/' + ticker + '.xlsx'

    data = pd.read_excel(ruta)
    data = data.sort_values('timestamp', ascending=True)
    data.set_index('timestamp', inplace=True)
    return data

data = leerExcel('AAPL')
data.head()
```

	open	high	low	close	adjusted_close	volume
timestamp						
2000-03-06	126.00	129.13	125.00	125.69	3.8960	1880000
2000-03-07	126.44	127.44	121.12	122.87	3.8086	2437600
2000-03-08	122.87	123.94	118.56	122.00	3.7816	2421700
2000-03-09	120.87	125.00	118.25	122.25	3.7894	2470700
2000-03-10	121.69	127.94	121.00	125.75	3.8979	2219700

Fíjense que interesante esto que se puede hacer en las funciones, si ven detenidamente el ejemplo sencillo que acabo de poner, en el mismo al definir la función leerExcel, le paso dos argumentos, una el ticker, y el segundo el nombre de la supuesta carpeta donde se encuentra,

pero en este segundo argumento, notarán que le puse "=" en la definición de la función, de esta forma estoy definiendo el valor `pode default` que es un string vacío

Al estar vacío por default, presupongo que el archivo debe estar en la carpeta donde se ejecuta el script, pero si luego quiero leer un archivo de otra carpeta, le tengo que pasar ese argumento a mi función, veámoslo (yo tengo la carpeta ADRs con los Excel de los ADRs argentinos, tal cual como lo puse en la carpeta de recursos de este libro:

- [bit.ly/39Am64T](http://bit.ly/39Am64T)

Obviamente deben bajar los archivos y carpetas al mismo lugar (directorio) donde estén ejecutando el código, tengan la precaución de verificar que si les descarga un archivo comprimido (.zip) deben descomprimirlo para poder leerlo

```
data = leerExcel(ticker='GGAL', carpeta='ADRs')
data.head()
```

	open	high	low	close	adjusted_close	volume
timestamp						
2000-07-25	17.48	17.75	16.75	17.50	16.5492	126200
2000-07-26	17.25	17.56	17.19	17.50	16.5492	28900
2000-07-27	17.50	17.63	17.38	17.50	16.5492	61200
2000-07-28	17.56	17.56	17.13	17.38	16.4357	146100
2000-07-31	17.50	17.69	16.88	17.69	16.7289	178400

Como pueden ver, al tener la función ya definida, solo tengo que llamarla y listo, así que si ahora quisiera leer varios tickers miren que facil y limpio se hace el código:

```
tickers = ['GGAL', 'YPF', 'TEO']
data = {}
for ticker in tickers:
    data[ticker] = leerExcel(ticker=ticker, carpeta='ADRs')
```

Lo que voy a hacer iterando dentro de ese FOR, fíjense que recorro todos los tickers de la lista "tickers", es generar una nueva clave por cada iteración del FOR, para el diccionario "data" que acabo de crear antes de entrar al FOR.

Ese diccionario tendrá una clave por ticker, y el valor de cada clave será un DataFrame entero (recuerden que los valores de un diccionario, podían ser datos, listas o cualquier objeto de estructura de datos, en este caso un dataframe de pandas)

Entonces, "data" es un diccionario que tendrá los dataFrames de todos los tickers leídos de sus respectivos Excels, comprobemos pidiendo que imprima alguno

```
data[ 'YPF' ].head()
```

	open	high	low	close	adjusted_close	volume
timestamp						
2000-04-03	35.25	35.44	34.94	35.25	10.4442	17800
2000-04-04	35.25	35.26	34.44	34.69	10.2783	25700
2000-04-05	34.75	34.75	34.06	34.13	10.1124	8000
2000-04-06	34.25	35.13	34.25	34.75	10.2961	31200
2000-04-07	34.50	35.00	34.50	34.88	10.3346	9400

Sigamos avanzando, no se si recuerdan, cuando graficábamos velas o cuando queríamos hacer algún tipo de análisis de velas, gaps o cualquier cosa que tenga que evaluar los precios OHLC teníamos la dificultad de tener los precios de OHLC sin ajustar por dividendos o splits.

Y yo les mostré un código básico de ajustar todos los datos de OHLC en función del AdjClose que suele ser una manera estandar de presentar los datos de series financieras, con el OHLC sin ajustar y el precio de cierre ajustado aparte, bueno, vamos a armar una función que no sólo lea el excel sino que además haga lo siguiente:

- \* Que me ajuste los datos de OHLC
- \* Que me devuelva el volumen en millones de dólares en lugar de nominales

Aclaro que este tipo de ajuste básico de las series de precios financieros, es super común, lo de los precios OHLC es obvio el por qué(\*), y lo del volumen en millones de dólares se hace mas evidente cuando los precios sufren grandes variaciones o cuando hay importantes splits, en ambos casos si comparo el volumen en diferentes épocas en nominales voy a estar comparando peras con manzanas, es por ello que se suele normalizar la serie usando el volumen en millones de dólares

(\*) Por si no se entiende, el punto es el siguiente: los precios de cierre ajustados nos traen los ajustes por dividendos por ejemplo, pero si no me ajustan toda la serie OHLC, el precio de apertura de la siguiente rueda luego del pago de un dividendo tendrá un gap que en realidad no es un gap de precio sino el reflejo el pago de un dividendo, por lo tanto estoy alterando la morfología de la vela siguiente a cada dividendo y ni hablar de los splits, es por ello que se debe ajustar por el coeficiente de dividendo o Split cada dato hacia atrás desde el último dato que se cuenta.

La manera de calcularlo es calcular el % de salto del precio ajustado respecto del sin ajustar y por el mismo % ajustar el OHLC, obviamente los días normales donde no haya dividendos ni splits, el factor de ajuste será 1 (ratio) o 0 en caso de tomar un % directo.

Bien, entonces hagamos la función:

```
def ajustarExcel(ticker, carpeta=''):

    if carpeta == '':
        ruta = ticker+'.xlsx'
    else:
        ruta = carpeta + '/' + ticker + '.xlsx'

    data = pd.read_excel(ruta)
    data = data.sort_values('timestamp', ascending=True)
    data.set_index('timestamp', inplace=True)
    data['factor'] = data.adjusted_close / data.close
    data['volMlnUSD'] = data.close * data.volume /1000000

    cols = [data.open*data.factor, data.high*data.factor,
            data.low*data.factor, data.adjusted_close, data.volMlnUSD]

    dataAj = pd.concat(cols, axis=1)
    dataAj.columns = ["Open", "High", "Low", "Close", "volMlnUSD"]
    return dataAj.round(2)
```

Como ven la función calcula el factor de ajuste, cuando no pasa nada ese factor es 1, y lo multiplica por todo el OHLC para obtener la serie ajustada

La probamos:

```
data = ajustarExcel('YPF', 'ADRs')
data.head()
```

	Open	High	Low	Close	volMlnUSD
timestamp					
2000-04-03	10.44	10.50	10.35	10.44	0.63
2000-04-04	10.44	10.44	10.20	10.28	0.89
2000-04-05	10.30	10.30	10.09	10.11	0.27
2000-04-06	10.15	10.41	10.15	10.30	1.08
2000-04-07	10.22	10.37	10.22	10.33	0.33

Recordemos como era la serie original, citada justamente en la página anterior:

	open	high	low	close	adjusted_close	volume
timestamp						
2000-04-03	35.25	35.44	34.94	35.25	10.4442	17800
2000-04-04	35.25	35.25	34.44	34.69	10.2783	25700
2000-04-05	34.75	34.75	34.06	34.13	10.1124	8000
2000-04-06	34.25	35.13	34.25	34.75	10.2961	31200
2000-04-07	34.50	35.00	34.50	34.88	10.3346	9400

Bien, supongamos que un amigo nos pasa un archivo .py (o que lo bajamos de un blog de python) con una función supuestamente para graficar un heatmap de correlación, y contiene la siguiente función:

```
def graficaCorr(dfCorr, title=''):

    # La función recibe como argumento un DataFrame de nxn
    # con valores de correlación

    # Devuelve un gráfico tipo heatMap de ese DataFrame, bien estilizado

    import matplotlib.pyplot as plt
    import numpy as np

    fig = plt.figure(figsize=(12, 8))
    plt.matshow(dfCorr, fignum=fig.number, cmap='binary')
    plt.xticks(range(dfCorr.shape[1]), dfCorr.columns, fontsize=12, rotation=90)
    plt.yticks(range(dfCorr.shape[1]), dfCorr.columns, fontsize=12)

    cb = plt.colorbar(orientation='vertical', label="Factor Correlación 'r'")
    cb.ax.tick_params(labelsize=12)
    plt.title(title, fontsize=16, y=1.15)

    # gca() captura los objetos pares de ejes
    ax = plt.gca()

    ax.set_xticks(np.arange(-.5, len(dfCorr), 1), minor=True);
    ax.set_yticks(np.arange(-.5, len(dfCorr), 1), minor=True);
    ax.grid(which='minor', color='w', linestyle='-', linewidth=3)

    # recorro la matriz horizontalmente (j) y verticalmente (i)

    for i in range(dfCorr.shape[0]):

        for j in range(dfCorr.shape[1]):

            if dfCorr.iloc[i,j] > 0.6:
                color = 'white'
            else:
                color = 'black'
            fig.gca().text(i,j, "{:.2f}".format(dfCorr.iloc[i,j]), ha="center",
                           va="center", c=color, size='14')

    return(plt)
```

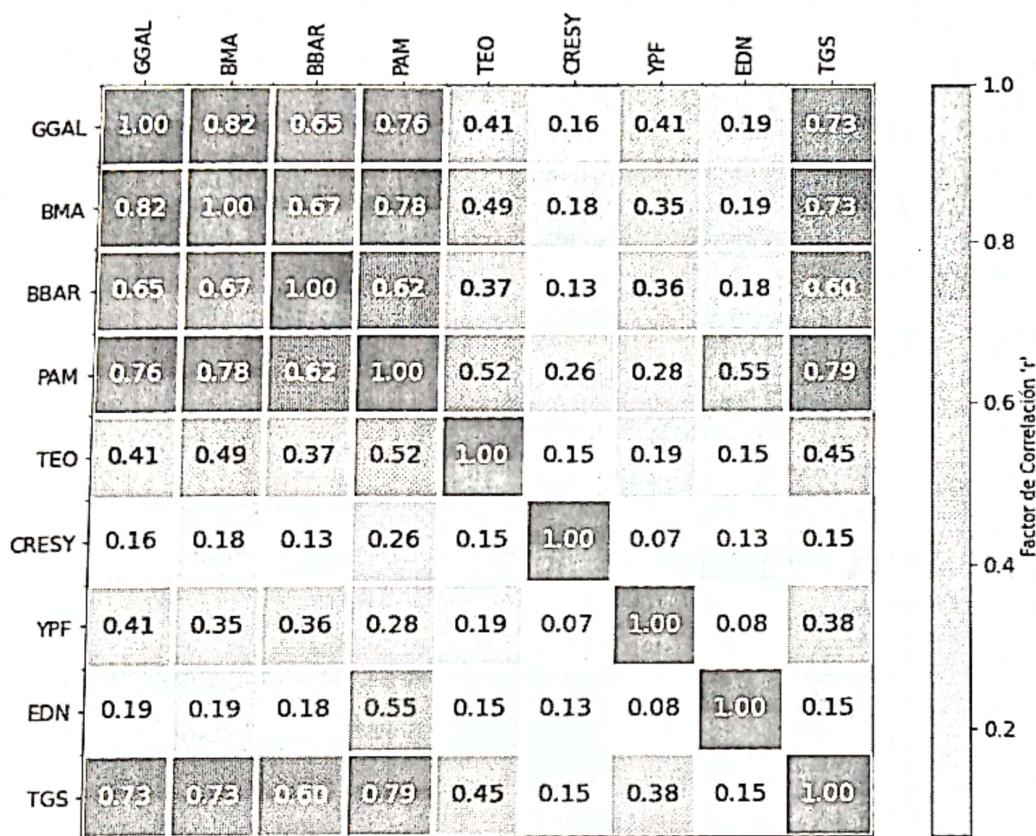
Ahora lo que vamos a hacer es probar la función pasándole como argumento una tabla de correlaciones que vamos a generar usando la función nuestra de leer el excel y ajustar el OHLC y el volumen en millones de dólares, y vamos a graficar correlación entre volúmenes

Si fueron observadores habrán notado que la función venía con un segundo argumento que era el título del gráfico que lo obvié pasar y por ende puso un título en blanco

```
tickers = ['GGAL', 'BMA', 'BBAR', 'PAM', 'TEO', 'CRESY', 'YPF', 'EDN', 'TGS']

tabla = pd.DataFrame()
for ticker in tickers:
    data = ajustarExcel(ticker, carpeta='ADRs')
    tabla = pd.concat([tabla, data['volMlnUSD']], axis=1)
tabla.columns = tickers

plt = graficaCorr(tabla.corr())
plt.show()
```



¿Se van dando cuenta la importancia de las funciones, no?

Obviamente que no vamos a estar sentados esperando que un amigo nos pase funciones copadas ni encontrarlas por ahí en un blog de python perdido en la web, cada tanto pasa y genial, pero la idea que les quiero ir presentando es que se vayan armando su propio directorio de archivos con funciones, si, así como digo, "funciones sueltas", ya mas adelante vamos a ver que esto de las funciones sueltas así nomás tampoco es lo mas copado, sino que se usan "Objetos" pero ya vamos a llegar, por ahora las funciones son una manera bastante piola de empezar a separar y ordenar el código que vayamos generando

De hecho yo recomiendo de ahora en mas que lo saben, empezar a pensar en términos de funciones, de inputs y outputs de cada "sub-proceso", por ejemplo si estamos aprendiendo a

generar HeatMaps, cuando generen uno elaborado como el del ejemplo (que de hecho lo saqué del libro anterior) directamente piensen en que reciba un DataFrame, y trabajar con eso, así luego le podemos pasar cualquier otro DataFrame de correlaciones y me lo grafica igual

Sigamos jugando con estas dos funciones, recordemos que tenemos una función `ajustarExcel()` que nos lee los datos de nuestro excel, y les ajusta los OHLC y calcula el volumen en USD; y ademas tenemos otra función que grafica matrices de correlación como HeatMaps

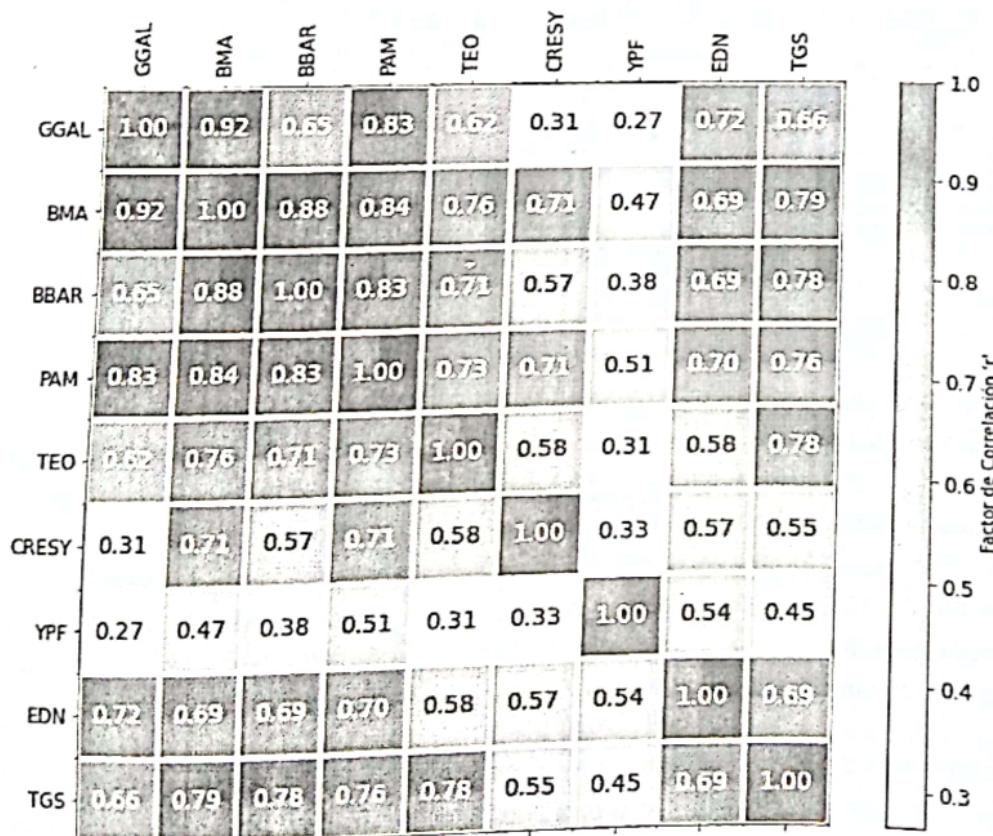
Bien, usando ambas funciones, ahora grafiquemos la correlación entre las volatilidades de los ADRs

```
tickers = ['GGAL', 'BMA', 'BBAR', 'PAM', 'TEO', 'CRESY', 'YPF', 'EDN', 'TGS']

tabla = pd.DataFrame()
for ticker in tickers:
    data = ajustarExcel(ticker, carpeta='ADRs')
    data['variacion'] = data.Close / data.Close.shift(1)-1
    data['volatilidad'] = data.variacion.rolling(40).std()
    tabla = pd.concat([tabla, data['volatilidad']], axis=1)
tabla.columns = tickers

plt = graficaCorr(tabla.corr(), title='Correlación de Volatilidades')
plt.show()
```

Correlación de Volatilidades



# Pensando funcionalmente

Si repasamos un poco lo que venimos viendo hasta este 4º Libro de la serie, tenemos que en el primer libro vimos las estructuras lógicas de flujo de un script, en el segundo vimos el trabajo con dataFrames o matrices de datos, y en el tercero vimos gráficas. Bien, con esa simple estructura ya puedo armar un primer proxy de "esquema funcional", vamos a ver un poco que quiero decir

Supongamos que se me ocurre un determinado análisis de datos, con determinados parámetros y quiero estudiar una determinada métrica.. Bien, así dicho parece medio un chino, pero pensemos así, ¿que necesito?

- \* 1- Conseguir los datos de entrada (input, dataFeed)
- \* 2- Trabajar esos datos matricialmente (DataSet Prepare)
- \* 3- Calcular las métricas (Research)
- \* 4- Reportar los resultados (Report)

Bien, ahora pensemos lo siguiente

1. Son los excels, en este bloque vamos a ver como conseguir date Feed de APIs y webs, pero por ahora esta parte viene de los excels que tenga guardados en mi disco rígido, así y todo puedo definir una función que los lea y me devuelva los datos en una matriz
2. Esta es la parte de trabajo de matriz, filtro, resampleo, cálculos con rolling, cummulative, shifts etc.. Todo esto puede entrar en una función donde haga este amasado de los datos
3. Las métricas finales suelen ser cálculos estadísticos, ratios, tablas de resumen etc que los calculo con las herramientas básicas de python por lo general
4. Acá se suele usar una librería gráfica como matplotlib para reportar de manera visual el resultado que genere

Esta lista no es taxativa ni tiene por que ser así, es un ejemplo de ordenamiento que les sugiero

Lo bueno ahora que estamos aprendiendo funciones es que vamos a poder ordenar mucho mejor el código que escribimos por ahí, si de entrada ya pensamos en separar las funciones por un criterio de "funcionalidad" valga la redundancia, a la larga se nos va a ir haciendo todo el coding mas natural y sencillo

Mientras que si de entrada ya partimos mal las funciones y dejamos un pedazo de script en una función que no tiene mucho que ver, luego la vamos a sufrir se los aseguro porque me pasó mil veces, por eso ahora si bien parece denso es el momento para empezar a generar buenas prácticas porque a la larga nos van a ayudar un montón

## Aplicaciones Prácticas

Antes de pasar al tema de las APIs y los data Feed, vamos a meter alguna cosita mas de estudio quant, si bien bastante básica, con los elementos que vimos hasta ahora es algo interesante que se puede ir haciendo para que esto no se torne tan teórico

Bien, vamos a leer los datos de los ADRs bancarios, a los que llamaremos "ADRs" en una lista. Luego vamos a hacer los cálculos necesarios para poder decidir cual de los dos tuvo mejor rendimiento en un supuesto método de trading que consista en estar comprado cuando el activo supera a una media móvil exponencial en "n" ruedas (siendo "n" un número entre fromN y toN, obviamente dos variables a parametrizar en los inputs del script) y estar vendido cuando pasa lo contrario (para ello tomaremos el rendimiento diario del día siguiente al día en que se cumple la condición de comprado o vendido al cierre)

Ahora bien, a fin de comprar los rendimientos para cada ADR por separado, vamos a calcular el rendimiento medio por rueda, de los métodos de comprado y vendido (en la posición vendida vamos a sumar el rendimiento de los cierres ajustados cambiando de signo obviamente, y sin considerar tasa para simplificar)

Por último necesitaremos para visualizar no solo el número medio del ADR para todos sus "n" sino toda la performance para los distintos "n" un gráfico de líneas que superponga el rendimiento de ambos activos para cada "n"

```
# FUNCION 1 - dataFeed y preparación
# Esta función solo abre el archivo y le calcula unas columnas

import matplotlib.pyplot as plt, pandas as pd

def abrirExcel(ticker, carpeta=''):

    if carpeta == '':
        ruta = ticker+'.xlsx'
    else:
        ruta = carpeta + '/' + ticker + '.xlsx'
    data = pd.read_excel(ruta)
    data = data.sort_values('timestamp', ascending=True)
    data.set_index('timestamp', inplace=True)
    ret = data.adj_close.to_frame()
    ret.columns = ['Close']
    ret['Yield']=(ret['Close']/ret['Close'].shift()-1)*100

    return ret.round(2).dropna()
```

A continuación les muestro las otras dos funciones..

```
# FUNCION 2 - Calculo rendimientos de estrategias

def getYields(data, fromEMA, toEMA):
    yields = [] timeIn = []
    for i in range(fromEMA, toEMA+1):
        key = 'EMA_'+str(i)
        data[key]=data.Close.ewm(span=i).mean()
        data['comprado'] = data.Close.shift() > data[key].shift()
        data['vendido'] = data.Close.shift() < data[key].shift()
        allIn = data.loc[data.comprado == True]['Yield']
        allOut = data.loc[data.vendido == True]['Yield']
        qIn = allIn.count()
        qOut = allOut.count()
        qTot = allIn.count() + allOut.count()
        yields.append((allIn.mean()*qIn-allOut.mean()*qOut)/qTot)
        timeIn.append(100*qIn/qTot)
    return yields, timeIn
```

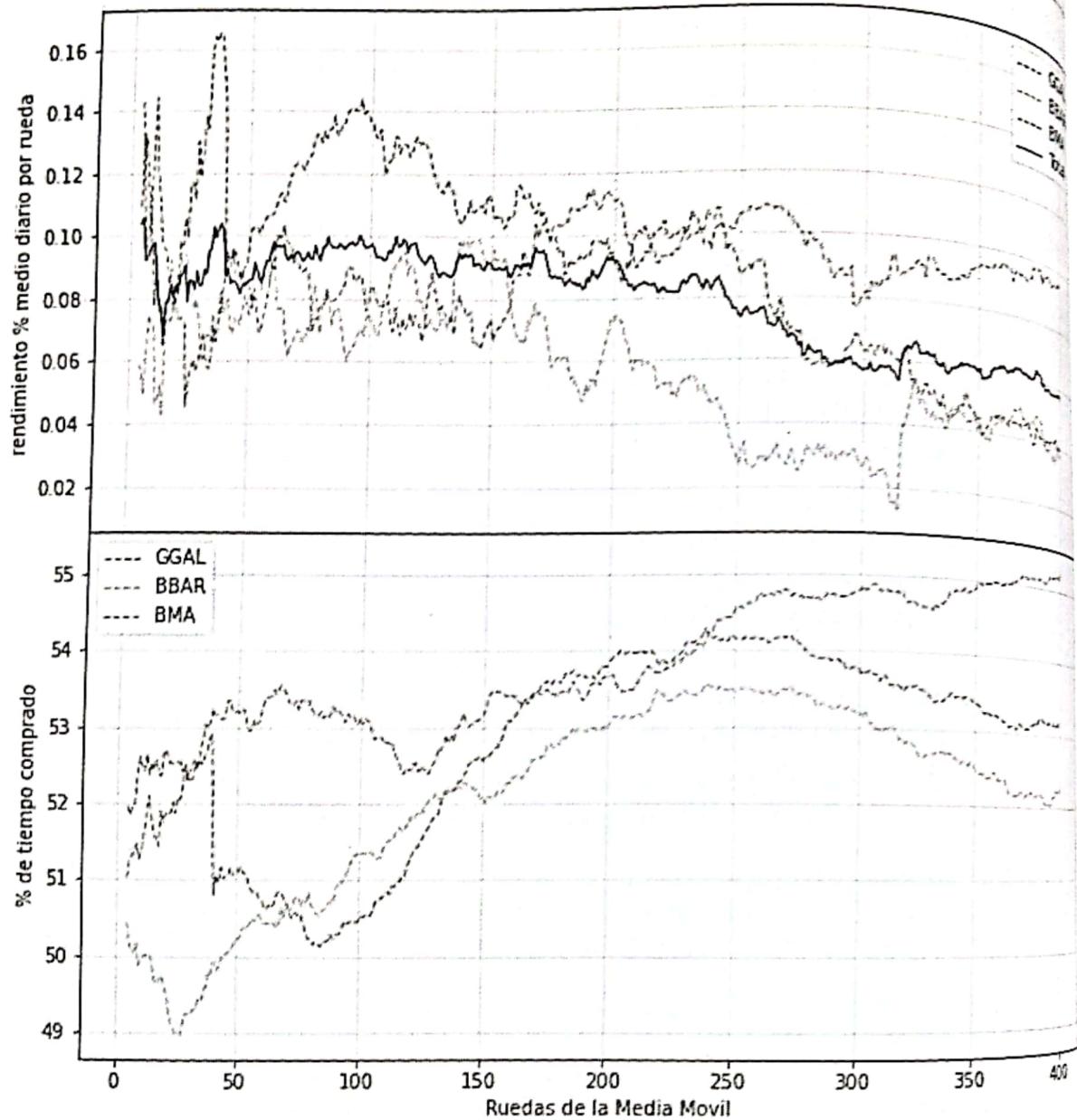
```
# FUNCION 3 - Gráfico final
def Graficar(ADRs, fromEMA, toEMA):
    ejeX = [i for i in range( fromEMA, toEMA+1)]
    fig, (ax1,ax2) = plt.subplots(figsize=(10,10), nrows=2)
    r , yieldsMedios = [], []
    for ADR in ADRs:
        #Llamo a la función1
        data = abrirExcel(ADR, 'ADRs')
        #Llamo a la función2
        yields,timeIn = getYields(data,fromEMA, toEMA)
        # Grafico serie de rendimientos y tiempos adentro
        ax1.plot(ejeX, yields, ls='--', lw=1, label=ADR)
        ax2.plot(ejeX, timeIn, ls='--', lw=1, label=ADR)
        # Acumulo series para rendimiento promedio total al final
        r.append(yields)
        yieldsMedios.append(sum(yields)/len(yields))

    # Grafico Rendimiento medio
    yieldsTotal = [(x+y+z)/3 for x, y, z in zip(r[0], r[1], r[2])]
    ax1.plot(ejeX, yieldsTotal, color='k', lw=1, label='Total')

    # Configuro Ejes
    ax1.legend()
    ax1.set_ylabel('rendimiento % medio diario por rueda')
    ax1.grid(which='major', axis='both', color='black', alpha=0.15)
    ax2.legend()
    ax2.set_xlabel('Ruedas de la Media Movil')
    ax2.set_ylabel('% de tiempo comprado')
    ax2.grid(which='major', axis='both', color='black', alpha=0.15)
    fig.subplots_adjust(hspace=0)
    return plt, yieldsMedios
```

Bueno, probemos esto:

```
# Defino Tickers  
ADRs = ['GGAL', 'BBAR', 'BMA']  
  
# Llamo a La función3  
plt, yields = Graficar(ADRs, 5, 400)  
  
# Muestro Resultados plt.show()  
for i in range(3):  
    print(ADRs[i],round(yields[i],4),end='')
```



GGAL 0.0893    BBAR 0.0576    BMA 0.0907

Lo interesante de tener el código ordenado así es que

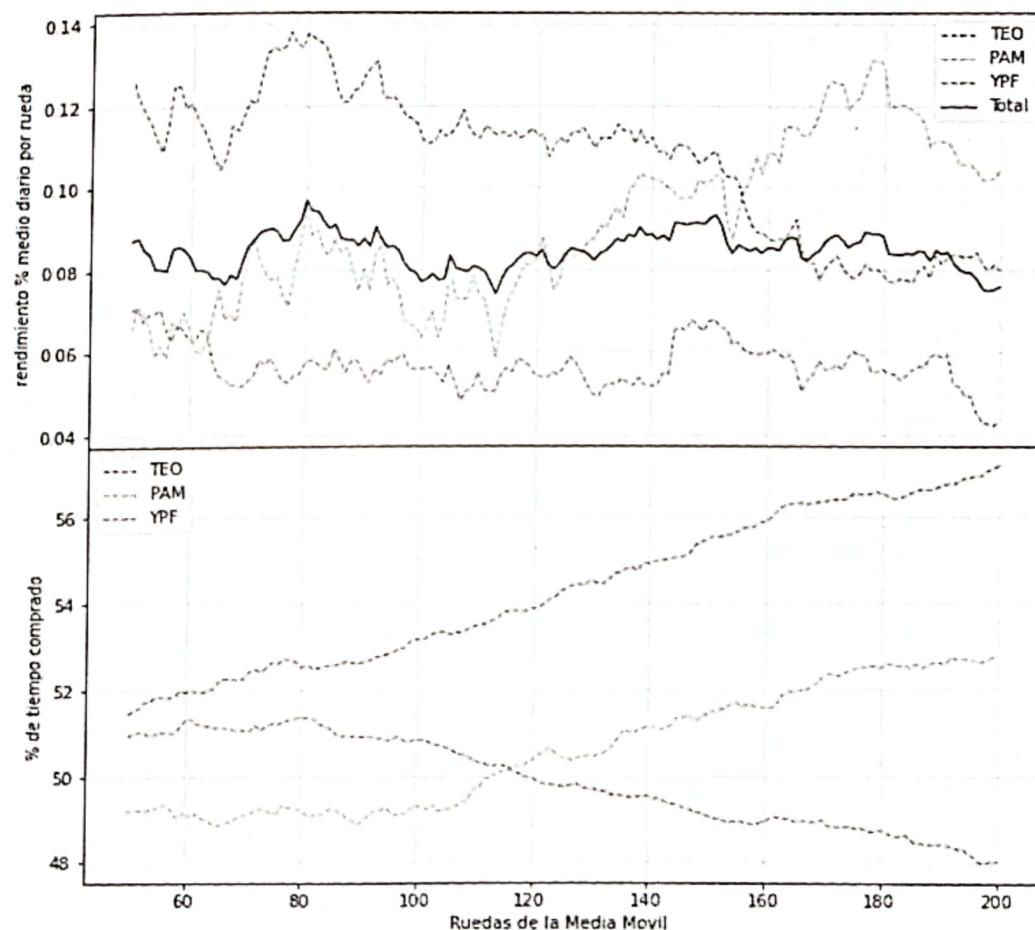
- \* Puedo agarrar cualquier función suelta y reutilizarla en otra estrategia
- \* Puedo generar funciones similares con pequeños cambios y tener mi propia biblioteca de funciones
- \* Puedo usar todas las funciones ya escritas con solo llamarlas
- \* El código se hace más fácil de entender y más manejable

Por ejemplo, probemos el mismo código para otros ADRs por ejemplo TEO, PAM e YPF, y para el rango 50 a 250 ruedas

```
# Defino Tickers
ADRs = ['TEO', 'PAM', 'YPF']

# Llamo a la función3
plt, yields = Graficar(ADRs, 50, 200)

# Muestro Resultados
plt.show()
for i in range(3):
    print(ADRs[i],round(yields[i],4),end=' ')
```



TEO 0.1064    PAM 0.0904    YPF 0.0566

# Ejercicios

1- Definir 4 funciones que se encarguen respectivamente de

- Leer el archivo Excel con los datos historicos para una acción, por ejemplo AAPL
- Calcular el indicador de análisis técnico RSI (en función de la cantidad de ruedas hacia atrás tomando como default 14)
- Calcular la recta de regresión entre RSI/RendimientoDiario en la siguiente rueda y su coeficiente de correlación y  $R^2$
- Graficar la correlación RSI/Rendimiento diario y su recta de regresión lineal

2- Empecemos a trabajar el concepto de la sobreventa y sobrecompra, hacer una función (reutilizando obviamente en las funciones del ejercicio anterior que nos sirvan) para medir el factor de correlación "r" del RSI contra el rendimiento de la rueda posterior pero filtrando hasta un valor tope del RSI (como por ejemplo <30 que indicarían en principio sobreventa)

- La función debe tomar el factor de correlación para todos los "n" (número de ruedas de la configuración del cálculo del RSI) de 2 hasta "maxRuedas" que es una variable o argumento input de la función.
- La tope "maxRSI" que indicaría sobreventa, también debe ser un argumento de la función
- El nombre del ticker y su carpeta también deben ser un argumento de la función
- Si la cantidad de puntos filtrados es menor a 20, la función debe devolver "None" en lugar de calcular el cator de correlación
- Hacer una función similar pero con topes mínimos para el RSI (SobreCompra)
- Hacer una función similar con ambos topes (Zona Media)
- Aplicarle las tres funciones a GGAL y graficar las 3 series
- Extraer conclusiones

3- Reutilizando las funciones que sean útiles de los ejercicios anteriores, hacer un grafico de líneas que muestre como varía el coeficiente de correlación entre el valor del RSI y el rendimiento al dia siguiente, en función de la cantidad de ruedas hacia atrás que se use en calcular el RSI

Hacer este ejercicio ya pensando que conviene meter lo que pueda ser reutilizable en otra función (esto va a servir para el ejercicio siguiente, pero la idea es que no lo lean aun, y solos craneen como pueden separar una parte del código de este ejercicio en una función cosa que les pueda servir para otros análisis

4- Usando las funciones de los ejercicios anteriores armar un script que recorra 12 ADRs de la carpeta ADRs y que grafique la sensibilidad del coeficiente de correlacion RSI/Rendimiento Siguiente Rueda, en función de la cantidad de ruedas hacia atrás que se toman para cálculo del RSI

5- Usando las funciones de los ejercicios anteriores, armar un script que recorra los 12 ADRs y grafique la sensibilidad del coeficiente de correlación RSI/Resndimiento sig rueda, pero segmentado según se encuentre en zona de sobreVenta, sobreCompra o neutral, y hacer cada gráfico en un recorrido de 2 a 40 ruedas de la configuracion del RSI

6- Definir dos funciones para grficar luego haciendo uso previo de las mismas, la correlación entre un cruce de medias (mediaRapida/mediaLenta) y el rendimiento futuro de la acción para los siguientes parámetros:

- Parámetro Input: Media móvil rápida, cantidad de ruedas
- Parámetro Input: Media móvil lenta, cantidad de ruedas
- La cantidad de ruedas fw futuras a observar debe variar entre 2 y 250
- Parámetro Input: Ticker y Carpeta (de donde lee el excel)

Para optimizar el script se pide recorrer de 2 a 250 ruedas forward pero de a 3

7- Armar un script bien sencillo que arme pares de cruces de medias con las siguientes restricciones:

- \* Definir de antemano la media mas baja de todos los cruces y la mas alta
- \* Que la primera media de cada par sea menor que la segunda
- \* Que la segunda media sea un 50% mas alta que la primera como mínimo
- \* Que la diferencia sea al menos de 10
- \* Ej para 5(min),17(max) debe devolver:  
[[5, 15], [5, 16], [5, 17], [6, 16], [6, 17], [7, 17]]
- \* Ej para 40(min),63(max) debe devolver:  
[[40, 61], [40, 62], [40, 63], [41, 62], [41, 63]]

8- Siguiendo con la idea del cruce de medias, un script de 3 funciones para encontrar la mejor correlación entre cruces de medias y rendimiento futuro, cuyas 3 funciones sean:

- DataFeed: Por ahora sería la función que lee el excel sabiendo el ticker y la carpeta donde se ubica
- Generador de cruces: La función del ejercicio anterior que pasándole un minimo y un maximo, nos devuelve los pares de cantidades de velas para la media rápida lenta del cruce
- Buscador de mejor cruce (esta función acepta como input, la dataFeed de la función1, los cruces de la función2, y la cantidad de ruedas a futuro con la que quiero correlacionar al cruce Me debe devolver cual de los cruces es el óptimo en factor de correlación y el coeficiente de correlación propiamente dicho)

# Respuestas

```
#-----#
# Rta Ejercicio 1 #
#-----#

import matplotlib.pyplot as plt, pandas as pd, numpy as np
def abrirExcel(ticker, carpeta=''):
    if carpeta == '':
        ruta = ticker+'.xlsx'
    else:
        ruta = carpeta + '/' + ticker + '.xlsx'
    data = pd.read_excel(ruta)
    data = data.sort_values('timestamp', ascending=True)
    data.set_index('timestamp', inplace=True)
    ret = data.adjusted_close.to_frame()
    ret.columns = ['Close']
    ret['Yield']=(ret['Close']/ret['Close'].shift()-1)*100
    return ret.round(2).dropna()

def rsi(data, ruedas=14):
    rsi = pd.DataFrame(data)
    rsi['dif'] = rsi['Close'] - rsi['Close'].shift(1)
    rsi['win'] = np.where(rsi['dif']>=0, rsi['dif'], 0)
    rsi['loss'] = np.where(rsi['dif']<=0, abs(rsi['dif']), 0)
    rsi['ema_win'] = rsi.win.ewm(alpha=1/ruedas).mean()
    rsi['ema_loss'] = rsi.loss.ewm(alpha=1/ruedas).mean()
    rsi['rs'] = rsi.ema_win / rsi.ema_loss
    rsi['rsi'] = 100 - (100 / (1+rsi.rs))
    rsi['nextYield'] = rsi.Yield.shift(-1)
    return rsi.reset_index().dropna()

def calcReg(serie1, serie2):
    regresion = {}
    b_1 = serie1.cov(serie2) / serie1.var()
    b_0 = serie2.mean() - b_1* serie1.mean()
    regresion['recta'] = b_0 + b_1*serie1
    regresion['r'] = round(serie1.corr(serie2),4)
    regresion['r2'] = round(serie1.corr(serie2)**2,4)
    return regresion

def grafCorr(serie1, serie2, regresion):
    fig, ax = plt.subplots(figsize=(8,8))
    lab = 'Coef r: '+str(regresion['r'])
    lab += '\nR^2: '+str(regresion['r2'])
    ax.plot(serie1,regresion['recta'], color='k', label=lab)
    ax.scatter(serie1, serie2, s=1)
    xmin = serie1.quantile(0.005)
    xmax = serie1.quantile(0.995)
    ymin = serie2.quantile(0.005)
    ymax = serie2.quantile(0.995)
    ax.set_xlim(xmin,xmax)
    ax.set_ylim(ymin,ymax)
    plt.legend(loc='upper right')
    return plt
```

```

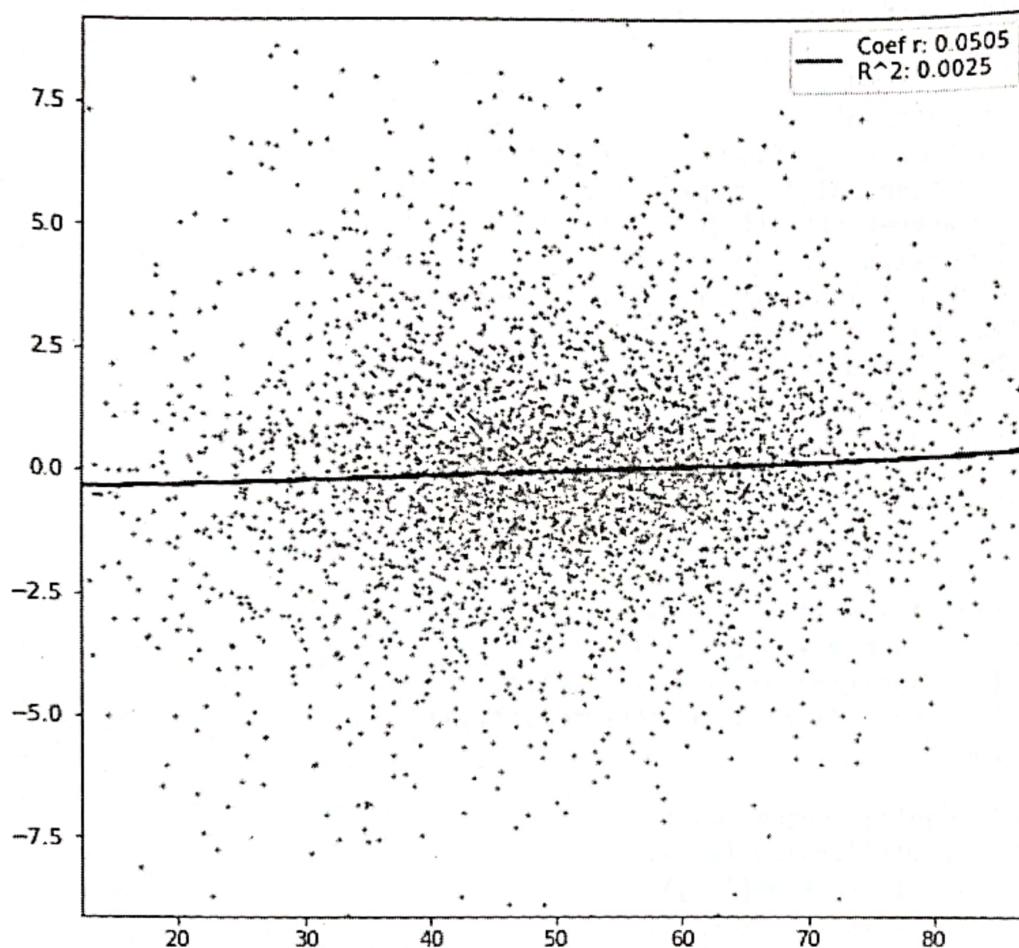
#-----#
# Ejercicio 1 Cont. #
#-----#
# esto es cuando te manda por salida o consola
# Advertencias que no queremos que nos siga mostrando

import warnings
warnings.filterwarnings('ignore')

# Ahora uso las funciones previamente armadas:

data = abrirExcel('YPF', 'ADRs')
rsiDF = rsi(data, 14)
regresion = calcReg(rsiDF.rsi, rsiDF.nextYield)
plt = grafCorr(rsiDF.rsi, rsiDF.nextYield, regresion)
plt.show()

```



```

#-----#
# Ejercicio 2 #
#-----#

def corrRSIsobreventa(ticker, carpeta='', maxRuedas=60, maxRSI=30):

    ticker = ticker
    data = abrirExcel(ticker, carpeta=carpeta)
    rs=[]
    rsEjeX = [i for i in range(2,maxRuedas)]

    for ruedas in range(2,maxRuedas):

        #Usamos La funcion "rsi" del ejercicio anterior
        rsiDF = rsi(data, ruedas=ruedas)
        rsiDF = rsiDF.loc[rsiDF.rsi < maxRSI]

        #Verificamos que el filtro tenga al menos 20 datos
        if rsiDF.nextYield.count() > 20:
            # Usamos la función "calcReg" del ejercicio anterior
            r = calcReg(rsiDF.rsi, rsiDF.nextYield)['r']
        else:
            r=None

        rs.append(r)
    return(rsEjeX, rs)

# Lo mismo para sobreCompra cambiando solo filtro
def corrRSIsobrecompra(ticker, carpeta='', maxRuedas=60, minRSI=70):

    ticker = ticker
    data = abrirExcel(ticker, carpeta=carpeta)
    rs=[]
    rsEjeX = [i for i in range(2,maxRuedas)]

    for ruedas in range(2,maxRuedas):
        rsiDF = rsi(data, ruedas=ruedas)
        rsiDF = rsiDF.loc[rsiDF.rsi > minRSI]

        if rsiDF.nextYield.count() > 20:
            r = calcReg(rsiDF.rsi, rsiDF.nextYield)['r']
        else:
            r=None
        rs.append(r)
    return(rsEjeX, rs)

```

Continúa en la página siguiente..

```

def corrRSIzonamedia(ticker, carpeta='', maxRuedas=60, minRSI=30, maxRSI=70):
    ticker = ticker
    data = abrirExcel(ticker, carpeta=carpeta)
    rs=[]
    rsEjeX = [i for i in range(2,maxRuedas)]

    for ruedas in range(2,maxRuedas):
        rsiDF = rsi(data, ruedas=ruedas)
        rsiDF = rsiDF.loc[rsiDF.rsi > minRSI]
        rsiDF = rsiDF.loc[rsiDF.rsi < maxRSI]
        if rsiDF.nextYield.count() > 20:
            r = calcReg(rsiDF.rsi, rsiDF.nextYield)['r']
        else:
            r=None
        rs.append(r)

    return(rsEjeX, rs)

ticker = 'GGAL'
fig, ax = plt.subplots(figsize=(10,5))

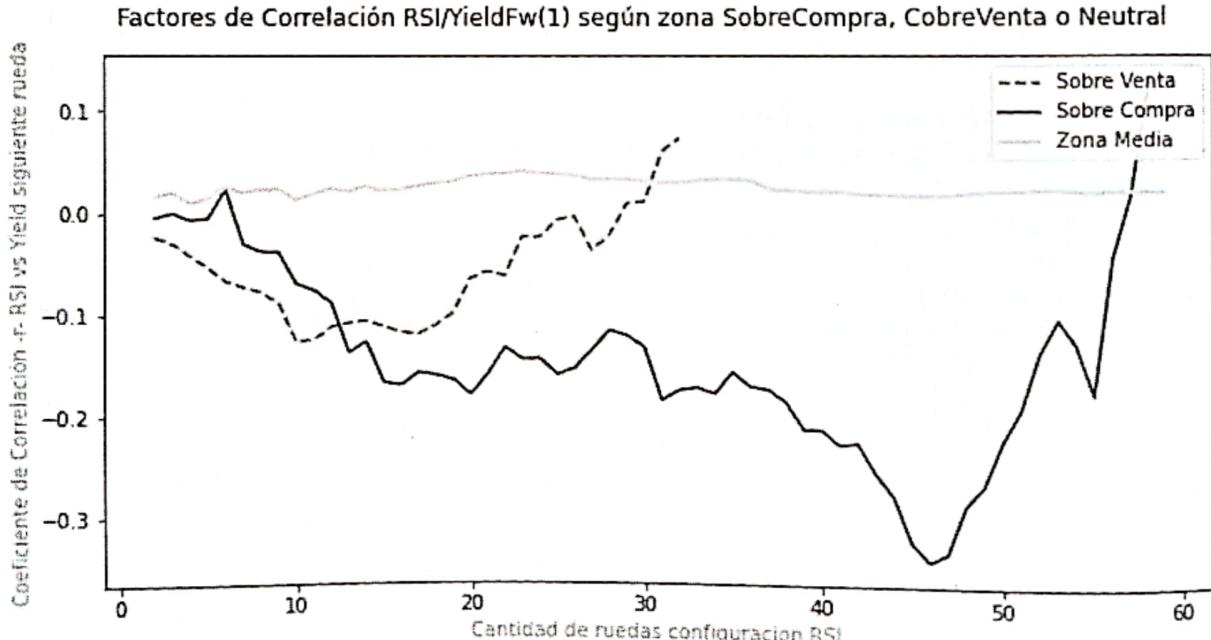
rsEjeX, rs = corrRSIsobreventa(ticker, carpeta='ADRs', maxRSI=30)
ax.plot(rsEjeX, rs, 'r--', label='Sobre Venta')

rsEjeX, rs = corrRSIsobrecompra(ticker, carpeta='ADRs', minRSI=70)
ax.plot(rsEjeX, rs, 'k-', label='Sobre Compra')

rsEjeX, rs = corrRSIzonamedia(ticker, carpeta='ADRs', minRSI=30, maxRSI=70)
ax.plot(rsEjeX, rs, 'lightgray', lw=2, label='Zona Media')

ax.set_ylabel('Coeficiente de Correlación ...', color='gray')
ax.set_xlabel('Cantidad de ruedas configuracion RSI', color='gray')
plt.legend(loc='upper right')
plt.suptitle('Factores de Correlación RSI/YieldFw(1) según zona ...', y=0.95)
plt.show()

```



Conclusión: Obviamente no hace falta mucha observación para darse cuenta que claramente hay una gran diferencia entre la correlación que tiene el indicador RSI en las zonas de sobrecompra o sobreventa con la que tiene en las zonas medias, claramente hay una correlación mucho más fuerte en las zonas extremas, al parecer en GGAL negativas (a menor RSI comprar en sobreVenta y mayor RSI vender en sobreCompra)

Obviamente, los invito a ponerse a jugar con el código en diferentes Acciones para ver que observan en cada una, si bien este es un ejemplo demasiado básico, es un buen proxy al tipo de análisis que se hace al evaluar que métricas se van a tomar, en que rangos y con qué lógica, para una estrategia de trading del tipo cuantitativa

Como verán el código se va poniendo "denso" o largo digamos, con lo cual si no empezara a usar funciones que se encarguen cada una de algo concreto, se empieza a descontrolar la cantidad de líneas de código y se hace muy poco entendible y "mantenible"

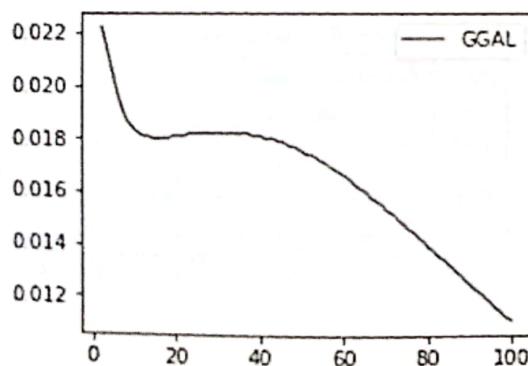
Una buena métrica a tener en cuenta, en python, sería que para un script de análisis de datos que haga una sola persona (no hablo de equipos de trabajo), no deberíamos superar las 100 líneas, si pasa eso es porque mandamos mucho código innecesario que podemos meter dentro de ciclos, o bien porque no estamos usando funciones donde deberíamos usarlas, después, desde ya, podemos importar funciones pero el código en sí sin contar las funciones que usemos, no debería superar las 100 líneas

```
#-----#
# Rta Ejercicio 3 #
#-----#

def rsiSensibilidad(ticker, carpeta='', nMax=30):
    data = abrirExcel(ticker, carpeta)
    rs = []
    for i in range(2,nMax+1):
        rsiDF = rsi(data, i)
        regresion = calcReg(rsiDF.rsi, rsiDF.nextYield)
        rs.append(abs(regresion['r']))

    x = [i for i in range(2,nMax+1)]
    return ticker, x, rs

ticker, x, rs = rsiSensibilidad(ticker='GGAL', carpeta='ADRs', nMax=100)
fig, ax = plt.subplots(figsize=(4,3))
ax.plot(x,rs, label=ticker)
plt.legend(loc='upper right')
plt.show()
```



```

#-----#
# Rta Ejercicio 4 #
#-----#



import math
tickers = ['BBAR', 'BMA', 'CEPU', 'CRESY', 'EDN', 'GGAL', 'LOMA', 'PAM',
           'SUPV', 'TEO', 'TGS', 'YPF']

fig, ax = plt.subplots(figsize=(14,10), nrows=3, ncols=4)

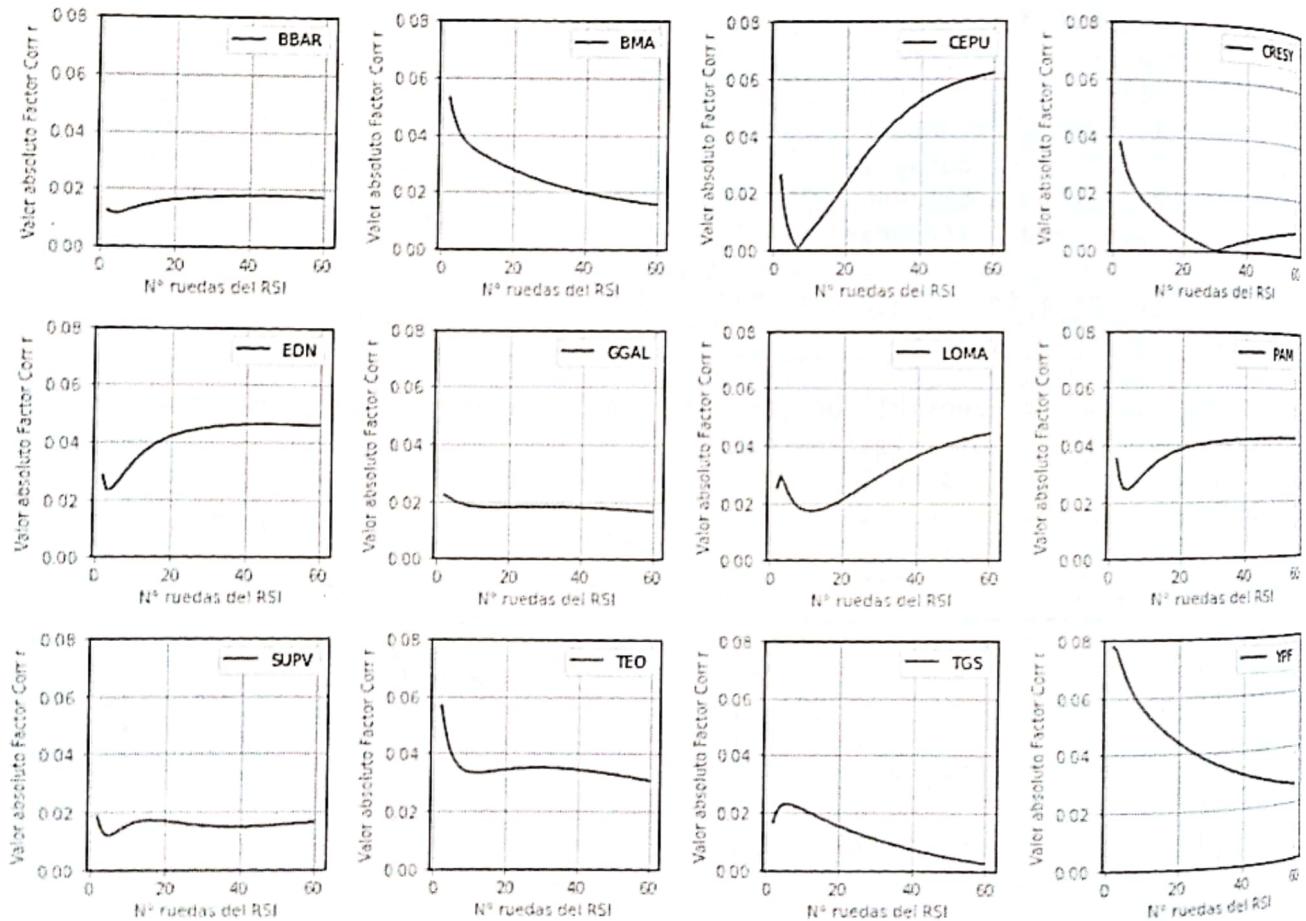
nMax=60
for i in range(len(tickers)):
    fila = math.floor(i/4)
    col = i%4
    ticker, x, rs = rsiSensibilidad(tickers[i], carpeta='ADRs', nMax=nMax)
    ax[fila][col].plot(x,rs, label=tickers[i])
    ax[fila][col].set_ylimit(0,0.08)
    ax[fila][col].legend(loc='upper right')
    ax[fila][col].grid(which='major')
    ax[fila][col].set_xlabel('Nº ruedas del RSI', color='gray')
    ax[fila][col].set_ylabel('Valor absoluto Factor Corr r', color='gray')

```

plt.suptitle('ABS(r), de Corr entre RSI y el rendimiento de la rueda siguiente, en función del "n" del RSI', y=0.93)

plt.subplots\_adjust(wspace=0.45, hspace=0.35)  
plt.show()

ABS(r), de Corr entre RSI y el rendimiento de la rueda siguiente, en función del "n" del RSI



```

#-----#
# Rta Ejercicio 5 #
#-----#


tickers = ['BBAR', 'BMA', 'CEPU', 'CRESY', 'EDN', 'GGAL', 'LOMA', 'PAM',
           'SUPV', 'TEO', 'TGS', 'YPF']

fig, ax = plt.subplots(figsize=(15,12), nrows=3, ncols=4)

n=40
for i in range(len(tickers)):
    fila = math.floor(i/4)
    col = i%4
    ticker = tickers[i]
    x, rs = corrRSIsobreventa(ticker, maxRuedas=n, carpeta='ADRs', maxRSI=30)
    ax[fila][col].plot(x,rs, 'r--', label='SobreVenta')

    x, rs = corrRSIsobrecompra(ticker, maxRuedas=n, carpeta='ADRs', minRSI=70)
    ax[fila][col].plot(x,rs, 'k-', label='SobreCompra')

    x,rs = corrRSIzonamedia(ticker,maxRuedas=n,carpeta='ADRs',minRSI=30,m axRSI=70)
    ax[fila][col].plot(x,rs, 'k-.', alpha=0.3, label='Neutral')
    ax[fila][col].set_ylimit(-0.3,0.3)
    ax[fila][col].legend(loc='upper right', fontsize=10)
    ax[fila][col].grid(which='major')
    ax[fila][col].set_xlabel('Nº ruedas del RSI', color='gray')
    ax[fila][col].set_ylabel('Valor absoluto Factor Corr r', c='gray')
    ax[fila][col].set_title(tickers[i],y=0.03, fontweight='bold', alpha=0.3,
                           fontsize=25, c='gray')

plt.suptitle("Coeficiente de Correlación entre RSI y el rendimiento de la rueda
siguiente\nEn función del 'n' del RSI, para estados de SobreCompra y
SobreVenta", y=0.96)

plt.subplots_adjust(wspace=0.45, hspace=0.35)
plt.show()

```

A grandes rasgos se ve claramente que en las zonas de sobrecompra y sobreventa las correlaciones son mucho mayores, incluso se nota que tienen a ser negativas (corroborando la hipótesis de manual que en la zona de sobrecompra hay mayor probabilidad de baja inminente y en la zona de sobreventa mayor probabilidad de rebote inminente)

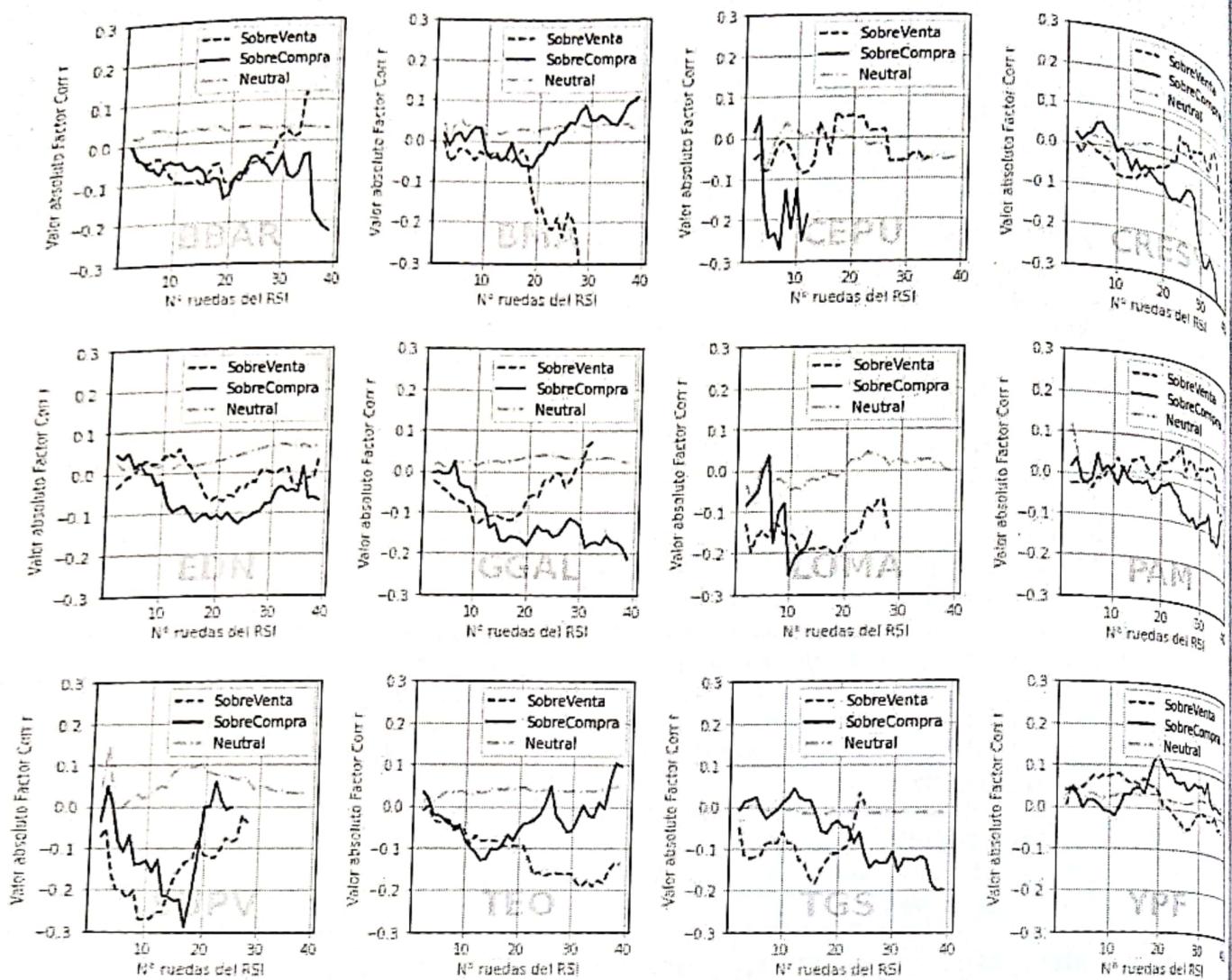
Bueno, como pueden ver, esto de tener funciones nos va permitiendo escalar la complejidad de los análisis sin necesidad de que el código final sea un loquero inentendible

Usamos 3 funciones (una para cada una de las zonas del RSI)

Pero a su vez cada una de las 3 funciones usa otras 3, la del RSI y la de correlaciones y la de abrir el Excel con los datos

Imaginen si no hubiéramos usado funciones lo que sería el código entero de esas 30 líneas

**Coeficiente de Correlación entre RSI y el rendimiento de la rueda siguiente  
En función del 'nº del RSI, para estados de SobreCompra y SobreVenta**



Pego primero la del ejercicio 7 para aprovechar el espacio 😊

```
#-----#
# Rta Ejercicio 7 #
#-----#
def crearCruces(mmMin, mmMax):
    cruces = [[i,j] for i in range (mmMin,mmMax+1)
              for j in range(mmMin+1,mmMax+1)]
    if (j <= mmMax) & (j > i*1.5) & (j >= i+10)]
        return cruces
crearCruces(40,63)
[[40, 61], [40, 62], [40, 63], [41, 62], [41, 63]]
```

```

#-----#
# Rta Ejercicio 6 #
#-----#

import pandas as pd
import matplotlib.pyplot as plt

def cruce(data, fast=50, slow=200, fw=20):
    cruce = pd.DataFrame(data['Close'])
    cruce['fwYield'] = (cruce['Close'].shift(-fw)/cruce['Close']-1)*100
    cruce['smaFAST'] = cruce['Close'].rolling(fast).mean()
    cruce['smaSLOW'] = cruce['Close'].rolling(slow).mean()
    cruce['cruce'] = (cruce['smaFAST']/cruce['smaSLOW']-1)*100
    return cruce.dropna().round(2)

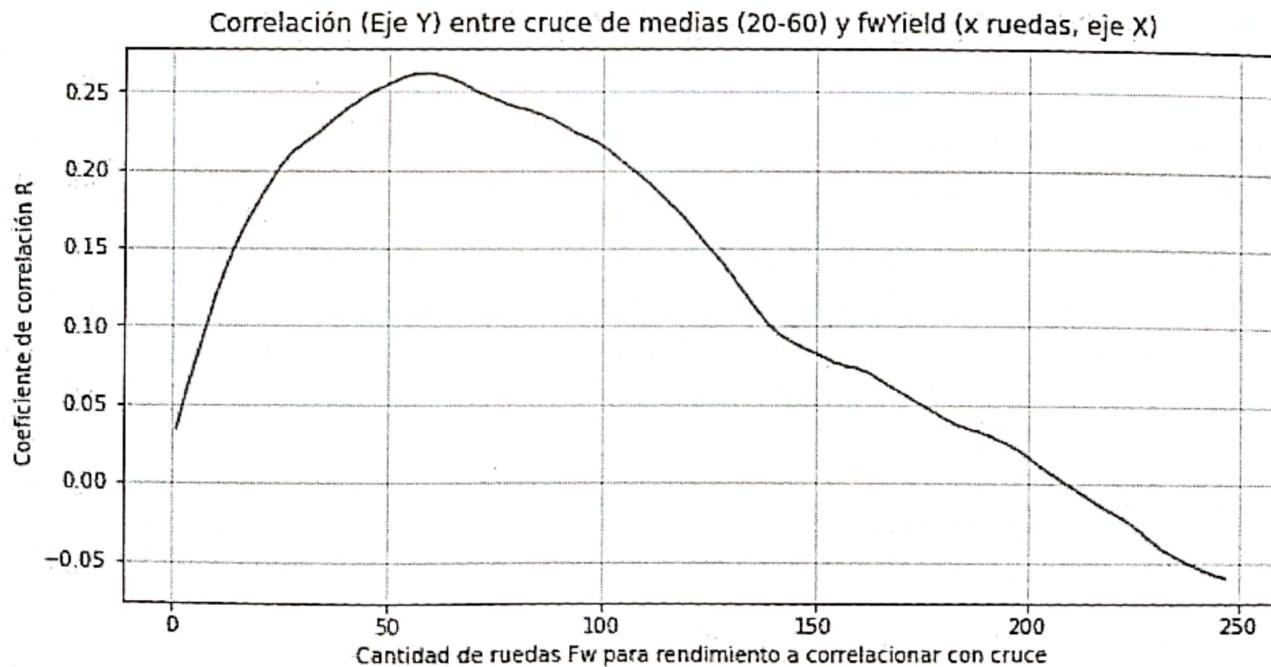
def corrCruce(data, fast, slow):
    lista=[]
    for i in range(1,250,3):
        c = cruce(data, fast=fast, slow=slow, fw=i)
        regresion = calcReg(c.cruce, c.fwYield)
        lista.append(regresion['r'])
    return lista

data = abrirExcel(ticker, carpeta)
lista = corrCruce(data, fast=20, slow=60)

fig, ax = plt.subplots(figsize=(10,5))
ax.plot([i for i in range(1,250,3)], lista)
ax.set_xlabel('Cantidad de ruedas Fw para rendimiento a correlacionar..')
ax.set_ylabel('Coeficiente de correlación R')
ax.grid()
plt.suptitle('Correlación (Eje Y) entre cruce de medias ('+str(fast)+'-'+str(slow)+') y fwYield (x ruedas, eje X)', y=0.93)

plt.show()

```



```
#-----#
# Rta Ejercicio 8 #
#-----#  
  
def mejorCruce(data, cruces, fw):  
    maximo = 0  
    for mm in mmCruces:  
        data['fwYield'] = (data['Close'].shift(-fw)/data['Close']-1)*100  
        data['mmFast'] = data.Close.rolling(mm[0]).mean()  
        data['mmSlow'] = data.Close.rolling(mm[1]).mean()  
        data['cruce'] = (data['mmFast'] / data['mmSlow'] -1)*100  
        r = round(data['cruce'].corr(data['fwYield']),4)  
  
        if r > maximo:  
            mejor = [mm[0],mm[1],r]  
            maximo = r  
    return mejor  
  
data = abrirExcel('TEO', 'ADRs')  
cruces = crearCruces(5,50)  
mejor = mejorCruce(data, cruces, 100)  
mejor
```

[6, 12, 0.1079]

# APIs

El significado de la sigla es Application Programming Interfaces, o sea son como "puentes" interfaces para unir programas, así como los puentes unen ciudades las APIs unen aplicaciones, por ejemplo la aplicación de un broker manda las órdenes de sus clientes al mercado, para ello el broker se genera su propia aplicación para interactuar con el mercado, ahora bien, si nosotros programamos un bot y queremos que nuestro broker mande al mercado las órdenes que decide nuestro bot, hay que unir ambos programas (aplicaciones), es decir necesitamos algo que "una" nuestro programa (nuestro bot) con el programa del broker (su plataforma de acceso al mercado), de eso por ejemplo se encargan las APIs

Lo distintivo de las APIs es que son agnósticas a la tecnología de "los programas que unen" es decir, mi broker puede tener su acceso al mercado escrito en C++ (un lenguaje de programación) y yo puedo tener mi bot en python, nada que ver, y sin embargo ambos entendemos el estandar de la API, y por lo tanto mi programa escrito en un lenguaje se puede comunicar perfectamente y recibir inputs y generarle outputs al programa del broker

Obviamente hay APIs de todo tipo, hay APIs que me permiten manejar una cuenta en una red social y hay APIs que me permiten comprar por un marketplace, como mercadolibre, en nuestro caso nos vamos a ocupar en este capítulo de las APIs que facilitan acceso a market data y las APIs que facilitan la conexión a mercados

Voy a dejarle una lista de mis APIs favoritas que cumplen esta función específica, solo nombrando aquellas que ofrecen una cuenta gratuita (al menos hasta cierta cantidad de uso, ya veremos este tema en detalle)

Como les digo, por lo general un modelo de negocio en este mundo muy utilizado es el freemium, es decir que ofrecen una versión FREE sin cargo con alguna limitación que para uso profesional te termina conviniendo pagar algo

De todos modos también hay un mundo en tema precios, yo acá les pongo solo APIs gratuitas y que si tienen alguna funcionalidad premium, esta no supera los usd 10 a usd 50 por mes.

En ningún caso hace falta la versión paga, los servicios de conexión son excelentes y los datos de precios muy precisos, algunas APIs tienen inconsistencias en algunos datos de ratios de fundamentales y ese tipo de datos mas difíciles de homogeneizar pero por lo general son todas excelentes, por eso no pongo APIs de menor calidad como "yahoo finance" por ejemplo

Desde ya no pongo acá las APIs caras (de miles de dolares mensuales) ya que para tratar seriamente ese tema requeriría un libro entero y en este capítulo simplemente quiero hacer un repaso de las mejores APIs para uso personal o institucional de presupuesto acotado

Sin mas presentación Les dejo estos listados, me llevó un tiempo prepararlos pero creo que valió la pena, porque hay muchas APIs y como vamos a ver las funcionalidades que nos ofrece cada una son muy variadas

#### Equity, futures, options, bonds, etc

API	WEB	Mercados	Precio	Datos	Indicadores	Comunidad	TF/Datos	Crypto	Trading	RealTime	Otros
Alpha Vantage	alphavantage.co	USA	~	~	~	~	158	540	~	~	AI
FinnHub	finnhub.io	65 Mkts	✓	~	~	~	154	794	~	✓	AI-Economics
FMP	fmpcloud.io	8 Mkts	✓	~	~	Varios	108	107	~	✓	13F SEC
IEX	iexcloud.io	14 Mkts	✓	✓	✓	Oil&Gas	Solo Pago	174	~	✓	Bonds Economics
Alpaca	alpaca.markets	USA	~	~	~	~	~	~	✓	✓	Paper Trading
IOL	api.invertironline.com	ARG	~	✓	~	~	~	~	✓	✓	Bonos
Rofex/Primary	aphub.primary.com.ar	ARG	~	✓	Varios	USDARS	~	~	✓	Futuros	Agro, Oro, WTI

#### Crypto

API	WEB	Recibido	Datos	Crypto	Trading	Stock	Futuros	Opciones	RealTime	Otros
CryptoCompare	min-api.cryptocompare.com	~	~	4660	~	~	~	~	✓	Social, coinData
HitBTC	hitbtc.com	0.065%	✓	377	✓	✓	~	~	✓	
Binance	binance.com	0.075%	~	189	✓	✓	✓	~	✓	
FTX	ftx.com	0.045%	~	78	✓	✓	✓	✓	✓	OIL
Deribit	deribit.com		✓	2	✓	~	✓	✓	✓	

Como verán no hay ninguna que las tenga todas, lo mas recomendable es aprender a usar todas, o tener una carpetita con archivos de APIs donde se vayan armando sus funciones y demás y recomiendo ampliamente que este listado que puse estén TODAS, si todas, siempre es bueno tener mas de una que brinde la misma data, para chequear cuando vemos resultados raros, o por si se cae una etc..

Es un listado de lo que considero las mejores en su rubro o fuerte digamos, y obviamente hay miles de APIs mas, y recomiendo siempre no dormirse y estar atento, esto es muy dinámico, hace tres años, del listado hay varias que no existían aún o eran demasiado mas básicas de lo que son hoy, asi que atentis siempre a la evolución de esto

Mi consejo es que para fines de práctica y didácticos estas APIs están mas que bien, y no necesitan nada de pago

- AlphaVantage: Es ideal para indicadores técnicos, buenas series históricas de USA
- FinnHub: Mucha cantidad de mercados en tiempo real, muy completa data de indicadores económicos
- FMP: Es la mas fuerte en Análisis fundamental
- IEX: Es la mas profesional, la mas trabajada y completa, pero la versión gratuita es bastante limitada y la de usd 9 tampoco permite un uso fuerte, asi que es ideal para un trabajo en una empresa o inversores profesionales
- Alpaca: Es sencilla y práctica y permite realizar trading algorítmico simulado sin plata real pero con precios y entornos reales
- InvertirOnline: Es privada de un bróker pero es lo mejorcito que hay en Argentina para trading algorítmico
- Rofex: Tiene datos de futuros en tiempo real, en pleno desarollo a mayo 2020 está bastante aceptable para trading de futuros en Argentina

## **¿Conviene usar una librería o paquete o usar las APIs con funciones propias?**

Prometo que esto es lo último teórico que comenté antes de meternos de lleno en cada una, perdón, pero quiero ir aclarando algunas cuestiones, porque sé que en algún momento van a pasar por esta duda o planteo

Conviene siempre, saber usar las APIs a "modo manual" es decir armando ustedes mismos las funciones y les pongo algunos por qué:

- Porque a veces actualizan las APIs y no los paquetes, con lo cual nos perdemos "potencia" de API por usarla del paquete
- Porque muchas veces necesitamos personalizar alguna función con algún parámetro extra que el paquete no lo brinda y nos terminamos limitando o haciendo un código menos eficiente
- Porque si bien uno se puede fijar las "estrellas" que tiene el repositorio del paquete que baja, muchas veces tienen bugs que no se notan o no tiran error y puede ser un problemón

## **Tipoos de API y OAuth**

Bien, como verán hay varios tipos de APIs que vamos a trabajar, en el caso de las primeras 7 que no son exclusivas del mundo crypto, veremos que hay APIs de solo marketData (las primeras 4) y APIs de trading (las otras 3), obviamente las APIs de trading van a tener que ligarse de algún modo a una cuenta de trading

Por otro lado verán que las que puse en la lista de cytos, solo una es de marketData y las otras son de exchanges que ofrecen cuentas de trading con conexión vía API.

El tema de la authentication es poco intuitivo, ustedes dirán (con bastante lógica) "Ahh seguro que requieren autenticación las de trading y las de marketData no"

La respuesta es No. jaja, es así, las de equity y mercados financieros (sacando crypto) son difíciles de armar porque recolectar y homogeneizar la data disponible no es tarea sencilla, por lo cual esas APIs (la primera lista) suelen tener un nivel gratuito y a partir de cierto uso un nivel pago, igual nunca superan los USD 50 para uso incluso de bots

Les decía en la primera lista al tener un modelo que empezas gratis y si querés mucho uso te empiezan a cobrar, tienen que si o si medir el uso desde un principio, por lo cual son con autenticación

Ahora en el caso de las APIs de crypto, la info de cryptos es gratuita en todos lados, y acá no hay regulaciones densas, cosas diferentes que homogeneizar ni balances, ni fillings de la SEC, con lo cual abunda la info gratuita y de excelente calidad.

Con lo cual por mas que se trate de una API con data en tiempo real y demás para lo que es marketData no piden autenticación es decir podes pedir datos a la API sin identificarte, pero para trading tiene una autenticación

Y acá viene lo importante, el mundo crypto es un mundo hermoso pero en el que hay que tener mucho cuidado del tema seguridad de claves y esas cosas, si yo dejo por ahí una clave de acceso a mi broker de acciones, no pasa nada, porque si alguien me quiere robar los fondos de mi comitente en algun momento para extraer los fondos va a tener que salir por una cuenta bancaria y en casi todos los países del mundo tiene que ser de la misma titularidad del de la comitente, así que de la seguridad de nuestros fondos en la comitente se encarga el broker

Pero en el mundo crypto es diferente, si alguien accede a mi cuenta de un exchange crypto y no tomo las medidas de seguridad necesarias, se transfiere los fondos y a cantarle a gardel

Entonces como van a ver la autenticación en las APIs de exchanges cryptos es muchisimo mas engorrosa y vuelta, incluso exigen si o si el estandar de 2FA, es decir que van a tener que andar siempre con el celu a mano para logearse cada vez que quieran cambiar algo de esa API, no cada vez que la API se loguee que es otra cosa

Entonces, resumiendo, se podria decir que:

- APIs de acciones, futuros y bonos, son casi siempre con logueo light que es un simple token
- Ojo que hay intermedios que tienen tokens que se refrescan cada tanto tiempo que son un poco mas seguras que un token único, pero siguen siendo vulnerables para la seguridad que requiere el trading crypto
- APIs de exchanges cryptos tienen una parte de marketData sin logueo, y una parte de trading con logueo Hard

## Generalidades de una API

Lo primero siempre que vamos a hacer al analizar la conveniencia de usar o no una API, despues de ver el precio obvio, es ir a la documentación y chusmear por arriba que onda la API, acá solo a modo de generalidades vamos a ver las cosa que tienen en común

Al entrar a la documentación de una API vamos a tener por lo general siempre la misma estructura de organización de la información, esta estandarizado (salvando los casos de APIs argentinas que quedaron en el tiempo jaja)

Les pego una a modo de ejemplo (la de IEX)

The screenshot shows the IEX Cloud API documentation. On the left is a sidebar with a navigation menu:

- Introduction
- API Reference
- API Versioning
- Attribution
- Authentication
- Batch Requests
- Data Formats
- Disclaimers
- Error Codes
- Filter results
- How Messages Work
- Query Parameters
- Request Limits
- Security
- SSE Streaming
- Support
- Developer Tools
  - Client and server libraries
  - Testing Sandbox

The main content area has a large heading "Introduction" and a sub-section "API Reference". The "API Reference" section contains a note about RESTful URLs, JSON responses, and standard HTTP codes. It also lists two bullet points:

- The base url for the API is: <https://cloud.iexapis.com/>
- We support JSONP for all endpoints.

Below the "API Reference" section is another sub-section "API Versioning" with a note about releasing new versions and supporting up to three active versions. It also lists two bullet points under "Backwards compatible changes":

- Adding new response attributes
- Adding new endpoints

Como les decía, la estructura general es mas o menos siempre la misma:

- Barra de navegación a la izquierda con una especie de "índice" de contenidos o funcionalidades de la API, generalmente está organizada por secciones y dentro de cada sección está cada función
- Los primeros contenidos son por lo general informativos (version, créditos, roadmap etc)
- Las primeras funciones suelen ser un "Getting Started" o primeros pasos, por lo general acá tenemos el tema de la autenticación y a veces una función de ejemplo
- Luego vienen los RateLimits, que son la cantidad de llamadas máximas que se puede hacer a la API por segundo, por minuto, por hora, por día, mes etc.. Cada API lo maneja a su manera, pero siempre hay un límite a la cantidad de veces que puedo mandarle un request a la API
- Luego arranca la lista de funciones del índice digamos
- Ultimamente se está estilando una segunda barra lateral a la derecha con ejemplos prácticos, pero no son muchas las APIs que tienen ese formato de documentación aún

Esto es lo mas usual, pero siguen habiendo APIs con documentación vieja, por ejemplo la de ROFEX en Argentina que es la única API con datos oficiales de futuros, tiene su documentación en un word o pdf 😞

## Explicación básica del funcionamiento de una comunicación con una API

Voy a explicar muy a groso modo como funciona una API REST

Lo que tienen las APIs es una base de datos que se actualiza a veces en tiempo real a veces una vez por dia etc, y lo que hacen estos servicios de APIs es proveernos de una serie de "funciones" o "endpoints" a los cuales podemos hacer "llamados" o "requests"

Acostúmbrense a esas palabritas porque las vamos a nombrar un montón a partir de ahora

- Endpoint: Es una URL (dirección web) a la que vamos a apuntar para que me devuelva un dato
- Llamados/Requests: Es la acción de entrar a esa URL
- Response: Es la respuesta de la API a ese llamado
- StatusCode: Es un código estandarizado que nos devuelve junto con el Response (abajo los explico mejor)
- Servidor: Es la compu de la API
- Cliente: Es la compu nuestra o nuestro programa que hace el request
- Credenciales: Son las claves o tokens de autenticación
- Body: Es el cuerpo de los mensajes (contenido) entre cliente y servidor
- Headers: Son encabezados de los mensajes entre cliente y servidor (A veces aquí viajan las credenciales)
- Parametros: Son justamente los parámetros (variables) que necesita el endpoint que le mandemos en el mensaje, hay opcionales y obligatorios
- SandBox: Son ambientes de prueba, para testear las funciones sin efecto real (como un simulador)
- POST, GET: Son los principales métodos de comunicación cliente/servidor, a grandes rasgos POST es mas seguro y se usa para autenticación o para envío de info sensible ya que cuando viaja por GET podría ser interceptada más fácilmente

O sea que en definitiva una API rest es como entrar a una web y ver un dato que el servidor de la web lo tiene guardado en una base de datos, pero como se hace todo automatizado el servidor tiene que definir digamos el fino del "cómo" (parametros) navegar cada endpoint (subpagina) de la web, y para saber quien es quien para evitar abusos y medir el uso, generalmente nos pide una autenticación en los headers del request o en el body

Suena a chino? no se preocupen, no es para que se aprendan esas palabras ahora, se las tiro acá porque se que van a aparecer mas adelante y queria dejar una especie de glosario inicial para que tengan a donde recurrir cuando aparezca mucha palabrería que no sepan que significa

## Rate Limits

Bien, como ya dije, toda API tiene una rate limit, y esto es así porque imaginense que al ser todo automatizado, si me descontrolo y le mando bots a una API que se queden eternamente haciendo requests a lo loco, les saturo el servicio, con lo cual el servidor nos va a bloquear al pasar determinado rateLimit, ya sea por segundo, minuto, hora, dia etc..

Este parámetro es de lo primero que debemos fijarnos, por lo general alcanza y sobra con los que nos deja, es muy común que sean al menos de 1 request por segundo, y alguna cantidad fija por mes también en los modelos freemium, así que siempre chequen este dato a la hora de evaluar la conveniencia del uso de una API

Weightening: Hay muchas APIs que miden el "peso" de las respuestas, y para las respuestas largas o pesadas (por ejemplo una serie de precios histórica muy larga) les pone un peso mayor, otras APIs no le dan bola a eso y tienen un costo de peso flat digamos para medir la cantidad de requests

## Status Code

Si bien ya vamos a ver mas adelante que pasa cuando falla una conexión y esas cosas, voy dejando por acá que es la parte de generalidades una tabla de códigos de respuesta Http, que nos devuelven todas las APIs a cada llamada salga bien o mal justamente diciendo eso (si salio bien o mal) y un mínimo estandar que nos orienta de donde está la falla en caso de haberla

- 1xx: Mensaje informativo.
- 2xx: Exito
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
- 3xx: Redirección
  - 300 Multiple Choice
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- 4xx: Error del cliente
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- 5xx: Error del servidor
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable

## Algunas aclaraciones de los códigos de error

El 404 creo que lo conocen, es el tipico error cuando buscamos una wew (endpoint) que no existe, como ven está catalogado como "error de cliente" es decir error nuestro, ya que somos nosotros los que le pedimos una dirección que no existe, cuando es 5xx son siempre errores de servidor,

fallas que cuando vemos que emplezan con 5 el status code, olvidate, no podés hacer nada vos.  
El status code que devuelve cuando está todo ok es el 200 y en el caso de APIs los del 3xx no  
los vamos a ver nunca

Es importante al menos entender esta diferencia entre la serie 400s y 500s de errores, mas allá  
de las APIs en general en el mundo web, la arquitectura cliente-servidor hace referencia a las  
computadoras "en tu casa" – "en la casa de la pagina que visitas"

Es decir "cliente" es la compu de tu casa

"Servidor" es la compu de la página web que estás visitando

Una comunicación http, que es lo que tenemos en las APIs o cuando visitamos una web, es una  
comunicación entre esas dos computadoras, cuando hay un error de comunicación siempre hay  
una de las dos que lo origina (a menos que sea un error de redireccionamiento las series 300s)

Entonces vamos a lo importante, cuando nos devuelve un error de la serie 400s, ponele 404, 403,  
401, etc, es porque algo le pifiamos nosotros, el servidor esta respondiendo bien pero nos dice  
"man le pifiaste con lo que me pedis"

En cambio cuando recibimos un error del tipo 500s, es porque el servidor se está colgando, no  
es culpa nuestra, no revisen ahí el código porque cuando aparecen ese tipo de errores es culpa  
de la compu del otro lado.

Ojo que a veces nos devuelve 200 y sale error igual, que nos devuelva status code 200, no  
significa que haya entendido la API lo que le pedimos o haya aceptado los parámetros sino que  
significa que hubo comunicación entre la API y nosotros, es decir que hubo una respuesta  
asignada a nuestro llamado, pero a veces vamos a ver que esa respuesta no siempre tiene los  
datos que esperábamos

Bueno, voy por cerrada esta breve intro así ya empezamos a ver una por una y a navegar por  
sus funciones, vamos a ir dentro de cada grupo de menor a mayor en cuanto a nivel de dificultad  
o complejidad de uso y de los instrumentos

# API AlphaVantage

Vamos a arrancar con la famosa API de alphaVantage, es una de las pioneras, si no me equivoco con la primera versión de IEX y con las discontinuadas GoogleFinance y YahooFinance fueron las pioneras en este mundo de las APIs financieras

Como les dije en la intro, todas las APIs del mundo del equity requieren autenticación, así que tienen que ir a [www.alphavantage.co](http://www.alphavantage.co) y pedir su "api\_key"

En este caso como son keys gratuitas les voy a empezar a mostrar ejemplos con una "api\_key" que a partir de ahora llamaremos "token", que ya saqué yo con un mail de prueba para estos libros

Obviamente el primer paso va a ser definir la variable token y asignarle el string que me da la API y ya lo dejo en la primera línea del código y me olvido del tema

```
token = '2RG2NEF3IPXMIPX3'
```

The screenshot shows the AlphaVantage API documentation for the `TIME_SERIES_INTRADAY` function. At the top right, there is a **High Usage** badge. Below it, a brief description states: "This API returns intraday time series (timestamp, open, high, low, close, volume) of the equity specified." The main content is organized into sections: **API Parameters**, **Required:** `function` (The time series of your choice. In this case, `function=TIME_SERIES_INTRADAY`), **Required:** `symbol` (The name of the equity of your choice. For example, `symbol=IBM`), **Required:** `interval` (Time interval between two consecutive data points in the time series. The following values are supported: `1min`, `5min`, `15min`, `30min`, `60min`), **Optional:** `outputsize` (By default, `outputsize=compact`. Strings `compact` and `full` are accepted with the following specifications: `compact` returns only the latest 100 data points in the intraday time series; `full` returns the full-length intraday time series. The "compact" option is recommended if you would like to reduce the data size of each API call.), **Optional:** `datatype` (By default, `datatype=json`. Strings `json` and `csv` are accepted with the following specifications: `json` returns the intraday time series in JSON format; `csv` returns the time series as a CSV (comma separated value) file.), and **Required:** `apikey` (Your API key. Claim your free API key here.).

Bueno, como verán esta API es super sencilla, por eso quise empezar por esta, vamos a arrancar directamente por la primera función

Como verán es la serie de precios intradiarios, y me da una serie de parámetros obligatorios:

- `function`: El nombre de la función (en esta API es obligatorio siempre este parámetro)
- `symbol`: El ticker por ejemplo "AAPL"
- `interval`: El intervalo entre cada vela (1min, 5min, 15min, 30min 60min)
- `apikey`: Es nuestra clave, o token como definimos antes

Y una serie de parámetros optativos:

- **outputsize**: Compacto o Full (aclara que por default compacto devuelve solo 100 datos)
- **datatype**: Nos da la posibilidad de descargarlo en un CSV (tipo excel) o devolver un JSON (texto que leerá python), por default es JSON y es lo que siempre vamos a usar, a menos que sea la API para descargarse excels, pero no tendría mucho sentido

Bueno, antes de codear esto, vamos a aclarar una cosa, como dijimos un request de una API REST es ni mas ni menos como visitar una pagina web, asi que estos parámetros los podríamos poner en la barra del navegador web y tendría que funcionar, veamos un ejemplo, vamos a usar esta API para saber los precios intradiarios de AAPL en velas de 15 minutos

LA URL base es: <https://www.alphavantage.co/query>

Luego le debemos poner los parámetros, para ello:

Al primer parámetro lo antecedemos por el signo "?"

Y para concatenar los otros usamos el signo "&"

Entonces la URL final quedaría así:

[https://www.alphavantage.co/query?function=TIME\\_SERIES\\_INTRADAY&symbol=AAPL&interval=15min&apikey=2RG2NEF3IPXMIP](https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol=AAPL&interval=15min&apikey=2RG2NEF3IPXMIP)

Y si ponemos eso en la barra del navegador (Chrome, Firefox etc..) vamos a ver algo así:

```
[{"Meta Data": {  
    "1. Information": "Intraday (15min) open, high, low, close prices and volume",  
    "2. Symbol": "AAPL",  
    "3. Last Refreshed": "2020-05-01 16:00:00",  
    "4. Interval": "15min",  
    "5. Output Size": "Compact",  
    "6. Time Zone": "US/Eastern"  
},  
"Time Series (15min)": {  
    "2020-05-01 16:00:00": {  
        "1. open": "289.5100",  
        "2. high": "290.2200",  

```

Que obviamente esa respuesta es el JSON que esperamos, pero calmamente nosotros no queremos entrar a una web y verlo, sino que queremos que python lo haga por nosotros y guarde en memoria los datos esos para procesarlo eventualmente luego

Así que veamos como sería eso, vamos a empezar por la manera mas burda de hacerlo y después lo emprolijamos

Vamos a tener importar tres librerías:

- requests: Para hacer el llamado HTTP a la API
- json: Para trabajar con la respuesta que nos da
- pandas: Para guardar los datos en un DataFrame

Obviamente solo necesito la librería requests para comunicarme con la API el resto es para trabajar con los datos que me devuelve

```
#Importamos Las Librerías
import pandas as pd
import requests
import json

#Definimos Las variables (parámetros)
function = 'TIME_SERIES_INTRADAY'
symbol = 'AAPL'
interval = '15min'
token = "2RG2NEF3IPXMXIPX3"

# Concateno La URL a visitar:
urlBase = 'https://www.alphavantage.co/query'
url = urlBase+'?function='+function url +=
'&symbol='+str(symbol)
url += '&interval='+interval
url += '&outputszie=compact'
url += '&apikey=' + token

# Acá hago el Llamado/Request
r = requests.get(url)

print(r)
```

```
<Response [200]>
```

Bien, pero lo que vemos ahí cuando imprimimos el response, ¿que es?

Es un objeto response de la librería requests, que obviamente tiene varios elementos y funciones:

- text o content: Es el texto plano como lo vemos en la web
- headers: Los encabezados
- status\_code: Como vimos el código que ya vimos que es 200, es decir que la comunicación esta ok
- url: la url
- cookies: Las cookies
- json(): El método para obtener el objeto json, o diccionario del contenido

Veamos algún ejemplo:

```
print(r.url, '\n\n', r.cookies, '\n\n', r.status_code, '\n\n', r.headers)

https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&
symbol=AAPL&interval=15min&outputsize=compact&apikey=2RG2NEF3IPX MIPX3

<RequestsCookieJar[]>

200

{'Connection': 'keep-alive', 'Server': 'gunicorn/19.7.0', 'Date': 'Sun, 03 May 2020 11:12:25 GMT', 'Transfer-Encoding': 'chunked', 'X-Frame-Options': 'SAMEORIGIN', 'Vary': 'Cookie', 'Allow': 'GET, HEAD, OPTIONS', 'Content-Type': 'application/json', 'Via': '1.1 vegur'}
```

Pero lo que nos interesa a nosotros es el JSON, entonces:

Pero si miramos bien como está el JSON imprimiendo `r.json()`, vemos que de todo el diccionario a mí la metadata no me interesa, sino que quiero la serie, así que de ese diccionario solo queremos la clave `['Time Series (15min)']`

```
{
    "Meta Data": {
        "1. Information": "Intraday (15min) open, high, low, close prices and volume",
        "2. Symbol": "AAPL",
        "3. Last Refreshed": "2020-05-01 16:00:00",
        "4. Interval": "15min",
        "5. Output Size": "Compact",
        "6. Time Zone": "US/Eastern"
    },
    "Time Series (15min)": {
        "2020-05-01 16:00:00": {
            "1. open": "289.5100",
            "2. high": "290.2200",
            "3. low": "288.2000",
            "4. close": "289.0900",
            "5. volume": "2769168"
        },
        "2020-05-01 15:45:00": {
            "1. open": "290.9300",
            "2. high": "291.1100"
        }
    }
}
```

así que vamos a asignar a la variable "data" `r.json()` que era la respuesta, pero su clave `['Time Series (15min)']`

Y eso lo vamos a estructurar en un DataFrame de pandas al que llamaremos `dataDF`

Y como en este caso está orientado con una clave para cada fecha y valor para cada dato, vamos a usar el método `from_dict()` de los dataFrames para indicarle que la orientación es del índice así no me pone cada fecha en una columna, lo cual sería muy engorroso de leer y trabajar luego

```
data = r.json()['Time Series (15min)']
dataDF = pd.DataFrame.from_dict(data, orient='index')
dataDF.head()
```

	1. open	2. high	3. low	4. close	5. volume
2020-05-01 16:00:00	289.5100	290.2200	288.2000	289.0900	2769168
2020-05-01 15:45:00	290.9300	291.1100	289.2300	289.4900	1458126
2020-05-01 15:30:00	291.9700	292.5600	290.5400	290.9473	1115531
2020-05-01 15:15:00	291.3600	292.3600	291.1300	291.9800	1137866
2020-05-01 15:00:00	291.2900	291.7200	290.6700	291.4100	756106

Bien, ahora emprolijemos un poco el código, en lugar de concatenar la URL entera con todos los parámetros, lo que podemos hacer, y es una buena práctica hacer, es mandar todos los parámetros como parámetros del request en lugar de mandarlos concatenando la URL

```
function = 'TIME_SERIES_INTRADAY'
symbol = 'AAPL'
interval = '15min'
size = 'compact'
token = "2RG2NEF3IPXMIPX3"

url = 'https://www.alphavantage.co/query'
parametros = {'function': function, 'symbol': symbol, 'interval': interval,
              'outputsize': size, 'apikey': token}

r = requests.get(url, params=parametros)
data = r.json()['Time Series (15min)']
dataDF = pd.DataFrame.from_dict(data, orient='index')
dataDF.head()
```

	1. open	2. high	3. low	4. close	5. volume
2020-05-01 16:00:00	289.5100	290.2200	288.2000	289.0900	2769168
2020-05-01 15:45:00	290.9300	291.1100	289.2300	289.4900	1458126
2020-05-01 15:30:00	291.9700	292.5600	290.5400	290.9473	1115531
2020-05-01 15:15:00	291.3600	292.3600	291.1300	291.9800	1137866
2020-05-01 15:00:00	291.2900	291.7200	290.6700	291.4100	756106

Y para terminar lo podemos meter en una función

```

def getIntra(function, symbol, interval, size, token):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol, 'interval': interval,
                  'outputsize': size, 'apikey': token}

    r = requests.get(url, params=parametros)
    data = r.json()['Time Series (15min)']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    return dataDF

data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL', interval='15min',
                size='compact', token="2RG2NEF3IPXMXIPX3")
data.head()

```

	1. open	2. high	3. low	4. close	5. volume
2020-05-01 16:00:00	289.5100	290.2200	288.2000	289.0900	2769168
2020-05-01 15:45:00	290.9300	291.1100	289.2300	289.4900	1458126
2020-05-01 15:30:00	291.9700	292.5600	290.5400	290.9473	1115531
2020-05-01 15:15:00	291.3600	292.3600	291.1300	291.9800	1137866
2020-05-01 15:00:00	291.2900	291.7200	290.6700	291.4100	756106

pará pará pará, momento, esto funciona porque pasé como argumento a la función un intervalo de 15min pero ¿que pasaba si le pedía un intervalo de 1 min?

```

data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL',
                interval='1min', size='compact', token="2RG2NEF3IPXMXIPX3")
data.head()

```

```

-----
-----
KeyError Traceback (most recent call last)

<ipython-input-84-d91142f4a65a> in <module>
      1 data = getIntra(function='TIME_SERIES_INTRADAY', symbol
      2                   ='AAPL', interval='1min',
      3                   size='compact', token="2RG2NEF3IPXMXIPX
      4                   3")
      5 data.head()

<ipython-input-82-1778a20a7b8b> in getIntra(function, symbol,
      1 in terval, size, token)
      2
      3
      4
      5
      6     r = requests.get(url, params=parametros)
      7 data = r.json()['Time Series (15min)']
      8 dataDF = pd.DataFrame.from_dict(data, orient='index'
      9 )
      9 return dataDF

KeyError: 'Time Series (15min)'

```

Bien, me está diciendo que no encuentra en el diccionario JSON esa clave "15min" y oobvio porque al hacer el request a la API con un intervalo de 1min, el diccionario va a tener en la serie la clave "Time Series (1min)" y no "Time Series (15min)", así que arreglemos la función para que funcione con cualquier intervalo válido

```
def getIntra(function, symbol, interval, size, token):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol, 'interval': interval,
                  'outputsize': size, 'apikey': token}

    r = requests.get(url, params=parametros)

    # En esta Línea arreglo ese tema
    data = r.json()['Time Series ('+interval+')']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    return dataDF

data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL', interval='1min',
                 size='compact', token="2RG2NEF3IPXMIWX3")
data.head()
```

	1. open	2. high	3. low	4. close	5. volume
2020-05-01 16:00:00	288.7600	290.0300	288.6500	289.0700	561655
2020-05-01 15:59:00	288.5200	289.0000	288.4000	288.7700	258854
2020-05-01 15:58:00	288.6200	288.7200	288.5200	288.5213	241296
2020-05-01 15:57:00	288.3200	288.7000	288.2900	288.6500	175712
2020-05-01 15:56:00	288.2600	288.5800	288.0200	288.2950	277477

## HardCodeando una Función

Ahora y ¿qué pasa si mando un intervalo que no es aceptado por la API?

```
data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL',
interval='12min', size='compact', token="2RG2NEF3IPXMIWX3")
data.head()
```

```
-----  
KeyError           Traceback (most recent call last)
```

```
<ipython-input-86-4daf4359aad2> in <module>
----> 2 data = getIntra(function='TIME_SERIES_INTRADAY', symbol
```

```
KeyError: 'Time Series (12min)'
```

Ok, me dice que no encuentra esa serie tampoco en el JSON pero es efectivamente el intervalo que le pasamos en la función ¿como sabemos si la API está andando? Bien acá no queda otra que usar el concepto de "HARDCODEAR" la función para encontrar el error

Esto de hardcodear es muy común y es meterse en medio de una función ya hecha (en nuestro caso getIntra) y cambiarle algo a mano a propósito para ver que hace el programa, como investigar para rastrear el error (en nuestro caso ya sabemos que el error es porque mandamos un intervalo no válido pero supongamos que no lo sabemos y pensamos que la API tenía ese intervalo, es a modo didáctico el planteo)

Bien, ahí lo que hacemos es esto:

```
def getIntra(function, symbol, interval, size, token):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol, 'interval': interval,
                  'outputsize': size, 'apikey': token}

    r = requests.get(url, params=parametros)

    #Comento La Linea que falla en adelante

    #data = r.json()['Time Series ('+interval+')']
    #dataDF = pd.DataFrame.from_dict(data, orient='index')
    #return dataDF

    #Y mando impresión en pantalla del
    response print(r)

data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL',
                interval='12min', size='compact', token="2RG2NEF3IPXMIPX3")
```

<Response [200]>

Bien, en primer lugar me dice que la API se comunicó y me dio una respuesta válida, pero sigo sin entender donde está el error, así que veamos que respuesta es

```
def getIntra(function, symbol, interval, size, token):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol, 'interval': interval,
                  'outputsize': size, 'apikey': token}

    r = requests.get(url, params=parametros)

    #Comento La Linea que falla en adelante
    #data = r.json()['Time Series ('+interval+')']
    #dataDF = pd.DataFrame.from_dict(data, orient='index')
    #return dataDF

    #Y mando impresión del atributo TEXT
    print(r.text)

data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL',
                interval='12min', size='compact', token="2RG2NEF3IPXMIPX3")

{
    "Error Message": "Invalid API call. Please retry or visit the documentation
    (https://www.alphavantage.co/documentation/) for TIME_SERIES_INTRADAY."
}
```

Bien, ahora si, me da una explicación la API de que pasa que no me devolvió la serie cada 12 minutos, es una cagada porque no me sirve de mucho la explicación que me da, me dice solo que la llamada a la API que hice es inválida pero no me aclara que es por el intervalo que le pedí ni me da mayor explicación, pero bueno, quería mostrarles como rastrear un error con esta API que es sencilla, hay APIs que cuando haces una llamada con algún parámetro mas configurado te dice que es por ese parámetro el error y te devuelve en el mismo mensaje saliente de la api los parámetros válidos

## Sacando los token o keys fuera de las funciones

Bien, ahí va tomando forma, pero podríamos pensar que si vamos a ir definiendo una función para cada consulta, y para todas vamos a tener que pasar el token, y por otro lado el token va a ser siempre el mismo, en este caso, podríamos definir la variable "token" global al principio del código y que cada función la use libremente sin tener que pasarlala como argumento siempre, este es un recurso válido, veamos como queda:

Además por razones obvias de seguridad, por lo general los tokens o keys vienen de importar archivos con estos datos para que no queden expuestos en todos los archivos o vienen de bases de datos, por ahora lo ponemos así como recurso pedagógico pero por lo general no vienen en el mismo código como les digo así nomás expuestos

```
TOKEN="2RG2NEF3IPXMPX3"

def getIntra(function, symbol, interval, size):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol, 'interval':
                  interval, 'outputsize': size, 'apikey': TOKEN}

    r = requests.get(url, params=parametros)
    data = r.json()['Time Series ('+interval+')']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    return dataDF

data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL',
                interval='15min', size='compact')
data.head()
```

	1. open	2. high	3. low	4. close	5. volume
2020-05-04 16:00:00	292.8000	293.8600	292.6200	293.1800	1878798
2020-05-04 15:45:00	292.4900	293.1500	292.4000	292.7950	1191711
2020-05-04 15:30:00	291.8700	292.7300	291.7500	292.4800	1044066
2020-05-04 15:15:00	291.5300	292.1900	291.1500	291.8590	827574
2020-05-04 15:00:00	290.0500	291.7000	289.9700	291.5500	840035

## Ojo siempre con los tipos

Un "temita" a considerar siempre son los tipos de datos cuando descargamos de una API nueva que no conocemos, porque muchas veces nos confiamos que está todo ok, y después resulta que no tanto, por ejemplo, en el caso que mostré recién todo indicaría que los datos están bien y que ya podemos trabajar con ellos, peeeeero, no es así, veamos bien

Vamos a probar "redondear" los datos a solo 2 decimales

```
def getIntra(function, symbol, interval, size):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol, 'interval': interval,
                  'outputsize': size, 'apikey': TOKEN}

    r = requests.get(url, params=parametros)
    data = r.json()['Time Series ('+interval+')']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    return dataDF

data = getIntra(function='TIME_SERIES_INTRADAY', symbol='AAPL',
                 interval='15min', size='compact')
data.round(2).head()
```

	1. open	2. high	3. low	4. close	5. volume
2020-05-01 16:00:00	289.5100	290.2200	288.2000	289.0900	2769168
2020-05-01 15:45:00	290.9300	291.1100	289.2300	289.4900	1458126
2020-05-01 15:30:00	291.9700	292.5600	290.5400	290.9473	1115531
2020-05-01 15:15:00	291.3600	292.3600	291.1300	291.9800	1137866
2020-05-01 15:00:00	291.2900	291.7200	290.6700	291.4100	756106

¿Que onda? no redondéo nada..

No insistan probando ejecutarlo de nuevo, va a salir lo mismo, el punto es que si no redondea es porque no interpreta los tipos de datos como numeros sino como strings, es re común esto en muchas APIs por eso lo muestro acá

La manera de solucionarlo rápido con Pandas es usar el método "astype()" que transforma los tipos siempre que se pueda de todo el DataFrame

```
data = data.astype('float')
data.round(2).head()
```

	1. open	2. high	3. low	4. close	5. volume
2020-05-01 16:00:00	289.51	290.22	288.20	289.09	2769168.0
2020-05-01 15:45:00	290.93	291.11	289.23	289.49	1458126.0
2020-05-01 15:30:00	291.97	292.56	290.54	290.95	1115531.0
2020-05-01 15:15:00	291.36	292.36	291.13	291.98	1137866.0
2020-05-01 15:00:00	291.29	291.72	290.67	291.41	756106.0

Así que lo vamos a meter directamente en la función

- Y otra cosita, no se si habrán observado que esta API devuelve los datos ordenados de nuevo a viejo, cuando para trabajar con este tipo de series lo mas cómodo a nuestro sentido común es trabajar de antiguo a nuevo, quizás para visualizar si, es más útil tener al principio de la tabla lo nuevo, pero como son series de tiempo el menor a mayor es de viejo a nuevo, así que se suele trabajar así, por lo tanto vamos a meter este ordenamiento también dentro de la función
- Y otra cosa más, le vamos a poner los nombres a las columnas más estandarizados
- Y una más, vamos a pasar el índice a un tipo "datetime" porque también viene como un string
- Y ahora sí la última cosita, si ven bien, le estamos pasando como parámetro de la función el parámetro del request "function" cuando es innecesario que se lo pasemos como parámetro a la función de Python porque `getIntra()` siempre va a hacer el llamado a la API con `function='TIME_SERIES_INTRADAY'` así que eso lo vamos a poner como variable dentro de la función (para poder reusar ese código luego) pero no lo vamos a pedir como parámetro, sigo me que después se va a entender joya)

```
def getIntra(symbol, interval, size):  
    function='TIME_SERIES_INTRADAY'  
    url = 'https://www.alphavantage.co/query'  
    parametros = {'function': function, 'symbol': symbol, 'interval': interval,  
                 'outputsize': size, 'apikey': TOKEN}  
  
    r = requests.get(url, params=parametros)  
    data = r.json()['Time Series ('+interval+')']  
    dataDF = pd.DataFrame.from_dict(data, orient='index')  
    dataDF = dataDF.astype('float')  
    dataDF.index.name = 'Date'  
    dataDF.columns = ['Open', 'High', 'Low', 'Close', 'Volume']  
    dataDF = dataDF.sort_values('Date', ascending=True)  
    dataDF.index = pd.to_datetime(dataDF.index)  
    return dataDF  
  
data = getIntra(symbol='AAPL', interval='15min', size='compact')  
data.round(2).head()
```

Date	Open	High	Low	Close	Volume
2020-04-28 10:45:00	283.29	284.15	282.21	282.33	958502.0
2020-04-28 11:00:00	282.36	282.50	280.64	281.10	1321274.0
2020-04-28 11:15:00	281.08	281.90	280.87	281.46	809673.0
2020-04-28 11:30:00	281.47	282.18	280.56	281.97	733579.0
2020-04-28 11:45:00	281.97	282.36	281.35	281.69	553574.0

Listo ahí ya tenemos una función que me devuelve el DataFrame tal cual como lo necesito para hacer el análisis cuantitativo que quiera con la mayor comodidad

## Mas funciones de la API AlphaVantage

Bien, pasemos a la siguiente función, los precios históricos, según la doc, solo hay que cambiar el parámetro "function" que en lugar de "TIME\_SERIES\_INTRADAY" que ahora pasa a ser "TIME\_SERIES\_DAILY", por eso empecé con esta API es demasiado sencilla usarla, a saber esta función no acepta el parámetro "interval" porque es obvio que es diario, así que vamos a copiar y pegar la función anterior pero cambiándole este parámetro "function" y volando el parámetro "interval"

Bueno, otra cosa que le tenemos que cambiar es que al json, antes de pasarlo a DataFrame, en lugar de ir a la clave del diccionario ['Time Series (XX min)'] debemos tomar ['Time Series (Daily)'], para ver esto en realidad lo que hice yo es ver que me devuelve imprimiendo solo el JSON y al mirar el JSON me di cuenta que era esa la clave que tenía que pedir para pasar la serie a un DataFrame

Veamos el código:

```
def getDaily(symbol, size):
    function='TIME_SERIES_DAILY'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function' : function, 'symbol': symbol,
                  'outputsize': size, 'apikey': TOKEN}

    r = requests.get(url, params=parametros)
    data = r.json()['Time Series (Daily)']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    dataDF = dataDF.astype('float')
    dataDF.index.name = 'Date'
    dataDF.columns = ['Open','High','Low','Close','Volume']
    dataDF = dataDF.sort_values('Date', ascending=True)
    dataDF.index = pd.to_datetime(dataDF.index)
    return dataDF

data = getDaily(symbol='AAPL', size='compact')
data.head()
```

	Open	High	Low	Close	Volume
Date					
2019-12-09	270.00	270.8000	264.910	266.92	32182645.0
2019-12-10	268.60	270.0700	265.860	268.48	22632383.0
2019-12-11	268.81	271.1000	268.500	270.77	19723391.0
2019-12-12	267.78	272.5599	267.321	271.46	34437042.0
2019-12-13	271.46	275.3000	270.930	275.15	33432806.0

## Series Ajustadas

Obviamente así como tiene una función para las series de precios corrientes, también tiene una para series de precios ajustadas, por lo general la mayoría de las APIs tiene una sola función donde nos muestra la serie de precios sin ajustar y solo el cierre ajustado, tal como vimos en los archivos esos de excel con los que trabajábamos hasta ahora

Pero veamos como es la serie de datos ajustados en esta API

Podemos usar exactamente la misma función de python "getDaily" cambiando una sola cosa, así que la vamos a llamar getDailyAdj()

- A la variable "function" le pongo "TIME\_SERIES\_DAILY\_ADJUSTED" en lugar de "TIME\_SERIES\_DAILY"
- Y en los nombres de columnas debo agregar mas nombres porque este Endpoint me devuelve mas columnas que el no ajustado

```
def getDailyAdj(symbol, size):
    function='TIME_SERIES_DAILY_ADJUSTED'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol,
                  'outputsize': size, 'apikey': TOKEN}

    r = requests.get(url, params=parametros)
    data = r.json()['Time Series (Daily)']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    dataDF = dataDF.astype('float')
    dataDF.index.name = 'Date'
    dataDF.columns=['Open','High','Low','Close','AdjClose','Volume','Div','Split']
    dataDF = dataDF.sort_values('Date', ascending=True).round(2)
    dataDF.index = pd.to_datetime(dataDF.index)
    return dataDF

data = getDailyAdj(symbol='AAPL', size='compact')
data.head()
```

	Open	High	Low	Close	AdjClose	Volume	Div	Split
Date								
2019-12-09	270.00	270.80	264.91	266.92	266.28	32182645.0	0.0	1.0
2019-12-10	268.60	270.07	265.86	268.48	267.84	22632383.0	0.0	1.0
2019-12-11	268.81	271.10	268.50	270.77	270.12	19723391.0	0.0	1.0
2019-12-12	267.78	272.56	267.32	271.46	270.81	34437042.0	0.0	1.0
2019-12-13	271.46	275.30	270.93	275.15	274.49	33432806.0	0.0	1.0

Por ejemplo, si con este request quiero ver todas las ruedas donde se aplicó split a AAPL, solo pongo:

```
data = getDailyAdj(symbol='AAPL', size='full')
data.loc[data.Split>1]
```

Date	Open	High	Low	Close	AdjClose	Volume	Div	Split
2000-06-21	50.50	56.94	50.31	55.63	3.45	4375000.0	0.0	2.0
2005-02-28	44.68	45.14	43.96	44.86	5.56	11635900.0	0.0	2.0
2014-06-09	92.70	93.88	91.75	93.70	85.09	75414997.0	0.0	7.0

O si quiero saber la cantidad de ruedas donde KO pagó dividendos en los ultimos 20 años (la API me trae valores de los ultimos 20 años)

```
data = getDailyAdj(symbol='KO', size='full')
data.loc[data.Div>0].Div.count()
```

80

Bueno, si seguimos explorando la Doc de la API vemos que tiene las siguientes funciones:

- TIME\_SERIES\_WEEKLY
- TIME\_SERIES\_WEEKLY\_ADJUSTED
- TIME\_SERIES\_MONTHLY
- TIME\_SERIES\_MONTHLY\_ADJUSTED

Obviamente se pueden hacer funciones similares a las "Daily" adaptando esa línea y la que filtra el json() del response no lo hago acá para no ser reiterativo y seguir explorando otras funciones piolas de la API

## Cotización actual

La llamada para esta función es function='GLOBAL\_QUOTE' y el parámetro obligatorio aparte del token es el "symbol"

```
def quote(symbol):
    function='GLOBAL_QUOTE'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    return r.json()

data = quote(symbol='AAPL')
data
```

```
{'Global Quote': {'01. symbol': 'AAPL',
  '02. open': '286.2500',
  '03. high': '299.0000',
  '04. low': '285.8500',
  '05. price': '289.0700',
  '06. volume': '60154175',
  '07. latest trading day': '2020-05-01',
  '08. previous close': '293.8000',
  '09. change': '-4.7300',
  '10. change percent': '-1.6099%'}}
```

Para ser un poco mas prolijos, le sacamos la clave['Global Quote'] y lo pasamos a un DataFrame:

```
def quote(symbol):
    function='GLOBAL_QUOTE'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    js= r.json()['Global Quote']
    df = pd.DataFrame.from_dict(js, orient='index')
    return df

data = quote(symbol='AAPL')
data
```

0	
01. symbol	AAPL
02. open	286.2500
03. high	299.0000
04. low	285.8500
05. price	289.0700
06. volume	60154175
07. latest trading day	2020-05-01
08. previous close	293.8000
09. change	-4.7300
10. change percent	-1.6099%

En este caso me queda un DataFrame con los campos como index y los valores en una única columna, si le podía agregar al final la función .transpose() y me quedaba transpuesta la matriz

También podría haber usado en lugar de usar from\_dict, definir por separado clavey y valores para el armado del dataframe de esta forma: pd.DataFrame(js.values(),index=js.keys())

## Cotizaciones de otros mercados

Ojo, les muestro esto porque acabo de darme cuenta que la API esta agregando data de mercados de fuera de USA pero esto es nuevo nuevo (mayo 2020) y nislquiera tienen en su documentacion por ahora cuales mercados estan disponibles ni que tickers de cada mercado, calculo que en pocos meses esta info estara disponible asi que por ahora les muestro que estan incorporando mercados externos como el nuestro

Argentina:

```
data = quote(symbol='GGAL.ARG')
data
```

	0
01. symbol	GGAL.ARG
02. open	83.0000
03. high	86.0000
04. low	79.1500
05. price	80.6500
06. volume	1649582
07. latest trading day	2020-04-30
08. previous close	82.5500
09. change	-1.9000
10. change percent	-2.3016%

Brasil:

```
data = quote(symbol='PETR3.SAO')
data
```

	0
01. symbol	PETR3.SAO
02. open	18.7300
03. high	19.1800
04. low	18.4300
05. price	18.6500
06. volume	23209200
07. latest trading day	2020-04-30
08. previous close	19.0000
09. change	-0.3500
10. change percent	-1.8421%

## Búsqueda de Activos

La función de la API "SYMBOL\_SEARCH" nos permite buscar coincidencias de un activo en el nombre o ticker, vuelvo a repetir que todo esto que les pongo está en la documentación de la API yo acá les facilito la implementación de las funciones para usar la API pero la gracia es que se vayan ponendo cancheros con este tipo de cosas y leyendo la documentación para que el día de mañana salga una nueva API o salgan funciones nuevas de estas mismas APIs y ustedes mismos sepan como aprovecharlas

Así que no se malacostumbren a tomar estas líneas como palabra final, comparén lo que digo con la doc de la API y sigan mi código y modifiquénlo a su gusto así el día de mañana saben a hacerlo por su cuenta con otras APIs

```
def search(keywords):
    function='SYMBOL_SEARCH'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'keywords':keywords, 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    js= r.json()
    return js

data = search(keywords='galicia')
data

{'bestMatches': [
    {'symbol': 'GGAL',
     'name': 'Grupo Financiero Galicia S.A.',
     'type': 'Equity',
     'region': 'United States',
     'marketOpen': '09:30',
     'marketClose': '16:00',
     'timezone': 'UTC-05',
     'currency': 'USD',
     'matchScore': '0.6364'},
    {'symbol': 'GGAL.ARG',
     .... y así sigue
}]]
```

Como hice en las anteriores, como no se la estructura del JSON De respuesta, primero escribo la función devolviendo el JSON entero, y una vez que tengo eso, me fijo que es lo que me sirve y de ahí recién lo paso a una salida más copada tipo DataFrame.

Y como ven, es un diccionario que me devuelve la data en la clave ['bestMatches']

O sea que voy a usar a continuación esa clave para extraer del json y sacar un DataFrame luego usando js['bestMatches'] como datos del DataFrame

Como esta API no tiene una doc muy completa, no nos queda otra que probar como viene organizada la info, esto lo podemos hacer como mostré recién en una función, o con los ejemplos que me sugiere en la misma API y veo en el navegador directamente

Veamos el ejemplo buscando "Galicia"

```
def search(keywords):
    function='SYMBOL_SEARCH'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'keywords':keywords, 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    js= r.json()['bestMatches']
    df = pd.DataFrame(js)
    ret = df[['1. symbol', '2. name','3. type','4.region']]
    return ret

data = search(keywords='galicia')
data
```

	1. symbol	2. name	3. type	4. region
0	GGAL	Grupo Financiero Galicia S.A.	Equity	United States
1	GGAL.ARG	Grupo Financiero Galicia S.A.	Equity	Argentina
2	GGALD.ARG	GPO FIN GALICIA	Equity	Argentina
3	BYC4O.ARG	BCO DE GALICIA BUE FRN 05/2020	Equity	Argentina
4	GFG.ARG	GPO FIN GALICIA	Equity	Argentina
5	S2841.MDR	ACCIONES TITULOS GALICIA.S.A.SI	Mutual Fund	Madrid
6	S3117.MDR	REALES DE GALICIA, SICAV, S.A.	Mutual Fund	Madrid
7	GF4.FRK	Grupo Financiero Galicia S.A.	Equity	Frankfurt
8	GGALN.MEX	Grupo Financiero Galicia S.A.	Equity	Mexico
9	GF4.BER	GRUPO FIN. GALICIA B ADR	Equity	Berlin

Finalmente logramos el objetivo entonces de meter en un DataFrame toda la clave del json devuelto

# FOREX

Veamos alguna de las funciones de Forex (FX) de ALphaVantage

Empezando obviamente por el precio en tiempo real del par de Forex

Nos ofrece una función 'CURRENCY\_EXCHANGE\_RATE' Que nos devuelve el precio actual,, vemos como armar el request en una función a la que llamaremos fx() y le pasaremos como argumentos los dos tickers de monedas, en las variables frmC y toC

Veamos cómo queda el código:

```
def fx(fromC, toC):
    function='CURRENCY_EXCHANGE_RATE'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function,
                  'from_currency':fromC, 'to_currency':toC,
                  'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    return r.json()

data = fx('USD', 'ARS')
print(json.dumps(data, indent=3))

{
    "Realtime Currency Exchange Rate": {
        "1. From_Currency Code": "USD",
        "2. From_Currency Name": "United States Dollar",
        "3. To_Currency Code": "ARS",
        "4. To_Currency Name": "Argentine Peso",
        "5. Exchange Rate": "66.61000000",
        "6. Last Refreshed": "2020-05-04 08:27:32",
        "7. Time Zone": "UTC",
        "8. Bid Price": "66.61000000",
        "9. Ask Price": "66.63000000"
    }
}
```

También la API me permite obtener el intradiario del par con la función FX\_INTRADAY

```
def fxIntra(fromC, toC, interval):
    function='FX_INTRADAY'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'from_symbol':fromC,
                  'to_symbol':toC, 'interval':interval, 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    return r.json()

data = fxIntra('USD', 'ARS', '5min')
print(json.dumps(data, indent=3))
```

Nos devuelve esto:

```
{  
    "Meta Data": {  
        "1. Information": "FX Intraday (5min) Time Series",  
        "2. From Symbol": "USD",  
        "3. To Symbol": "ARS",  
        "4. Last Refreshed": "2020-05-04 08:00:00",  
        "5. Interval": "5min",  
        "6. Output Size": "Compact",  
        "7. Time Zone": "UTC"  
    },  
    "Time Series FX (5min)": {  
        "2020-05-04 08:00:00": {  
            "1. open": "66.6100",  
            "2. high": "66.6100",  
            "3. low": "66.6100",  
            "4. close": "66.6100"  
        },  
        "2020-05-04 07:55:00": {  
            "1. open": "66.6100",  
            ....  
            ....  
        }  
    }  
}
```

Y así sucesivamente las distintas fechas

Con lo cual tengo que filtrar la clave ['Time Series FX (5min)'] del diccionario para poder meterlo en un DataFrame, y ya que estamos vamos a hacerlo que dependa del intervalo que le pasamos como parámetro como lo hicimos con las series de equity

```
def fxIntra(fromC, toC, interval):  
    function='FX_INTRADAY'  
    url = 'https://www.alphavantage.co/query'  
    parametros = {'function':function, 'from_symbol':fromC,  
                 'to_symbol':toC, 'interval':interval,'apikey':TOKEN}  
  
    r = requests.get(url, params=parametros)  
    js = r.json()['Time Series FX ('+ interval +')']  
    df = pd.DataFrame.from_dict(js, orient='index')  
    df = df.astype('float')  
    df.index.name = 'Date'  
    df.columns = ['Open', 'High', 'Low', 'Close']  
    df = df.sort_values('Date', ascending=True).round(2)  
    df.index = pd.to_datetime(df.index)  
    return df  
  
data = fxIntra('USD','EUR','1min')  
data.head()
```

Date	Open	High	Low	Close
2020-05-04 07:02:00	0.91	0.91	0.91	0.91
2020-05-04 07:03:00	0.91	0.91	0.91	0.91
2020-05-04 07:04:00	0.91	0.91	0.91	0.91
2020-05-04 07:05:00	0.91	0.91	0.91	0.91
2020-05-04 07:06:00	0.91	0.91	0.91	0.91

De modo muy similar a las funciones de equity historicos tenemos para Forex

```
def fxDaily(fromC, toC, size):
    function='FX_DAILY'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'from_symbol':fromC, 'to_symbol':toC,
                  'interval':interval,'outputsize':size , 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    js = r.json()['Time Series FX (Daily)']
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df.columns = ['Open', 'High', 'Low', 'Close']
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

data = fxDaily('USD', 'ARS', 'full')
data.tail()
```

	Open	High	Low	Close
Date				
2020-04-28	66.45	66.63	66.38	66.49
2020-04-29	66.49	66.74	66.49	66.67
2020-04-30	66.67	66.83	66.61	66.77
2020-05-01	66.77	66.77	66.02	66.61
2020-05-04	66.61	66.61	66.61	66.61

Y no se los muestra pero es exactamente igual para monthly y weekly con las funciones:

- FX\_WEEKLY
- FX\_MONTHLY

Lo único que hay que cambiar de la función anterior para obtener el sampleo en semanal o mensual es la variable function que en lugar de ser 'FX\_DAILY' será 'FX\_WEEKLY' y también tendríamos que cambiar la clave del JSON, esta línea:

```
js = r.json()['Time Series FX (Daily)']
```

por esta otra:

```
js = r.json()['Time Series FX (Weekly)']
```

# Crypto

Como vamos a ver funciones específicas de cryptos en las APIs especializadas en cryptos no me voy a detener en estas funciones pero al menos les muestro una de AlphaVantage que vale la pena como proxy a "fundamentals" es una especie de rating que hace la API en función de varios parámetros para medir que tanto potencial de uso puede tener cada crypto

```
def cryptoRank(symbol):
    function='CRYPTO_RATING'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    return r.json()

data = cryptoRank('BTC')
print(json.dumps(data, indent=3))

{

    "Crypto Rating (FCAS)": {
        "1. symbol": "BTC",
        "2. name": "Bitcoin",
        "3. fcas rating": "Attractive",
        "4. fcas score": "887",
        "5. developer score": "843",
        "6. market maturity score": "839",
        "7. utility score": "951",
        "8. last refreshed": "2020-05-04 00:00:00",
        "9. timezone": "UTC"
    }
}
```

```
data = cryptoRank('LAMB')
print(json.dumps(data, indent=3))

{

    "Crypto Rating (FCAS)": {
        "1. symbol": "LAMB",
        "2. name": "Lambda",
        "3. fcas rating": "Caution",
        "4. fcas score": "501",
        "5. developer score": "590",
        "6. market maturity score": "775",
        "7. utility score": "337",
        "8. last refreshed": "2020-05-04 00:00:00",
        "9. timezone": "UTC"
    }
}
```

La verdad no es gran cosa el análisis que hace la API pero es un proxy a tres aspectos básicos como la comunidad de programadores o desarrolladores trabajando en el proyecto, el uso o utilidad que tiene o que le dan los usuarios y la madurez de su mercado en cuanto a liquidez y demás, para mayor info de este índice pueden ver las especificaciones de la consultora que lo elabora: <https://flipsidecrypto.com/fcas/>

# Indicadores Técnicos

Bueno, llega la parte mas interesante que tiene esta API en particular que son los indicadores técnicos, aca les explico brevemente que significan cada uno de los aproximadamente 50 indicadores técnicos que la API calcula y devuelve en formato serie temporal:

Medias Móviles:

- SMA, EMA, WMA (Medias móviles simple, exponencial y ponderada)
- DEMA, TEMA (Media móvil doble exponencial y triple)
- TRIMA (Media exponencial triangular)
- KAMA (Media móvil adaptativa: Kaufman)
- MAMA (MESA adaptive moving average, otra media móvil adaptativa)

Indicadores Super conocidos:

- VWAP (Precio ponderado por volumen)
- MACD (El conocidísimo oscilador ideal para ver divergencias)
- STOCH, STOCHF (El conocido oscilador estocástico)
- RSI (El conocidísimo oscilador de Fuerza relativa ideal para divergencias)
- ADX (Indicador de tendencia, conocidísimo como Average Directional Movement Index)
- DX (Es el ADX pero sin suaviza)
- CCI (Commodity channel index, Indicador de inicio y fin de tendencias, zonas de sobrecompra/sobreventa o divergencias)
- AROON (Anticipador de inicios de tendencias compuesto por dos curvas)
- BBANDS (Famosas bandas de bollinger, bandas de rango de movimiento a +/- 2 sigma de media móvil)
- SAR (famoso Parabolic SAR, se usa como Trailing StopLoss para cerrar posiciones) ATR (Average true range, es la media móvil del True Range, es decir la media móvil del rango de movimiento, proxy de la volatilidad pero en términos absolutos)
- OBV (On Balance Volume, acumulador del volumen, cuando cierra Up se suma vol, y cuando cierra abajo se resta)
- AD (Chaikin Acumulación/Distribución, una mejora del OBV con una ponderación de la forma de la vela)

## Otros indicadores menos conocidos

- T3 (Una media móvil basada en la doble exponencial)
- MACDEXT (Es el MACD y se puede usar otros tipos de medias móviles que la EMA)
- STOCHRSI (Oscilador estocástico del oscilador RSI)
- WILLR (Oscilador de Williams, similar al estocástico)
- ADXR (Promedio del ADX contra el ADX de "n" ruedas atrás)
- APO (Absolute price oscillator, Es una diferencia de medias móviles, simples, exponenciales, etc)
- PPO (Percentage price oscillator, Es la diferencia porcentual de las medias móviles)
- MOM (El famoso "Momentum", es la diferencia entre el precio y el precio "n" ruedas atrás)
- BOP (Balance of power) dado el OHLC es la media móvil de  $(C-O)/(H-L)$ , se supone que cuando pasa de negativo a positivo marca señal de compra y viceversa
- CMO (Chande momentum oscillator, es una variante del oscilador RSI)
- ROC (Es como el Momentum pero %, mide variación % respecto a "n" ruedas atrás)
- AROONOSC (Es la diferencia entre las curvas del AROON)
- MFI (Money flow index, es un oscilador que se interpreta como el RSI y que tiene en cuenta el volumen)
- TRIX (Es un oscilador de la TEMA respecto de "n" ruedas atrás)
- ULTOSC (Ultimate oscillator, es un oscilador de 3 períodos, como promediar 3 RSIs con 3 "n" diferentes)
- MINUS\_DI, PLUS\_DI, MINUS\_DM, PLUS\_DM (Componentes del ADX)
- MIDPOINT (El promedio entre el Máximo y Mínimo de una serie de "n" velas puede ser de el "H" el "O", "L" o "C")
- MIDPRICE (El promedio entre el Máximo "High" y Mínimo "Low" de la serie de "n" velas)
- TRANGE (True range, es un estimador del rango normal de trading o movimiento)
- ADOSC (Oscilador del Chaikin AD)
- HT TRENDLINE, HT SINE, HT TRENDMODE, HT DCPERIOD, HT DCOPHASE, HT PHASOR, son todos indicadores de la "descomposición" de los precios en la transformada de Hilbert, es un método usado para el procesamiento de señales

Ante cualquier duda de estos indicadores técnicos, se puede ver la fórmula u operatoria iterativa de cálculo en el siguiente link de indicadores técnicos:

<https://www.fmlabs.com/reference/default.htm>

Obviamente no les voy a mostrar la implementación en funciones de python de todos, porque de ese modo el libro sería un ladrillo infumable, y además no hace falta porque una vez que entienden la lógica de como implementar una función el resto son todas similares

## Medias Móviles

Bueno, para no mostrar 10 filas en cada ejemplo uso esta configuración así solo imprime las dos primeras, las dos últimas y el resumen de la cantidad de filas

```
pd.options.display.max_rows= 4
```

Empecemos con las típicas medias móviles. Puedo hacer una función para cada tipo de media móvil, de la siguiente forma por ejemplo para la SMA:

```
def SMA(symbol, interval, series_type, time_period):
    function='SMA'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval':interval,
                  'series_type':series_type,'time_period':time_period , 'apikey':TOKEN}

    r = requests.get(url, params=parametros) js
    = r.json()['Technical Analysis: SMA']
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

data = SMA('AAPL','5min','close', 50)
data
```

SMA	
Date	
2020-04-13 13:40:00	267.96
2020-04-13 13:45:00	267.97
...	...
2020-05-01 15:55:00	290.94
2020-05-01 16:00:00	290.84

1121 rows × 1 columns

De esa forma obtuve la media móvil de 50 velas de los precios de cierre, en un timeframe de 5min de AAPL

Ahora bien, si quisiera calcular en lugar de la SMA la EMA, si veo la doc de la API son exactamente los mismo parámetros que la SMA, solo cambia el parámetro "function" que en lugar de ser "SMA" es "EMA" por lo tanto andar haciendo funciones diferentes para cada tipo de media móvil, lo mejor es definir una sola función que tome el tipo de media móvil como parámetro y listo, sería algo así:

```
def movingAv(symbol, interval, series_type, time_period, mov_type):
    function=mov_type
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval':interval,
                  'series_type':series_type,'time_period':time_period , 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    js = r.json()['Technical Analysis: '+mov_type]
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

data = movingAv('AAPL','5min','close', 50, 'SMA')
data
```

#### SMA

##### Date

2020-04-13 13:40:00 267.96

2020-04-13 13:45:00 267.97

... ...

2020-05-01 15:55:00 290.94

2020-05-01 16:00:00 290.84

Ahora con esta función si quiero la Triple Exponencial solo cambio ese argumento:

```
data = movingAv('AAPL','daily','close', 50, 'TEMA')
data
```

#### TEMA

##### Date

2000-11-28 14.88

2000-11-29 14.79

... ...

2020-04-30 269.27

2020-05-01 271.18

# MACD

Como ya les dije no me voy a poner a mostrarles una función para cada uno de los 50 indicadores técnicos, de hecho este es el último que voy a mostrar, uso este porque es super conocido y porque tiene varios parámetros

```
def MACD(symbol, interval, series_type='close', fastperiod=12,
slowperiod=26, signalperiod=9):
    function='MACD'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval':interval,
                  'series_type':series_type, 'fastperiod': fastperiod,
                  'slowperiod':slowperiod, 'signalperiod':signalperiod,
                  'apikey':TOKEN }

    r = requests.get(url, params=parametros)
    js = r.json()['Technical Analysis: MACD']
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

data = MACD('AAPL','5min')
data
```

MACD\_Hist MACD MACD\_Signal

Date

2020-04-13 12:20:00	0.25	0.17	-0.08
2020-04-13 12:25:00	0.21	0.19	-0.03
...	...	...	...
2020-05-01 15:55:00	-0.29	-0.44	-0.15
2020-05-01 16:00:00	-0.28	-0.50	-0.22

1137 rows × 3 columns

# Ejercicios

1- Implementar una función que usando la dataFeed de AlphaVantage, me devuelva una serie con el valor del ADX de simbolo que le pase a la función, que también debe aceptar como argumentos el intervalo (1min, 5min daily etc) y la cantidad de velas, siendo el valor de default si no se le pasa ninguna a la función de 50 velas

2- Implementar una función que devuelva cualquiera de los tres siguientes indicadores técnicos:

- ADX
- AROON
- CCI

La función debe recibir el nombre de dicho indicador como parámetro ademas de los siguientes parámetros:

- Símbolo
- Intervalo (1min, 5min, daily etc)
- Cantidad de velas

¿Para que otros indicadores se podría usar la misma función?

3- Esta función de la API no se las mostré pero quiero creer que con lo visto hasta acá son capaces de poder resolver este ejercicio:

Armar una función que devuelva la performance de los sectores del SP500 del día corriente e tiempo real. El endpoint de la performance de los sectores está en la doc de la API al final de todo

4- Basados en el ejercicio anterior, implementar una función que tome como argumento el plazo como un string y que soporte los siguientes valores:

- RT: Real Time
- 1D: 1 día
- 5D: 5 días
- 1M: 1 mes
- 3M: 3 meses
- YTD: El año corriente
- 1Y: 1 Año
- 3Y: 3 Años
- 5Y: 5 Años
- 10Y: 10 Años

5- Basado en el ejercicio Anterior Armar un gráfico de barras donde pueda comparar los rendimientos de los diferentes sectores este año corriente vs los últimos 10 Y

6- Armar un script para comparar "fundamentals" de las cryptos según la clasificación de la API alphaVantage, donde tenga como input una lista de 5 cryptos (no se puede acer con mas porque el token gratuito de AlphaVantage permite hasta 5 requests por minuto) y que de reporte me muestre un gráfica de barras tipo stack con los tres componentes de la clasificación

7- Graficar el Dolar CCL implícito para la acción de YPF, sabiendo que cada ADR en USA equivale a 1 acción local usando la API de AlphaVantage

8- Usando la API de AlphaVantage, calcular la Brecha cambiaria CCL vs Dolar Oficial, imprimir la tabla y graficar dicha brecha

9- Armar una script que me devuelva una tabla, para el ticker "symbol" que contenga el precio histórico de cierre y los indicadores técnicos MACD y RSI, realizar todo usando la API de AlphaVantage

10- Con la tabla del ejercicio anterior armar un grafico que muestre la serie de precios y el MACD en un subplot y el RSI en el otro y que marque con líneas verdes y rojas las supuestas entradas y salidas del MACD (considerando entrada cuando su histograma pasa de negativo a positivo y salida cuando pasa lo contrario, es solo a modo didáctico)

# Respuestas

```
#-----#
# Rta Ejercicio 1 #
#-----#

def ADX(symbol, interval, time_period=50):
    function='ADX'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval':interval,
                  'time_period':time_period, 'apikey':TOKEN }

    r = requests.get(url, params=parametros) js
    = r.json()['Technical Analysis: ADX']
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

ADX('AAPL','daily').head()
```

```
ADX
Date
-----
2000-10-24 34.51
2000-10-25 34.66
2000-10-26 34.83
2000-10-27 34.98
2000-10-30 35.11
```

```
#-----#
# Rta Ejercicio 2 #
#-----#

def indicadoresAT(function, symbol, interval, period=50):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval':interval,
                  'time_period':period, 'apikey':TOKEN }

    r = requests.get(url, params=parametros)
    js = r.json()['Technical Analysis: '+function]
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

data = indicadoresAT(function='CCI',symbol='AAPL',interval='daily', period=20)
```

Output de la respuesta anterior:

CCI	
Date	
2000-05-31	-100.04
2000-06-01	-88.97
...	...
2020-05-01	107.42
2020-05-04	105.81

```
#-----#
# Rta Ejercicio 3 #
#-----#
pd.options.display.max_rows=20
def sectores():
    function='SECTOR'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'apikey':TOKEN }
    r = requests.get(url, params=parametros)
    js = r.json()['Rank A: Real-Time Performance']
    df = pd.DataFrame.from_dict(js, orient='index')
    df.columns = ['Performance']
    return df

data = sectores()
data
```

Performance	
Energy	3.71%
Information Technology	1.42%
Utilities	0.75%
Consumer Discretionary	0.69%
Materials	0.40%
Communication Services	0.32%
Health Care	0.07%
Consumer Staples	0.00%
Real Estate	-0.02%
Financials	-0.92%
Industrials	-1.33%

```

#-----#
# Rta Ejercicio 4 #
#-----#


claves = {'RT':'Rank A: Real-Time Performance',
          '1D': 'Rank B: 1 Day Performance',
          '5D': 'Rank C: 5 Day Performance',
          '1M': 'Rank D: 1 Month Performance',
          '3M': 'Rank E: 3 Month Performance',
          'YTD': 'Rank F: Year-to-Date (YTD) Performance',
          '1Y': 'Rank G: 1 Year Performance',
          '3Y': 'Rank H: 3 Year Performance',
          '5Y': 'Rank I: 5 Year Performance',
          '10Y': 'Rank J: 10 Year Performance'}


def sectores(plazo):
    clave = claves[plazo]
    function='SECTOR'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'apikey':TOKEN }
    r = requests.get(url, params=parametros) js = r.json()[clave]
    df = pd.DataFrame.from_dict(js, orient='index')
    df.columns = ['Performance']
    return df

data = sectores('10Y')
data

```

Performance	
<b>Information Technology</b>	307.40%
<b>Consumer Discretionary</b>	227.96%
<b>Health Care</b>	216.17%
<b>Consumer Staples</b>	107.07%
<b>Utilities</b>	82.66%
<b>Industrials</b>	78.79%
<b>Financials</b>	63.60%
<b>Communication Services</b>	56.33%
<b>Materials</b>	56.14%
<b>Energy</b>	-37.75%

```

#-----#
# Rta Ejercicio 5 #
#-----#

claves = {'RT':'Rank A: Real-Time Performance',
          '1D': 'Rank B: 1 Day Performance',
          '5D': 'Rank C: 5 Day Performance',
          '1M': 'Rank D: 1 Month Performance',
          '3M': 'Rank E: 3 Month Performance',
          'YTD': 'Rank F: Year-to-Date (YTD) Performance',
          '1Y': 'Rank G: 1 Year Performance',
          '3Y': 'Rank H: 3 Year Performance',
          '5Y': 'Rank I: 5 Year Performance',
          '10Y': 'Rank J: 10 Year Performance'}

def sectores(plazo):
    clave = claves[plazo]
    function='SECTOR'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'apikey':TOKEN }
    r = requests.get(url, params=parametros) js = r.json()[clave]
    df = pd.DataFrame.from_dict(js, orient='index')
    df.columns = ['Performance']
    df['Performance'] = df['Performance'].str.replace('%','')
    df['Performance'] = pd.to_numeric(df['Performance'])
    return df

# Calculo las series
p10Y = sectores('10Y')
pYTD = sectores('YTD')

# Armo el gráfico
fig, ax = plt.subplots(figsize=(8,8), nrows=2)

ax[0].bar(p10Y.index, p10Y.Performance, width=0.5, label='10Y Yield/SP500 Sector')
ax[0].legend()

ax[1].bar(pYTD.index, pYTD.Performance, width=0.5, label='YTD Yield/SP500 Sector')
ax[1].legend()

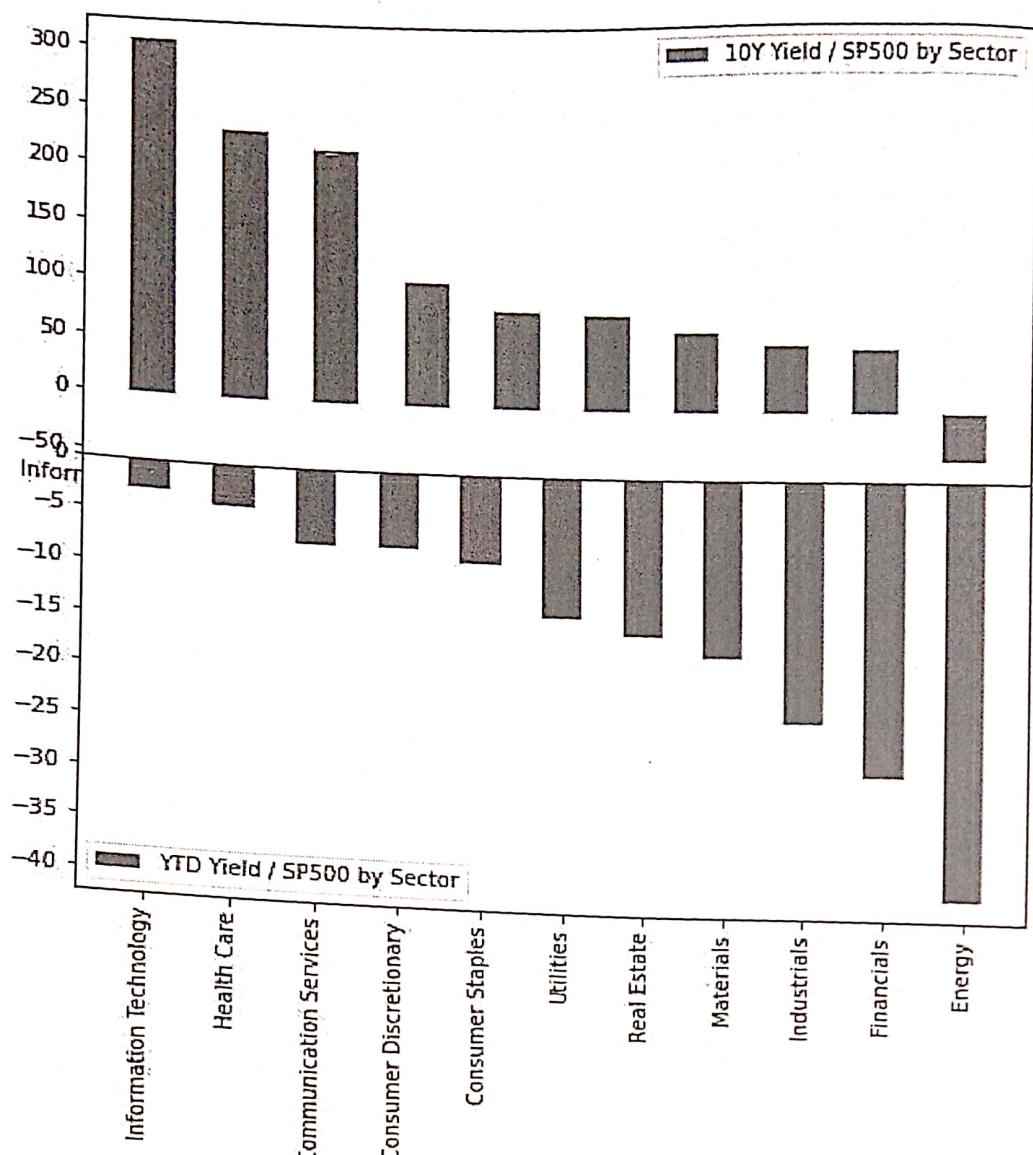
plt.subplots_adjust(hspace=0)
plt.xticks(rotation=90)

plt.show()

```

Grafico en la página siguiente

## Output de la respuesta al ejercicio 5



```

#-----#
# Rta Ejercicio 6 #
#-----#

import matplotlib.pyplot as plt
import pandas as pd

cryptos = ['BTC', 'ETH', 'XRP', 'BCH', 'BNB']

df = pd.DataFrame()
for i in range(len(cryptos)):
    data = cryptoRank(cryptos[i])['Crypto Rating (FCAS)']
    dataDF = pd.DataFrame(data, index=[i])
    df = pd.concat([df, dataDF])

df = df.drop(['3. fcas rating', '8. last refreshed', '9. timezone'], axis=1)
df.columns = ['Symbol', 'Name', 'FCAS Score', 'Developer', 'Mkt Maturity', 'Utility']

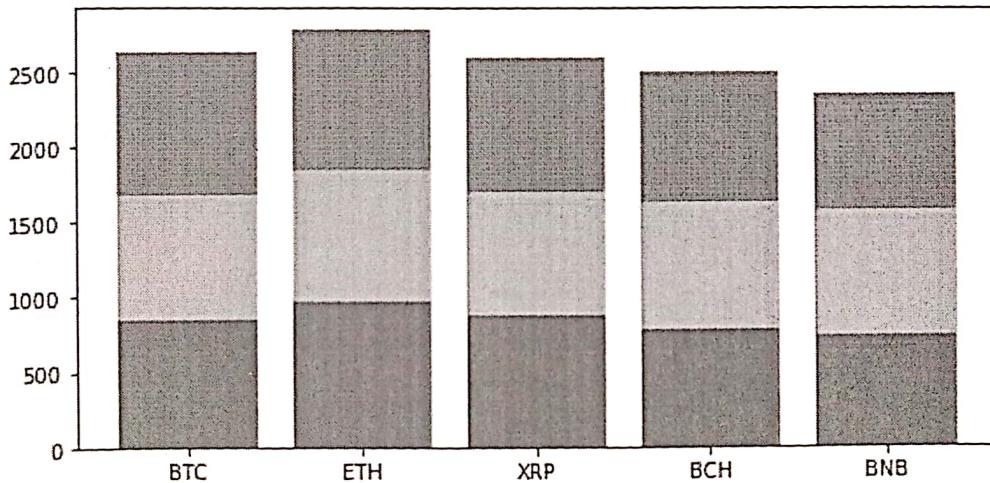
claves = ['FCAS Score', 'Developer', 'Mkt Maturity', 'Utility']

for clave in claves:
    df[clave] = df[clave].astype('float')

fig, ax = plt.subplots(figsize=(8,4))
ax.bar(df.Symbol, df['Developer'])
ax.bar(df.Symbol, df['Mkt Maturity'], bottom=df['Developer'])
ax.bar(df.Symbol, df['Utility'], bottom=(df['Mkt Maturity']+df['Developer']))

plt.show()

```



```

#-----#
# Rta Ejercicio 7 #
#-----#
```

```

import matplotlib.pyplot as plt
import pandas as pd
import requests

TOKEN = '2RG2NEF3IPXMIPX3'
def getDailyAdj(symbol, size):
    function='TIME_SERIES_DAILY_ADJUSTED'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol,
                  'outputsize': size, 'apikey': TOKEN}

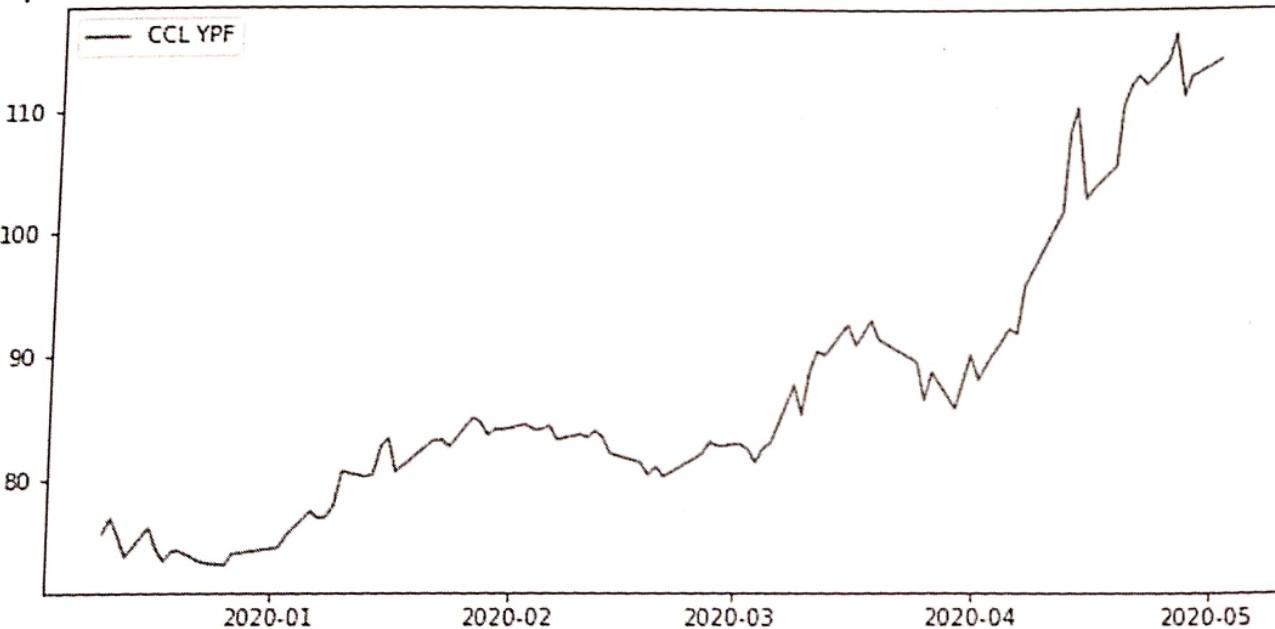
    r = requests.get(url, params=parametros)
    data = r.json()['Time Series (Daily)']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    dataDF = dataDF.astype('float')
    dataDF.index.name = 'Date'
    dataDF.columns=['Open','High','Low','Close','AdjClose','Volume','Div','Split']
    dataDF = dataDF.sort_values('Date', ascending=True).round(2)
    dataDF.index = pd.to_datetime(dataDF.index)
    return dataDF

dataUSA = getDailyAdj(symbol='YPF', size='compact')
dataLoc = getDailyAdj(symbol='YPFD.ARG',
                      size='compact') ccl = dataLoc.AdjClose/dataUSA.AdjClose
ccl = ccl.dropna().round(2)

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(ccl.index, ccl, label='CCL YPF')
plt.legend()

plt.show()

```



```

#-----#
# Rta Ejercicio 8 #
#-----#
import matplotlib.pyplot as plt
import pandas as pd
import requests

TOKEN = '2RG2NEF3IPXMIPX3'

def getDailyAdj(symbol, size):
    function='TIME_SERIES_DAILY_ADJUSTED'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function': function, 'symbol': symbol,
                  'outputsize': size, 'apikey': TOKEN}

    r = requests.get(url, params=parametros)
    data = r.json()['Time Series (Daily)']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    dataDF = dataDF.astype('float')
    dataDF.index.name = 'Date'
    dataDF.columns=['Open','High','Low','Close','AdjClose','Volume','Div','Split']
    dataDF = dataDF.sort_values('Date', ascending=True).round(2)
    dataDF.index = pd.to_datetime(dataDF.index)
    return dataDF

def fxDaily(fromC, toC, size):
    function='FX_DAILY'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'from_symbol':fromC, 'to_symbol':toC,
                  'interval':interval,'outputsize':size , 'apikey':TOKEN}

    r = requests.get(url, params=parametros)
    js = r.json()['Time Series FX (Daily)']
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df.columns = ['Open','High','Low','Close']
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

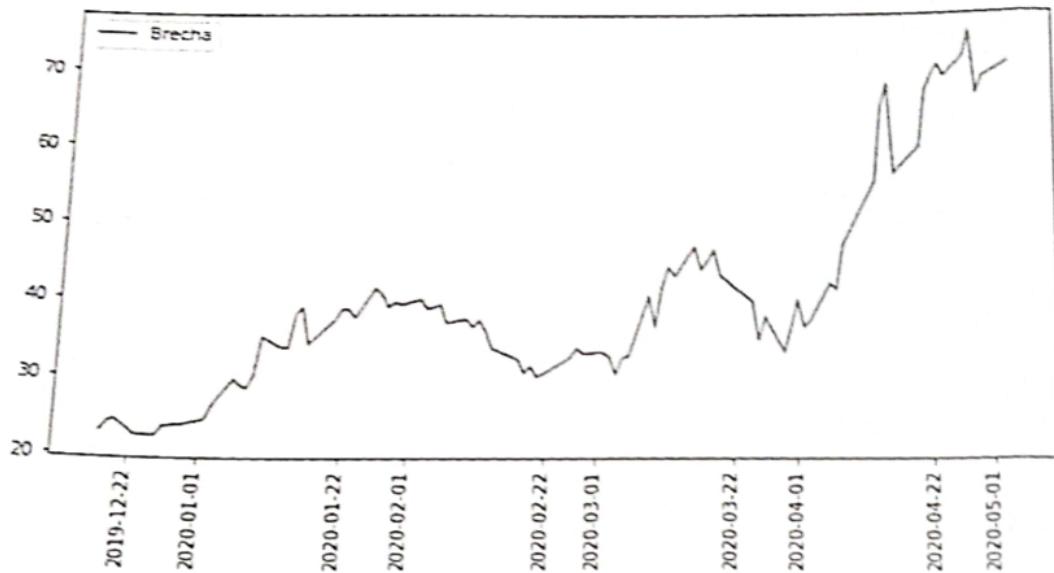
# Toma de datos de la API local, USA y FX
dataUSA = getDailyAdj(symbol='YPF', size='compact')
dataLoc = getDailyAdj(symbol='YPFD.ARG', size='compact')
oficial = fxDaily('USD','ARS','compact')

# Cálculos de los diferentes TDC
tdc = dataLoc.AdjClose/dataUSA.AdjClose
tdc = tdc.to_frame()
tdc.columns=['CCL']
tdc['Oficial'] = oficial['Close']
tdc['Brecha'] = (tdc.CCL/tdc.Oficial-1)*100
tdc = tdc.dropna().round(2)

```

= Continuación rta Ejercicio 8

```
# Grafico
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(tdc.index, tdc.Brecha, label='Brecha')
plt.xticks(rotation=90)
plt.legend()
plt.show()
print(tdc)
```



Date	CCL	Oficial	Brecha
2019-12-18	73.31	59.63	22.94
2019-12-19	74.12	59.74	24.07
2019-12-20	74.20	59.63	24.43
2019-12-23	73.23	59.80	22.46
2019-12-26	73.03	59.67	22.40
...	...	...	...
2020-04-27	114.24	66.45	71.92
2020-04-28	116.49	66.49	75.20
2020-04-29	111.24	66.67	66.86
2020-04-30	112.89	66.77	69.07
2020-05-04	114.29	66.86	70.93

[85 rows x 3 columns]

```

# Rta Ejercicio 9

def RSI(symbol, interval, series_type, time_period):
    function='RSI'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval':interval,
    'series_type':series_type,'time_period':time_period , 'apikey':TOKEN}

    r = requests.get(url, params=parametros) js
    = r.json()['Technical Analysis: RSI']
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

def MACD(symbol, interval, series_type='close', fastperiod=12,
         slowperiod=26, signalperiod=9):

    function='MACD'
    url = 'https://www.alphavantage.co/query'
    parametros = {'function':function, 'symbol':symbol, 'interval': interval,
    'series_type': series_type, 'fastperiod': fastperiod,
    'slowperiod': slowperiod, 'signalperiod': signalperiod,
    'apikey':TOKEN }

    r = requests.get(url, params=parametros)
    js = r.json()['Technical Analysis: MACD']
    df = pd.DataFrame.from_dict(js, orient='index')
    df = df.astype('float')
    df.index.name = 'Date'
    df = df.sort_values('Date', ascending=True).round(2)
    df.index = pd.to_datetime(df.index)
    return df

def getDailyAdj(symbol, size):
    url = 'https://www.alphavantage.co/query'
    parametros = {'function' : 'TIME_SERIES_DAILY_ADJUSTED', 'symbol':
    symbol, 'outputsize': size, 'apikey': TOKEN}
    r = requests.get(url, params=parametros)
    data = r.json()['Time Series (Daily)']
    dataDF = pd.DataFrame.from_dict(data, orient='index')
    dataDF = dataDF.astype('float')
    dataDF.index.name = 'Date'
    dataDF.columns = ['Open','High','Low','Close','AdjClose','Volume','Div','Split']
    dataDF = dataDF.sort_values('Date', ascending=True).round(2)
    dataDF.index = pd.to_datetime(dataDF.index)
    return dataDF

symbol = 'AAPL'
rsi = RSI(symbol,'daily','close', 14)
macd = MACD(symbol,'daily')
precios = getDailyAdj(symbol=symbol, size='full')
tabla = pd.concat([precios.Close,macd,rsi],axis=1).dropna()
tabla

```

Date	Close	MACD_Hist	MACD_Signal	MACD	RSI
2000-06-20	101.25	2.57	-5.11	-2.53	54.00
2000-06-21	55.63	-0.21	-5.16	-5.37	26.05
...	...	...	...	...	...
2020-05-01	289.07	1.94	3.91	5.84	57.56
2020-05-04	293.16	1.93	4.39	6.32	59.44

```

#-----#
# Rta Ejercicio 10 #
#-----#

data = tabla.loc[tabla.index>'2019'].copy()
data['Buy'] = (data.MACD_Hist>0)&(data.MACD_Hist.shift()<0)
data['Sell'] = (data.MACD_Hist<0)&(data.MACD_Hist.shift()>0)

import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(12,12), nrows=3)
ax[0].plot(data.index, data.Close, label='Precio Close', c='k')

ax[1].plot(data.index, data.MACD, label='MACD Fast', ls='--', color='red')
ax[1].plot(data.index, data.MACD_Signal, label='MACD Slow', color='green')
ax[1].bar(data.index, data.MACD_Hist, color='gray', label='MACD Signal')

ax[2].plot(data.index, data.RSI, label='RSI Index')
ax[2].axhline(y=70, xmin=0, color='gray', linestyle='--')
ax[2].axhline(y=30, xmin=0, color='gray', linestyle='--')

for i in range(3):
    ax[i].legend(loc='upper left',bbox_to_anchor=(1, 1), frameon=True)

buySig = data.MACD.loc[data.Buy==True]
sellSig = data.MACD.loc[data.Sell==True]
ax[1].plot(buySig.index, buySig, "^", markersize=15, c='g')
ax[1].plot(sellSig.index, sellSig, "v", markersize=15, c='r')

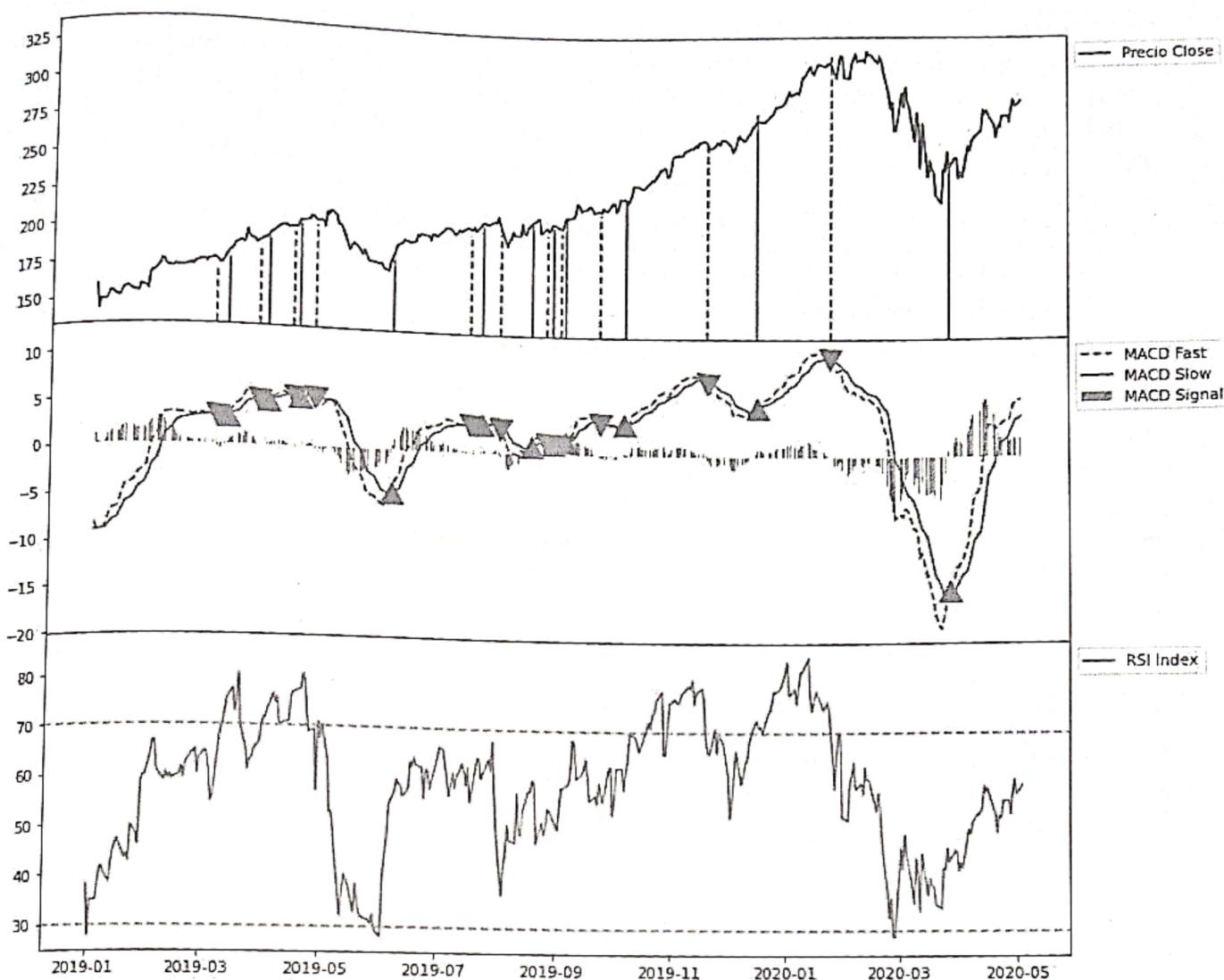
for idx, row in data.iterrows():
    rango = data.Close.max()-data.Close.min()
    h = (row.Close-data.Close.min())/rango

    if row.Buy==True:
        ax[0].axvline(x=idx, ymin=0, ymax=h , c='green')
    if row.Sell:
        ax[0].axvline(x=idx, ymin=0, ymax=h, c='red', ls="--")

plt.subplots_adjust(hspace=0)
plt.show()

```

output de la rta al ejercicio 10

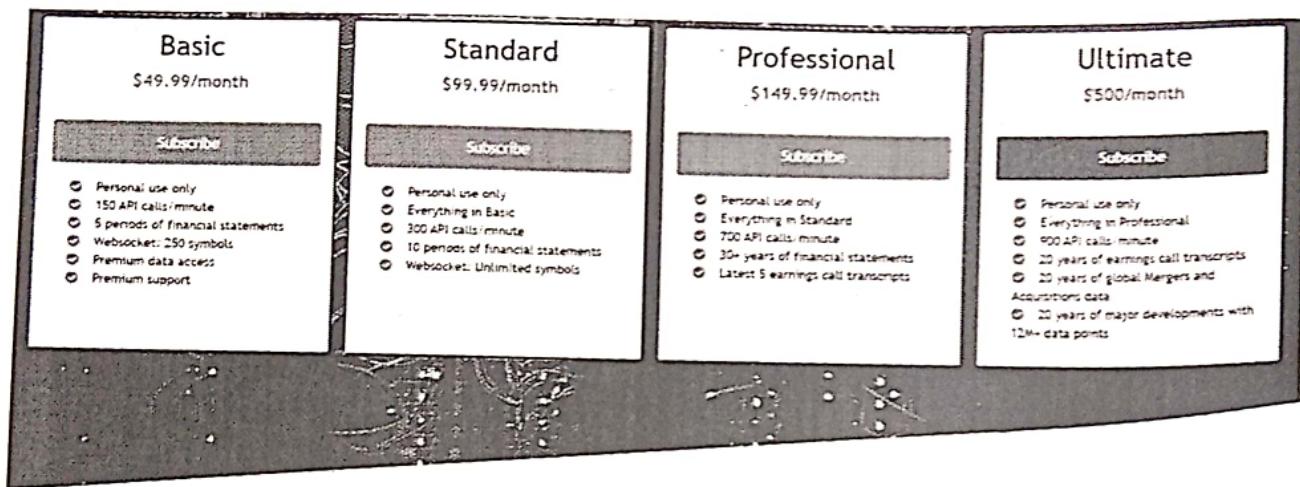


# FinnHub API

Esta es una API que tiene terrible potencia, tiene hasta WebSockets que no lo vamos a usar por ahora (es muy útil por ejemplo para proyectos que requieren streaming constante de datos) y la verdad que el perfil gratuito lo único que tiene limitado son por ejemplo acceso a estados contables alguna info más de perfil y fundamentals como el % de los principales holders de cada acción, y sobre todo lo más copado que le falta y si tienen los planes pagos es el historial de ticks, pero bueno todo gratis es mucho pedir

Importante: Al parecer esta API tiene toda la pinta de que mucha de la info que tiene gratis por ahora en algún momento será paga, al momento de escribir esto (mayo 2020), todos los endpoints son gratuitos, pero llama la atención la cantidad y calidad de info sin cargo, con lo cual mientras dure hay que aprovechar

Arrancamos, veamos las limitaciones primero



Get started with our Free plan

\$0/month

Test your idea or build a low-usage personal app at no cost. Upgrade anytime.

[Get FREE API Key](#)

- Personal use only
- 60 free API calls/minute
- Websocket: 50 symbols
- Access most data

Como les decía pocas diferencias, en eso que llaman "Premium Data Access" está la info de tick by tick, esto es operación por operación, independientemente de las velas, es una data super importante para bots de trading de alto volumen y ni hablar para bots de HFT, es decir bots de arbitrajes y bots que dependen mucho de la velocidad de colocación de las órdenes, que son los típicos bots de market makers

Sacando eso, nos dicen que podemos hacer hasta 60 requests por minuto es decir uno por segundo, es un montón para algo gratis, si recuerdan la de AlphaVantage nos permitía solo 5 requests por minuto

Lo primero que hay que hacer es sacar un token, yo ya saqué uno con un mail temporal que puedo compartir

```
TOKEN = 'bqoji3nrh5rced4gaukg'
```

Lo segundo es ir a la documentación y ver como hacer los requests, la eb es: <https://finnhub.io/> (<https://finnhub.io/>)

Como ven la documentación está super clara, de hecho hasta tiene los ejemplos en python de como hacer los requests

The screenshot shows two sections of the Finnhub API documentation:

**Stock Exchanges**  
List supported stock exchanges.  
**Method:** GET  
**Examples:**  
`/stock/exchange`  
**Arguments:**  
**Response Attributes:**  
Array  
Array of supported forex exchanges.  
**name**  
Exchange name.  
**code**  
Exchange code.  
**currency**  
Currency.

**Stock Symbol**  
List supported stocks.  
**Method:** GET  
**Examples:**  
`/stock/symbol/exchange/US%`

**Sample code:**

```
1 import requests
2 r = requests.get('https://finnhub.io/api/v1/stock/exchange?token=TOKEN')
3 print(r.json())
```

**Sample response:**

```
1 [
2   {
3     "code": "BSE",
4     "currency": "INR",
5     "name": "BSE exchanges"
6   },
7   {
8     "code": "NSE",
9     "currency": "INR",
10    "name": "NSE and NSEI"
11  }
12 ]
```

Vamos a arrancar con la función de consultar los mercados, para ello vamos a usar funciones, recuerden que siempre es buena práctica "empaquetar" en funciones los conjuntos de instrucciones que hacen a un mismo objetivo en este caso bajar datos de la API

```
import requests
import pandas as pd

def getExchanges():
    url = 'https://finnhub.io/api/v1/stock/exchange'
    p = {'token': TOKEN}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

exchanges = getExchanges()
exchanges
```

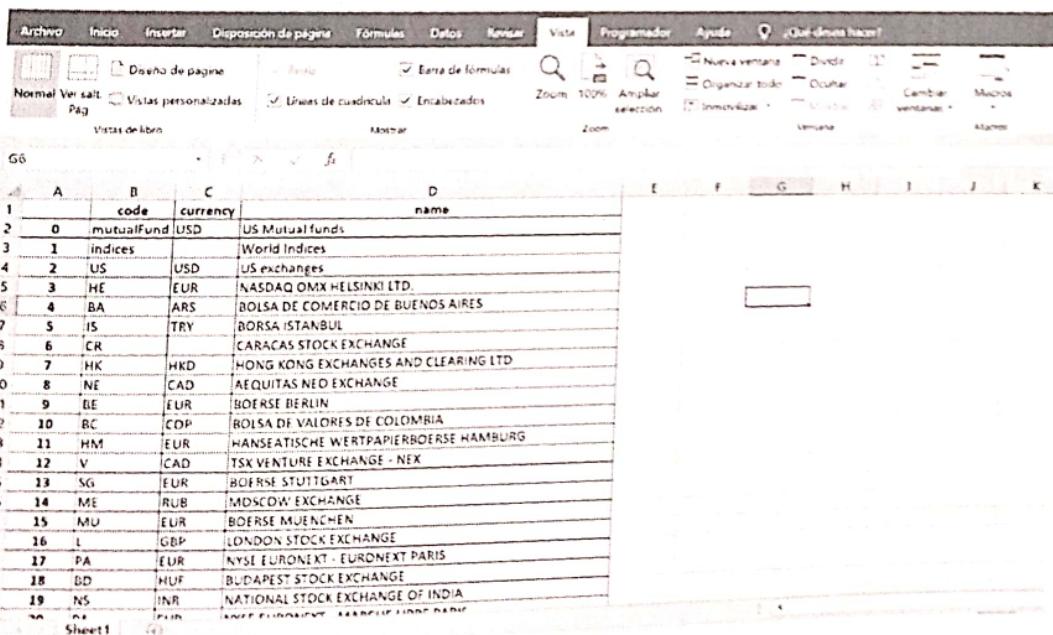
	code	currency	name
0	mutualFund	USD	US Mutual funds
1	indices		World Indices
2	US	USD	US exchanges
3	LS	EUR	NYSE EURONEXT - EURONEXT LISBON
4	PR	CZK	PRAGUE STOCK EXCHANGE
...	...	...	...
61	QA	QAR	QATAR EXCHANGE
62	MU	EUR	BOERSE MUENCHEN
63	ST	SEK	NASDAQ OMX NORDIC
64	BD	HUF	BUDAPEST STOCK EXCHANGE
65	VN	VND	Vietnam exchanges including HOSE, HNX and UPCOM

66 rows × 3 columns

Nada mal ehhh, tenemos 66 mercados con todas las acciones listadas en cada uno de ellos al alcance de una linea de código

Como ven aprovechamos y metimos en un DataFrame el JSON que devuelve. Ojo si trabajan con diccionarios de mercados como este ejemplo que puse es indiferente usar el diccionario o el JSON o pasarlo a un DataFrame, no nos agrega mucho, pero cuando trabajamos con series de tiempo o series largas y debemos usar filtros y esas cosas, la verdad que ahí conviene mucho usar DataFrames, así que les sugiero acostumbrarse a meter los diccionarios en DataFrames, les va a ahorrar mucho a futuro, de hecho por ejemplo si quisiera guardarme estos mercados en un Excel no se si recuerdan la siguiente función

```
exchanges = getExchanges()
exchanges.to_excel('exchanges_finnhub.xlsx')
```



	A	B	C	D
1	code	currency		name
2	0	mutualFund	USD	US Mutual funds
3	1	indices		World Indices
4	2	US	USD	US exchanges
5	3	HE	EUR	NASDAQ OMX HELSINKI LTD.
6	4	BA	ARS	BOLSA DE COMERCIO DE BUENOS AIRES
7	5	IS	TRY	BORSA ISTANBUL
8	6	CR		CARACAS STOCK EXCHANGE
9	7	HK	HKD	HONG KONG EXCHANGES AND CLEARING LTD.
10	8	NE	CAD	AEQUITAS NEO EXCHANGE
11	9	DE	EUR	BOERSE BERLIN
12	10	BC	COP	BOLSA DE VALORES DE COLOMBIA
13	11	HM	EUR	HANSEATISCHE WERTPAPIERBOERSE HAMBURG
14	12	V	CAD	TSX VENTURE EXCHANGE - NEX
15	13	SG	EUR	BOERSE STUTTGART
16	14	ME	RUB	MOSCOW EXCHANGE
17	15	MU	EUR	BOERSE MUENCHEN
18	16	I	GBP	LONDON STOCK EXCHANGE
19	17	PA	EUR	NYSE EURONEXT - EURNEXT PARIS
20	18	BD	HUF	BUDAPEST STOCK EXCHANGE
21	19	NS	INR	NATIONAL STOCK EXCHANGE OF INDIA
22	20	CA	JPY	TAKEI FUTUROCENTRE MARKET PLACE

Así que bueno, ese es mi consejo, pasar siempre los JSON de las respuestas a DataFrames como hicimos con la API de AlphaVantage

Como habrán notado esta API tiene las acciones de nuestro mercado y se puede decir que la API es bastante mas confiable que la de Yahoo

Bueno, ya que vimos la consulta a los mercados, veamos ahora como consultar a la API los instrumentos de cada mercado, en este caso le tenemos que pasar el exchange como parámetro

```
def getSymbols(exchange):
    url = 'https://finnhub.io/api/v1/stock/symbol'
    p = {'token': TOKEN, 'exchange' : exchange}

    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

symbols = getSymbols('BA')
symbols.head()
```

	description	displaySymbol	symbol
0	APPLE INC CEDEAR	AAPL.BA	AAPL.BA
1	ABBOTT LABORATO/SH	ABT.BA	ABT.BA
2	BARRICK GOLD Co/SH	ABX.BA	ABX.BA
3	Agrometal SAI	AGRO.BA	AGRO.BA
4	Amern Intl Grp /SH	AIG.BA	AIG.BA

Bueno, muestra algunos CEDEARS y AGRO del panel general porque imprimí solo en head() pero tiene todos los tickers del panel general y todos los CEDEARS también, interesante

## Perfil básico de la compañía

Podemos acceder a los datos básicos de un ticker como marketCap con este request, vemos como sería la función para obtener esos datos

```
import requests
import pandas as pd

def getProfile(symbol):
    url = 'https://finnhub.io/api/v1/stock/profile2'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    return js
data = getProfile('AAPL')
print(json.dumps(data, indent=5))

{
    "country": "US",
    "currency": "USD",
    "exchange": "NASDAQ NMS - GLOBAL MARKET",
    "finnhubIndustry": "Technology",
    "ipo": "1980-12-12",
    "logo": "https://static.finnhub.io/logo/87cb30d8-80df-11ea...."
    "marketCapitalization": 1333458,
    "name": "Apple Inc",
    "phone": "14089961010",
    "shareOutstanding": 4334.335,
    "ticker": "AAPL",
    "weburl": "https://www.apple.com/"
}
```

Como ven nos devuelve la fecha del IPO, un string con la URL del logo, el MktCap, la página web y alguna cosita mas.

## Ejecutivos de una compañía

Bueno arrancamos con las consultas por empresa, veamos antes de pasar a las series de precios algunas cositas interesantes que nos permite consultar esta API

Acá tenemos una consulta de los principales directivos de cada compañía, esto sale mayormente de los fillings que están obligadas a presentar a la SEC las compañías que cotizan en USA, por lo que es probable que si consultamos empresas de otros países no tengan info pero si vamos a ver info de las de EEUU

```

import requests, pandas as pd

def getEx(symbol):
    url = 'https://finnhub.io/api/v1/stock/executive'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['executive'])
    return df

data = getEx('AAPL')
data.head()

```

	age	compensation	currency	name	position	sex	since
0	69.0	557922.0	USD	Dr. Arthur Levinson	Independent Chairman of the Board	male	2011
1	59.0	11555466.0	USD	Mr. Timothy Cook	Chief Executive Officer, Director	male	2011
2	56.0	25209637.0	USD	Mr. Luca Maestri	Chief Financial Officer, Senior Vice President	male	2014
3	56.0	25207919.0	USD	Mr. Jeffrey Williams	Chief Operating Officer	male	2018
4	55.0	25231800.0	USD	Ms. Katherine Adams	Senior Vice President, General Counsel, Secretary	female	2017

Y es inevitable hacer lo siguiente no?

```
data.groupby('sex').mean()
```

sex	age	compensation
female	57.500000	1.348203e+07
male	62.314286	1.181220e+07

Vamos a poner una línea de código para que nos devuelva los números sin notación científica

```

pd.options.display.float_format = '{:,}'.format
data.groupby('sex').mean().round(1)

```

sex	age	compensation
female	57.5	13,482,026.6
male	62.3	11,812,203.3

No se a ustedes, a mi me sorprendió, veamos otra empresa

```
data = getEx('AMZN')
data.groupby('sex').mean().round(1)
```

age	compensation
sex	
female	57.8 812,473.0
male	58.2 3,698,851.2

Pelado amarrete jaja, a ver el colo..

```
data = getEx('FB')
data.groupby('sex').mean().round(1)
```

age	compensation
sex	
female	56.4 12,078,306.0
male	51.8 7,765,778.6

Y también podemos chusmear data Argentina, pero obviamente acá no se informa nada de remuneraciones

```
data = getEx('GGAL.BA')
data.head()
```

	age	compensation	currency	name	position	sex	since
0	69.0		None	Mr. Eduardo Escasany	Chairman of the Board	male	2010
1	66.0		None	Mr. Pedro Richards	Chief Executive Officer, Director	male	2017
2	59.0		None	Mr. Pablo Gutierrez	Vice Chairman of the Board	male	2010
3	47.0		None	Mr. Jose Ronsini	Chief Financial Officer	male	2018
4	80.0		None	Mr. Abel Ayerza	Director	male	1999

Bueno, basta de joda, seguimos con lo nuestro, veamos más Endpoints

## Obtener Noticias

Obviamente que esto a nosotros mucho no nos interesa, pero ya que lo tenemos a mano veamos cómo se obtiene esta data

```
def getNews(category):
    url = 'https://finnhub.io/api/v1/news'
    p = {'token': TOKEN, 'category' : category}
    r = requests.get(url, params = p) js = r.json()
    df = pd.DataFrame(js)
    return df

#Categorías disponibles: general, forex, crypto, merger
data = getNews('general')
print(data.columns)

Index(['category', 'datetime', 'headline', 'id', 'image', 'related',
       'source', 'summary', 'url'], dtype='object')
```

Veamos un código para mostrar noticias:

```
from datetime import datetime
for id, row in data.iterrows():
    fecha = datetime.fromtimestamp(row.datetime)
    print(row.category,'--',row.source,'--',fecha,'\n',row.headline,'\n',row.url)
    if id==3:
        break

earnings -- CNBC -- 2020-05-05 07:37:06
Tenet Healthcare warns of significant Covid-19 impact in the second quarter
https://www.cnbc.com/2020/05/05/tenet-healthcare-thc-earnings-q1-2020.html

earnings -- CNBC -- 2020-05-05 07:25:06
DuPont slashes capital expenses, increases cost savings
https://www.cnbc.com/2020/05/05/dupont-dd-earnings-q1-2020.html

company news -- Reuters -- 2020-05-05 07:20:05
DuPont slashes capital expenses, increase cost savings
https://www.reuters.com/article/us-dupont-results/dupont-slashe...

company news -- Reuters -- 2020-05-05 07:00:00
U.S. Senate panel to question Trump's pick to oversee coronavirus money
https://www.reuters.com/article/health-coronavirus-usa-....
```

Por una cuestión de espacio recorté la salida de las URL completas, pero en un script que se ejecute en un entorno windows esa URL por lo general al hacer clic te abre el navegador predeterminado y te lleva a la noticia desde su fuente original

# Noticias de una Empresa

```
def getSymbolNews(symbol, fromDate, toDate):
    url = 'https://finnhub.io/api/v1/company-news'
    p = {'token': TOKEN, 'symbol' : symbol, 'from':fromDate, 'to':toDate}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

data = getSymbolNews('XOM','2020-01-01','2020-05-05')
for id, row in data.iterrows():
    fecha = datetime.fromtimestamp(row.datetime)
    print(row.category,'--',row.source,'--',fecha,'\n',row.headline,'\n',row.url)
    if id==4:
        break
```

company -- seekingalpha -- 2020-05-05 06:54:49  
Q1 Earnings Calls Show Exxon Mobil Taking Cover, While ØRsted Is Flying  
<https://seekingalpha.com/article/4342952-q1-earnings-calls-....>

company -- cnbc -- 2020-05-05 03:28:19  
Total's net profit falls 35% in the first quarter as oil prices slide  
<https://www.cnbc.com/2020/05/05/total-q1-earnings-oil-majors-...>

company -- cnbc -- 2020-05-04 21:55:39  
Retail investors bought airline stocks even as travel slowed to a trickle  
<https://www.cnbc.com/2020/05/04/td-ameritrades-retail-investors...>

company -- investing -- 2020-05-04 21:00:00  
Earnings Of Big 3 Oil Supermajors Sending Mixed Signals On Demand Recovery  
<https://www.investing.com/analysis/earnings-of-big-3-oil-...>

## Mayores eventos

```
def getSymbolDev(symbol, fromDate, toDate):
    url = 'https://finnhub.io/api/v1/major-development'
    p = {'token': TOKEN, 'symbol' : symbol, 'from':fromDate, 'to':toDate}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['majorDevelopment'])
    return df

data = getSymbolDev('XOM','2020-01-01','2020-05-05')
for id, row in data.iterrows():
    print(row.datetime,'\n',row.headline,'\n')
    if id==4:
        break
```

```
-- Output del Script anterior -  
2020-05-01 12:26:12  
ExxonMobil Sees Margin Support From Lower Liquids Feed Costs In Q2 For Chemical Segment  
2020-05-01 12:21:02  
ExxonMobil Reports Q1 Loss Of $0.14 Per Share  
2020-05-01 08:13:30  
Oil Search To Undertake Further Measures To Reduce Breakeven Costs  
2020-04-29 21:43:18  
Exxon Mobil Sets Quarterly Cash Dividend Of $0.87Per Share  
2020-04-24 12:32:37  
Exxonmobil Modifies Facilities To Produce Medical-Grade Sanitizer For Covid-19 Response
```

## News Sentiment

En este caso por como viene presentada la data, no me conviene tanto usar un DataFrame, sino que voy a dumper el JSON

En un momento crítico para el petróleo y las petroleras corroboremos el sentiment

```
import json  
  
def getSentiment(symbol):  
    url = 'https://finnhub.io/api/v1/news-sentiment'  
    p = {'token': TOKEN, 'symbol' : symbol}  
    r = requests.get(url, params = p)  
    js = r.json()  
    return js  
data = getSentiment('XOM')  
print(json.dumps(data, indent=5))  
  
{  
    "buzz": {  
        "articlesInLastWeek": 120,  
        "buzz": 1.2903,  
        "weeklyAverage": 93  
    },  
    "companyNewsScore": 0.2372,  
    "sectorAverageBullishPercent": 0.5813,  
    "sectorAverageNewsScore": 0.4951,  
    "sentiment": {  
        "bearishPercent": 0.6819,  
        "bullishPercent": 0.3181  
    },  
    "symbol": "XOM"  
}
```

Bastante elocuente, este tipo de rastreo se hacen con bots tipo spyders que reastrean y contabilizan todas las palabras que hay asociadas a una acción en noticias y edios financieros, incluso hasta redes sociales, y le asignan a cada palabra un peso bullish o bearish, por ejemplo a la palabra "crisis" seguramente se le asignará un peso "bearish" importante, mientras que a la palabra "bueno" un peso bullish moderado o a la palabra "excelente" un peso bullish alto

Obviamente no es para nada preciso porque se asigna peso a palabras sacadas de contexto, pero en el grosso, es decir en una hora te barren millones de palabras en redes sociales y algo te están diciendo

## Pares similares

Esta consulta nos sirve para encontrar acciones que pueden estar correlacionadas con la acción que estamos estudiando, cuando nos metemos en mercados que no conocemos bien y estamos indagando suelen ser bastante útiles estas consultas

```
def getPeers(symbol):
    url = 'https://finnhub.io/api/v1/stock/peers'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    return js

data = getPeers('GGAL.BA')
data
```

```
['BMA.BA', 'GGAL.BA', 'BRIO.BA', 'BBAR.BA', 'BPAT.BA', 'SUPV.B A', 'BHIP.BA']
```

```
data = getPeers('AMZN')
data
```

```
['AMZN', 'BKNG', 'EBAY', 'CHWY', 'W', 'EXPE', 'ETSY', 'GRUB', 'Q RTEA', 'STMP']
```

## Métricas de Análisis Fundamental

Bueno, este EndPoint es realmente interesante, te devuelve las principales métricas de AF y están agrupadas en los siguientes grupos de métricas:

- price
- valuation
- growth
- margin
- management
- financialStrength
- perShare

Vamos a definir la función y veamos una por una que nos devuelve:

```
def getMetrics(symbol, metric):
    url = 'https://finnhub.io/api/v1/stock/metric'
    p = {'token': TOKEN, 'symbol' : symbol, 'metric':metric}
    r = requests.get(url, params = p) js = r.json()
    return js

data = getMetrics('GGAL.BA','price')
print(json.dumps(data,indent=5))

{
    "metric": {
        "10DayAverageTradingVolume": 2.58033,
        "13WeekPriceReturnDaily": -35.06441,
        "26WeekPriceReturnDaily": -12.43214,
        "3MonthAverageTradingVolume": 43.07904,
        "52WeekHigh": 175.9,
        "52WeekHighDate": "2019-08-09",
        "52WeekLow": 53.5,
        "52WeekLowDate": "2020-03-18",
        "52WeekPriceReturnDaily": -15.73154,
        "5DayPriceReturnDaily": 12.63966,
        "beta": 1.22207,
        "marketCapitalization": 92388.04,
        "monthToDatePriceReturnDaily": 39.1717,
        "priceRelativeToS&P50013Week": -19.88875,
        "priceRelativeToS&P50026Week": -6.4084,
        "priceRelativeToS&P5004Week": -0.31197,
        "priceRelativeToS&P50052Week": -23.89381,
        "priceRelativeToS&P500Ytd": -15.59024,
        "yearToDatePriceReturnDaily": -33.67599
    },
    "metricType": "price",
    "symbol": "GGAL.BA"
}
```

Bueno, claramente la métrica de precios no es una métrica de análisis fundamental, pero me devuelve

- $\beta$  el Beta es un indicador del movimiento del activo respecto a su mercado
- El Mkt Cap (El valor por bolsa de la empresa)
- Devuelve tambien el movimiento de la empresa respecto al SP500
- Devuelve los valores y las fechas de picos altos y bajos en las últimas 52 semanas
- Medias de volúmenes y retornos de 5 dias, 1, 2, 3, 6 y 12 meses
- Retornos MTD y YTD (del mes y año corriente)

```

data = getMetrics('GGAL.BA', 'valuation')
print(json.dumps(data, indent=5))

{
    "metric": {
        "currentDividendYieldTTM": 0,
        "dividendYield5Y": null,
        "dividendYieldIndicatedAnnual": 1.73809,
        "netDebtAnnual": -90070.18,
        "netDebtInterim": -90070.18,
        "pbAnnual": 1.21592,
        "pbQuarterly": 1.21592,
        "pcfShareTTM": 2.04974,
        "peBasicExclExtraTTM": 4.17485,
        "peExclExtraAnnual": 2.76893,
        "peExclExtraHighTTM": 18.41728,
        "peExclExtraTTM": 2.76893,
        "peExclLowTTM": 3.17683,
        "peInclExtraTTM": 2.76893,
        "peNormalizedAnnual": 2.76893,
        "pfcfShareAnnual": null,
        "pfcfShareTTM": null,
        "psAnnual": 0.64721,
        "psTTM": 0.647,
        "ptbvAnnual": 1.03656,
        "ptbvQuarterly": 1.03656
    },
    "metricType": "valuation",
    "symbol": "GGAL.BA"
}

```

Acá tenemos datos de:

- Dividendos
- Deuda neta
- pb (ratio precio de mercado sobre precio de libros)
- pe (ratio precio de mercado sobre ganancias)
- ps (ratio precio de mercado sobre ventas)

La sigla TTM significa que toma los últimos 12 meses hacia atrás independientemente del mes en el que se encuentre, que a diferencia del YTD toma desde principio del año corriente al mes actual.

Bueno, así como tenemos las métricas de valuación, también están las de crecimiento que lo que miden es su variación porcentual respecto al período anterior o presentación anterior

```

data = getMetrics('GGAL.BA','growth')
print(json.dumps(data,indent=5))

{
    "metric": {
        "bookValueShareGrowth5Y": 53.12003,
        "capitalSpendingGrowth5Y": 81.02543,
        "dividendGrowthRate5Y": null,
        "ebitdaCagr5Y": 55.69636,
        "ebitdaInterimCagr5Y": 55.70019,
        "epsGrowth3Y": 84.62633,
        "epsGrowth5Y": 62.5448,
        "epsGrowthQuarterlyYoy": 111.3791,
        "epsGrowthTTMYoy": 189.0901,
        "focfCagr5Y": null,
        "netMarginGrowth5Y": 13.33256,
        "revenueGrowth3Y": 52.83869,
        "revenueGrowth5Y": 45.88154,
        "revenueGrowthQuarterlyYoy": 45.22597,
        "revenueGrowthTTMYoy": 64.10203,
        "revenueShareGrowth5Y": 42.17158,
        "tbvCagr5Y": 60.20488,
        "totalDebtCagr5Y": 26.62086
    },
    "metricType": "growth",
    "symbol": "GGAL.BA"
}

```

Estos son todos indicadores de crecimiento, es decir los indicadores de epsGrowth no da una idea de la dinámica de los EarningsPerShare (EPS) es decir de la ganancia por acción, no de como es el valor, sino de que tanto está creciendo (obviamente respecto al período anterior)

Las métricas de growth son métricas que se asemejan mas a una película que a una foto, es decir nos muestran mas una dinámica

Los indicadores de crecimiento son útiles para detectar small caps con alto potencial en su fase temprana, obviamente acompañados de otro tipo de análisis, por eso esta bueno este resumen de todos ratios "growth" porque podemos comparar el ritmo de crecimiento de las ganancias (EPS) vs los ingresos (revenue) vs el patrimonio neto (bookValue) vs la deuda (total Debt) y así sucesivamente con cualquier ratio

### Ratios de margen

Si tenemos buenas métricas de crecimiento va a ser muy interesante adentrarnos en las métricas fundamentales de margen que nos dan una idea de los márgenes operativos y de ganancias de la compañía, veremos esto en las siguientes llamadas

```

data = getMetrics('GGAL.BA', 'margin')
print(json.dumps(data, indent=5))
{
    "metric": {
        "freeOperatingCashFlow/revenue5Y": 19.47705,
        "freeOperatingCashFlow/revenueTTM": -38.48505,
        "grossMargin5Y": null,
        "grossMarginAnnual": null,
        "grossMarginTTM": null,
        "netProfitMargin%Annual": 29.78355,
        "netProfitMargin5Y": 22.70966,
        "netProfitMarginTTM": 29.77368,
        "operatingMargin5Y": 30.14253,
        "operatingMarginAnnual": 35.74474,
        "operatingMarginTTM": 35.7329,
        "pretaxMargin5Y": 30.14253,
        "pretaxMarginAnnual": 35.74474,
        "pretaxMarginTTM": 35.7329
    },
    "metricType": "margin",
    "symbol": "GGAL.BA"
}

```

Los ratios de "margin" son muy útiles para filtros sobre otros screenings, por ejemplo si buscamos dentro de determinado rango de precio/ganancias (PER) empresas que tengan mejores márgenes operativos que otros porque proveemos una caída económica y estimamos que los negocios de margen serán más sólidos que los de volumen, entonces este tipo de ratio será ideal para mejorar ese filtro

### Ratios de Solvencia Financiera

```

data = getMetrics('GGAL.BA', 'financialStrength')
print(json.dumps(data, indent=5))
{
    "metric": {
        "currentEv/freeCashFlowAnnual": null,
        "currentEv/freeCashFlowTTM": null,
        "freeCashFlowAnnual": -56954.85,
        "freeCashFlowTTM": -54954.85,
        "longTermDebt/equityAnnual": 40.38971,
        "longTermDebt/equityQuarterly": 40.38971,
        "netInterestCoverageTTM": null,
        "payoutRatioAnnual": 4.81265,
        "payoutRatioTTM": 4.81265,
        "quickRatioAnnual": null,
        "quickRatioQuarterly": null,
        "totalDebt/totalEquityAnnual": 40.38971,
        "totalDebt/totalEquityQuarterly": 40.38971
    },
    "metricType": "financialStrength",
    "symbol": "GGAL.BA"
}

```

## Ratios de Management

```
data = getMetrics('GGAL.BA','management')
print(json.dumps(data,indent=5))

{
    "metric": {
        "assetTurnoverAnnual": null,
        "assetTurnoverTTM": null,
        "inventoryTurnoverAnnual": null,
        "inventoryTurnoverTTM": null,
        "netIncomeEmployeeAnnual": 4325509,
        "netIncomeEmployeeTTM": null,
        "receivablesTurnoverAnnual": null,
        "receivablesTurnoverTTM": null,
        "revenueEmployeeAnnual": 14523150,
        "revenueEmployeeTTM": null,
        "roaRfy": 6.91661,
        "roaa5Y": 4.62581,
        "roae5Y": 40.95633,
        "roaeTTM": 55.58977,
        "roeRfy": 55.58977,
        "roeTTM": 6.91661,
        "roi5Y": null,
        "roiAnnual": null,
        "roiTTM": null
    },
    "metricType": "management",
    "symbol": "GGAL.BA"
}
```

Acá tenemos los típicos ratios de tablero:

- ROA: Returns on Assets (retorno sobre Activos)
- ROE: Returns on Equity (retorno sobre Patrimonio neto)
- ROI: Returns on Investment (retorno sobre inversión)  
Y despues el giro de inventarios y activos, util por ej, en el análisis de Dupont

## Ratios por Acción

Los ratios por acción nos permiten poner en contexto al precio por acción por ejemplo las ganancias, el famoso EPS (ganancias por acción) o los dividendos o el valor de efectivo en caja o el valor libros siempre comparado por cada acción, sobre todo puede ser muy útil para tener una referencia en empresas muy castigadas en sus precios

```
data = getMetrics('GGAL.BA','perShare')
print(json.dumps(data,indent=5))
```

-- Output en la siguiente página -

```

{
  "metric": {
    "bookValuePerShareAnnual": 66.32856,
    "bookValuePerShareQuarterly": 66.32856,
    "cashFlowPerShareAnnual": 31.59103,
    "cashFlowPerShareTTM": 31.59103,
    "cashPerSharePerShareAnnual": 89.91888,
    "cashPerSharePerShareQuarterly": 89.91888,
    "dividendPerShare5Y": null,
    "dividendPerShareAnnual": null,
    "dividendsPerShareTTM": 0,
    "ebitdPerShareTTM": 52.18668,
    "epsBasicExclExtraItemsAnnual": 29.12681,
    "epsBasicExclExtraItemsTTM": 29.12681,
    "epsExclExtraItemsAnnual": 29.12681,
    "epsExclExtraItemsTTM": 29.12681,
    "epsInclExtraItemsAnnual": 29.12681,
    "epsInclExtraItemsTTM": 29.12681,
    "epsNormalizedAnnual": 29.12681,
    "freeCashFlowPerShareTTM": -38.5171,
    "revenuePerShareAnnual": 100.0501,
    "revenuePerShareTTM": 100.0833,
    "tangibleBookValuePerShareAnnual": 62.4694,
    "tangibleBookValuePerShareQuarterly": 62.4694
  },
  "metricType": "perShare",
  "symbol": "GGAL.BA"
}

```

Esta serie de ratios es genial porque permite llevar todo a la unidad de acción flotante, es muy útil para observar cuando las cotizaciones llega a valores límites muy bajos comparando con fundamentales mas del tipo de mercado que del tipo financieras, para ver si hay una oportunidad real o una potencial quiebra

## Filings (SEC)

Bueno, para los que aún no lo saben la SEC obliga a las empresas cotizantes en EEUU a presentar ciertos formularios

La base de datos de la SEC es pública se llama EDGAR que es una sigla que significa (Electronic Data Gathering Analysis and Retrieval System) que en realidad surge de un proyecto mucho más amplio, pero en nuestro campo de acción es digamos la fuente de información pública de donde salen los datos de las compañías y los grandes tenedores de acciones que a partir de los 1000 millones de dólares están obligados a informar trimestralmente a la SEC sus cambios de posiciones

Todo ese movimiento de informacion da lugar a los llamados "filings" que son formularios con info presentada, tienen números o códigos que indican que tipo de formulario es

```
def filings(symbol):
    url = 'https://finnhub.io/api/v1/stock/filings'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

data = filings('AAPL')
print(data.loc[0], '\n\n', data.loc[0]['filingUrl'])
```

```
accessNumber      0001193125-20-050884
symbol           AAPL
cik              320193
form             8-K
filedDate        2020-02-27 00:00:00
acceptedDate     2020-02-27 06:14:21
reportUrl        https://www.sec.gov/ix?doc=/Archives/edg...
filingUrl        https://www.sec.gov/Archives/edgar/data...
```

Name: 0, dtype: object

<https://www.sec.gov/Archives/edgar/data/320193/000119312520050884/0001193125-20-050884-index.html>

Las URL en la salida de DataFFrame salen cortados, pero les muestro a modo de referencia como sacar la salida de la URL completa, para chusmear un poco la base de datos EDGAR

Les muestro solo el primer filing, y luego la URL completa del mismo, es un mundo inmenso esto de los filings, para el que hace análisis fundamental recomiendo fuertemente ponerse a estudiar todo esta info en detalle, no me voy a explayar acá porque no es una info tan rica desde el punto de vista de data science ya que es mas cualitativa que cuantitativa

Veamos por ejemplo un filtro o agrupamiento por tipo de filing, para que vean que tan rica y variada es esta info

```
data = filings('AAPL')
porTipo = data.groupby('form').symbol.count()
porTipo
```

form

10-K	16
10-K/A	1
10-Q	41
10-Q/A	1
25	1
3	4
4	59
424B2	39
424B5	1
8-A12B	6
8-K	110
8-K/A	1
CERT	2
CERTNYS	4
CORRESP	12
DEF 14A	12
DEFA14A	10
DEFR14A	1
DFAN14A	1
FWP	19
NO ACT	13
PRE 14A	3
PX14A6G	4
S-3ASR	3
S-8	14
S-8 POS	1
SC 13G	1
SC 13G/A	1
SC TO-I/A	1
SD	6
UPLOAD	18

Name: symbol, dtype: int64

O sea que me muestra que hay formularios que tiene mas de 100 cargados o mas de 10 un montón y recuerden que por cada uno esta la URL con la referencia completa

## IPO Calendar

Este es un endpoint interesante para quienes tradean estos eventos. Se trata ni mas ni menos que del calendario de Ofertas públicas iniciales próximas

Veamos como sería la función en python y la respuesta de la API .Fíjense que en este caso voy a dar dos salidas, una en formato json y la otra en formato DataFrame

Al sacar dos salidas desde el return, claramente luego tengo que poner dos nombre de variables cuando ejecuta la función para que le asigne una salida a cada una, a mi gusto es despropósito hacerlo así pero a veces es útil, y se los pongo porque en algún momento se van a topar con ese tipo de cosas así que me pareció buen momento para meter esto

```
def IPOs(fromDate, toDate):
    url = 'https://finnhub.io/api/v1/calendar/ipo'
    p = {'token': TOKEN, 'from' : fromDate, 'to' : toDate}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['ipoCalendar'])
    return js, df

dataJS, dataDF = IPOs(fromDate='2020-04-01',toDate='2020-04-03')
print(json.dumps(dataJS, indent=5))
```

```
{
  "ipoCalendar": [
    {
      "date": "2020-04-03",
      "exchange": "NASDAQ Global",
      "name": "ZENTALIS PHARMACEUTICALS, INC.",
      "numberOfShares": 9180000,
      "price": "18.00",
      "status": "priced",
      "symbol": "ZNTL",
      "totalSharesValue": 165240000
    },
    {
      "date": "2020-04-01",
      "exchange": "NASDAQ Global",
      "name": "WIMI HOLOGRAM CLOUD INC.",
      "numberOfShares": 4750000,
      "price": "5.50",
      "status": "priced",
      "symbol": "WIMI",
      "totalSharesValue": 26125000
    }
  ]
}
```

Aprovechando que generamos la función con la salida de DataFrame podríamos hacer un conteo de IPOs por meses dado un determinado año.. interesante no? bueno se las dejo picando para la ejercitación viene eso..

## Recomendaciones de "Analistas"

Sorry por lo de las comillas, pero cuando aparecen este tipo de información y no me dicen quienes son los analistas y básicamente en que criterio se basan, la verdad le tengo que poner las comillas obligadamente, yo les muestro las herramientas ustedes decidan cual es la mejor manera de usarlas, claramente tener más datos es mejor que tener menos, pero muchas veces no hay que leerlos tan literal como parecería que se supone que hay que leerlos, se puede ver por ejemplo la serie de los mismos "analistas" si no confiamos en el criterio absoluto, es decir medir lo relativo mes a mes de los mismos "analistas" jaja insisto con las comillas perdón

O también lo que se puede hacer con este tipo de datos es comparar entre mismos "analistas" distintas empresas, o bien contra benchmarks, es decir comparar la performance de las recomendaciones versus la performance de los índices de referencia de esas empresas, en fin solo les tiro ideas para que no tengan esa tentación que tuve yo de entrada de descartar este tipo de información de una

Hecho esas aclaraciones como esta serie viene con una fecha, vamos a pasar esa fecha (Que viene como string) a formato fecha y la vamos a poner como índice, hacemos esto dentro de la función, y como les dije antes, vamos a usar este criterio cada vez que sea posible, es decir cada vez que los responses de las APIs sean series de tiempo o series que tengan una fecha en alguna columna, ya que si ponemos la fecha como índice del dataframe podremos acceder a infinidad de funciones de agrupamiento y resampleo

```
def recommendation(symbol):
    url = 'https://finnhub.io/api/v1/stock/recommendation'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df['period'] = pd.to_datetime(df.period)
    df.set_index('period', inplace=True)
    df.drop(['symbol'], axis=1, inplace=True)
    return df

data = recommendation('MELI')
data.head()
```

		buy	hold	sell	strongBuy	strongSell
period						
2020-05-01	4	9	0	2	0	0
2020-04-01	11	9	1	2	0	0
2020-03-01	11	9	1	2	0	0
2020-02-01	5	9	1	1	0	0
2020-01-01	5	9	1	1	0	0

Por ejemplo, yendo al tema anterior que hablaba de usar estos datos de un modo mas amplio que ver si hoy un papel está o no con buenas recomendaciones, vamos a construir una función que le asigne un puntaje a cada estado:

- Buy = 2 puntos
- Hold = 1 puntos
- Sell = -1 puntos
- Strong Buy = 3 puntos
- Strong Sell = -2 puntos

En función de los puntajes, vamos construir un DataFrame que me muestre la evolución de ese puntaje, y que pueda comparar el puntaje de una empresa A con una empresa B, yo acá les hago la primera parte porque la segunda la dejo para la ejercitación para que practiquen ustedes (Ejercicio 3 de este bloque)

Para que no dependa el puntaje de la cantidad de recomendaciones (AAPL va a tener muchas más que una small cap) vamos a generar una columna de puntajes, otra de conteo de recomendaciones totales del período y una tercera columna, que es la importante, que es el puntaje medio por recomendación

Ojo, nada de esto es una recomendación de análisis, son solo ideas como disparadores y como recursos para enseñar las herramientas, claro está que si a una empresa la analizan 20 "especialistas" y a otra solo 5 "especialistas" no hay una homogeneidad en el puntaje medio que saque entre ambas

Pero bueno, nada, ningún análisis es tajante tampoco, siempre se trata de herramientas que nos dan un proxy y luego usando mucha combinación de datos diferentes puedo armar un esquema de decisión, pero falta mucho para eso, por ahora nos concentraremos en ir armando sets de datos y "comparables"

```
def puntaje(recomendaciones):  
    pesos = [2,1,-1,3,-2]  
    puntaje = recomendaciones.mul(pesos, axis=1).sum(axis=1)  
    conteo = recomendaciones.sum(axis=1)  
    puntajeMedio = puntaje/conteo  
    resumen = pd.concat([puntaje,conteo,puntajeMedio],axis=1)  
    resumen.columns = ['Puntaje','Conteo','PuntajeMedio']  
    resumen = resumen.sort_values('period', ascending=True)  
    inicial = float(resumen[0:1]['PuntajeMedio'])  
    resumen['Idx100'] = 100*resumen['PuntajeMedio']/inicial  
    return resumen  
  
data = recommendation('AAPL')  
puntajes = puntaje(data)  
puntajes.tail()  
  
-- Output en la página siguiente --
```

period	Puntaje	Conteo	PuntajeMedio	Idx100
2020-01-01	74	44	1.681818	86.093074
2020-02-01	74	44	1.681818	86.093074
2020-03-01	76	41	1.853659	94.889663
2020-04-01	78	40	1.950000	99.821429
2020-05-01	81	38	2.131579	109.116541

Y ahí está tenemos una serie de puntajes medios que nos da un proxy del humor de los "analistas" respecto a esta empresa y en la columna "normalizada como índice base 100" para poder comparar evolución entre varias

Consejo, es una API bastante nueva que recién le empezó a dar bola a este tipo de datos en 2020, así que los datos previos de esta serie es probable que hayan hecho un copy-paste de alguna fuente, pero habrá que seguir mas adelante esta serie como la manejan

## Target de Precio

Bueno, voy mostrando variantes en la construcción de la salida del request que les van a ser útiles en función de las necesidades que tengan con respecto a esa salida, en este caso le saco la columna "symbol" y uso el ticker de índice de la serie que en este caso es de un solo ticker pero bien podría pedir targets de varios

```
def priceTarget(symbol):
    url = 'https://finnhub.io/api/v1/stock/price-target'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js, index=[symbol]).drop(['symbol'],axis=1)
    df['lastUpdated'] = pd.to_datetime(df.lastUpdated)
    return df

target = priceTarget('AAPL')
target
```

	lastUpdated	targetHigh	targetLow	targetMean	targetMedian
AAPL	2020-05-07	370.8	207.77	303.56	310

Y así gracias al uso de funciones y un poco de maña con solo un par de líneas nos podemos armar una tabla de target Prices

```

def tablaTarget(tickers):
    tabla = pd.DataFrame()
    for ticker in tickers:
        target = priceTarget(ticker)
        tabla = pd.concat([tabla,target], axis=0)
    return tabla

tablaTarget(['GGAL','YPF','PAM'])

```

	lastUpdated	targetHigh	targetLow	targetMean	targetMedian
GGAL	2020-05-07	24.1	9	13.91	13.00
YPF	2020-05-07	18.0	1	7.56	7.60
PAM	2020-05-07	33.0	7	18.11	17.25

## Upgrades & DownGrades

Tiene mucha data esta ref, la implementación es así

```

def upDown(symbol):
    url = 'https://finnhub.io/api/v1/stock/upgrade-downgrade'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df['gradeTime'] = pd.to_datetime(df.gradeTime, unit='s')
    return df
upDown('JPM').head()

```

	symbol	gradeTime	company	fromGrade	toGrade	action
0	JPM	2020-05-04	UBS		Neutral	main
1	JPM	2020-04-15	Morgan Stanley		Overweight	main
2	JPM	2020-04-15	BMO Capital		Market Perform	main
3	JPM	2020-04-06	Wells Fargo		Overweight	main
4	JPM	2020-04-06	UBS		Neutral	main

Obviamente se refieren a los upgrades y downgrades de calificaciones, nos indican las calificacoras, en realidad analistas porque no son calificaciones de deuda sino análisis de price targeting, es decir de si recomiendan comprar vender o mantener en función de si la ver sobrecomprada sobre vendida etc, pero siempre respecto a su propio análisis anterior, por lo tanto es una valoración subjetiva y relativa.

## Estimaciones

Tenemos estimaciones de ventas y de Ganancias por acción, les dejo ambas funciones, son muy similares.

```
def estRev(symbol, freq='quarterly'):
    url = 'https://finnhub.io/api/v1/stock/revenue-estimate'
    p = {'token': TOKEN, 'symbol' : symbol, 'freq':freq}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['data'])
    df.set_index('period', inplace=True)
    return df
estRev(symbol='JPM', freq='annual').head()
```

	numberAnalysts	revenueAvg	revenueHigh	revenueLow
period				
2022-12-31	11	121989559110	135345000000	110274450000
2021-12-31	19	120780000000	132153000000	110360780000
2020-12-31	22	117680000000	128313150000	107971670000
2019-12-31	23	116104275570	124167142050	110483000000
2018-12-31	24	110756672570	117906600000	105697990000

```
def estEPS(symbol, freq='quarterly'):
    url = 'https://finnhub.io/api/v1/stock/eps-estimate'
    p = {'token': TOKEN, 'symbol' : symbol, 'freq':freq}
    r = requests.get(url, params = p) js = r.json()
    df = pd.DataFrame(js['data'])
    df.set_index('period', inplace=True)
    return df
estEPS('GGAL')
```

	epsAvg	epsHigh	epsLow	numberAnalysts
period				
2020-06-30	0.93	1.0100	0.8624	5
2020-03-31	0.83	0.9696	0.6468	5

Como ven en las salidas las estimaciones de ingresos son absolutas es decir estiman directamente el valor que preveen en los balances en función de temas macro y del sector y empresa en si, mientras que las estimaciones de EPS ya son relativas porque obviamente el denominador es el precio de la acción que es dinámico

## EPS Surprise

Atenti a este endpoint, es muy util para backtestear saltos de volatilidad y analizar estrategias de opciones, como verán en la función le agrego la columna que calcula la sorpresa en % respecto al esperado

```
def epsS(symbol):
    url = 'https://finnhub.io/api/v1/stock/earnings'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('period', inplace=True)
    df['surprise'] = round((df.actual/df.estimate-1)*100,2)
    df.drop(['symbol'],axis=1, inplace=True)
    return df
epsS('GGAL')
```

	actual	estimate	surprise
period			
2019-12-31	1.11	0.99	12.12
2019-09-30	1.61	0.92	75.00
2019-06-30	1.85	0.98	88.78
2019-03-31	1.62	0.74	118.92
2018-12-31	0.84	0.70	20.00

## Earnings Calendar

Idem, atenti a los operadores de volatilidad, es sabido que los eventos de presentación de ganancias pueden ser disparadores de volatilidad o más bien todo lo contrario si hay mucho miedo y no hay sorpresas

```
def earningsCalendar(FROM, TO, international=False):
    url = 'https://finnhub.io/api/v1/calendar/earnings'
    p = {'token': TOKEN, 'from' : FROM, 'to':TO, 'international':international}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['earningsCalendar'])
    df.set_index('date', inplace=True)
    df.drop(['epsActual','epsEstimate','revenueActual','revenueEstimate'],
            axis=1,inplace=True)
    return df

# Lista de Earnings
calendario = earningsCalendar('2020-05-05', '2020-06-31')

# Cantidad por fecha
calendario.groupby('date').symbol.count()

-- Output en la página siguiente --
```

```
date
2020-05-07    135
2020-05-08     87
2020-05-11    183
2020-05-12     56
2020-05-13     39
Name: symbol, dtype: int64
```

## Precios

### Cotizaciones

El response de estas series está super comprimido, le ponen:

- pc: Previous close
- t: Timestamp (momento de cierre de la vela)
- o: Open
- h: High
- l: Low
- c: Close

Para acciones que cotizan en USA es en RealTime, para otros mercados generalmente no, por ejemplo, para Arg es con 20 minutos de delay, cada mercado tiene su propio delay, ojo algunos es muy muy bajo ese delay

```
def getQuote(symbol):
    url = 'https://finnhub.io/api/v1/quote'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    return js

data = getQuote('GGAL.BA')
print(json.dumps(data,indent=5))
```

```
{
    "c": 84,
    "h": 84.5,
    "l": 83,
    "o": 83.05,
    "pc": 81.45,
    "t": 1588688732
}
```

## Series históricas

Si buscan series intradiarias de muchos años, esta es la función y la API que necesitaban, super útil para los que buscan trading intradiario de stock

Primera implementación posible con un contador de cantidad de Días, o intervalos

Miren que interesante que está esto que hasta puedo pedir directamente la serie Ajustada y me devuelve todo (OHLC) ajustado

```
def getHistory(symbol, interval, count, adj):
    url = 'https://finnhub.io/api/v1/stock/candle'
    p = {'token': TOKEN, 'symbol' : symbol, 'resolution':interval,
         'count':count, 'adjusted':adj}

    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df['date'] = pd.to_datetime(df['t'], unit='s')
    df.set_index('date', inplace=True)
    df.drop(['s','t'],axis=1, inplace=True)
    df.columns = ['Close','High','Low','Open','Volume']
    return df

getHistory('YPF','D',8000,'true').head()
```

		Close	High	Low	Open	Volume
	date					
1998-06-12	13:30:00	8.6199	8.8074	8.3074	8.6824	2634200
1998-06-15	13:30:00	8.3522	9.2272	8.3522	8.7897	1744200
1998-06-16	13:30:00	8.2808	8.8433	8.0308	8.5933	2143700
1998-06-17	13:30:00	8.5663	8.9413	7.9413	8.0038	1622900
1998-06-18	13:30:00	8.2273	9.2273	8.1023	9.2273	1258900

La otra forma es implementarlo pasándole como argumentos el "from" y "to", esto obviamente porque así lo permite la API.

El tema es que la API pide que le mandemos el "from" y "to" como timestamps, y obviamente para nosotros es imposible pensar una fecha en "timestamps" así que tenemos que implementar dentro de la función que pase los strings que reciba a timestamps

Esta manera de llamar a la API es más eficiente para el response (tarda menos en responder y es más lógica) además la otra forma si le mandábamos una cantidad fuera de rango nos devuelve error

```
def getHistoryDates(symbol, interval, fromS,toS, adj):  
  
    # Pasamos a formato timestamp los strings YYYY-MM-DD  
    import datetime  
    fromDT = datetime.datetime.strptime(fromS, '%Y-%m-%d')  
    fromTS = int(datetime.datetime.timestamp(fromDT))  
    toDT = datetime.datetime.strptime(toS, '%Y-%m-%d')  
    toTS = int(datetime.datetime.timestamp(toDT))  
  
    # Hacemos el request http  
    url = 'https://finnhub.io/api/v1/stock/candle'  
    p = {'token': TOKEN, 'symbol' : symbol, 'resolution':interval,  
         'from':fromTS, 'to':toTS,'adjusted':adj}  
  
    r = requests.get(url, params = p)  
    js = r.json()  
  
    # Acomodamos el JSON de respuesta en un DataFrame  
    df = pd.DataFrame(js)  
    df['date'] = pd.to_datetime(df['t'], unit='s')  
    df.set_index('date', inplace=True)  
    df.drop(['s','t'],axis=1, inplace=True)  
    df.columns = ['Close','High','Low','Open','Volume']  
    return df  
  
getHistoryDates('YPF','D','1990-01-01','2020-05-10','true').head()
```

	Close	High	Low	Open	Volume
date					
1993-06-29 13:30:00	5.1827	5.4327	4.4327	4.5577	34346800
1993-06-30 13:30:00	5.1531	6.4031	5.0281	5.4031	9060600
1993-07-01 13:30:00	5.0346	5.4096	4.7846	5.1596	4273100
1993-07-02 13:30:00	5.0050	5.1300	4.6300	5.0050	2504500
1993-07-06 13:30:00	4.9161	5.1661	4.7911	5.0411	2030400

Bueno, después tiene para consultar dividendos y splits por acción, se las dejo así avanço con otras cosas más usadas

## FOREX

Tenemos un endpoint para pedir por los mercados, otro por los pares de cada mercado

```
def getExchanges():
    url = 'https://finnhub.io/api/v1/forex/exchange'
    p = {'token': TOKEN}
    r = requests.get(url, params = p)
    js = r.json()
    return js
getExchanges()
```

```
['oanda',
 'fxcm',
 'forex.com',
 'icmtrader',
 'ic markets',
 'fxpro',
 'octafx',
 'fxpig',
 'pepperstone']
```

```
def symbolsFX(exchange):
    url = 'https://finnhub.io/api/v1/forex/symbol'
    p = {'token': TOKEN, 'exchange' : exchange}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

symbolsFX('oanda').head()
```

	description	displaySymbol	symbol
0	US SPX 500	SPX500/USD	OANDA:SPX500_USD
1	Gold/CAD	XAU/CAD	OANDA:XAU_CAD
2	CHF/JPY	CHF/JPY	OANDA:CHF_JPY
3	US 2Y T-Note	USB02Y/USD	OANDA:USB02Y_USD
4	Copper	XCU/USD	OANDA:XCU_USD

Ahora sabiendo los mercado y teniendo como acceder a los instrumentos de cada mercado de Forex, puedo recorrer ambas listas para armar o bien listas personalizadas o bien diccionarios de cada mercado

Con un simple FOR podemos ver la cantidad de pares de cada mercado:

```
exchanges = getExchanges()
totals = {}
for exchange in exchanges:
    totals[exchange] = symbolsFX(exchange).symbol.count()
totals

{'oanda': 124, 'fxcm': 71,
'forex.com': 84,
'icmtrader': 154,
'ic markets': 82,
'fxpro': 105,
'octaafx': 30,
'fxpig': 110,
'pepperstone': 95}
```

Y la consulta por par

```
import datetime
def historyFX(exchange,symbol, interval, fromS,toS):
    s = exchange+':'+symbol

    # Pasamos a formato timestamp los strings YYYY-MM-DD
    fromDT = datetime.datetime.strptime(fromS, '%Y-%m-%d')
    fromTS = int(datetime.datetime.timestamp(fromDT))
    toDT = datetime.datetime.strptime(toS, '%Y-%m-%d')
    toTS = int(datetime.datetime.timestamp(toDT))

    # Hacemos el request http
    url = 'https://finnhub.io/api/v1/forex/candle'
    p = {'token': TOKEN, 'symbol' : s, 'resolution':interval, 'from':fromTS,
        'to':toTS}
    r = requests.get(url, params = p)
    js = r.json()

    # Acomodamos la salida a un dataframe prolijo
    df = pd.DataFrame(js)
    df['date'] = pd.to_datetime(df['t'], unit='s')
    df.set_index('date', inplace=True)
    df.drop(['s','t'],axis=1, inplace=True)
    df.columns = ['Close','High','Low','Open','Volume']
    return df

historyFX('OANDA','SPX500_USD',60,'2020-01-01','2020-05-05').head()
```

	Close	High	Low	Open	Volume
date					
2020-01-01 23:00:00	3237.8	3240.6	3234.8	3237.1	382
2020-01-02 00:00:00	3239.1	3239.3	3237.3	3238.0	114
2020-01-02 01:00:00	3238.6	3240.6	3236.3	3239.3	306
2020-01-02 02:00:00	3239.8	3240.3	3238.0	3238.6	152
2020-01-02 03:00:00	3240.8	3241.1	3239.8	3239.8	86

## Crypto

Bueno, arrancando con el mundo crypto veamos los exchanges o mercado que tiene acceso esta API de pares de cryptomonedas:

```
def exchangesCrypto():
    url = 'https://finnhub.io/api/v1/crypto/exchange'
    p = {'token': TOKEN}
    r = requests.get(url, params = p)
    js = r.json()
return js
exchangesCrypto()
```

```
['BITTREX',
 'GEMINI',
 'POLONIEX',
 'COINBASE',
 'Binance',
 'ZB',
 'HUOBI',
 'HITBTC',
 'KUCOIN',
 'BITFINEX',
 'KRAKEN',
 'OKEX']
```

Bien, tenemos acceso a esos 12 mercados, son mercados muy grandes y muy líquidos todos, pero ahora veamos los símbolos de cada Exchange, rimimos solo el head() de uno:

```
def symbolsCrypto(exchange):
    url = 'https://finnhub.io/api/v1/crypto/symbol'
    p = {'token': TOKEN, 'exchange' : exchange}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

symbolsCrypto('HITBTC').head()
```

	description	displaySymbol	symbol
0	Hitbtc BCN/BTC	BCN/BTC	HITBTC:BCNBTC
1	Hitbtc MKR/DAI	MKR/DAI	HITBTC:MKRDAI
2	Hitbtc EOS/DAI	EOS/DAI	HITBTC:EOSDAI
3	Hitbtc USD/DAI	USD/DAI	HITBTC:USDDAI
4	Hitbtc MKR/BTC	MKR/BTC	HITBTC:MKRBTC

Vamos a hacer ahora lo mismo que hicimos con Forex, que es recorrer mercado por mercado para contabilizar la cantidad de pares de cada exchange

Recorremos los exchanges:

```
exchanges = exchangesCrypto()
totals = {}
for exchange in exchanges:
    totals[exchange] = symbolsCrypto(exchange).symbol.count()
totals

{'BITTREX': 205,
 'GEMINI': 27,
 'POLONIEX': 137,
 'COINBASE': 59,
 'Binance': 797,
 'ZB': 249,
 'HUOBI': 573,
 'HITBTC': 912,
 'KUCOIN': 435,
 'BITFINEX': 290,
 'KRAKEN': 155,
 'OKEX': 859}
```

Acá para variar un poco la implementación la hago fijando el timestamp actual y tomando solo el "from", está bueno hacerlo así para aprovechar el hecho de que las series son RealTime

```
import datetime
def historyCrypto(exchange,symbol, interval, fromS):
    s = exchange+':'+symbol
    fromDT = datetime.datetime.strptime(fromS, '%Y-%m-%d')
    fromTS = int(datetime.datetime.timestamp(fromDT))
    toTS = int(datetime.datetime.now().timestamp())
    url = 'https://finnhub.io/api/v1/crypto/candle'
    p = {'token': TOKEN, 'symbol' : s, 'resolution':interval, 'from':fromTS,
         'to':toTS}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df['date'] = pd.to_datetime(df['t'], unit='s')
    df.set_index('date', inplace=True)
    df.drop(['s','t'],axis=1, inplace=True)
    df.columns = ['Close','High','Low','Open','Volume']
    return df
historyCrypto('HITBTC','ETHBTC','5','2019-01-01').tail()
```

	Close	High	Low	Open	Volume
date					
2020-05-06 14:40:00	0.022563	0.022570	0.022557	0.022566	1953.7231
2020-05-06 14:45:00	0.022559	0.022567	0.022545	0.022566	163.9938
2020-05-06 14:50:00	0.022553	0.022573	0.022545	0.022563	97.2218
2020-05-06 14:55:00	0.022537	0.022542	0.022524	0.022540	8.9356
2020-05-06 15:00:00	0.022546	0.022551	0.022519	0.022533	19.8206

Lo super interesante para los que quieren ver temas de arbitrajes, o arbitrajes triangulares, es que en crypto es todo en tiempo real, y en esta API tenés 12 mercados simultáneos para cruzar y hasta 900 pares en algunos mercados

## Reconocimiento de patrones

Bueno, para la parte de análisis técnico esta API tiene todos los mismos indicadores que la API de AlphaVantage, no los vamos a ver acá porque ya lo vimos en la otra API

Otras cosas interesantes que tienen por ejemplo son los reconocimientos de patrones, ellos mismos aclaran que está en Beta, aparentemente el reconocimiento es vía machine learning, yo estuve testeando y falla un poco pero reconoce bastantes cosas, imagino que lo irán mejorando con el tiempo, es con precios de cierre, veamos en crudo el JSON que tira el response:

```
url = 'https://finnhub.io/api/v1/scan/pattern'
p = {'token': TOKEN, 'symbol' : 'AAPL', 'resolution':'D'}
r = requests.get(url, params = p)
data = r.json()['points']
data[0]
```

```
{
    "aprice": 256.29,
    "atime": 1575383400,
    "bprice": 317.57,
    "btime": 1579012200,
    "cprice": 304.88,
    "ctime": 1580135400,
    "dprice": 327.85,
    "dtime": 1580308200,
    "entry": 311.6286,
    "entry_date": 1580740200,
    "intersect_price": 392.6205681229509,
    "intersect_time": 1588167000,
    "mature": 1,
    "patternname": "Wedge",
    "patterntype": "bearish",
    "profit1": 250.3486,
    "profit2": 0,
    "sortTime": 1580308200,
    "status": "complete",
    "stoploss": 328.1778,
    "symbol": "AAPL.US",
    "terminal": 0
}
```

Como ven viene con demasiada info que confunde mas de lo que informa, la verdad lo que nos interesa son más que nada los puntos que forman el patrón (precio/fecha) y el nombre del patrón, y a su vez vemos que las fechas las informa en timestamps que mucho no nos dice. Así que limpiemos un poco esa data para nuestra implementación:

```

def patterns(symbol, interval):
    from datetime import datetime as dt
    url = 'https://finnhub.io/api/v1/scan/pattern'
    p = {'token': TOKEN, 'symbol' : symbol, 'resolution':interval}
    r = requests.get(url, params = p)
    js = r.json()['points']
    keys = ['atime','btime','ctime','dtime','etime']

    # Recorremos y convertimos los timestamps en fecha legible
    for i in range(5):
        for key in keys:
            try:
                if js[i][key]==0:
                    del js[i][key]
                    keyP = key.replace('time','price')
                    del js[i][keyP]
                js[i][key] = dt.fromtimestamp(js[i][key]).strftime('%Y-%m-%d')
            except:
                pass

    df = pd.DataFrame(js)
    df2 = pd.concat([df.atime,df.btime,df.ctime,df.dtime],axis=1)
    df2['patron'] = df['patternname']
    df2['type'] = df['patterntype']
    df2['status'] = df['status']
    df2.set_index('patron',inplace=True)
    return df2

patterns('PAM','D')

```

		atime	btime	ctime	dtime	type	status
	patron						
	Triangle	2020-03-23	2020-03-26	2020-04-27	2020-04-29	bullish	complete
	Flag	2020-03-23	2020-03-23	2020-03-26	2020-03-26	bearish	complete
	two black gapping	2020-03-04		NaN	2020-03-10	bearish	complete
	Double Bottom	2019-10-29	2019-11-05	2019-11-15	NaN	bullish	failed
	channel	2019-09-25	2019-09-25	2019-12-09	2019-12-09	bearish	complete

Bueno, ahora si se entiende mucho mejor

Ojo, al momento de escribir esto, este endpoint está en beta, puede que esté medio inestable un tiempo pero promete bastante por eso no quise dejar de ponerlo aunque debo decir que al momento no está andando muy bien el reconocimiento, calculo que irá mejorando

## Soportes y Resistencias

Esto si que puede ser realmente útil, es sencillo se entiende solo:

```
def supRes(symbol, interval):
    url = 'https://finnhub.io/api/v1/scan/support-resistance'
    p = {'token': TOKEN, 'symbol' : symbol, 'resolution':interval}
    r = requests.get(url, params = p)
    js = r.json()['levels']
    return js

supRes('GGAL','D')

[5.65999847412109,
 8.189995803833,
 12.47000267028809,
 14.60999656677246,
 17.70000762939453]
```

Lo que reconoce son precios donde hay soportes o resistencias fuertes, donde el precio se trabaa el precio y obviamente los que están por debajo del precio actual son soportes y los que están por encima son resistencias

## Resumen de indicadores de AT

Este es el resumen que usé de ejemplo cuando mostré lo que es un JSON, vamos a ver el endpoint de la API y como implementar una función para obtener esta data ordenada

```
def resumenAT(symbol, interval):
    url = 'https://finnhub.io/api/v1/scan/technical-indicator'
    p = {'token': TOKEN, 'symbol' : symbol, 'resolution':interval}
    r = requests.get(url, params = p)
    js = r.json()
    return js

resumen = resumenAT('GGAL','D')
print(json.dumps(resumen, indent=4))

{
    "technicalAnalysis": {
        "count": {
            "buy": 6,
            "neutral": 10,
            "sell": 1
        },
        "signal": "neutral"
    },
    "trend": {
        "adx": 14.952580051235499,
        "trending": false
    }
}
```

Lo que nos muestra es que hay 6 indicadores que muestran compra, 10 neutrales, uno que indica venta, y en resumen la señal es neutral, mientras que la tendencia dada por el valor de ADX=14.95 se considera sin tendencia concreta

## Indicadores técnicos

Luego la API brinda todos los indicadores técnicos que brinda AlphaVantage, de hecho es una integración a su librería pero al día de hoy no está andando del todo bien así que salteo esta parte de esta API

## Indicadores económicos

Bueno, atentí economistas, esta base de datos es inmensa, vamos a mostrar primero todos los "códigos" de datos económicos que tiene la base de datos, si observan bien son mas de 8000 series.. está bien contando todos los países

```
def ecoCodes():
    url = 'https://finnhub.io/api/v1/economic/code'
    p = {'token': TOKEN}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df
```

```
ecoCodes()
```

	code	country		name	unit
0	MA-ABW-667984	Aruba		Balance Of Trade	AFI Million
1	MA-ABW-678073	Aruba	Consumer Price Index Cpi	points 2019=100	
2	MA-ABW-67807367	Aruba		Inflation Rate	percent 2006=100
3	MA-ABW-678482	Aruba		Corporate Tax Rate	percent
4	MA-ABW-6988867976	Aruba		Exports	AFI Million
...	...	...		...	...
8240	MA-ZWE-80738482	Zimbabwe	Personal Income Tax Rate		percent
8241	MA-ZWE-807980	Zimbabwe		Population	million
8242	MA-ZWE-838482	Zimbabwe		Sales Tax Rate	percent
8243	MA-ZWE-84798582	Zimbabwe		Tourist Arrivals	
8244	MA-ZWE-857882	Zimbabwe		Unemployment Rate	percent

8245 rows × 4 columns

Mas razonable es buscar series por país, obviamente

```
def ecoCodesByCountry(country):
    url = 'https://finnhub.io/api/v1/economic/code'
    p = {'token': TOKEN}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df.loc[df.country==country]
```

```
ecoCodesByCountry('Argentina')
```

	code	country	name	unit
230	MA-ARG-667984	Argentina	Balance of Trade	USD Million
231	MA-ARG-6765	Argentina	Current Account	USD Million
232	MA-ARG-6765716880	Argentina	Current Account to GDP	percent
233	MA-ARG-67658283	Argentina	Domestic Car Sales	Thousand Volume, NSA
234	MA-ARG-6770	Argentina	Capital Flows	USD Million Current Prices, NSA
...	...	...	...	...
287	MA-ARG-847984	Argentina	Terms of Trade	points NSA, 2004=100
288	MA-ARG-84798582	Argentina	Tourist Arrivals	
289	MA-ARG-857880	Argentina	Unemployed Persons	Thousand Volume, NSA
290	MA-ARG-857882	Argentina	Unemployment Rate	percent
291	MA-ARG-87657169	Argentina	Average Monthly Wage for registered employees ...	ARS/Month NSA

62 rows × 4 columns

```
list(ecoCodesByCountry('Argentina')['name'])
```

```
['Balance of Trade',
 'Current Account',
 'Current Account to GDP',
 'Domestic Car Sales',
 'Capital Flows',
 'Competitiveness Index',
 'Consumer Confidence',
 'Consumer Price Index (CPI)',
 'Inflation Rate',
 'Inflation Rate MoM',
 'Consumer Spending',
 'Corporate Tax Rate',
 'Capacity Utilization',
 'Argentinean Peso',
 'Changes in Inventories',
 'Total External Debt',
 'Employed Persons',
 'Export Prices',
 'Exports',
 'Foreign Direct Investment',
 'GDP',
 'GDP Annual Growth Rate',
```

-- Continua en la página siguiente -

```
'Government Budget',
'Government Budget Value',
'GDP Constant Prices',
'GDP Deflator',
'Government Debt to GDP',
'Gross Fixed Capital Formation',
'GDP Growth Rate',
'Gold Reserves',
'GDP per capita',
'GDP per capita PPP',
'Government Spending',
'Permits for Floor Space',
'up to 15 Days Interbank Rate',
'Import Prices',
'Imports',
'Industrial Production',
'35-Day Lebac Rate',
'Leading Economic Index',
'Money Supply M0',
'Money Supply M1',
'Money Supply M2',
'Money Supply M3',
'Wages In Manufacturing Index',
'Minimum Monthly Wage',
'Crude Oil Production',
'Personal Income Tax Rate',
'Population',
'Producer Prices',
'Remittances',
'Retail Sales MoM',
'Retail Sales YoY',
'Social Security Rate',
'Social Security Rate For Companies',
'Social Security Rate For Employees',
'Sales Tax Rate',
'Terms of Trade',
'Tourist Arrivals',
'Unemployed Persons',
'Unemployment Rate',
'Average Monthly Wage for registered employees in the private sector']
```

Fíjense lo inmenso que es esto, de Argentina solo me trae 62 endpoints diferentes a los que luego podré consultar, de Estados Unidos tiene el doble..

Obviamente ante este tipo de cosas inmensas, por ahora que no vimos aun bases de datos, podemos usar el querido excel para exportar esto y chusmearlo de una forma mas cómoda que en una consola de python

```
ecoArgentina = ecoCodesByCountry('Argentina')
ecoArgentina.to_excel('ecoArgentina.xlsx')
```

Ejecutado esto, tenemos nuestro excel para chusmear tranca lo que podemos consultar

## Implementación de función para datos económicos

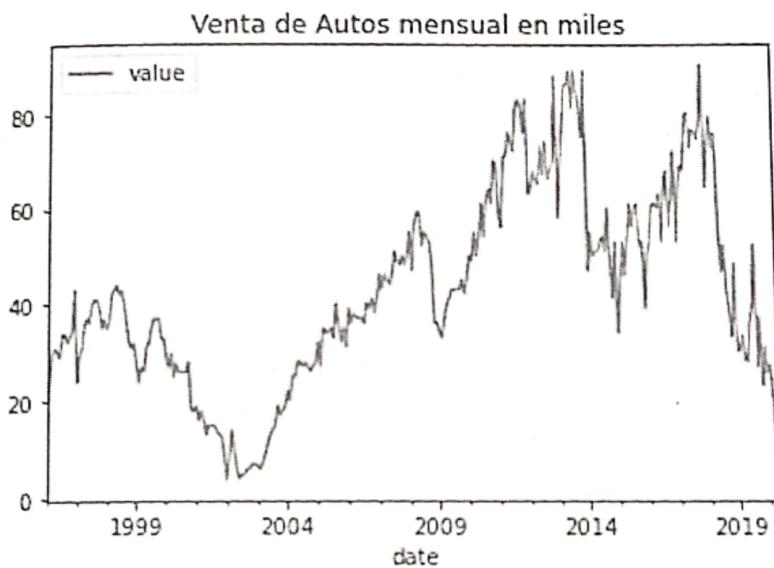
```
def economic(code):
    url = 'https://finnhub.io/api/v1/economic'
    p = {'token': TOKEN, 'code':code}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df
```

Vamos a usar un código cualquiera por ejemplo: 'MA-ARG-67658283' que es la venta de autos mensuales, y lo vamos a graficar

Para ello, simplemente debemos pasar a datetime la columna con las fechas que esta como un string así el eje X me queda bien ordenado

```
data = economic('MA-ARG-67658283')
data['date'] = pd.to_datetime(data['date'])
data.set_index('date', inplace=True)

#Graficamos la serie
data.plot(title='Venta de Autos mensual en miles')
```



Bien podríamos meter dentro de la función original este tema del pasaje de la columna date a datetime y mandarlo como índice del dataframe, pero se los dejo también a ustedes

Otra fuente de datos muy interesante de esta API es el calendario económico que nos informa de los hechos venideros de relevancia

## Calendario económico

El endpoint es el siguiente:

```
def ecoCalendar():
    url = 'https://finnhub.io/api/v1/calendar/economic'
    p = {'token': TOKEN}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['economicCalendar']['result'])
    return df
data = ecoCalendar()
data.columns
```

```
['actual', 'comment', 'country', 'currency', 'date', 'forecast', 'id',
'importance', 'indicator', 'link', 'period', 'previous', 'scale', 'source',
'title', 'unit']
```

Mostramos una sola columna porque la cantidad de datos es muy grande y la tabla se hace inmensa:

```
data.loc[0]
```

```
actual -7.3
comment The coincident index consists of the following...
country JP
currency JPY
date 2020-06-05T05:00:00.000Z
forecast NaN
id 13056164 2020-06-05 05:00:00 +0000 UTC JPY Jap...
importance 0
indicator Coincident Index*
period Apr. 2020
previous -4.9
source Cabinet Off
title Japan-Leading indicator-Coincident Index*
unit Index
```

También puedo agrupar por país:

```
data.groupby('country').country.count()
```

country	
AU	7
CA	12
CH	3
CN	17
ERL	8
GB	20
JP	31
NZ	10
US	132

# Ejercicios

1- Usando los pares y la función que lista los ejecutivos, ver la comparativa de proporción de hombres y mujeres en los directorios de los bancos argentinos que cotizan en bolsa. Empaquetar las instrucciones en una función así con una línea de código podemos comparar otros sectores además del bancario y en otros países

2- Armar una función que me devuelva una tabla con la cantidad de IPOs del año y el monto en millones y los millones promedio por IPO del mes

3- Usando el endpoint de la API de recomendación de analistas y las funciones usadas en la explicación, armar una función que grafique la comparativa de los índices base=100 del inicio de "n" acciones, pasándolas los tickers como argumento de la función como un listado de tickers

4 - Implementar una función que tome como argumento el ticker y una cierta cantidad de días hacia atrás y grafique desde esa cantidad de días la serie de precios como una nube entre los máximos y mínimos de cada rueda y marque las resistencias y soportes que devuelve la API finnhub. Indicar también en el gráfico el valor de las resistencias y soportes y del ultimo precio de cierre. Hacer todo en escala diaria

5- Graficar la serie de desempleo de Argentina, Brasil y México en el mismo gráfico.

6- Hacer un script con sus funciones respectivas para que dado un determinado ticker, por ejemplo "XOM", obtener una matriz de correlación con sus pares

- Usar la funciones de pares de la API
- Que la función permita tomar como parámetro no solo los pares devueltos por la API sino agregar una lista personalizada de tickers que queramos agregar en la matriz de correlación
- Que admita desde/hasta como parámetros tipo string

# Respuestas

```
#-----#
# Rta Ejercicio 1 #
#-----#

import requests, pandas as pd
TOKEN = 'bqoji3nrh5rced4gaukg'

def getEx(symbol):
    url = 'https://finnhub.io/api/v1/stock/executive'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['executive'])
    return df

def getPeers(symbol):
    url = 'https://finnhub.io/api/v1/stock/peers'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    return js

def compareExSex(symbol):
    tickers = getPeers(symbol)
    df = pd.DataFrame()
    for ticker in tickers:
        data = getEx(ticker)
        res = data.groupby('sex').sex.count()
        res['Female/Tot'] = str(round(res.female/(res.female+res.male)*100,1))+'%'
        res['Male/Tot'] = str(round(res.male/(res.female+res.male)*100,1))+'%'
        df = pd.concat([df,res], axis=1)
    df.columns=tickers
    f = df.sum(axis=1).female
    m = df.sum(axis=1).male
    female_total = round (100* f / (f+m) ,2)
    male_total = round (100* m / (f+m) ,2)
    return df,female_total, male_total

compareExSex('GGAL.BA')
```

```
(      BMA.BA GGAL.BA BRIO.BA BBAR.BA BPAT.BA SUPV.BA BHIP.BA
sex
female      11      1      8      6      8      8      4
male       65     22     35     58     56     21     66
Female/Tot  14.5%   4.3%  18.6%   9.4%  12.5%  27.6%  5.7%
Male/Tot    85.5%  95.7%  81.4%  90.6%  87.5%  72.4%  94.3%,
12.47,
87.53)
```

```
compareExSex('JPM')
```

```
(      JPM     BAC     WFC      C      USB     FINN     HFBA     CIBH     HABK
sex
female      19     18     26     19     19      1      1      3      5
male       72     79     87    106     65      8      8     20     23
Female/Tot  20.9%  18.6%  23.0%  15.2%  22.6%  11.1%  11.1%  13.0%  17.9%
Male/Tot    79.1%  81.4%  77.0%  84.8%  77.4%  88.9%  88.9%  87.0%  82.1%,
19.17,
80.83)
```

```

#-----#
# Rta Ejercicio 2 #
#-----#
def contarIPOs(año):
    import datetime
    fromDate=datetime.date(año,1,1)
    toDate=datetime.date(año,12,31)
    dataJS, dataDF = IPOs(fromDate,toDate)
    dataDF['date'] = pd.to_datetime(dataDF.date, format='%Y-%m-%d')
    dataDF.set_index('date',inplace=True)
    iposCount = dataDF.symbol.groupby(dataDF.index.month).count()
    iposMln = dataDF.totalSharesValue.groupby(dataDF.index.month).sum()/1000000
    iposMedia = iposMln/iposCount
    agrupados = pd.DataFrame([iposCount,iposMln,iposMedia]).transpose().round()
    agrupados.columns = ['Cantidad','Millones','Millones/IPO']
    agrupados.index.name = 'mes'
    return agrupados

data = contarIPOs(2020)
data

```

	Cantidad	Millones	Millones/IPO
mes			
1	22.0	3,341.0	152.0
2	35.0	6,273.0	179.0
3	13.0	3,171.0	244.0
4	20.0	3,397.0	170.0
5	6.0	708.0	118.0

```

#-----#
# Rta Ejercicio 3 #
#-----#

import pandas as pd, requests, matplotlib.pyplot as plt

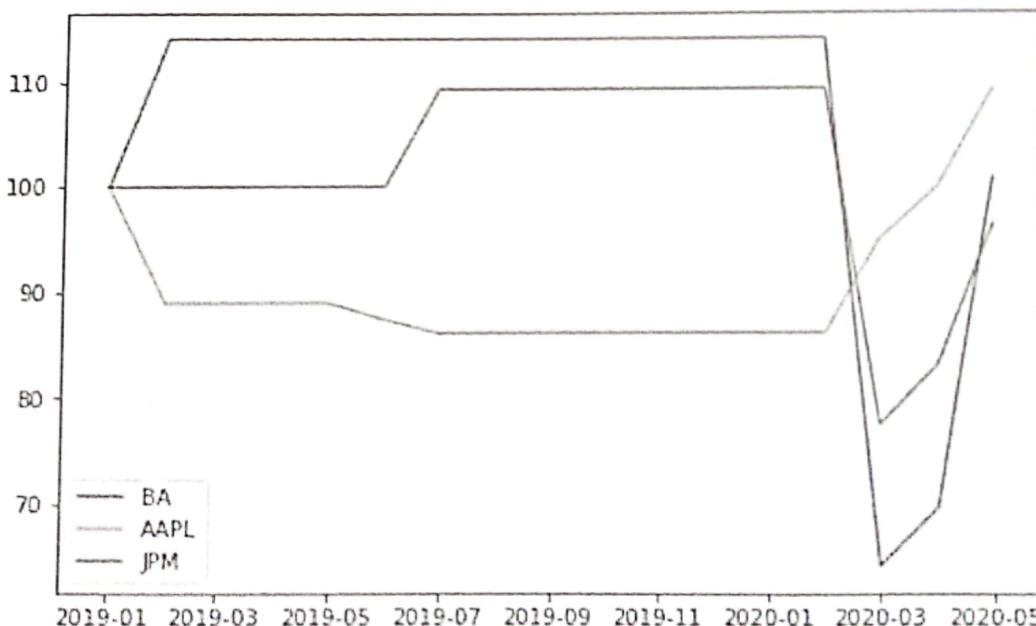
def recommendation(symbol):
    url = 'https://finnhub.io/api/v1/stock/recommendation'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df['period'] = pd.to_datetime(df.period)
    df.set_index('period',inplace=True)
    df.drop(['symbol'],axis=1,inplace=True)
    return df

def puntaje(recomendaciones):
    pesos = [2,1,-1,3,-2]
    puntaje = recomendaciones.mul(pesos, axis=1).sum(axis=1)
    conteo = recomendaciones.sum(axis=1)
    puntajeMedio = puntaje/conteo
    resumen = pd.concat([puntaje,conteo,puntajeMedio],axis=1)
    resumen.columns = ['Puntaje','Conteo','PuntajeMedio']
    resumen = resumen.sort_values('period', ascending=True)
    inicial = float(resumen[0:1]['PuntajeMedio'])
    resumen['Idx100'] = 100*resumen['PuntajeMedio']/inicial
    return resumen

def compararRec(tickers):
    fig, ax = plt.subplots(figsize=(8,5))
    for i in range(len(tickers)):
        serie = puntaje(recommendation(tickers[i]))['Idx100']
        ax.plot(serie, label=tickers[i])
    plt.legend()
    plt.show()

tickers=['BA','AAPL','JPM']
compararRec(tickers)

```



```

#-----#
# Rta Ejercicio 4 #
#-----#
def sopRes(symbol, days=200):
    url = 'https://finnhub.io/api/v1/scan/support-resistance'
    p = {'token': TOKEN, 'symbol' : symbol, 'resolution':'D'}
    r = requests.get(url, params = p)
    js = r.json()['levels']

    import matplotlib.pyplot as plt
    from datetime import datetime as dt, timedelta as delta

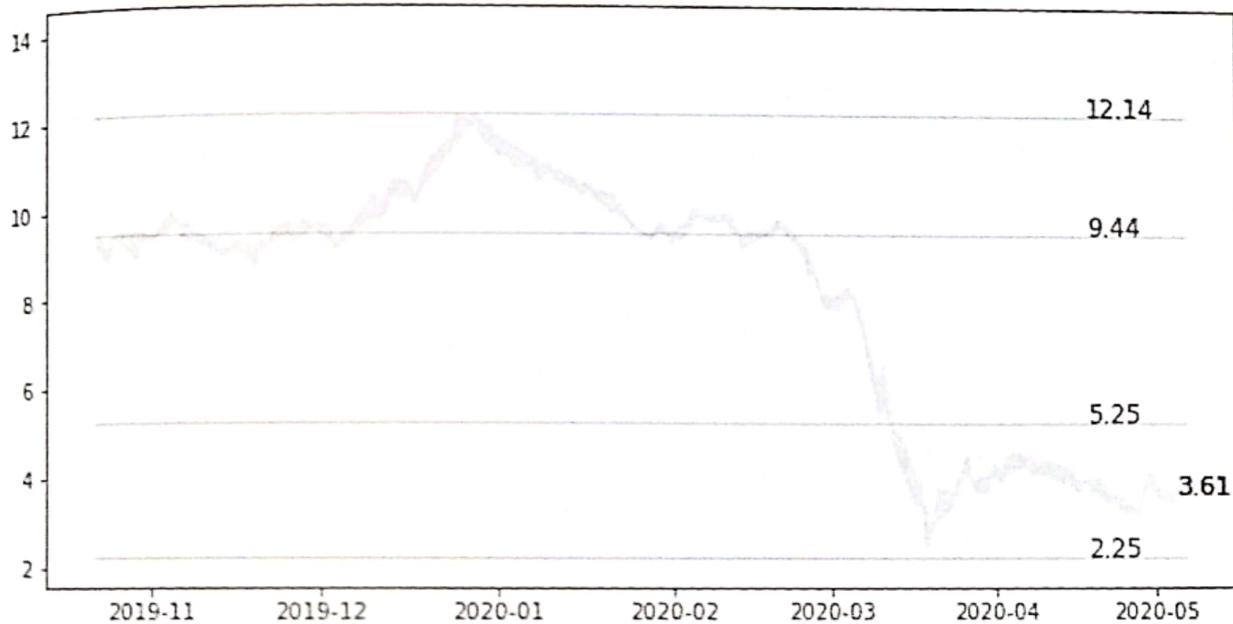
    desde = (dt.now()-delta(days=days)).strftime('%Y-%m-%d')
    hasta = dt.now().strftime('%Y-%m-%d')

    fig, ax = plt.subplots(figsize=(10,5))
    serie = getHistoryDates(symbol,'D',desde,hasta,'true')
    last = serie[-1:].Close
    plt.fill_between(serie.index, serie.High, serie.Low, alpha=0.2)

    for i in range(len(js)):
        lines = [js[i] for j in range(len(serie))]
        ax.plot(serie.index,lines,'gray', lw=1, alpha=0.5)
        plt.text(dt.now()-delta(days=20),js[i], round(js[i],2), fontsize=12)
    ax.set_ylim([serie.Low.min()*0.7,serie.High.max()*1.2])
    plt.text(dt.now()-delta(days=3),float(last), float(last), fontsize=12)

sopRes(symbol='YPF', days=200)

```



```

#-----#
# Rta Ejercicio 5  #
#-----#

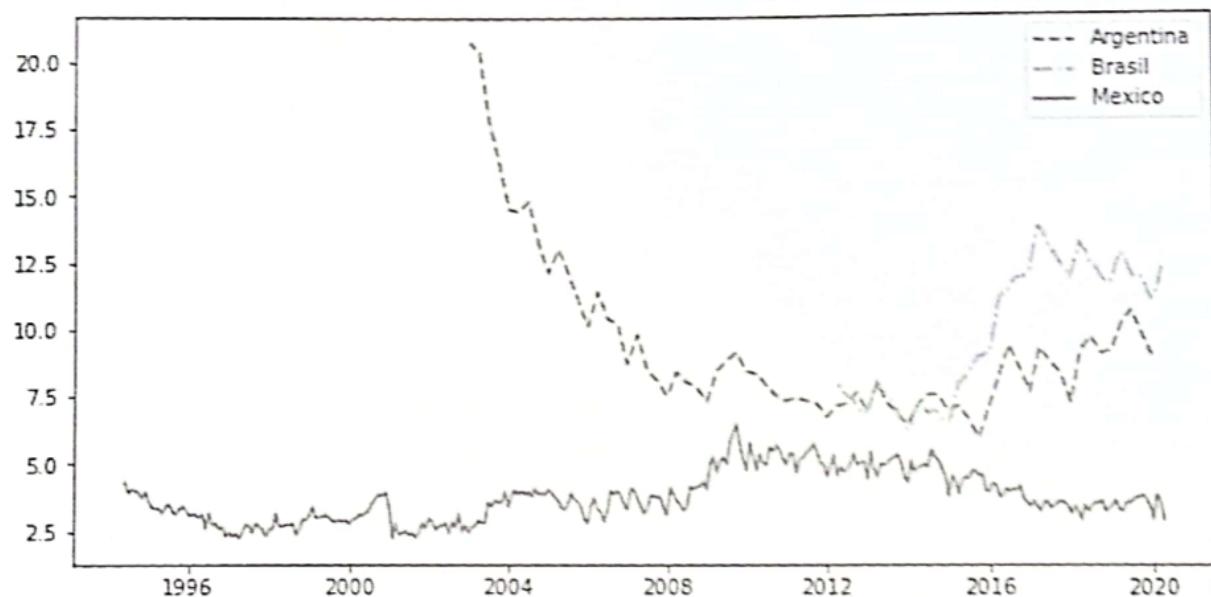
def economic(code):
    url = 'https://finnhub.io/api/v1/economic'
    p = {'token': TOKEN, 'code':code}
    r = requests.get(url, params = p)
    js = r.json()
    data = pd.DataFrame(js)
    data['date'] = pd.to_datetime(data['date'])
    data.set_index('date', inplace=True)
    return data

import matplotlib.pyplot as plt
import pandas as pd
import requests

dataAr = economic('MA-ARG-857882')
dataBr = economic('MA-BRA-857882')
dataMx = economic('MA-MEX-857882')

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(dataAr, label='Argentina', ls = '--')
ax.plot(dataBr, label='Brasil', ls='-.')
ax.plot(dataMx, label='Mexico', ls='--')
plt.legend()
plt.show()

```



```

#-----#
# Rta Ejercicio 6  #
#-----#

import requests, pandas as pd
TOKEN = 'bqoji3nrh5rced4gaukg'

def getPeers(symbol):
    url = 'https://finnhub.io/api/v1/stock/peers'
    p = {'token': TOKEN, 'symbol' : symbol}
    r = requests.get(url, params = p)
    js = r.json()
    return js

def getHistoryDates(symbol, interval, fromS,toS, adj):
    import datetime
    fromDT = datetime.datetime.strptime(fromS, '%Y-%m-%d')
    fromTS = int(datetime.datetime.timestamp(fromDT))
    toDT = datetime.datetime.strptime(toS, '%Y-%m-%d')
    toTS = int(datetime.datetime.timestamp(toDT))
    url = 'https://finnhub.io/api/v1/stock/candle'
    p = {'token': TOKEN, 'symbol' : symbol, 'resolution':interval,
          'from':fromTS, 'to':toTS,'adjusted':adj}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df['date'] = pd.to_datetime(df['t'], unit='s')
    df.set_index('date', inplace=True)
    df.drop(['s','t'],axis=1, inplace=True)
    df.columns = ['Close','High','Low','Open','Volume']
    return df

def verCorr(symbol, lista, desde, hasta):
    pares = getPeers(symbol)
    pares = pares + lista
    matriz = pd.DataFrame()
    for ticker in pares:
        data = getHistoryDates(ticker, 'D', desde, hasta, 'true')
        data[ticker] = data.Close.pct_change()
        matriz = pd.concat([matriz,data[ticker]], axis=1)
    return matriz.corr().round(2)

matriz = verCorr('XOM', ['YPF','PBR'], '2019-01-01','2019-12-31')
matriz

```

	XOM	CVX	OXY	UNT	YPF	PBR
XOM	1.00	0.75	0.53	0.41	0.30	0.53
CVX	0.75	1.00	0.45	0.34	0.28	0.50
OXY	0.53	0.45	1.00	0.44	0.32	0.45
UNT	0.41	0.34	0.44	1.00	0.23	0.35
YPF	0.30	0.28	0.32	0.23	1.00	0.35
PBR	0.53	0.50	0.45	0.35	0.35	1.00

# API FMP - Análisis Fundamental

Ni bien entramos a su web: <https://fmpcloud.io/> (<https://fmpcloud.io/>) nos encontramos con esta pantalla

Fmp Cloud

Home Documentation Dashboard Subscription

Fmp Cloud

- Realtime and historical stock data
- Financial statements, financial ratios and enterprise value
  - Historical 13F data back to 2001
  - Cusip and CIK mapper ticker symbols
- Stock screener
- FX and cryptocurrency feeds
- Global coverage

Su plan de precios es bastante sencillo, arranca gratis, como todas las APIs que muestro en este libro, pero con la limitación de 250 requests por mes, que a nosotros para testear ideas nos alcanza y sobra

Fmp Cloud

Home Documentation Dashboard Subscription Logout

FmpCloud scales with your application

Everything you need to succeed with stock market data.

Free	Starter	Teams	Business
<ul style="list-style-type: none"><li>Ideal for testing your ideas</li><li>Limited</li><li>Personal use</li><li>Upgrade anytime</li></ul>	<ul style="list-style-type: none"><li>Best for individuals</li><li>Personal use &amp; commercial use</li><li>Premium Data &amp; Add-ons</li><li>Email support</li></ul>	<ul style="list-style-type: none"><li>Best for teams</li><li>Up to 5 Team members</li><li>Personal use &amp; commercial use</li><li>Premium Data &amp; Add-ons</li><li>Email support &amp; chat support</li><li>Priority Support</li></ul>	<ul style="list-style-type: none"><li>Best for companies</li><li>Unlimited Team</li><li>Bloomberg</li><li>Personal use &amp; commercial use</li><li>Premium Data &amp; Add-ons</li><li>Email support &amp; chat support</li><li>Priority Support</li><li>99.9% Uptime SLA</li><li>Multi-Region use</li><li>Custom on-demand endpoint</li></ul>
250 requests	UNLIMITED requests / mo	UNLIMITED requests / mo	UNLIMITED requests / mo
\$0	\$19 mo	\$39 mo	\$99 mo
<a href="#">Get Started</a>	<a href="#">Get Started</a>	<a href="#">Get Started</a>	<a href="#">Get Started</a>

Les resalto los puntos salientes de esta API respecto al resto:

- La mejor API de AF gratuita
- Tiene commodities
- Data de Filings 13F (posiciones de grandes fondos, +1B)
- Posibilidad de Batch-Requests (muchos tickers en un request)
- Data histórica de muchos años atrás

No se cuanto tiempo mas va a durar online pero les paso el dato de la API anterior que fue donde nació este proyecto, sin límites de requests: <https://financialmodelingprep.com/developer/docs/> (<https://financialmodelingprep.com/developer/docs/>)

Cambia el https de base pero tiene casi los mismos endpoints con alguna pequeña diferencia, el tema es que como esta API esta migrando a un modelo de base gratis limitada, es posible que esta API original sin límites de requests no dure mucho tiempo mas, pero por ahora se puede disfrutar de ella

## Datos disponibles

Empecemos viendo los mercados y sus datos disponibles:

- ETFs (424)
- Commodities (28)
- Equity Europa: Euronext (1248)
- Equity Nyse (4380)
- Equity Amex (274)
- Equity Nasdaq (3825)
- Equity Canadá: TSX (1413)
- Indices (56)
- Fondos Mutuales (1504)

Vamos a definir una función que le pase que item quiero y me traiga los intrumentos de ese item

Lo primero que hacemos es asignar a la variable apiKey el key que me de la página al registrarme con cuenta gratuita o paga, en mi caso estoy poniendo un key que saqué a modo de prueba con lo cual obviamente cada uno debería entrar a <https://fmpcloud.io/> (<https://fmpcloud.io/>) y sacar el suyo

```
apikey = '66c44b016256f8b9e507838f32434ece'
```

Hecho eso les pongo aquí los paneles/mercados disponibles, se los dejo en una lista

```
exchanges = ['etfs','commodities','euronext','nyse','amex','nasdaq','tsx','indices','mutual-funds']
```

Luego debemos importar la librería requests como venimos haciendo hasta acá

```
import requests
```

Y en este caso vamos a necesitar también pandas para sacar los resultados como un DataFrame que es mucho más cómodo de tratar que un JSON

```
import pandas as pd
```

Listo, ahora sí puedo armar una función para listar los tickers o instrumentos de cada panel

```
def getList(exchange):
    url = 'https://fmpcloud.io/api/v3/symbol/available-'+exchange
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.drop(['exchangeShortName'], axis=1, inplace=True)
    return df
getList('commodities')
```

	symbol	name	currency	stockExchange
0	CLUSD	Crude Oil Jul 20	USD	NY Mercantile
1	ZIUSD	Silver 5000 oz. May 20	USD	COMEX
2	HOUSD	Heating Oil Jul 20	USD	NY Mercantile
3	B0USD	Mont Belvieu LDH Propane (OPIS)	USD	NY Mercantile
4	SIUSD	Silver Jul 20	USD	COMEX
5	PLUSD	Platinum Jul 20	USD	NY Mercantile
6	CTUSX	Cotton Oct 20	USX	NYBOT
7	NGUSD	Natural Gas Jul 20	USD	NY Mercantile
8	BZUSD	Brent Crude Oil Last Day Financ	USD	NY Mercantile
9	LHUSX	Lean Hogs Jul 20	USX	CME
10	PAUSD	Palladium Sep 20	USD	NY Mercantile
11	FCUSX	Feeder Cattle Aug 20	USX	CME
12	CCUSD	Cocoa Sep 20	USD	NYBOT
13	OJUSX	Orange Juice Sep 20	USX	NYBOT
14	KCUSX	Coffee Sep 20	USX	NYBOT
15	SMUSD	Soybean Meal Aug 20	USD	CBOT
16	BOUSX	Soybean Oil Aug 20	USX	CBOT
17	CUSX	Corn Sep 20	USX	CBOT
18	RBUSD	RBOB Gasoline Jul 20	USD	NY Mercantile
19	SUSX	Soybeans Aug 20	USX	CBOT

20	HGUSD	Copper May 20	USD	COMEX
21	GCUSD	Gold Aug 20	USD	COMEX
22	LCUSX	Live Cattle Aug 20	USX	CME
23	KWUSX	KC HRW Wheat Futures,Sep-2020,C	USX	CBOT
24	ZGUSD	Gold 100 oz. Apr 20	USD	COMEX
25	OUSX	Oats Sep 20	USX	CBOT
26	LBUSD	Lumber Sep 20	USD	CME
27	SBUSX	Sugar #11 Jul 20	USX	NYBOT
28	RRUSD	Rough Rice Sep 20	USD	CBOT

## Paneles de precios en tiempo real

Antes de entrar a lo que son los batch requests, vamos haciendo un proxy, supongamos que necesito hacer un barrido a todo el panel para ver cual activo se movió mas de un "X"% por decir algo, hasta ahora lo que vimos son requests por un precio concreto de un activo o ticker pero no vimos requests de muchos tickers al mismo tiempo

Bueno de eso se tratan los batch requests, no todas las APIs lo permiten porque el peso de la info es grande y consume mas ancho de banda pero como anticipamos esta API si que lo permite así que lo vamos a aprovechar

En este caso vamos a consultar paneles de precios enteros, no es un batch requests exactamente porque no mandamos una lista de tickers sino que pedimos en panel pero es algo muy similar, veamos:

```
paneles = ['etf','commodity','euronext','nyse','amex','nasdaq','tsx','index','mutual_fund']
```

Esos son los paneles disponibles como vimos antes, y a continuación les dejo la función para llamar a la API y pedir esos paneles.

Como verán tiene muchas columnas y no me van a entrar por lo que imprimo solo el precio y mando el listado de columnas aparte para que vean todo lo que podemos traer en un solo request

```
def getPanel(exchange):
    url = 'https://fmpcloud.io/api/v3/quotes/' + exchange
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('symbol', inplace=True)
    return df

data = getPanel('commodity')
print(data.columns, data['price'])
```

```
[ 'name', 'price', 'changesPercentage', 'change', 'dayLow', 'dayHigh', 'yearHigh',
  'yearLow', 'marketCap', 'priceAvg50', 'priceAvg200', 'volume', 'avgVolume',
  'exchange', 'open', 'previousClose', 'eps', 'pe', 'earningsAnnouncement',
  'sharesOutstanding', 'timestamp' ]
```

Vuelco la data de salida en dos columnas para aprovechar mejor el espacio del libro aunque obviamente el intérprete de python la devuelve en una sola 😊

Bueno como verás mandé a imprimir solo los precios, pero podría haber sacado un panel multicolumna o una tabla combinando este panel con el panel informativo anterior (el de la función getList para poner por ejemplo el mercado donde cotiza o el nombre del commodity)

symbol

ZGUSD	1694.3000
LCUSX	100.8000
CTUSX	58.1600
NGUSD	1.7210
LHUSX	60.2500
FCUSX	134.8000
B0USD	0.2175
HGUSD	2.3895
CLUSD	31.6900
RRUSD	16.4000
HOUSD	0.9445
CCUSD	2344.0000
SIUSD	17.8000
BZUSD	33.8600
SBUSX	10.7900
OJUSX	129.8500
KCUSX	102.1500
GCUSD	1713.0000
ZIUSD	15.1400
LBUSD	359.7000
SUSX	848.5000
PAUSD	1985.1000
RBUSD	0.9707
SMUSD	282.9000
BOUSX	27.4400
CUSX	321.0000
KWUSX	451.5000
OUSX	329.0000
PLUSD	876.4000

Como ven este request es potentísimo, tiene no solo los precios de todo un panel entero, sino que además me da el rango diario y anual, unas SMA de 50 y 200, volumen, volumen medio, cierre anterior, mktCap, eps, P/E, etc

Obviamente que estos últimos datos (mktCap, P/E etc) al traer lista de commodities o pares de Forex no aplican y vendrán vacíos pero en el panel del NySE o Euronext es impresionante la data que trae en una sola petición http

Está claro una vez que me descargué todo un panel, puedo acceder a cualquier instrumento de ese panel

```
nasdaq = getPanel('nasdaq')
```

```
accion = nasdaq.loc['MELI']  
accion
```

```
name           MercadoLibre, Inc.  
price          820.83  
changesPercentage -1.6  
change         -13.34  
dayLow         784.85  
dayHigh        836.285  
yearHigh       864.05  
yearLow        422.22  
marketCap      4.08035e+10  
priceAvg50    668.564  
priceAvg200   613.129  
volume         647723  
avgVolume     714598  
exchange       NASDAQ  
open            833.86  
previousClose  834.17  
eps             -4.211  
pe              NaN  
earningsAnnouncement 2020-08-01T00:00:00  
sharesOutstanding 4.971e+07  
timestamp      1590643028  
Name: MELI, dtype: object
```

O bien listar una columna cualquiera y sacar una media o cualquier estadística

Calculamos por ejemplo la media y la mediana:

```
nasdaq.changesPercentage.mean()
```

```
1.8794422623723483
```

```
nasdaq.changesPercentage.median()
```

```
0.91
```

Está claro una vez que me descargué todo un panel, puedo acceder a cualquier instrumento de ese panel

```
nasdaq = getPanel('nasdaq')
```

```
accion = nasdaq.loc['MELI']  
accion
```

```
name           MercadoLibre, Inc.  
price          820.83  
changesPercentage -1.6  
change         -13.34  
dayLow          784.85  
dayHigh          836.285  
yearHigh          864.05  
yearLow          422.22  
marketCap        4.08035e+10  
priceAvg50        668.564  
priceAvg200        613.129  
volume            647723  
avgVolume          714598  
exchange           NASDAQ  
open              833.86  
previousClose        834.17  
eps                -4.211  
pe                  NaN  
earningsAnnouncement 2020-08-01T00:00:00  
sharesOutstanding      4.971e+07  
timestamp          1590643028  
Name: MELI, dtype: object
```

O bien listar una columna cualquiera y sacar una media o cualquier estadística

Calculamos por ejemplo la media y la mediana:

```
nasdaq.changesPercentage.mean()
```

```
1.8794422623723483
```

```
nasdaq.changesPercentage.median()
```

```
0.91
```

## Cotizaciones intradiarias

Arrancamos con la implementación de una función que traiga las intradiarias, a saber lo que le implemento además de la query es

- Pasarlo a un datafrme
- Setearle como índice el dato de la fecha
- Transformar en objeto datetime ese índice
- Ordenar en forma ascendente

Siempre conviene realizar estos pequeños pasos en la implementación y acostumbrarse a seguir ese estándar de buenas prácticas y sobre todo ser homogéneo y hacerlo siempre así cada vez que agarremos una función que codeamos hace tiempo no nos tenemos que poner a arreglarle esas cositas

El hecho de tener como índice una fecha es ideal para graficar y para que en el orden y resampleo de las series no me tire ningún error inesperado

Bueno, les dejo la implementación de esta llamada a intradiarios

```
# Los intervalos posibles son: 1min, 5min, 15min, 30min, 1hour
def getIntra(symbol, period='1min'):
    url = 'https://fmpcloud.io/api/v3/historical-
    chart/' + period + '/' + symbol
    p = {'apikey': apikey}
    r = requests.get(url, params=p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('date', inplace=True)
    df.index = pd.to_datetime(df.index)
    df.sort_values('date', ascending=True, inplace=True)
    return df

data = getIntra('AAPL')
print(data)
```

	open	low	high	close	volume
date					
2020-05-07 15:52:00	303.5600	303.2873	303.610	303.2873	25116815
2020-05-07 15:53:00	303.2873	303.0400	303.360	303.1600	25230850
2020-05-07 15:54:00	303.1600	303.0100	303.160	303.0900	25355960
2020-05-07 15:55:00	303.0900	303.0315	303.300	303.3000	25530617
2020-05-07 15:56:00	303.3000	303.1900	303.360	303.1900	25681506
...	...	...	...	...	...
2020-06-05 15:56:00	330.5800	330.4900	330.800	330.7600	29072022
2020-06-05 15:57:00	330.7600	330.4400	330.806	330.4400	29271045
2020-06-05 15:58:00	330.4400	330.3800	330.770	330.5800	29421401
2020-06-05 15:59:00	330.5800	330.3600	330.960	330.9600	29663393
2020-06-05 16:00:00	330.9600	330.9001	331.560	331.5200	30119024

800 rows × 5 columns

Así como usamos un ticker cualquiera del nasdaq, podemos usar un commodity o cualquier instrumento de cualquiera de los otros panales disponibles

Veamos por caso el platino:

```
data = getIntra('PLUSD')
print(data.tail())
```

date	open	low	high	close	volume
2020-05-28 01:50:00	878.3	878.3	878.3	878.3	865
2020-05-28 01:51:00	878.3	878.3	878.9	878.9	867
2020-05-28 01:52:00	878.9	878.9	878.9	878.9	867
2020-05-28 01:53:00	878.9	878.9	879.4	879.4	868
2020-05-28 01:54:00	879.4	878.8	879.4	878.8	869

También podemos traer de un índice del panel de índices, veamos el Merval:

```
data = getIntra('^MERV')
print(data.tail())
```

date	open	low	high	close	volume
2020-06-05 15:56:00	38390.84	38390.84	38390.84	38390.84	0
2020-06-05 15:57:00	38390.84	38390.84	38390.84	38390.84	0
2020-06-05 15:58:00	38390.84	38390.84	38390.84	38390.84	0
2020-06-05 15:59:00	38390.84	38390.84	38390.84	38390.84	0
2020-06-05 16:00:00	38390.84	38390.84	38390.84	38390.84	0

## Cotizaciones históricas

El endpoint de las series históricas de esta API está bastante mejor que en otros casos, ya que me trae no solo el OHLC sino que me trae el precio ajustado y el precio ponderado por volumen (vwap), la variación diaria y porcentual y un label de fecha en formato amigable a la lectura humana, como contrapartida esto es más lento que otra API ya que trae toda esta info

```
def getIntra(symbol):
    url = 'https://fmpcloud.io/api/v3/historical-price-
full/'+symbol
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['historical'])
    df.set_index('date', inplace=True)
    df.index = pd.to_datetime(df.index)
    df.sort_values('date', ascending=True, inplace=True)
    return df
data = getIntra('AAPL')
print(data.head())
```

date	open	high	low	close	adjClose	volume	\
2015-05-28	131.86	131.95	131.10	131.78	121.45	30733300.0	
2015-05-29	131.23	131.45	129.90	130.28	120.07	50884500.0	
2015-06-01	130.28	131.39	130.05	130.54	120.31	32112800.0	
2015-06-02	129.86	130.66	129.32	129.96	119.77	33667600.0	
2015-06-03	130.66	130.94	129.90	130.12	119.92	30983500.0	

date	unadjustedVolume	change	changePercent	vwap	label	\
2015-05-28	30733300.0	-0.08	-0.061	131.61000	May 28, 15	
2015-05-29	50884500.0	-0.95	-0.724	130.54333	May 29, 15	
2015-06-01	32112800.0	0.26	0.200	130.66000	June 01, 15	
2015-06-02	33667600.0	0.10	0.077	129.98000	June 02, 15	
2015-06-03	30983500.0	-0.54	-0.413	130.32000	June 03, 15	

date	changeOverTime
2015-05-28	-0.00061
2015-05-29	-0.00724
2015-06-01	0.00200
2015-06-02	0.00077
2015-06-03	-0.00413

### Queries más pesadas y con más data

Esta API tiene otra ventaja que es que me permite "elegir" si quiero 5 años de data pesada como en el endpoint anterior o solo quiero los precios pero de 40 años o mas

Es exactamente el mismo endpoint que el anterior pero se agrega el parámetro 'serietype':'line' al diccionario de parámetros que tenía solo el apikey por ahora

```
def getHist(symbol):
    url = 'https://fmpcloud.io/api/v3/historical-price-full/'+symbol
    p = {'apikey': apikey, 'serietype':'line'}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js['historical'])
    df.set_index('date', inplace=True)
    df.index = pd.to_datetime(df.index)
    df.sort_values('date', ascending=True, inplace=True)
    return df

data = getHist('^GSPC')
print(data)
```

date	close
1927-12-30	17.660000
1928-01-03	17.760000
1928-01-04	17.719999
.....	

23210 rows × 1 columns

Como ven en el caso del S&P 500 tiene data desde 1927, así que ya saben si necesitan una serie larga, aquí tienen la mejor API gratuita que encontré hasta el momento para esa query, en equity también por ejemplo para Coca Cola tiene datos desde 1962, para el Merval Argentino desde 1996

## Batch Requests

Se pone interesante esto, con la siguiente función armamos un panel con absolutamente todo el equity de USA a fin del día, el ultimo precio de cada ticker, dato conocido como EOD por "End of day" de la fecha que le pase como argumento

Ya a esta altura dejé de comentar estas cosas, pero les recuerdo que la mejor manera de aprender es probando e investigando, no se queden con esta implementación que hice yo, armen las suyas, vayan a la documentación de la API y vean si se agregó algo o no al endpoint etc.

```
def getHistBatch(date):
    url = 'https://fmpcloud.io/api/v3/batch-request-end-of-day-prices/'
    p = {'apikey': apikey, 'date':date}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('symbol', inplace=True)
    df.date = pd.to_datetime(df.date)
    return df

# Pido todos los precios EOD del 20/5/2020
data = getHistBatch('2020-05-20')

# Muestro solo
# La columna del precio de cierre, primeros 5 datos
# La forma (cantidad de columnas y filas
# Los nombres de las columnas

data.close.head(), data.shape, data.columns
```

```
(symbol
MET      33.16
CLFD     13.50
AVGR     0.35
TVIX    149.51
IWY     100.70

(10149, 7),

['date', 'open', 'low', 'high', 'close', 'adjClose', 'volume'])
```

# Datos de Análisis Fundamental. Perfil básico

Bueno, nos empezamos a meter en el corazón de esta API que son los datos de las empresas que se usan en el análisis fundamental.

Arrancamos consultando el perfil básico de la empresa, esto trae en una sola consulta, datos de precio actual más un par de métricas básicas como el Beta (variación respecto al mercado de referencia) el marketCap, el volumen medio, los últimos dividendos pagados, el DCF (discounted cash flow) y algunos datos más de perfil como la web, el sector e industria etc

Definamos primero la función:

```
# Importamos la librería para hacer los requests http
import requests

# Definimos función que toma de argumento el ticker y me trae el perfil de la empresa
def getProfile(symbol):
    apikey = 'demo'
    url = 'https://fmpcloud.io/api/v3/profile/' + symbol
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    return js
```

```
getProfile('AAPL')
```

```
[{'symbol': 'AAPL',
  'price': 318.25,
  'beta': 1.228499,
  'volAvg': 49183251,
  'mktCap': 1379400420000,
  'lastDiv': 3.08,
  'range': '170.27-327.85',
  'changes': 0.14,
  'companyName': 'Apple Inc.',
  'exchange': 'Nasdaq Global Select',
  'exchangeShortName': 'NASDAQ',
  'industry': 'Computer Hardware',
  'website': 'http://www.apple.com',
  'description': 'Apple Inc designs, manufactures and markets mobile communication and media devices and personal computers, and sells a variety of related software, services, accessories, networking solutions and third-party digital content and applications.',
  'ceo': 'Timothy D. Cook',
  'sector': 'Technology',
  'dcfDiff': 89.92,
  'dcf': 297.11,
  'image': 'https://fmpcloud.io/image-stock/AAPL.jpg'}]
```

Como vemos nos devuelve por ejemplo:

- Beta (Sensibilidad de la acción respecto al movimiento del mercado, cuando es mayor a 1 es mas sensible y Beta menor a 1 es una acción "defensiva")
- volAvg: Volumen Medio en nominales
- mktCap: El valor de la empresa por bolsa
- lastDiv: El último dividendo pagado
- dcf: La valuación por flujo de fondos
- dcf diff: La diferencia entre la valuación por fijo de fondos y el valor por bolsa
- nos devuelve hasta la URL de un logo de la empresa, su pagina web, el nombre del CEO, una descripción básica etc..

## Performance por sector

Este endpoint nos da la data de la variación diaria de cada sector de la economía, que obviamente lo calcula con un promedio de las variaciones de las acciones de cada sector

```
def sectors():
    url = 'https://fmpcloud.io/api/v3/sectors-performance'
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('sector', inplace=True)
    return df
sectors()
```

sector	changesPercentage
Basic Materials	1.8386%
Communication Services	0.5063%
Conglomerates	0.0000%
Consumer Cyclical	1.0135%
Consumer Defensive	0.4246%
Energy	0.8962%
Financial	-0.4138%
Financial Services	0.3486%
Healthcare	0.9898%
Industrial Goods	0.9986%
Industrials	-0.3007%
Real Estate	0.3677%
Services	0.0000%
Technology	0.2884%
Utilities	-0.3043%

## Performance por sector Histórica

Vamos a arrancar definiendo la función para traer los datos  
Ya que estamos le ponemos el índice como fecha datetime así lo graficamos

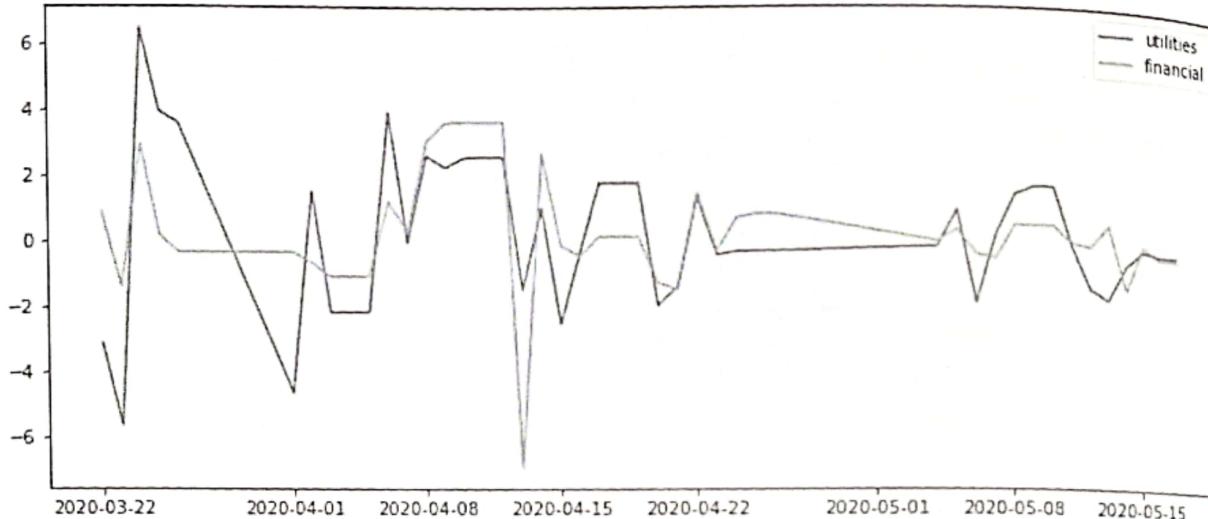
```
def sectorsHist(sector):
    url = 'https://fmpcloud.io/api/v3/historical-sectors-performance'
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('date', inplace=True)
    df.index = pd.to_datetime(df.index)

    return df[sector+'ChangesPercentage']
```

Ahora vamos a graficar

```
utilities = sectorsHist('utilities')
financial = sectorsHist('financial')

import matplotlib.pyplot as plt
plt.figure(figsize=(12,5))
plt.plot(utilities, label='utilities')
plt.plot(financial, label='financial')
plt.legend()
plt.show()
```



## Datos de análisis Fundamental

Bueno, acá viene el punto fuerte de esta API, es verdaderamente interesante todo lo que tiene y bastante prolífica la info, cabe mencionar que en ADRs y activos con balances en otras monedas no tienen info contable, por un lado es menos info pero por otro lado es mejor contar con info pulida y homogénea que contar con series más grandes pero con fallas por problemas de monedas, factores de conversión etc, cosa que vamos a ver es muy común en data de fundamentales

Tengamos en cuenta que conciliar datos de empresas que cotizan en muchos mercados en diferentes monedas y que presentan balances en diferentes monedas y además tienen equivalentes a ADRs o CEDEARs para muchos mercados con sus consiguientes factores de conversión, y ese tipo de cosas hacen que generalmente la data automatizada (la de bajo costo) tenga muchas fallas, por eso hay que ser siempre muy cuidadoso cuando se hace un filtro de data de AF con APIs baratas porque pueden traernos cualquier cosa

Arrancando con los datos de AF, vamos a mencionar que tenemos los siguientes datos:

- **Estados contables**

- Hoja de balances
  - Anual / Anual growth
  - Trimestral / Trimestral growth
- Estado de Resultados
  - Anual / Anual growth
  - Trimestral / Trimestral growth
- Flujo de Fondos
  - Anual / Anual growth
  - Trimestral / Trimestral growth

- **Cálculos**

- Ratios (+50)
  - Anual
  - Trimestral
  - TTM
- Métricas (+50)
  - Anual
  - Trimestral

- **Valuación**

- EV (Con desagregado de cash equivalente y deuda)
  - Anual
  - Trimestral
- DFC: Valuación Flujo de Fondos descontado
  - Anual
  - Trimestral
  - Diaria
- Rating por fundamentales (DFC, ROE, ROA, PER, PB)
- Filings 13F (declaración de posición de grandes holders)

# Hoja de Balance

Primero les muestro la función para traer las hojas de balance de todos los períodos

Les imprimo solo el primer balance de la serie (el más reciente)

```
def balanceSheet(symbol, period):
    url = 'https://fmpcloud.io/api/v3/balance-sheet-
statement/'+symbol
    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

bs = balanceSheet('AAPL', 'annual')
bs.columns
```

```
['date', 'symbol', 'fillingDate', 'acceptedDate', 'period', 'cashAndCashEquivalents',
'shortTermInvestments', 'cashAndShortTermInvestments', 'netReceivables', 'inventory',
'otherCurrentAssets', 'totalCurrentAssets', 'propertyPlantEquipmentNet', 'goodwill',
'intangibleAssets', 'goodwillAndIntangibleAssets', 'longTermInvestments', 'taxAssets',
'otherNonCurrentAssets', 'totalNonCurrentAssets', 'otherAssets', 'totalAssets',
'accountPayables', 'shortTermDebt', 'taxPayables', 'deferredRevenue',
'otherCurrentLiabilities', 'totalCurrentLiabilities', 'longTermDebt',
'deferredRevenueNonCurrent', 'deferredTaxLiabilitiesNonCurrent',
'otherNonCurrentLiabilities', 'totalNonCurrentLiabilities', 'otherLiabilities',
'totalLiabilities', 'commonStock', 'retainedEarnings',
'accumulatedOtherComprehensiveIncomeLoss', 'otherTotalStockholdersEquity',
'totalStockholdersEquity', 'totalLiabilitiesAndStockholdersEquity', 'totalInvestments',
'totalDebt', 'netDebt', 'link', 'finalLink']
```

Es muy interesante que puedan ir al link de las fuentes y chusmeen la base de datos original

```
bs.loc[0]['finalLink']
```

```
'https://www.sec.gov/Archives/edgar/data/320193/000032019319000119/a10-k20199282019.htm'
```

A los que están interesados en el análisis fundamental, recomiendo fuerte empezar a meterse en las fuentes originales de los datos, en este campo del análisis fundamental es muy complicado encontrar fuentes de datos fiables, a menos que paguen usd 10.000 por un terminal Bloomberg o cosa por el estilo, pero las fuentes son las fuentes y estos links son directamente los datos de la base de datos de la SEC, EDGAR

Como decía por lo general cuesta encontrar buena data de análisis fundamental, esta API es lo mejor de lo mejor en herramientas gratuitas de ese tipo, cuando hay complicaciones por ejemplo las monedas (si buscan balances de ADRs por ejemplo van a ver que no están listadas, al menos no al momento que yo estoy escribiendo esto) y eso porque evidentemente ante la complejidad

del tema de balances con series en diferentes monedas y criterios contables, prefirieron no poner estas empresas para cuidar la calidad de los datos de su API

Por otro lado como arriba solo les imprimí el último balance, veamos que trae la API en realidad, mostramos todos los datos que tiene la serie (columna) "date" del DataFrame, y vemos que tiene en el caso de AAPL casi 20 años de balances esta API!

```
bs.date
```

0	2019-09-28
0	2018-09-29
1	2017-09-30
2	2016-09-24
3	2015-09-26
4	2014-09-27
5	2013-09-28
6	2012-09-29
7	2011-09-24
8	2010-09-25
9	2009-09-26
10	2008-09-27
11	2007-09-29
12	2006-09-30
13	2005-09-24
14	2004-09-25
15	2003-09-27
16	2002-09-28
17	2001-09-29

Name: date, dtype: object

### Hoja de balance, variación

últimamente se está estilando mirar más los "growths" es decir los crecimientos o variaciones en las hojas de balances o mirarlo paralelamente a los valores absolutos ya que el "growth" me da una película o una idea más dinámica que los valores absolutos

Esta API cuenta con un endpoint para este llamado es el siguiente:

- <https://fmpcloud.io/api/v3/balance-sheet-statement-growth>  
(<https://fmpcloud.io/api/v3/balance-sheet-statement-growth>)

Implemento exactamente la misma función que la anterior pero con el cambio de la URL:

Aprovecho para tirar un gráfico, para ello le voy a cepillar las tres columnas que no son numéricas

Para simplificar tiro el gráfico directamente con la interfaz de pandas

Y de paso les muestro algo que les va a ser útil que es cuando tenés una serie toda con una determinada palabra que querés limpiar, o también otra que es pasar de camelCase toda una serie de indices a palabras separadas con espacios que es mucho más legible como label al ojo humano

```
import pandas as pd
```

```
def balanceSheetG(symbol, period):
    url = 'https://fmpcloud.io/api/v3/balance-sheet-statement-growth/' + symbol
    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

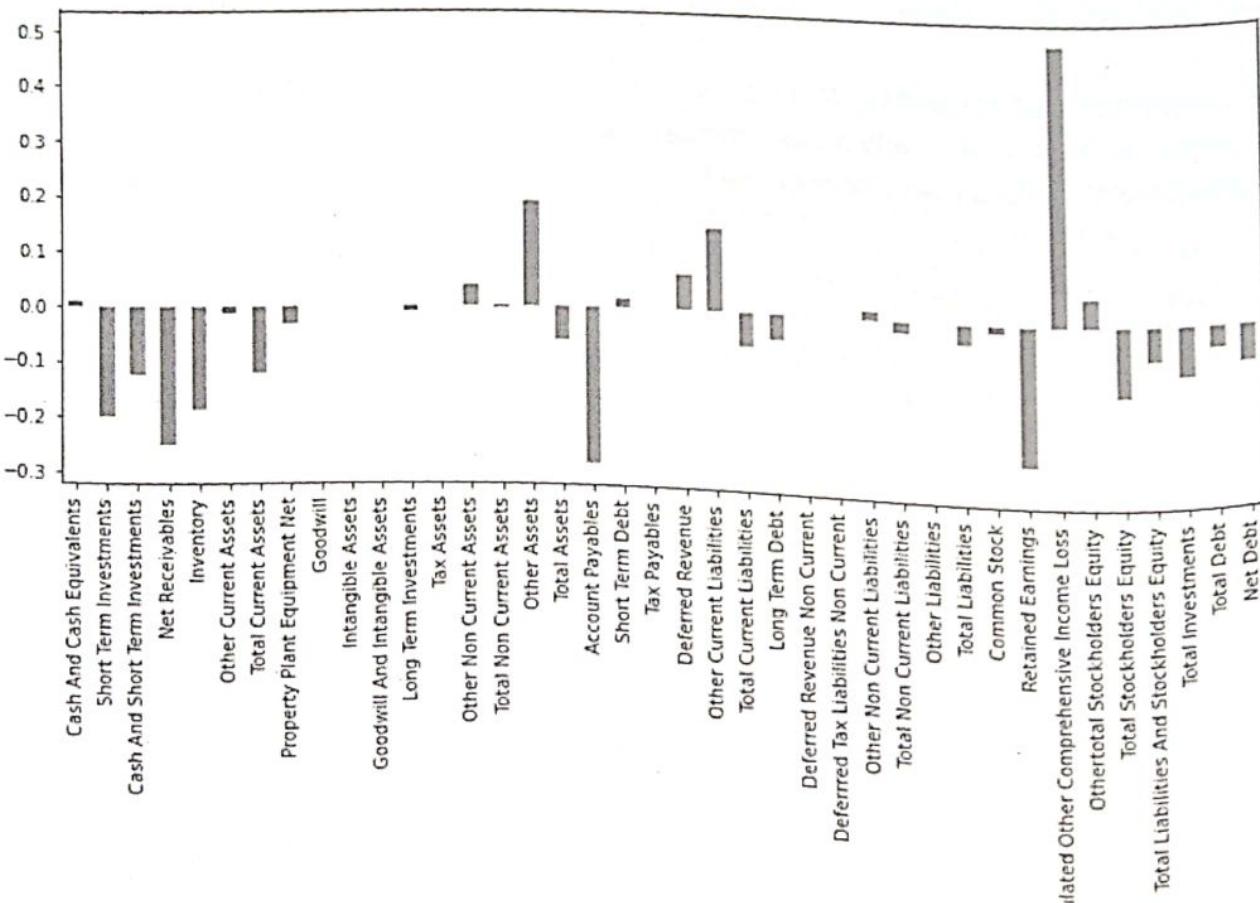
bs = balanceSheetG('AAPL', 'quarter')
ultimo = bs.loc[0].drop(['date', 'symbol', 'period'])

# Acoto por si hay valores muy fuera de escala a +/- 50%
acotado = ultimo.clip(lower=-0.5, upper=0.5)

# Con esto le saco la palabra "growth" a todas porque es redundante
acotado.index = acotado.index.str.replace('growth', '')

# Y aquí les dejo un truquito para pasar de "camelCase" a "camel case"
letras = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
for letra in letras:
    acotado.index = acotado.index.str.replace(letra, ' '+letra)

acotado.plot(kind='bar', figsize=(12,5))
```



## Estado de resultados

Bueno, no muy diferente de la implementación que vimos antes para la hoja de balance es lo mismo para el estado de resultados, trae una info más que pulida como decía antes y se puede verificar en el link a la base decía datos de la SEC que también trae la API

Les dejo la implementación ya más avanzada que permite pasar como parámetro si queremos o no como crecimiento (al estar growth predeterminada como False, significa que trae los valores absolutos como default)

```
def iStat(symbol, period, growth=False):

    # Este IF me permite 'decidir' si hago el request a un endpoint u otro
    if growth:
        url = 'https://fmpcloud.io/api/v3/income-statement-growth/' + symbol
    else:
        url = 'https://fmpcloud.io/api/v3/income-statement/' + symbol

    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params=p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

# Llamamos a la función que trae Los datos de la
# API data = iStat('AAPL', 'annual', False)

# Guardamos el link al balance
linkSEC = data.loc[0]['finalLink']

# traemos el último estado de resultados
ultimo = data.loc[0].drop(['link', 'finalLink'])
ultimo, linkSEC
```

(date	2019-09-28
symbol	AAPL
fillingDate	2019-10-31
acceptedDate	2019-10-30 18:12:36
period	FY
revenue	260174000000
costOfRevenue	161782000000
grossProfit	98392000000
grossProfitRatio	0.378178
researchAndDevelopmentExpenses	16217000000
generalAndAdministrativeExpenses	18245000000
sellingAndMarketingExpenses	0
otherExpenses	1.807e+09
operatingExpenses	34462000000
costAndExpenses	196244000000
interestExpense	3.576e+09
depreciationAndAmortization	12547000000
ebitda	76477000000
ebitdaratio	0.293946

operatingIncome	6.393e+10
operatingIncomeRatio	0.24572
totalOtherIncomeExpensesNet	1807000000
incomeBeforeTax	65737000000
incomeBeforeTaxRatio	0.252666
incomeTaxExpense	10481000000
netIncome	55256000000
netIncomeRatio	0.212381
eps	11.97
epsdiluted	11.89
weightedAverageShsOut	4617834000
weightedAverageShsOutDil	4648913000

<https://www.sec.gov/Archives/edgar/data/320193/000032019319000119/a10-k20199282019.htm>

## Cash Flow o Flujo de Fondos

El flujo de fondos es el corazón del estado financiero de la empresa, es la película de cómo se mueve o fluye el efectivo o el valor mejor dicho

```
def cashFlow(symbol, period, growth=False):

    # Este IF me permite 'decidir' si hago el request a un endpoint u otro
    if growth:
        url = 'https://fmpcloud.io/api/v3/cash-flow-statement-growth/' + symbol
    else:
        url = 'https://fmpcloud.io/api/v3/cash-flow-statement/' + symbol

    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

# Obtengo los cash flows
data = cashFlow('AAPL', 'annual')

    # Los reordeno e indexo por la fecha que paso a datetime objeto
    data.date = pd.to_datetime(data.date)
    data.sort_values('date', ascending=True)
data.set_index('date', inplace=True)
data.columns

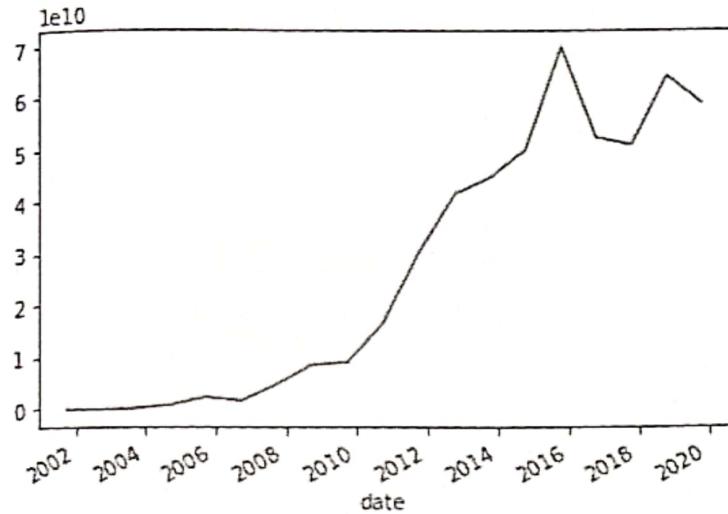
['symbol', 'fillingDate', 'acceptedDate', 'period', 'netIncome',
'depreciationAndAmortization', 'deferredIncomeTax', 'stockBasedCompensation',
'changeInWorkingCapital', 'accountsReceivables', 'inventory', 'accountsPayables',
'otherWorkingCapital', 'otherNonCashItems', 'netCashProvidedByOperatingActivities',
'investmentsInPropertyPlantAndEquipment', 'acquisitionsNet',
'purchasesOfInvestments', 'salesMaturitiesOfInvestments',
'otherInvestingActivites', 'netCashUsedForInvestingActivites', 'debtRepayment',
'commonStockIssued', 'commonStockRepurchased', 'dividendsPaid',
'otherFinancingActivites', 'netCashUsedProvidedByFinancingActivities',
'effectOfForexChangesOnCash', 'netChangeInCash', 'cashAtEndOfPeriod',
'cashAtBeginningOfPeriod', 'operatingCashFlow', 'capitalExpenditure',
'freeCashFlow', 'link', 'finalLink']
```

Recordemos que de todas esas columnas nos trae:

- \* El histórico de 20 años
- \* El Anual y trimestral
- \* El modo absoluto y modo growth

Obviamente podríamos graficar cualquier serie a un clic gracias a la magia de pandas y su interface con matplotlib

```
data.freeCashFlow.plot()
```



Claramente la potencia de esto es fenomenal porque con la misma API podemos:

- Consultar el ultimo estado contable
  - hoja de balance
  - estado de resultados
  - flujo de fondos
- Consultar los crecimientos del ultimo estado contable
- Consultar la evolución histórica de cualquier estado contable
- Armar ratios personalizados con las series históricas
- Cruzar toda esta data con la info de series de precios histórica
- Aún no llegamos, pero veremos cómo consultar más de 50 ratios financieros de cada activo

# Ratios financieros

Bueno, llegamos a los ratios, es la info más utilizada en el análisis fundamental, no nos vamos a detener en ninguno de ellos simplemente les muestro la implementación del llamado a la API y les muestro como queda lo que trae de respuesta

Antes de mostrarles la implementación les muestro la "pequeña dificultad" que nos trae

Este endpoint tiene las opciones de períodos:

- \* Anual (+50 Ratios, histórico)
  - \* Trimestral (+50 Ratios, histórico)
  - \* TTM (Trailing Twelve Months, últimos 12 meses independientemente del mes que estemos)
- Este último solo me trae 4 ratios y el último, no los históricos

El problema es que el anual y trimestral tienen un endpoint y TTM tiene otro (una mala práctica de quienes crearon la API probablemente el TTM lo agregaron después y les quedó así)

<https://fmpcloud.io/api/v3/ratios/AAPL>  
<https://fmpcloud.io/api/v3/ratios-ttm/AAPL>

A su vez el primer endpoint admite obviamente el parámetro period = annual/quarter, mientras que naturalmente el otro no

Para solucionarlo debemos meter un IF adentro de la función preguntando por el periodo pasado como parámetro y ahí definir la url y los parámetros

```
def ratios(symbol, period):  
  
    if period == 'TTM':  
        url = 'https://fmpcloud.io/api/v3/ratios-ttm/' + symbol  
    else:  
        url = 'https://fmpcloud.io/api/v3/ratios/' + symbol  
        p = {'apikey': apikey, 'period': period}  
        r = requests.get(url, params=p)  
        js = r.json()  
        df = pd.DataFrame(js)  
        return df  
  
ttmRatios = ratios('AAPL', 'TTM').loc[0]  
ttmRatios  
  
dividendYieldTTM      0.009835  
peRatioTTM            26.596671  
pegRatioTTM           2.063110  
payoutRatioTTM        0.261579  
Name: 0, dtype: float64
```

Bien, veamos los ratios que me trae para los anuales y trimestrales:

```
quarterRatios = ratios('AAPL', 'quarter')
quarterRatios.columns
```

```
Index(['symbol', 'date', 'currentRatio', 'quickRatio', 'cashRatio',
'daysOfSalesOutstanding', 'daysOfInventoryOutstanding', 'operatingCycle',
'daysOfPayablesOutstanding', 'cashConversionCycle', 'grossProfitMargin',
'operatingProfitMargin', 'pretaxProfitMargin', 'netProfitMargin', 'effectiveTaxRate',
'returnOnAssets', 'returnOnEquity', 'returnOnCapitalEmployed', 'netIncomePerEBT',
'ebtPerEbit', 'ebitPerRevenue', 'debtRatio', 'debtEquityRatio',
'longTermDebtToCapitalization', 'totalDebtToCapitalization', 'interestCoverage',
'cashFlowToDebtRatio', 'companyEquityMultiplier', 'receivablesTurnover',
'payablesTurnover', 'inventoryTurnover', 'fixedAssetTurnover', 'assetTurnover',
'operatingCashFlowPerShare', 'freeCashFlowPerShare', 'cashPerShare', 'payoutRatio',
'operatingCashFlowSalesRatio', 'freeCashFlowOperatingCashFlowRatio',
'cashflowCoverageRatios', 'shortTermCoverageRatios', 'capitalExpenditureCoverageRatio',
'dividendPaidAndCapexCoverageRatio', 'dividendPayoutRatio', 'priceBookValueRatio',
'priceToBookRatio', 'priceToSalesRatio', 'priceEarningsRatio', 'priceToFreeCashFlowsRatio',
'priceEarningsToGrowthRatio', 'priceSalesRatio', 'dividendYield',
'enterpriseValueMultiple', 'priceFairValue'], dtype='object')
```

Los ratios más comunes de esta lista son naturalmente:

- priceToBookRatio (PB o P/B)
- priceEarningsRatio (PER o P/E)
- priceSalesRatio (P/S)

Bueno, de toda esa data vamos a graficar algún ratio para mostrar la idea, pero las posibilidades son infinitas

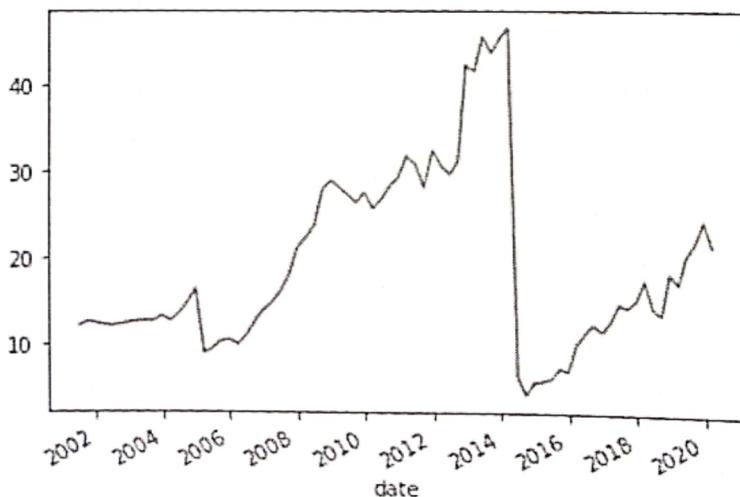
### Grafiquemos el ratio de Efectivo por Accion por ejemplo de AAPL

```
qRatiosSerie = ratios('AAPL', 'quarter')

# Pasamos a serie de tiempo el indice
qRatiosSerie['date'] = pd.to_datetime(qRatiosSerie['date'])
qRatiosSerie.set_index('date', inplace=True)

# Ordenamos de antiguo a reciente
qRatiosSerie.sort_values('date', ascending=True, inplace=True)

# Graficamos con pandas
qRatiosSerie.cashPerShare.plot()
```



## Métricas Financieras

Esta serie de métricas complementan la lista anterior, aun no entiendo por qué razón decidieron ponerlo en un endpoint diferente, estimo que por razones de arquitectura, cuando fueron construyendo la API llegaron a algún limitante y separaron este endpoint por alguna razón de ese estilo más que por un tema de la información que brindan

Como sea, son más ratios financieros, esto esta terriblemente completo para los amantes del análisis fundamental. Veamos primero la forma de implementar la función que traiga esto de la API

Y de salida voy a mostrar primero todas las columnas que nos devuelve:

```
def metricas(symbol, period):
    url = 'https://fmpcloud.io/api/v3/key-metrics/' + symbol
    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

metricas = metricas('AAPL', 'annual')
print(metricas.columns)
```

Index(['symbol', 'date', 'revenuePerShare', 'netIncomePerShare', 'operatingCashFlowPerShare', 'freeCashFlowPerShare', 'cashPerShare', 'bookValuePerShare', 'tangibleBookValuePerShare', 'shareholdersEquityPerShare', 'interestDebtPerShare', 'marketCap', 'enterpriseValue', 'peRatio', 'priceToSalesRatio', 'pocfratio', 'pfcfRatio', 'pbRatio', 'ptbRatio', 'evToSales', 'enterpriseValueOverEBITDA', 'evToOperatingCashFlow', 'evToFreeCashFlow', 'earningsYield', 'freeCashFlowYield', 'debtToEquity', 'debtToAssets', 'netDebtToEBITDA', 'currentRatio', 'interestCoverage', 'incomeQuality', 'dividendYield', 'pa\_youtRatio', 'salesGeneralAndAdministrativeToRevenue', 'researchAndDevelopmentToRevenue', 'intangiblesToTotalAssets', 'capexToOperatingCashFlow', 'capexToRevenue', 'capexToDepreciation', 'stockBasedCompensationToRevenue', 'grahamNumber', 'roic', 'returnOnTangibleAssets', 'grahamNetNet', 'workingCapital', 'tangibleAssetValue', 'netCurrentAssetValue', 'investedCapital', 'averageReceivables', 'averagePayables', 'averageInventory', 'daysSalesOutstanding', 'daysPayablesOutstanding', 'daysOfInventoryOnHand', 'receivablesTurnover', 'payablesTurnover', 'inventoryTurnover', 'roe', 'capexPerShare'],  
 dtype='object')

No me voy a detener en nada de esto porque este es un libro de python y no de análisis fundamental, pero al pasar les remarco un indicador que personalmente me parece interesante como referencia que es el **número de Graham** claramente en referencia a B. Graham el autor del famoso libro "el inversor inteligente" que combina el valor libro con la ganancia por acción en una fórmula, ese indicador está incluido en el endpoint pero lo que quiero remarcar como analista cuantitativo cuando hagan screeners de ratios e indicadores es que tomen ese tipo de cosas como referencia pero calculen sus propios ratios, info hay de sobra, en esta API tienen todo tipo de data de los estados contables, con lo que mi sugerencia es que se armen sus propios indicadores para los screeners, la originalidad y creatividad para elegir bien los datos en función de características cuantitativas puede hacer la diferencia en un buen screener y uno común y corriente

# Valuación Empresarial

Una de las valuaciones más consideradas por los analistas se la conoce como "Enterprise Value" que es simplemente el marketCap o el valor por bolsa corregido por: el efectivo o equivalente en cash (se resta) y la deuda (se suma), esto para ver qué es lo que está priceando el mercado exactamente

O sea que si una empresa tiene un mkCap de 1000 pero debe 500, el EV (enterprise value) es de 1500, y me está diciendo que el mercado está priceando la empresa en 1500 porque asume que devolviendo esos 500 quedan los 1000 limpios de valor por bolsa para los accionistas

Es una valuación que se usa en ratios como el EV/EBITDA que es una variación del PER porque expresa tanto la valuación como las ganancias en términos levemente diferentes

Bueno, sin más les muestro la implementación en python de la consulta a la API y lo que la misma devuelve:

Como verán es super sencilla viene tal cual el resto, en este caso como es una serie temporal voy a dejarle el índice de la fecha

```
def EV(symbol, period):
    url = 'https://fmpcloud.io/api/v3/enterprise-values/' + symbol
    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df = df.set_index('date')
    df = df.drop(['symbol'], axis=1)
    df.index = pd.to_datetime(df.index)
    return df
```

Y les dejo lo que trae mostrando el primer valor que es el más reciente

Le aplico la función .transpose() para verlo en una columna y varias filas en lugar de una fila y varias columnas ya que es más cómodo leerlo así

```
EValue = EV('AAPL','quarter')
print(EValue.head(1).transpose())
```

date	2020-03-28
stockPrice	2.831700e+02
numberOfShares	4.387570e+09
marketCapitalization	1.242428e+12
minusCashAndCashEquivalents	4.017400e+10
addTotalDebt	9.947800e+10
enterpriseValue	1.301732e+12

## Valuación por flujo de fondos descontado

Para los que han trabajado en proyectos de inversión de la economía real, y han armado carpetas de valuaciones, estarán bastante acostumbrados a la valuación por flujo de fondos descontado, para los que no conozcan esta valuación, es una forma de calcular el valor actual de la empresa trayendo a una tasa de descuento a hoy el valor futuro de todas las ganancias futuras proyectadas más un valor residual o final siempre traído a valor presente por una tasa de descuento

Es una manera más "realista" u objetiva de valuar, que no tiene nada que ver con la lógica del mercado y su mkCap que se rige por el humor de los mercados, pero de todos modos como se podrán dar cuenta en el DCF (flujo de fondos descontado en inglés) hay implícita una estimación subjetiva que son las ganancias futuras de la empresa

Obviamente este dato es totalmente desconocido y los valores allí volcados son absolutamente especulativos, pero tomando valores razonables de proyecciones según las ganancias pasadas podemos estimar una valuación al menos de cálculo actuarial si se quiere ver de ese modo

Bueno, les dejo la implementación de la llamada a la API

```
def DCF(symbol, period):
    url = 'fmpcloud.io/api/v3/historical-discounted-cash-flow-stateme nt/' + symbol
    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df = df.set_index('date')
    df = df.drop(['symbol'], axis=1)
    df.index = pd.to_datetime(df.index)
    return df
```

Imprimimos los primeros valores:

```
DCF('AAPL', 'annual').head()
```

	price	dcf
date		
2019-09-28	249.05	259.63
2018-09-29	212.24	217.47
2017-09-30	166.72	170.61
2016-09-24	117.65	121.39
2015-09-26	115.28	118.95

## Rating por valuaciones

Tenemos una llamada a la API interesante que nos va a devolver 5 "ratings" les pondría comillas quíntuples a esa palabra, ya que en realidad no es un rating sino un comparativo por valuaciones pero bueno, lo que me da es una idea de si está barata o cara la acción en función de esos ratios que veremos a continuación

```
def rating(symbol):
    url = 'https://fmpcloud.io/api/v3/historical-rating/'+symbol
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df = df.set_index('date')
    df = df.drop(['symbol'],axis=1)
    df.index = pd.to_datetime(df.index)
    return df
```

Imprimimos las columnas a ver que tiene

```
data.columns
```

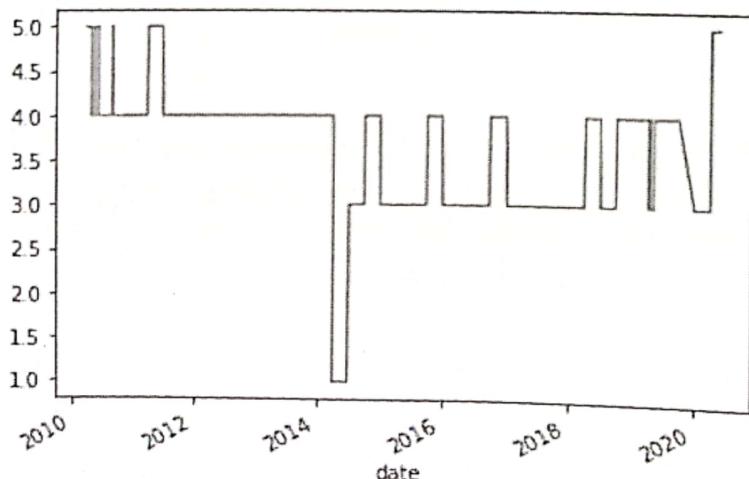
```
Index(['rating', 'ratingScore', 'ratingRecommendation', 'ratingDetailsDCFScore',
       'ratingDetailsDCFRecommendation', 'ratingDetailsROEScore', 'ratingDetailsROERecommendation',
       'ratingDetailsROAScore', 'ratingDetailsROARecommendation', 'ratingDetailsDEScore',
       'ratingDetailsDERecommendation', 'ratingDetailsPEScore', 'ratingDetailsPERecommendation',
       'ratingDetailsPBScore', 'ratingDetailsPBRecommendation'], dtype='object')
```

Como vemos me rankea por los siguientes criterios:

- \* Valuación de Flujo de fondos (DCF)
- \* Return on Equity (ROE)
- \* Return on Assets (ROA)
- \* Deuda sobre PN (DE)
- \* Precio sobre Ganancias (PER)
- \* Precio sobre Valor Libro (PB)

Y vemos el gráfico de alguno cualquiera de esos ratings:

```
data.ratingDetailsDCFScore.plot()
```



# 13F Filings

Los filings 13F famosos son unos formularios que están obligados a presentar a la SEC todos los tenedores de valores por montos superiores a 1000 millones de dólares (1B), se presentan trimestralmente y son nuestra guía para seguir "el dinero inteligente" odio esa frase, pero la uso acá porque se que me entienden lo que quiero decir.

Bueno, veamos, tengo varias cosas, una de ellas es un listado de los "grandes tenedores" de activos y su ID (cik), los consultamos así:

```
import requests
import pandas as pd

def f13F_list():
    url = 'https://fmpcloud.io/api/v3/cik_list?' p
    = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df
data = f13F_list()
data
```

	cik	name
0	0001694461	HARVEST GROUP WEALTH MANAGEMENT, LLC
1	0001583751	TCI Wealth Advisors, Inc.
2	0001356202	Beech Hill Advisors, Inc.
3	0001799859	Birch Capital Management, LLC
4	0001767045	Lindbrook Capital, LLC
...	...	...
6195	0001803397	Sculati Wealth Management, LLC
6196	0001079930	MCMILLION CAPITAL MANAGEMENT
6197	0001665337	Integrated Investment Consultants, LLC
6198	0001120048	STEGINSKY CAPITAL LLC
6199	0001791555	Hubbell Strickland Wealth Management, LLC

6200 rows x 2 columns

Y si queremos buscar por nombre de fondo o tenedor, también disponemos de una query para ello en esta API, les dejo el endpoint para que practiquen con alguna búsqueda puntual y corroboren con el método que les voy a mostrar a continuación

El endpoint para buscar por nombre es:

[https://fmpcloud.io/api/v3/cik-search/CADENA\\_A\\_BUSCAR?](https://fmpcloud.io/api/v3/cik-search/CADENA_A_BUSCAR?)

Obviamente no siempre vamos a contar con un endpoint para búsqueda, y de hecho siempre es mejor hacer una sola llamada y guardar esta data y luego buscar las veces que sea necesario sin tener que recurrir a más llamadas

Creo que se los mostré en el libro de Pandas pero por si no lo hice, les refresco esta función de pandas que me permite buscar dentro del dataframe diferentes cosas, en este caso alguna coincidencia con un string

```
busquedaPandas = data.loc[data.name.str.contains("black", case=False)]  
busquedaPandas
```

	cik	name
1277	0001582561	Blackhawk Capital Partners LLC.
1402	0001364742	BlackRock Inc.
3160	0001528593	Black Creek Investment Management Inc.
3815	0000872162	BLACKHILL CAPITAL
4183	0001800249	Blacksheep Fund Management Ltd
4494	0001720745	Blackcrane Capital, LLC
5371	0001282197	BLACK DIAMOND CAPITAL, L.L.C.
5436	0001666647	Blackstart Capital LP
5979	0001393818	Blackstone Group Inc
5980	0001393818	Blackstone Group L.P.

Bueno, y ahora si veamos la función para ver tenencias, supongamos que queremos ver de "BalckRock" como vimos tiene un CIK=0001364742 así que consultamos por ese CIK

```
def f13F(cik,year):  
    url = 'https://fmpcloud.io/api/v3/form-thirteen/'+cik  
    p = {'apikey': apikey, 'year':year}  
    r = requests.get(url, params = p)  
    js = r.json()  
    df = pd.DataFrame(js)  
    return df  
data = f13F('0001364742',2019)  
data.shape, data.columns  
  
((19888, 12),  
  
Index(['date', 'fillingDate', 'acceptedDate', 'cik', 'cusip', 'tickercusip','nameOfIssuer',  
'shares', 'titleOfClass', 'value', 'lin k', 'finalLink'], dtype='object'))
```

Como vemos, BlackRock tiene en 2019, 19888 posiciones declaradas a la SEC, chusmiemos un poco mas

```
data.groupby('date').fillingDate.count()

date
2019-03-31    4924
2019-06-30    5009
2019-09-30    4981
2019-12-31    4974
Name: fillingDate, dtype: int64
```

Eso nos muestra la cantidad de activos declarados en cada presentación, son presentaciones trimestrales

Filtremos las 10 mayores posiciones de fines de 2019:

```
# Filtramos la ultima presentacion de 2019
ultimos2019 = data.loc[data.date=='2019-12-31']

# Calculamos el monto total de BalckRock
montoTotal = ultimos2019.value.sum()

# Ordenamos esos valores por monto descendente
ultimos2019 = ultimos2019.sort_values('value', ascending=False)

# Elegimos los 10 primeros y las columnas de ticker y monto
top10 = ultimos2019.head(10).loc[:,['tickercusip','value']]

# Calculamos el % de su cartera en cada activo del top10
top10['% Cartera'] = top10.value/montoTotal *100

top10
```

		tickercusip	value	% Cartera
134	AAPL	81567328000	3.127740	
4320	MSFT	81246600000	3.115442	
74	AMZN	49351143000	1.892394	
3845	FB	32294030000	1.238331	
883	JPM	29097325000	1.115751	
2528	JNJ	28163322000	1.079937	
3391	GOOG	26698630000	1.023772	
3392	GOOGL	26373087000	1.011289	
3522	BRKB	24937451000	0.956239	
3269	V	23531052000	0.902310	

# Ejercicios

1- Reutilizar la función del principio de este libro usada para hacer gráficos tipo mapa correlación y graficar la correlación entre los rendimientos históricos devueltos por esta API de los distintos sectores de equity

2 - Implementar una función que devuelva las hojas de balances y que tome el argumento "growth" que por default sea False, para poder usar la misma función para traer el endpoint de los valores absolutos o el de los crecimientos de las hojas de balances:

3- Implementar una función que grafique la evolución del activo y pasivo de una empresa y su patrimonio neto con esta API de fmpcloud

4- Usando la función que vimos en esta guía iStat que copio a continuación  
Armar un script que muestre un gráfico del crecimiento del ultimo estado de resultados en cada item

```
def iStat(symbol, period, growth=False):
    apikey = 'd4bf02982fc44e1c4f0a33ccfe9c6c03'
    if growth:
        url = 'https://fmpcloud.io/api/v3/income-statement-growth/'+symbol
    else:
        url = 'https://fmpcloud.io/api/v3/income-statement/'+symbol
    p = {'apikey': apikey, 'period':period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df
```

5- Usando las funciones EV y DCF que vimos en este capítulo para llamar a la API y traer las valuaciones empresariales y de flujo de fondos respectivamente, armar un script que me grafique de 2015 en adelante el precio por acción según cada una de esas valuaciones y el precio de la acción en sí

Ya a esta altura no voy a seguir copiando las funciones vistas antes en el libro, se los dejo ya como parte de su trabajo, ya que deben acostumbrarse a leer rápido el código y funciones

6- Usando la llamada a la API de los ranks de valuación, armar con el siguiente diccionario de ponderaciones una función que devuelva un ranking ponderado y que acepte como argumento un diccionario como el dado, y me grafique la serie del ranking ponderado devuelto. Nótese que se pide que la función grafique y devuelva la serie como return

Ponderaciones: p = {'DCF':0.25, 'ROE':0.1, 'ROA':0.1, 'DE':0.15, 'PE':0.2, 'PB':0.2}

7- Sabiendo que el fondo Blackock tiene el cik='0001364742' graficar su tenencia en el tiempo de nominales de GGAL, YPF y TEO

# Respuestas

```
# Ejercicio 1
import pandas as pd, requests

def graficaCorr(dfCorr, title=''):
    import matplotlib.pyplot as plt
    import numpy as np

    fig = plt.figure(figsize=(14, 8))
    plt.matshow(dfCorr, fignum=fig.number, cmap='binary')
    plt.xticks(range(dfCorr.shape[1]), dfCorr.columns, rotation=90)
    plt.yticks(range(dfCorr.shape[1]), dfCorr.columns, fontsize=11)

    cb = plt.colorbar(orientation='vertical', label="Factor Correlación")
    cb.ax.tick_params(labelsize=12)
    plt.title(title, fontsize=16, y=1.15)

    ax = plt.gca()
    ax.set_xticks(np.arange(-.5, len(dfCorr), 1), minor=True);
    ax.set_yticks(np.arange(-.5, len(dfCorr), 1), minor=True);
    ax.grid(which='minor', color='w', linestyle='-', linewidth=3)

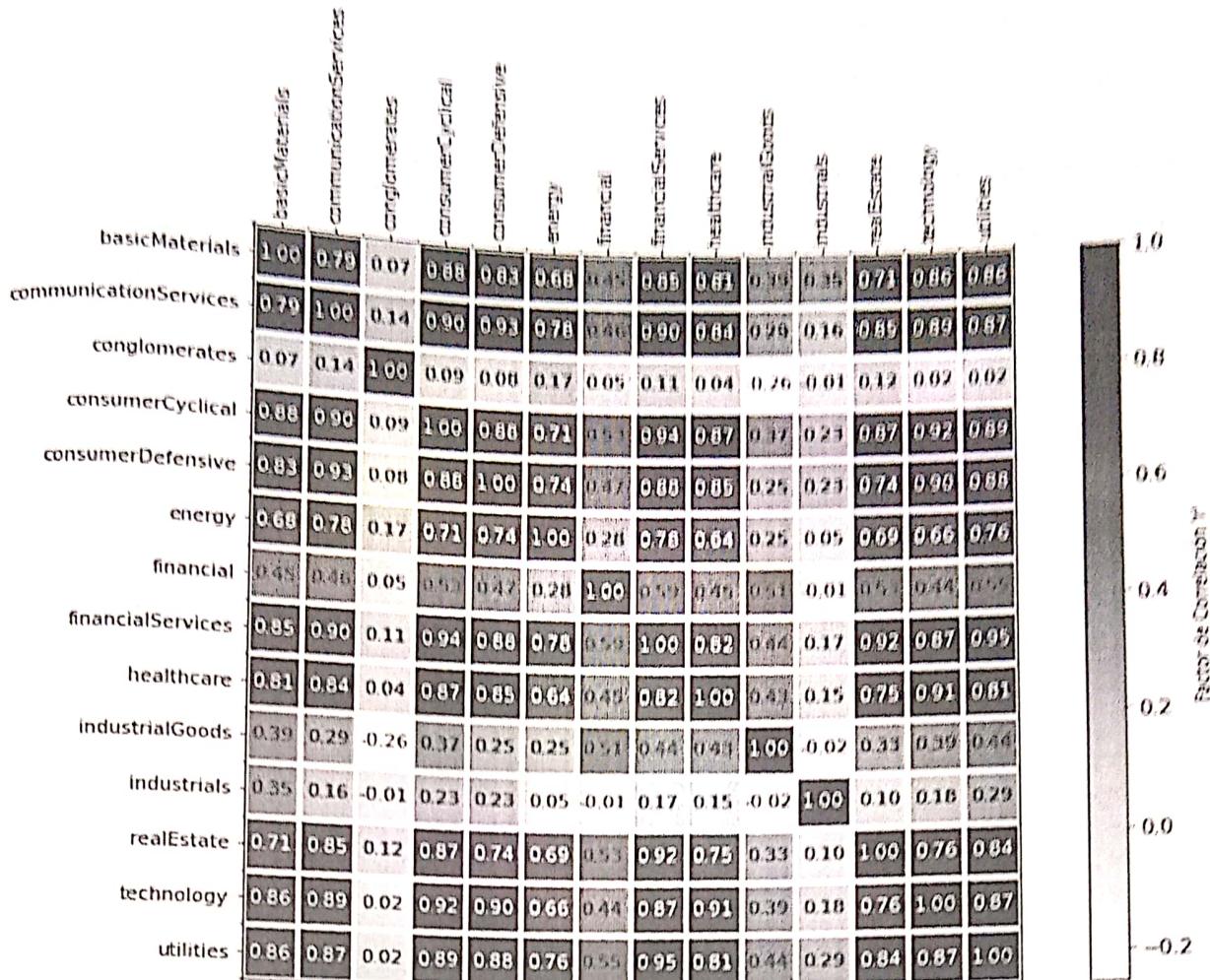
    for i in range(dfCorr.shape[0]):
        for j in range(dfCorr.shape[1]):
            if dfCorr.iloc[i,j] > 0.6:
                color = 'white'
            else:
                color = 'black'
            fig.gca().text(i,j, "{:.2f}".format(dfCorr.iloc[i,j]),
                           ha="center", va="center", c=color, size='11')
    plt.show()

def sectorsHist(sector):
    url = 'https://fmpcloud.io/api/v3/historical-sectors-
performance' p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('date', inplace=True)
    df.index = pd.to_datetime(df.index)
    return df[sector+'ChangesPercentage']

sectors = ['basicMaterials', 'communicationServices', 'conglomerates',
           'consumerCyclical', 'consumerDefensive', 'energy', 'financial',
           'financialServices', 'healthcare', 'industrialGoods',
           'industrials', 'realEstate', 'technology', 'utilities']

tabla = pd.DataFrame()
for sector in sectors:
    tabla = pd.concat([tabla, sectorsHist(sector)], axis=1)

tabla.columns=sectors
graficaCorr(tabla.corr())
```



```

#-----#
# Rta Ejercicio 2 #
#-----#


import pandas as pd, requests
import matplotlib.pyplot as plt

def bS(symbol, period, growth=False):
    apikey = 'd4bf02982fc44e1c4f0a33ccfe9c6c03'
    if growth:
        url = 'https://fmpcloud.io/api/v3/balance-sheet-statement-growth/'+symbol
    else:
        url = 'https://fmpcloud.io/api/v3/balance-sheet-statement/'+symbol
    p = {'apikey': apikey, 'period':period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

```

```

#-----#
# Rta Ejercicio 3 #
#-----#

import pandas as pd, requests, matplotlib.pyplot as plt

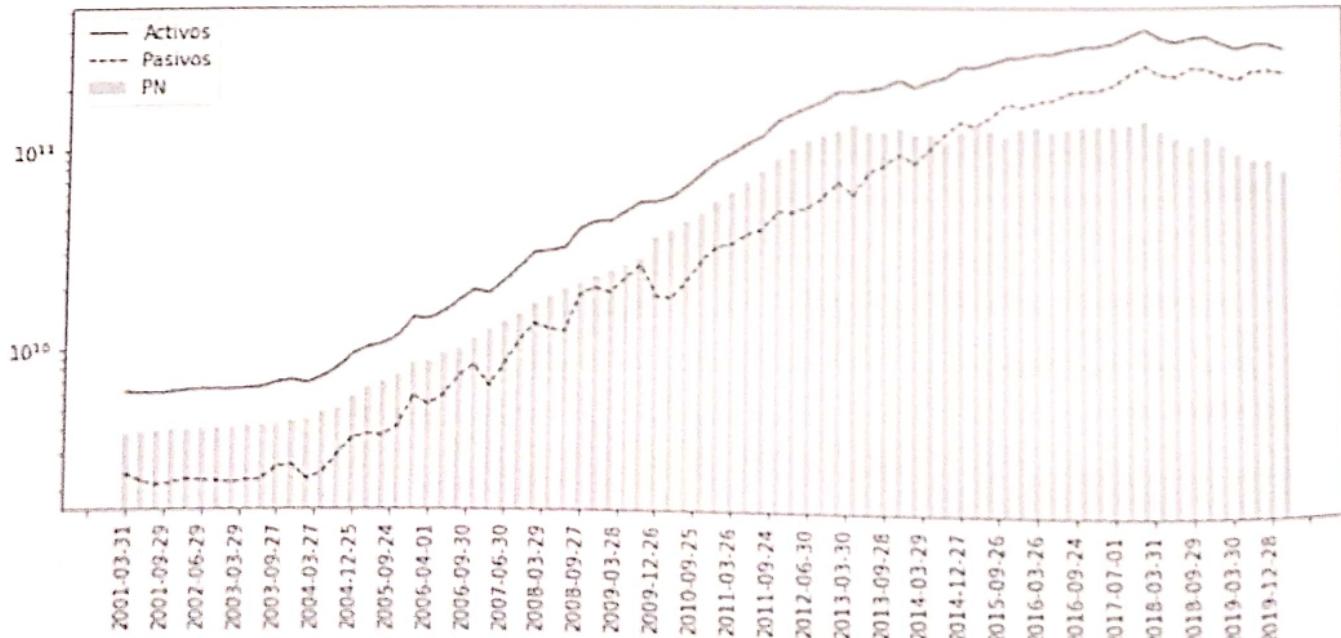
def bS(symbol, period, growth=False):
    apikey = 'd4bf02982fc44e1c4f0a33ccfe9c6c03'
    if growth:
        url = 'https://fmpcloud.io/api/v3/balance-sheet-statement-growth/'+symbol
    else:
        url = 'https://fmpcloud.io/api/v3/balance-sheet-statement/'+symbol
    p = {'apikey': apikey, 'period': period}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df.set_index('date', inplace=True)
    return df[::-1]

# Grafico
def graficarAyP(data):
    fig, ax = plt.subplots(figsize=(12, 5))
    activo = data.totalAssets
    pasivo = data.totalLiabilities
    ax.plot(activo, 'k-', lw=1, label = 'Activos')
    ax.plot(pasivo, 'k--', lw=1, label = 'Pasivos')
    ax.bar(data.index, activo-pasivo, label='PN', width=0.5, color='silver')
    ax.set_yscale('log')
    plt.xticks(rotation=90)
    ax.legend()

    # Esta Linea es para mostrar menos cantidad de elementos del eje X
    # muy util cuando tenemos muchos y se superponen los labels
    ax.xaxis.set_major_locator(plt.MaxNLocator(40))
    plt.show()

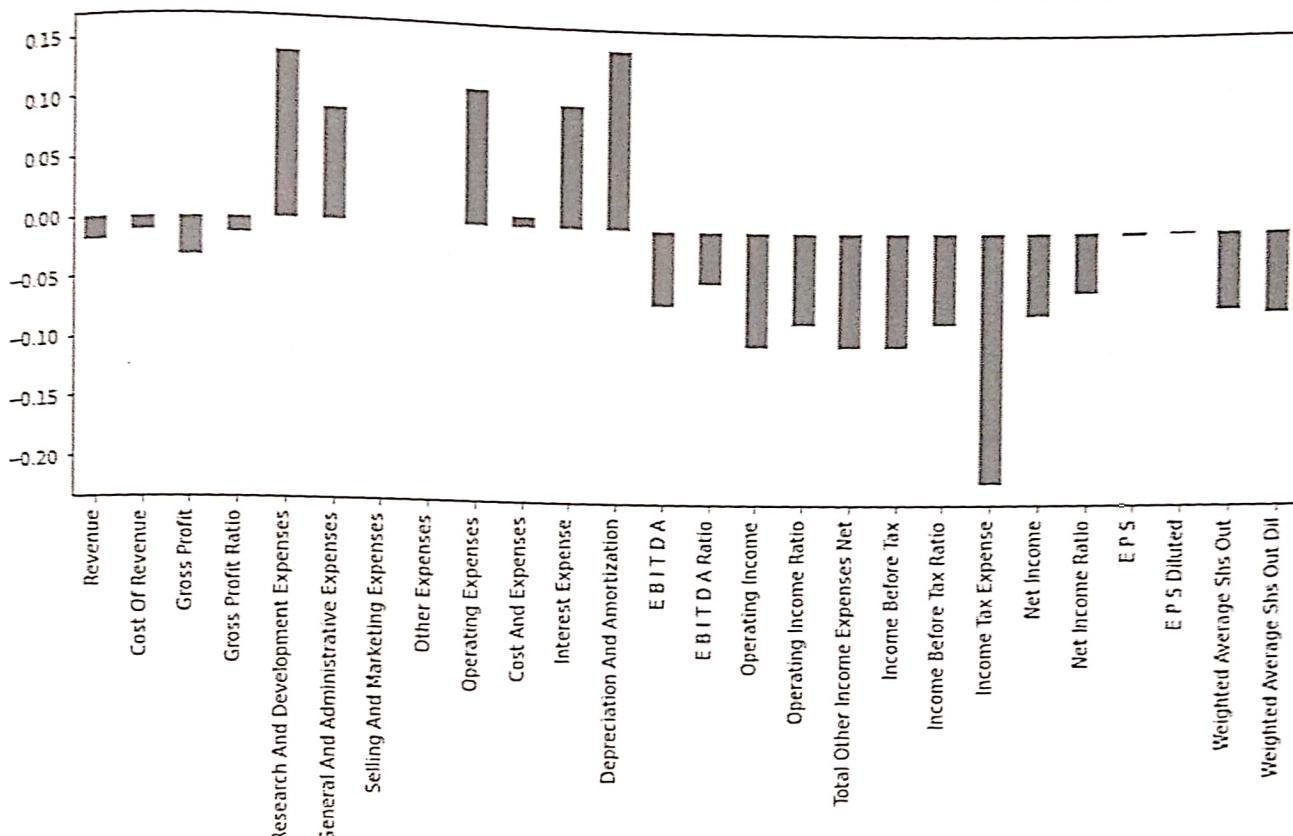
data = bS('AAPL', 'quarter', False)
graficarAyP(data)

```



# Rta Ejercicio 4

```
data = iStat('AAPL', 'annual', True)
iStat = data.loc[0].drop(['date','symbol','period'])
acotado = iStat.clip(lower=-0.5, upper=0.5)
acotado.index = acotado.index.str.replace('growth', '')
letras = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
for letra in letras:
    acotado.index = acotado.index.str.replace(letra, ' '+letra)
acotado.plot(kind='bar', figsize=(12,5))
```



```

#-----#
# Rta Ejercicio 5 #
#-----#
#No repetimos las funciones EV y DFC para no ser repetitivos

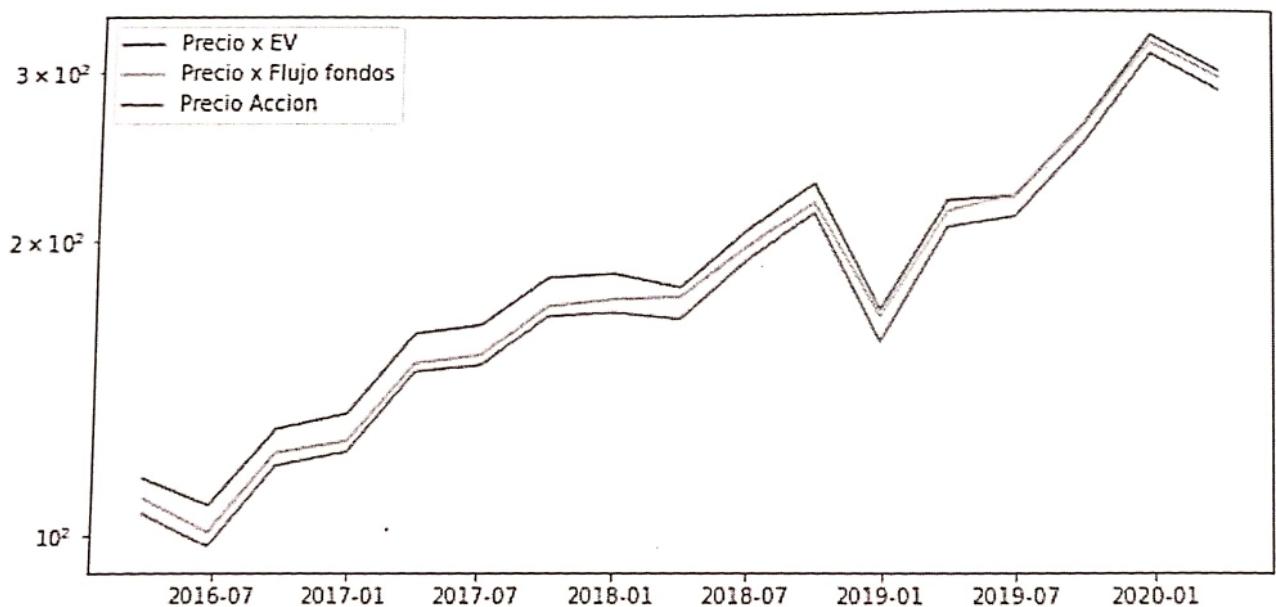
import matplotlib.pyplot as plt

EValue = EV('AAPL','quarter')
DCFValue = DCF('AAPL', 'quarter')

EValueAccion = round(EValue.enterpriseValue/EValue.numberOfShares,2)
Val = pd.concat([EValueAccion, DCFValue.dcf, DCFValue.price], axis=1)
Val.columns = ['EValue', 'DCFValue', 'Price']
Val = Val.loc[Val.index.year>2015]

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(Val['EValue'], label='Precio x EV')
ax.plot(Val['DCFValue'], label='Precio x Flujo fondos')
ax.plot(Val['Price'], label='Precio Accion')
ax.set_yscale('log')
ax.legend()
plt.show()

```



```

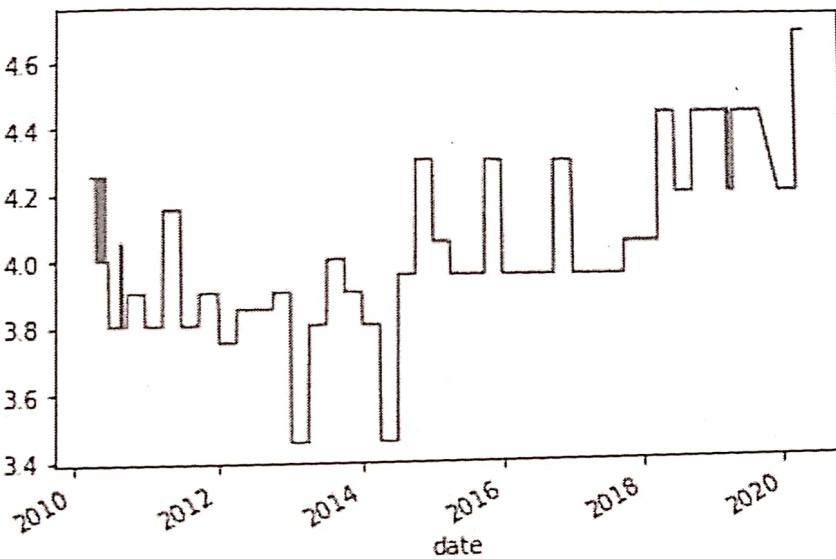
#-----#
# Rta Ejercicio 6   #
#-----#


def rating(symbol):
    url = 'https://fmpcloud.io/api/v3/historical-rating/'+symbol
    p = {'apikey': apikey}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    df = df.set_index('date')
    df = df.drop(['symbol'],axis=1)
    df.index = pd.to_datetime(df.index)
    return df

def graficarRanking(symbol, p):
    data = rating(symbol)
    score = data.ratingDetailsDCFScore*p['DCF']
    score += data.ratingDetailsROEScore*p['ROE']
    score += data.ratingDetailsROAScore*p['ROA']
    score += data.ratingDetailsDEScore*p['DE']
    score += data.ratingDetailsPEScore*p['PE']
    score += data.ratingDetailsPBScore*p['PB']
    score.plot()
    plt.show()
    return score

p = {'DCF':0.25, 'ROE':0.1, 'ROA':0.1, 'DE':0.15, 'PE':0.2, 'PB':0.2}
ret = graficarRanking('AAPL',p)
print(ret.head())

```



date	ratingDetailsDCFScore
2020-05-22	4.7
2020-05-21	4.7
2020-05-20	4.7
2020-05-19	4.7
2020-05-18	4.7

Name: ratingDetailsDCFScore, dtype: float64

```

#-----#
# Rta Ejercicio 7 #
#-----#

import matplotlib.pyplot as plt

def f13F(cik,year):
    url = 'https://fmpcloud.io/api/v3/form-thirteen/'+cik
    p = {'apikey': apikey, 'year':year}
    r = requests.get(url, params = p)
    js = r.json()
    df = pd.DataFrame(js)
    return df

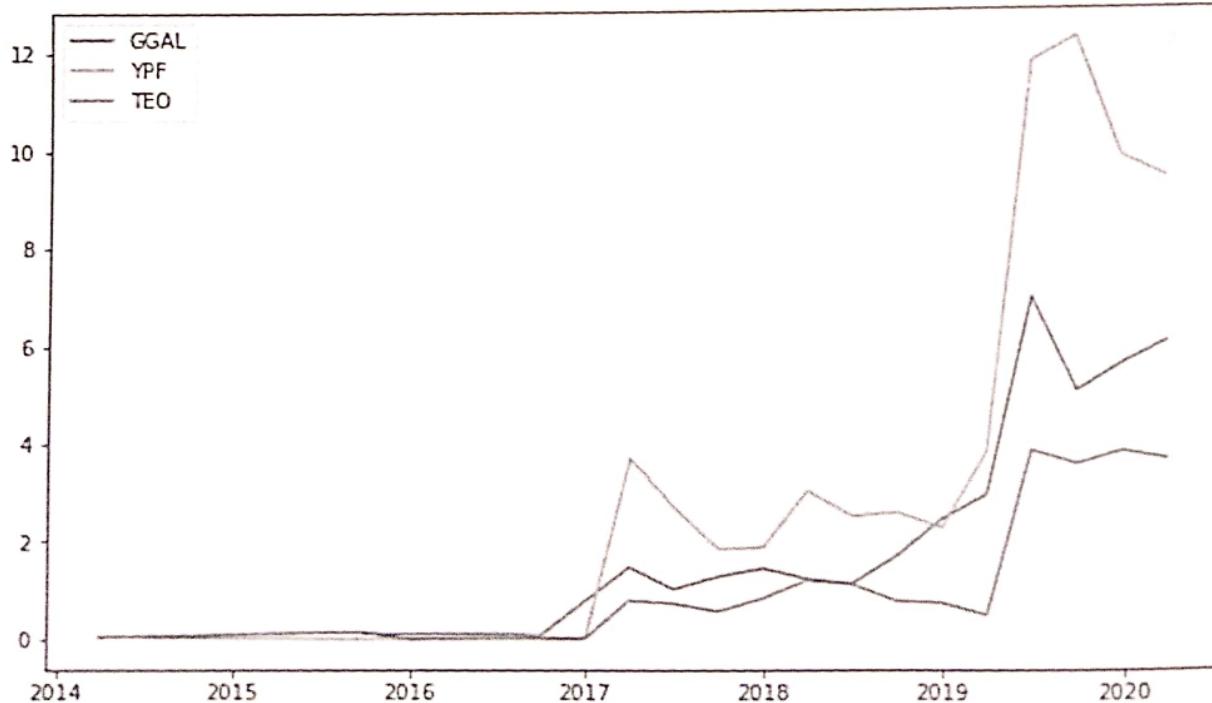
def tenencia(cik,ticker):
    data = f13F(cik,'')
    search = data.loc[data.tickercusip.str.contains(ticker, case=False)]
    search = search.set_index('date')
    search.index = pd.to_datetime(search.index)
    return search

cik = '0001364742'
ggal = tenencia(cik,'GGAL')
ypf = tenencia(cik,'YPF')
teo = tenencia(cik,'TEO')

fig, ax = plt.subplots(figsize=(10.,6))
ax.plot(ggal.shares/1000000, label='GGAL')
ax.plot(ypf.shares/1000000, label='YPF')
ax.plot(teo.shares/1000000, label='TEO')
plt.suptitle('BlackRock posición en millones de nominales ADRs', y=0.93)
plt.legend()

```

BlackRock posición en millones de nominales ADRs



# IEX Cloud API

Dejé esta API para el final de esta tanda porque es la que tiene mas cosas, la mas completa de todas con una cantidad de opciones y endpoints insuperable, así que como es inmensa la cantidad de cosas que tiene, solo voy a mostrar acá las mas únicas o el diferencial de esta respecto a otras APIs, por ejemplo no voy a mostrar una serie de precios histórica o las estadísticas básicas de ratios o los estados contables, porque eso lo pueden ver de otras APIs mas sencillas

De hecho les voy a mostrar en esta guía solo los 10 endpoints mas especiales de esta API que no verán en otras o que están en otras APIs pero de mucho menor calidad

Cabe destacar que como contrapartida es "la mas cara" obviamente acá a fines didácticos voy a mostrar solo las opciones gratuitas ya que la API como casi todas cuentan con un tipo de cuenta GRATIS, pero hay que saber que si vamos a darle un uso fuerte vamos a tener que optar por alguna de las opciones pagas, aunque arrancan de solo usd 9 al mes

## Peso de las Api Calls

Lo primero que vemos con esta API, y lo vamos a ver en general en las APIs mas profesionales, es que el peso de las llamadas a la API no es constante o Flat, sino que hay endpoints que tienen un peso y endpoints que tienen otro peso, esto lo vamos a ir viendo en cada endpoint en la documentación de la API en:

- <https://iexcloud.io/docs/api/>

A fines didácticos saqué un token gratuito que usaré en este libro, lo dejo por acá pero saquen el suyo en <https://iexcloud.io>

```
token = 'pk_7a042da030014a719cd7b12a69854ff2'
```

También vamos a usar un token\_sandbox que es como un token de pruebas, esto es porque esta API tiene endpoints gratuitos y endpoints pagos, es algo común con lo que se van a encontrar seguido, cuando hay endpoints pagos muchas veces te dan un token\_sandbox para que pruebes la función aunque los datos que te devuelven no son reales o están levemente distorsionados a propósito.

```
token_sandbox = 'Tsk_c33e06dd46af46a7b4d42130e87e5455'
```

Como decía antes, esta API tiene endpoints a los que se puede acceder con una cuenta gratuita y endpoints a los que solo se puede con una cuenta paga, en esos casos, lo que hago es usar el "sandbox" que es como una forma de probar la API por si les interesa ese tipo de data,

Usar el sandbox en realidad es apuntar a otra api, van a ver que es otra URL, otro subdominio en realidad, y como decía antes al usar el subdominio del "sandbox" la info que tira no es real real, es como si estuviera "levemente distorsionada" eso lo hacen para que puedas ver la info pero que si queres usarla pagues una suscripción, igual insisto arranca de usd 9 mensuales, no es gran cosa, aunque tampoco es que con usd 9 mensuales vas a poder darle un uso industrial, como verán cada llamada tiene un peso o costo (estos de iexcloud lo miden en "mensajes" y hay endpoints que cuestan 1 msg y otros que cuestan 10.000 msgs cada llamada, y los usd 9 mensuales te cubren 5.000.000 de msgs)

El costo de cada api call lo llaman "weight"

Como dije antes, ya llegado este punto del libro no voy a mostrar un endpoint de serie de precios histórico porque ya lo mostré en APIs anteriores, aquí voy a mostrar solo los endpoints más interesantes de esta API o que mas la diferencian del resto, y como podrán observar la mayoría son accesibles con suscripciones pagas, pero en realidad es porque estoy seleccionando justamente los endpoints "mas especiales"

### Importaciones básicas

Vamos a importar pandas así ya nos queda en el kernel, esta librería cuando trabajamos con estructuras de datos de series y diccionarios es casi inevitable de prescindir

```
import pandas as pd
```

Y como siempre hicimos hasta ahora para comunicarnos con la API vamos a necesitar la librería estandar requests

```
import requests
```

## Endpoint BASE

Vamos a tener una URL base común para todos los ENDpoints que es:abs

\* <https://cloud.iexapis.com/stable/>

Y para los endpoint pagos en los que usemos el SANDBOX el endpoint es:

\* <https://sandbox.iexapis.com/stable/>

# Libro de Precios

Bueno, veamos por ejemplo el endpoint del BOOK de precios lo que nos devuelve:

- quote
  - datos institucionales
  - precios demorados mercado oficial
  - precios real time IEX
  - puntas e info volumen IEX
  - Horario extendido en IEX
  - Referencia 52week, las price etc
  - MktCap, y ratio p/e
- bids (puntas de compra)
- asks (puntas de venta)
- trades (ultimos trades)

Implementemos una función para traer esta data:

```
def getBook(symbol):  
    url = 'https://cloud.iexapis.com/stable/stock/'+symbol+'/book'  params =  
    {'token':token}  
    r = requests.get(url, params)  
    return r.json()  
data = getBook('AMZN')  
data['quote']
```

Veamos por partes la respuesta ya que la cantidad de datos que trae esta sola llamada que tiene un costo de 1 msg es inmensa:

Primera parte mas "Institucional"

```
'symbol': 'AMZN',  
'companyName': 'Amazon.com, Inc.',  
'primaryExchange': 'NASDAQ',  
'calculationPrice': 'tops',  
'openSource': 'official',  
'closeSource': 'official',  
'highSource': '15 minute delayed price',  
'lowSource': '15 minute delayed price',
```

Esta parte nos indica el precio Oficial del mercado en que está inscripto, en este caso Nasdaq

```
'latestPrice': 2483.28,  
'latestSource': 'IEX real time price',  
'latestTime': '3:25:25 PM',  
'latestUpdate': 1591385125162,  
'latestVolume': None,
```

Esta Parte Nos Muestra el precio en RealTime de Iex Investors

```
'iexRealtimePrice': 2483.28,  
'iexRealtimeSize': 100,  
'iexLastUpdated': 1591385125162,
```

Esta es la parte que funciona en horario fuera de rueda

```
'extendedPrice': None,  
'extendedChange': None,  
'extendedChangePercent': None,  
'extendedPriceTime': None,
```

Primeras Puntas de IEX y volumen

```
'iexMarketPercent': 0.03228709052123007,  
'iexVolume': 79551,  
'avgTotalVolume': 4254940,  
'iexBidPrice': 2350,  
'iexBidSize': 300,  
'iexAskPrice': 2500,  
'iexAskSize': 175,
```

Estas claves nos informan acerca de la comparación con precios anteriores y alguna data general

```
'previousClose': 2460.6,  
'previousVolume': 2948710,  
'change': 22.68,  
'changePercent': 0.00922,  
'volume': None,  
'marketCap': 1238600465280,  
'peRatio': 116.51,  
'week52High': 2525.45,  
'week52Low': 1626.03,  
'ytdChange': 0.30563,  
'lastTradeTime': 1591385125162,  
'isUSMarketOpen': True
```

Creo que no hay mucho que agregar, tenemos en "quote" el post market (extendedPrice) acá no me lo está mostrando porque en este preciso momento que estoy haciendo la consulta estamos en horario de mercado pero si hacen la misma consulta a la API a la noche verán el precio post-market en esa clave "extendedPrice", y en "extendedPriceTime" verán la hora del último trade que generó ese precio, obviamente en el PRE-Market si lo consultan a las 9 AM verán en esa clave el valor del PRE-Market en lugar del post-market

Bueno, luego el mismo llamado nos trae las siguientes claves:

```
data['bids']
```

```
[{'price': 2350, 'size': 300, 'timestamp': 1591363800460},  
{'price': 2280, 'size': 100, 'timestamp': 1591363802335},  
{'price': 2270, 'size': 100, 'timestamp': 1591363802351},  
{'price': 2260, 'size': 100, 'timestamp': 1591363802318},  
{'price': 1700, 'size': 120, 'timestamp': 1591363802280},  
{'price': 1421, 'size': 490, 'timestamp': 1591363802302}]
```

```
data['asks']
```

```
[{'price': 2500, 'size': 175, 'timestamp': 1591367907503},  
{'price': 2589.15, 'size': 1000, 'timestamp': 1591363802295}]
```

Obviamente eso fueron las puntas de compra y venta, y ademas nos trae los últimos trades, que muestro a continuación, recordemos todo en un mismo llamado a la API

Como es un listado de trades con todos los mismos campos podemos pasarlo a DataFrame:

```
trades = pd.DataFrame(data['trades'], columns=['price','size','timestamp'])
trades.head()
```

	price	size	timestamp
0	2483.28	100	1591385125162
1	2481.90	50	1591385055156
2	2481.90	50	1591385055156
3	2481.92	40	1591385030141
4	2481.92	60	1591385029243

Y ya que estamos al ser una serie temporal lo mas lógica con esto sería tratarlo como tal, pasando el timestamp (recordemos una fecha epoch) a fecha objeto datetime, y mandarlo como índice de la serie.

Para ello usamos la función milagrosa de Pandas `to_datetime()` que agarra lo que le tires que sea una fecha y te lo convierte en objeto datetime como por arte de magia.. en este caso como el timestamp está en milisegundos le tenemos que indicar eso en el parámetro "unit" les comento esto porque he renegado viendo que no me lo convertía me ponía fechas de 1970 y era porque estaba interpretando el dato como segundos cuando eran milisegundos

```
trades = pd.DataFrame(data['trades'], columns=['price','size','timestamp'])

# Generamos una columna con la hora
trades['hora'] = pd.to_datetime(trades.timestamp, unit='ms')

# Laponemos como índice
trades.set_index('hora', inplace=True)

trades.head()
```

	price	size	timestamp
<b>hora</b>			
2020-06-05 19:25:25.162	2483.28	100	1591385125162
2020-06-05 19:24:15.156	2481.90	50	1591385055156
2020-06-05 19:24:15.156	2481.90	50	1591385055156
2020-06-05 19:23:50.141	2481.92	40	1591385030141
2020-06-05 19:23:49.243	2481.92	60	1591385029243

## Trades más grandes del día (sandBox)

Este endpoint te devuelve los trades de mayor cantidad de nominales del dia, y te muestra:

- \* El precio
- \* La cantidad de nominales operados
- \* En que exchange se produjeron
- \* A que hora fue cada trade

Como en este caso son datos para suscriptores pagos, usamos el sandbox que lo que hace es generar data "distorsionada" que en este caso nos sirve como recurso pedagógico y para que les pueda mostrar todo lo que se puede consultar en APIs pero obviamente en caso de querer usar este tipo de data, deberían contratar al menos el servicio de usd 9 al mes

La implementación es exactamente igual que como las otras, en este caso solo cambia (aparte del endpoint) que usamos el "token\_sandbox" en lugar del "token"

```
def getLargest(symbol):  
    url = 'https://sandbox.iexapis.com/stable/stock/' + symbol + '/largest-trades'  
    params = {'token': token_sandbox}  
    r = requests.get(url, params)  
    js = r.json()  
    df = pd.DataFrame(js)  
    return df  
data = getLargest('TWTR')  
data.head()
```

	price	size	time	timeLabel	venue	venueName
0	35.75	301505	1658616488971	12:20:17	noNe	f eOhEcfgxna
1	36.82	305061	1649173763714	12:21:22	oeNn	ahcfOxfg neE
2	36.70	179868	1609454308624	12:43:35	oNne	hgfOfaEcxne
3	35.74	165672	1653505021076	11:58:05	enoN	n egOxhfEafc
4	35.86	102727	1623210037266	12:11:41	oNne	cn xgfaeEOfh

## Volumen por Exchange (sandBox)

En consonancia con el dato de los mayores trades del día y chequeado contra los exchanges donde se produjeron, esa data nos viene muy util para combinarlo por ejemplo con el share o participación de un ticker por exchange total de la jornada y su comparación contra el promedio.

Recordemos que estamos usando el sandbox porque es un endpoint pago y la idea es mostrar las herramientas pero no tener que obligarlos a gastar en un proveedor de datos para probarlas. El sandbox mezcla las letras de los strings y deforma un poco los numeros con leves variaciones para que la data provista no sea real pero a fines didácticos nos sirve bien

Bueno, les muestro la implementación de una función que consulte el volumen diario por exchange, si ven lo que nos devuelve está no solo el "share" que los llaman "marketPercent" sino

que muestra el promedio lo cual me permitiría inferir en el caso de un movimiento atípico del activo, si hay un comprador concreto indidiendo demasiado en el mismo

```
def exchangeShare(symbol):
    url = 'https://sandbox.iexapis.com/stable/stock/'+symbol+'/volume-by-venue'
    params = {'token':token_sandbox}
    r = requests.get(url, params)
    js = r.json()
    df = pd.DataFrame(js)
    return df
data = exchangeShare('TWTR')
data.sort_values('marketPercent', ascending=False).head()
```

	volume	venue	venueName	date	marketPercent	avgMarketPercent
13	10209492	FRT	fcnhxearf gOE	2020-06-05	0.436053	0.458307
12	4226767	SYXN	ESYN	2020-06-05	0.181343	0.190274
11	2803840	SXGN	aqasdN	2020-06-05	0.121625	0.106317
10	1571930	GEXD	DEXCbe Go	2020-06-05	0.066086	0.054807
9	1363408	XCAR	NcSEa YrA	2020-06-05	0.061152	0.065719

## Top 10 Holders (SandBox)

Esta llada a la API nos devuelve los 10 tenedores del papel mayoritarios

```
def holders(symbol):
    url = 'https://sandbox.iexapis.com/stable/stock/'+symbol+'/fund-ownership'
    params = {'token':token_sandbox}
    r = requests.get(url, params)
    js = r.json()
    df = pd.DataFrame(js)
    return df
data = holders('GGAL')
data
```

	report_date	entityProperName	adjHolding	adjMv	reportedHolding	reportedMv
0	1023695807919	mgloGiMeE. k dcawgrthntuF senrrn	8945840	13269320	8936918	13154043
1	1002274087186	enrinAFut angd	5935079	8677563	5963342	8631786
2	1588122827760	tot ieohsbo( )gnntcSIAbn qeortiEoP yPsilluffpi...	4606733	9108753	4509564	9109111
3	1010248027856	HuyhuuDd sM(bseadtn(s )nerni iFirUi hEtc )Egkg...	2405514	3479406	2369950	3387782
4	1159117899603	nasuedeM IV ga tDnlmufEkner gsmroilainaFeD	2126279	1564011	2068788	1537976
5	1154318512953	t nsAne2N cc-rHrcimem er Eeds° LtggEo h. o oo...	1902479	1386262	1909192	1386840
6	1079198559313	,vv SPiicdnraea	1728490	363076	1742712	354404
7	1069375212606	am kengarIldcEngdMo Suxtu akr srFdN engVet	1183840	144663	1196107	144456
8	1082576958350	S,saalavdcRG a	966455	262603	985992	262074
9	1377547392229	kni ncBrTr sutkescmrelosl otaPven ctrlFt	952321	878604	975173	853367

## Estimaciones de EPS (SandBox)

Usualmente muchas consultoras y analistas de equipos de research publican estimaciones de las ganancias de las empresas, si bien es info cuantitativa, en realidad surgen de análisis cualitativos, por lo que tomamos entre comillas estos datos, pero como siempre digo, son datos, pueden servirnos para buscar correlacionar, para testear escenarios pre y post presentación de ganancias etc, no necesariamente tomar un dato significa usarlo directamente para tomar una decisión

Un potencial uso de este tipo de información sería recolectar datos de estimaciones y buscar correlación entre las siguientes variables:

```
# Diferencia EPS estimado vs EPS anterior  
# Volatilidad previa a presentación de ganancias
```

Esta correlación puede servirnos para tradear volatilidad en estrategias de opciones por ejemplo

Bueno, dicho eso, veamos la implementación en su sandbox, ya que también es un dato que para acceder a su dato real necesitamos una suscripción paga

```
def estimates(symbol):  
    url = 'https://sandbox.iexapis.com/stable/stock/'+symbol+'/estimates'  
    params = {'token':token_sandbox}  
    r = requests.get(url, params)  
    js = r.json()  
    df = pd.DataFrame(js['estimates'])  
    return df  
data = estimates('YPF')  
data
```

	consensusEPS	announceTime	numberOfEstimates	reportDate	fiscalPeriod	,
0	-0.5	MCA	6	2020-08-06	Q2 2020	

Lo que me devuelve es el promedio de las estimaciones y el número de estimaciones, recordemos que son estimaciones del EPS, es decir de la ganancia por acción, en el caso que el consenso es que la empresa dará pérdidas el EPS medio será negativo

## Estimaciones de Precio Target (Sandbox)

Esto también les pondría varias comillas, ya que si bien son datos cuantitativos, es decir un precio target, en realidad surgen de análisis subjetivos, pero bueno, nuevamente, son datos, cada uno sabrá como usarlos y como no usarlos

```
def priceTarget(symbol):
    url = 'https://sandbox.iexapis.com/stable/stock/' + symbol + '/price-target'
    params = {'token': token_sandbox}
    r = requests.get(url, params)
    js = r.json()
    return js
data = priceTarget('YPF')
data
```

```
{'symbol': 'YPF',
 'updatedDate': '2020-06-02',
 'priceTargetAverage': 6.8,
 'priceTargetHigh': 12.9,
 'priceTargetLow': 2.5,
 'numberOfAnalysts': 13,
 'currency': 'USD'}
```

Si lo quisieramos en un DataFrame, podríamos hacer algo así:

```
df = pd.DataFrame(data.values(), index=data.keys(), columns=['Valores'])
df
```

Valores	
symbol	YPF
updatedDate	2020-06-02
priceTargetAverage	6.8
priceTargetHigh	12.9
priceTargetLow	2.5
numberOfAnalysts	13
currency	USD

Obviamente en este caso no tiene sentido tener en un DataFrame datos sueltos de clave-valor, pero no viene mal repasar un poco cada tanto como manipular estructuras de datos

## Collections: Listas, Sectores, Tags

Las colecciones, son llamadas a un endpoint que nos devuelve múltiples valores de precios, algo parecido a "paneles", ahora bien estos "paneles" pueden ser de sectores (por ejemplo el tecnológico) de tipos de panel (los de mayor volumen en el dia) y directamente un custom search, con un tag que es mas específico que el sector

Las listas, sectores y tags nos permiten consultar datos de varios activos en un mismo llamado:

- Listas:
  - \* mostActive
  - \* gaines
  - \* losers
  - \* iexVolume \* iexPercent

- Sectores:
  - Electronic Technology
  - Distribution Services
  - Health Technology
  - Commercial Services
  - Industrial Services
  - Finance
  - Process Industries
  - Transportation
  - Technology Services
  - Producer Manufacturing
  - Retail Trade
  - Consumer Sevices
  - Non-Energy Minerals
  - Utilities
  - Miscellaneous
  - Health Services
  - Consumer Durables
  - Consumer Non-Durables
  - Communications
  - Energy Minerals
  - Government

- Tags

Son muchos (154 al dia de hoy) por lo que no los voy a poner acá sino que abajo les dejo como obtener su listado llamando justamente a la API

Veamos como es el endpoint y su impleentación

```
def panel(tipo, nombre):
    url = 'https://cloud.iexapis.com/stable/stock/market/collection/' + tipo
    params = {'token': token, 'collectionName': nombre}
    r = requests.get(url, params)
    js = r.json()
    df = pd.DataFrame(js)
    return df

panel = panel('sector', 'Retail Trade')
```

Empecemos por ver todo lo que me trae el panel:

```
panel.columns
```

```
Index(['symbol', 'companyName', 'primaryExchange', 'calculationPrice',  
'open', 'openTime', 'openSource', 'close', 'closeTime', 'closeSource',  
'high', 'highTime', 'highSource', 'low', 'lowTime', 'lowSource', 'latestPrice',  
'latestSource', 'latestTime', 'latestUpdate', 'latestVolume', 'iexRealtimePrice',  
'iexRealtimeSize', 'iexLastUpdated', 'delayedPrice', 'delayedPriceTime',  
'oddLotDelayedPrice', 'oddLotDelayedPriceTime', 'extendedPrice', 'extendedChange',  
'extendedChangePercent', 'extendedPriceTime', 'previousClose', 'previousVolume',  
'change', 'changePercent', 'volume', 'iexMarketPercent', 'iexVolume',  
'avgTotalVolume', 'iexBidPrice', 'iexBidSize', 'iexAskPrice', 'iexAskSize',  
'iexOpen', 'iexOpenTime', 'iexClose', 'iexCloseTime', 'marketCap', 'peRatio',  
'week52High', 'week52Low', 'ytdChange', 'lastTradeTime', 'sector'],  
dtype='object')
```

Y armamos un panel de precios por ejemplo de Comercio Minorista en dos renglones, como muestro en el siguiente ejemplo, o bien podria armar panel de puntas, o cualquier tipo de cálculo que me interese con cualquiera de las columnas mostradas

```
panelPrecios = panel.loc[:, ['symbol', 'open', 'high', 'low', 'close']]  
panelPrecios = panelPrecios.dropna().set_index('symbol').sort_index()  
panelPrecios
```

	open	high	low	close
symbol				
AAP	145.070	149.160	144.560	146.10
ABG	85.950	87.870	80.970	81.50
ADRNY	26.426	26.680	26.374	26.42
AEO	13.350	13.680	12.700	12.98
AMZN	2444.160	2488.645	2437.130	2483.00
...	...	...	...	...
YI	7.150	7.340	7.000	7.00
YJ	3.520	3.740	3.260	3.35
ZLDSF	71.700	71.700	71.700	71.70
ZLNDY	36.165	36.260	36.010	36.15
ZUMZ	30.500	31.450	29.010	29.46

295 rows × 4 columns

## Consulta de TAGS

Como son demasiados para no ponerles todos directamente les muestro como chequearlos directo a la API, que obviamente tiene un endpoint para consultar los TAGs accesibles, y les recomiendo siempre buscar esa "data de referencia" directo en las APIs porque siempre se va actualizando, yo hoy acá les puedo decir que hay 10 commodities disponibles, pero mañana puede haber 40

Lo mismo en lugar de "tags" se puede usar "sectors" y devuelve los sectores disponibles para consultar en colecciones

```
def tags():
    url = 'https://cloud.iexapis.com/stable/ref-data/tags'
    params = {'token':token}
    r = requests.get(url, params)
    js = r.json()
    df = pd.DataFrame(js)
    return df
tags()
```

	name
0	Electronic Technology
1	Distribution Services
2	Commercial Services
3	Health Technology
4	Industrial Services
...	...
149	Supranational
150	General Government
151	Government
152	Province/State
153	Municipality

154 rows × 1 columns

## IPOs, Ofertas públicas iniciales (Sandbox)

Cuando una empresa empieza a cotizar en la bolsa por primera vez lanza su IPO que es la primera oferta de su equity, estos eventos son "el debut" en la bolsa de las empresas y se pueden listar con esta API con el siguiente endpoint

Para variar un poco les muestro una salida combinada, el JSON de la respuesta me devuelve dos diccionarios

- rawData
- viewData

Por lo tanto, vamos a sacar dos DataFrames de return, para ello asigno al return dos variables y ya, de este modo si llamaramos "response" a la variable a la cual asignamos la salida de la función, podríamos usar "response[0]" para el primer valor y "response[1]" para el segundo

Veamos el código:

```
def ipos():
    url = 'https://sandbox.iexapis.com/stable/stock/market/upcoming-ipos'
    params = {'token':token_sandbox}
    r = requests.get(url, params)
    js = r.json()
    raw, view = pd.DataFrame(js['rawData']), pd.DataFrame(js['viewData'])
    return raw, view
dataRaw, dataView = ipos()
```

```
dataView.loc[0]
```

```
Company      NHARNM CR.A LPEITNA
Symbol       RTNL
Price        00.101 0-$ .57
Shares       0,6,03501
Amount       ,00820,050
Float        ,90,40536
Percent      9252%.
Market       QapiCSa AANltD
Expected     2020-05-30
Name: 0, dtype: object
```

Y les muestro simplemente las columnas que trae la otra parte de la respuesta, ya que es un paquete de info mucho mas completo que n viene al caso imprimir totalmente:

```
dataRaw.columns
```

```
Index(['symbol', 'companyName', 'expectedDate', 'leadUnderwriters', 'underwriters',
       'companyCounsel', 'underwriterCounsel', 'auditor', 'market', 'cik', 'address', 'city',
       'state', 'zip', 'phon e', 'ceo', 'employees', 'url', 'status', 'sharesOffered', 'priceLo
       w', 'priceHigh', 'offerAmount', 'totalExpenses', 'sharesOverAlloted', 'shareholderShares',
       'sharesOutstanding', 'lockupPeriodExpiration', 'quietPeriodExpiration', 'revenue',
       'netIncome', 'totalAssets', 'totalLiabilities', 'stockholderEquity', 'companyDescription',
       'businessDescription', 'useOfProceeds', 'competition', 'amount', 'percentOffered'],
       dtype='object')
```

## Opciones (SandBox)

Para los fanáticos de las opciones esta es una de las pocas APIs que tiene data de opciones, lamentablemente no está disponible para cuentas gratuitas, pero para cuentas de usd 9 sí, así que les muestro el API call para opciones en sandbox

Nota al margen, si están interesados en APIs pagas de opciones para uso intensivo, les recomiendo la siguiente API: <https://eodhistoricaldata.com/>

Otra opción es usar la API de TDA Ameritrade que tiene data de opciones en tiempo real, en el tomo que viene que vemos APIs de brokers está esta API

Volviendo a nuestra API en cuestión les dejo las funciones para acudir a las llamadas, en este caso como cada ticker (subyacente) tiene varios vencimientos, vamos a tener que primero llamar a los vencimientos y luego consultar por un subyacente y un vencimiento concreto

```
def opcionesVenc(symbol):
    url = 'https://sandbox.iexapis.com/stable/stock/' + symbol + '/options'
    params = {'token': token_sandbox}
    r = requests.get(url, params)
    js = r.json()
    return js
venc = opcionesVenc('AAPL')
venc
```

['204411', '208811', '204040', '209227', '211960', '204623', '204457', '209086',  
'206875', '203492', '209265', '207295']

Ahora, si no hubiera usado el "sandbox" me hubiera devuelto vencimientos reales, pero como es un endpoint pago y el token que estoy usando para este libro es gratuito y la idea es mostrar herramientas gratuitas para que no tengan que gastar en servicios de terceros hasta no probar las herramientas, esos vencimientos están recortados y distorsionados, así que simplemente es para mostrarles que me devuelve todas las fechas en una lista, por ejemplo la de May-2019 sería 201905

Seguimos viendo, esto de las opciones, vamos ahora a llamar a todas las opciones de AAPL de Jun 2020, suponiendo que ya sabemos que cuenta con ese vencimiento, que efectivamente lo tiene

Veamos:

```
def opciones(symbol, vencimiento):
    url = 'https://sandbox.iexapis.com/stable/stock/' + symbol + '/options/' + vencimiento
    params = {'token': token_sandbox}
    r = requests.get(url, params)
    js = r.json()
    df = pd.DataFrame(js)
    return df
chain = opciones('GGAL', '202006')
```

Veamos las columnas

```
chain.columns
```

```
['symbol', 'id', 'expirationDate', 'contractSize', 'strike Price', 'closingPrice', 'side',  
'type', 'volume', 'openInterest', 'bid', 'ask', 'lastUpdated', 'isAdjusted']
```

Bueno en la variable "chain" tenemos toda la cadena de opciones con todos sus datos, contrato por contrato, podríamos con esto armar una visión de straddle (que sería una tabla con el call y put de cada base a la izquierda y derecha, o bien simplemente listar contratos, o listar solo precios puntas como si fuera un panel o bien armar una tabla de volúmenes, en fin, vamos a ver alguna de las cosas más comunes a modo de ejemplo:

Dadas las columnas armamos una tabla práctica para su lectura cómoda, por ejemplo:

Para ellos transformamos la columna de vencimiento de string a formato fecha para poder ponerla como índice y ordenar la tabla por fecha de expiración

```
tabla = chain.loc[:,['expirationDate','strikePrice','closingPrice','side','bid','ask']]  
tabla['expirationDate'] = pd.to_datetime(tabla.expirationDate, format='%Y%m%d')  
tabla = tabla.set_index('expirationDate').round(2).sort_index()  
tabla
```

	strikePrice	closingPrice	side	bid	ask
expirationDate					
2020-06-05	10.0	0.79	alcl	0.78	0.90
2020-06-06	2.5	5.30	alcl	7.40	8.30
2020-06-06	7.9	0.10	tup	0.05	0.15
2020-06-07	16.0	0.05	call	0.00	0.16
2020-06-09	10.0	0.77	utp	0.30	0.90
2020-06-09	2.6	0.05	upt	0.00	0.05
2020-06-13	7.9	2.86	alcl	2.40	2.99
2020-06-13	15.0	6.30	put	4.70	5.60
2020-06-15	12.9	3.80	ptu	2.10	2.86
2020-06-16	5.0	0.06	upt	0.00	0.05
2020-06-17	13.1	0.15	llac	0.10	0.20
2020-06-18	5.0	5.40	lac	5.00	5.90

Varias cosas a comentar, obviamente como es un llamado al sandbox los datos que devuelve están "distorsionados" por ejemplo, el "side" que indicaría si es un call o un put, están desordenadas las letras pero fácilmente podríamos arreglarlo, no viene al caso porque es solo para mostrar lo que nos devuelve la API ya que no tendría sentido arreglar eso con los datos que no son reales.

Por lo demás está claro lo que devuleve, la fecha de expiración, la base, el ultimo precio, las puntas bid y ask etc, en el siguiente ejemplo les filtro solo columnas de volumen e interés abierto

```
Vols = chain.loc[:,['expirationDate','strikePrice','side','volume','openInterest']]  
Vols['expirationDate'] = pd.to_datetime(Vols.expirationDate, format='%Y%m%d')  
Vols = Vols.set_index('expirationDate').round(2).sort_index()  
Vols
```

		strikePrice	side	volume	openInterest
expirationDate					
2020-06-05	10.0	alcl	250	4538	
2020-06-06	2.5	alcl	0	0	
2020-06-06	7.9	tup	5	1242	
2020-06-07	16.0	call	0	203	
2020-06-09	10.0	utp	24	515	
2020-06-09	2.6	upt	0	11	
2020-06-13	7.9	alcl	76	1615	
2020-06-13	15.0	put	0	14	
2020-06-15	12.9	ptu	0	3	
2020-06-16	5.0	upt	0	276	
2020-06-17	13.1	llac	52	434	
2020-06-18	5.0	lacl	0	109	

## Social Sentiment (SandBox)

Otra de las cosas que diferencian esta API del resto es que es la única que conozco al momento yo que tiene data de social sentiment, este endpoint chequea tweets que refieren a los tickers de las acciones y en función de una calificación de las palabras "bearish" y palabras "bullish" nos da un score de -1 (bajista maximo) a +1 (alcista maximo)

Vamos a pedir por el día en formato YYYYMMDD

```
def sentiment(symbol, day):  
    url = 'https://sandbox.iexapis.com/stable/stock/'+symbol+'/sentiment/daily/'+day  
    params = {'token':token_sandbox}  
    r = requests.get(url, params)  
    js = r.json()  
    return js  
sentiment('PBR','20200505')  
  
{'sentiment': 0.06873894024417357,  
'totalScores': 18,  
'positive': 0.87,  
'negative': 0.18}
```

# Treasuries

Bueno, entre tantas cosas que nos ofrece esta API nos da los precios de los treasuries, y sus series históricas

Los simbolos que tenemos son:

- \* DGS30: 30 Year constant maturity rate
- \* DGS20: 20 Year constant maturity rate
- \* DGS10: 10 Year constant maturity rate
- \* DGSS5: 5 Year constant maturity rate
- \* DGS2: 2 Year constant maturity rate
- \* DGS1: 1 Year constant maturity rate
- \* DGS6MO: 6 Month constant maturity rate
- \* DGS3MO: 3 Month constant maturity rate
- \* DGS1MO: 1 Month constant maturity rate

Valor Actual:

```
def treasuriesQuote(symbol):  
    url = 'https://cloud.iexapis.com/stable/data-points/market/'+symbol  
    params = {'token':token}  
    r = requests.get(url, params)  
    js = r.json()  
    return js  
treasuriesQuote('DGS30')
```

1.61

Serie histórica:

```
def treasuriesHistory(symbol, desde, hasta):  
    url = 'https://cloud.iexapis.com/stable/time-series/treasury/'+symbol  
    params = {'token':token, 'from':desde, 'to':hasta}  
    r = requests.get(url, params)  
    js = r.json()  
    df = pd.DataFrame(js)  
    df.updated = pd.to_datetime(df.updated, unit='ms')  
    df.set_index('updated', inplace=True)  
    return df  
treasuriesHistory('DGS10', desde='2020-06-01', hasta='2020-06-05')
```

value	id	source	key	subkey	date
updated					
2020-06-02 23:00:03	0.66	TREASURY	IEX Cloud	DGS10	NONE 1590969600000
2020-06-03 23:00:03	0.68	TREASURY	IEX Cloud	DGS10	NONE 1591056000000
2020-06-04 23:00:02	0.77	TREASURY	IEX Cloud	DGS10	NONE 1591142400000
2020-06-05 23:00:03	0.82	TREASURY	IEX Cloud	DGS10	NONE 1591228800000

## Otros Endpoints

Les dejo alguna referencia a la fecha de otros endpoints, todos para suscriptores pagos que tiene esta API, la mayoría son caros, no es info para fines didácticos sino para uso mas profesional:

- Commodities Energia (WTI, Brent, Gas, HeatingOil, jetFuel, Diesel, Propane)
- Wall Street Horizons (Eventos corporativos relevantes)
- Extract Alpha (Modelos cuantitativos de selección de opciones)
- Precision Alpha (Modelos de pricing de activos via Machine Learning)
- BRAIN Company's (Consultora proveedora de Data Sentiment pricings y forecasts)
- kaVout (Machine learning para estimación de probabilidades de outperformance etc)
- Audit Analytics (Auditorias de cambios en los directorios de mas de 10.000 empresas)
- ValuEngine (Recomendaciones de compra/venta por modelos cuantitativos)

## Panel Completo de IEX

Bueno al final quería llegar a este endpoint que me parece fantástico, te trae el último trade de todos los tickers que se comercializan en IEX y su cantidad y fecha y hora exactas

Armo la función para hacer el llamado y le paso la fecha a datetime y le seteo el índice

```
def panelIEX():  
    url = 'https://cloud.iexapis.com/stable/tops/last'  
    params = {'token':token}  
    r = requests.get(url, params)  
    js = r.json()  
    df = pd.DataFrame(js)  
    df.time = pd.to_datetime(df.time, unit='ms')  
    df.set_index('symbol', inplace=True)  
    return df  
panelIEX = panelIEX()  
panelIEX
```

	price	size		time
symbol				
CMO-E	24.520	24	2020-06-05	19:42:45.303
SMG	134.630	100	2020-06-05	19:59:45.965
QSY	79.440	100	2020-05-27	19:52:15.711
SAFT	80.000	5	2020-06-05	19:57:23.355
ALL-E	25.435	25	2019-09-03	19:11:02.372
...	...	...	...	...
AVNS	33.170	95	2020-06-05	19:59:17.992
RWM	35.030	295	2020-06-05	19:59:50.569
LBDC	2.435	100	2020-03-30	19:40:59.977
HCACU	10.450	100	2020-05-15	18:06:52.962
GTS	20.390	100	2020-06-05	19:59:51.293

9980 rows × 3 columns

## Libro IEX

Les dejo el libro de bids y asks de IEX de un ticker, les dejo tres observaciones interesantes:

- Es en tiempo real
- Al ser un call a los datos de IEX solo, no tiene consumo de mensajes el token lo cual lo hace ilimitado

```
def getBook(symbol):
    url = 'https://cloud.iexapis.com/stable/deep/book'
    params = {'token':token, 'symbols':symbol}
    r = requests.get(url, params)
    js = r.json()
    try:
        bids = pd.DataFrame(js[symbol]['bids']).loc[:,['size','price']]
        asks = pd.DataFrame(js[symbol]['asks']).loc[:,['price','size']]
        book = pd.concat([bids, asks], axis=1)
        book = book.fillna('').head()
    except:
        book = 'No se encontraron puntas'
    return book

book = getBook('SPY')
book
```

	size	price	price	size
0	500	321.18	321.23	500
1	500	321.15	321.24	200
2	500	321.13	321.25	200
3	500	321.10	321.26	700
4	200	321.09	321.28	500

El mismo endpoint acepta pasar más de un ticker al mismo tiempo, si es más de un ticker deben pasarse separados con coma, por ej: 'GGAL,YPF,BMA'

Obviamente mi implementación en ese caso no serviría porque se partió de la base de que se ingresaba uno solo, para adaptar mi código a varios tickers debería hacerse un Split del string separado por las comas y meter un for que recorra todos los tickers una vez devuelto el response de la API

Algo así:

```
t_str = 'AAPL,QQQ,SPY'
t_lis = t_str.split(',')
t_lis
['AAPL', 'QQQ', 'SPY']
```

# Ejercicios

1- Ir a la documentación de la API, chequear el endpoint de estadísticas básicas <https://iexcloud.io/docs/api/#key-stats> y generar un script que me recorra todos los ADRs de la lista tickers dada, y me devuelva en un gráfico de barras todos los Beta de los ADRs y el Beta Medio de los tickers dados

```
tickers = ['BBAR','BMA','CEPU','CRESY','EDN','GGAL','LOMA','PAM','SUPV','TEO','TGS','YPF']
```

2- Basados en el ejercicio anterior crear una función que directamente tome como argumentos la lista de tickers y la clave de la estadística a graficar y me genere el gráfico de esa clave, por ejemplo "year5ChangePercent" o cualquier otra clave de las estadísticas siguientes:

3- Usando el Endpoint de las colecciones, buscar las acciones mas ganadoras de la última rueda e imprimir la tabla con los siguientes datos:

- r El ticker de la acción
- r El nombre de la compañía
- r El precio de cierre de la rueda anterior
- r El precio de cierre final
- r La variación porcentual

Mostrar la tabla con los valores redondeados a dos decimales y ordenad a por mayor variación diaria

4- Usar el endpoint last de IEX para armar un panel de los ADRs argentinos:

```
tickers = ['BBAR','BMA','CEPU','CRESY','EDN','GGAL','LOMA','PAM','SUPV','TEO','TGS','YPF']
```

# Respuestas

```
#-----#
# Rta Ejercicio 1 #
#-----#

import matplotlib.pyplot as plt

def keyStats(symbol,stat):
    url = 'https://cloud.iexapis.com/stable/stock/'+symbol+'/stats/'+stat
    params = {'token':token}
    r = requests.get(url, params)
    return float(r.json())

tickers = ['BBAR','BMA','CEPU','CRESY','EDN','GGAL','LOMA','PAM','SUPV','TEO','TGS','YPF']

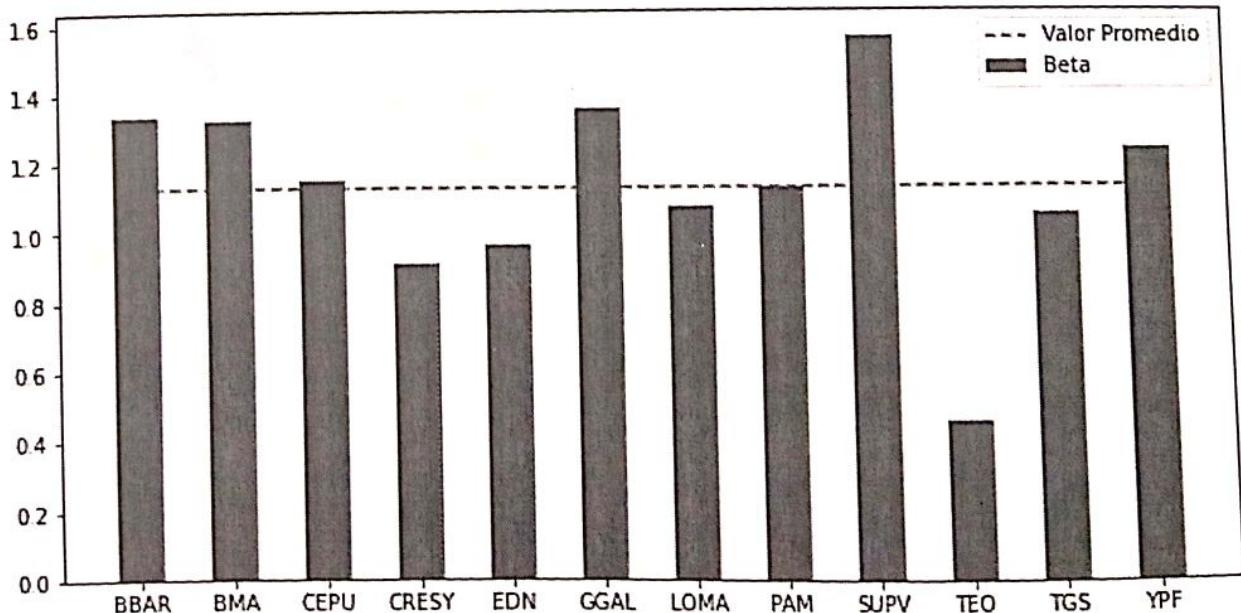
dic = {}
for ticker in tickers:
    print(ticker, end='.. ')
    dic[ticker] = keyStats(ticker,'beta')

media = sum(dic.values())/len(dic.values())

fig, ax = plt.subplots(figsize=(10,5))
ax.bar(dic.keys(), dic.values(), label='Beta', width=0.5)
ax.plot([media]*len(dic), label='Valor Promedio', ls='--')
ax.legend(loc='upper right')

plt.show()
```

BBAR.. BMA.. CEPU.. CRESY.. EDN.. GGAL.. LOMA.. PAM.. SUPV.. TEO.. TGS.. YPF..



```

#-----#
# Rta Ejercicio 2 #
#-----#


import matplotlib.pyplot as plt

def keyStats(symbol,stat):
    url = 'https://cloud.iexapis.com/stable/stock/' + symbol + '/stats/' + stat
    params = {'token': token}
    r = requests.get(url, params)
    return float(r.json())

def graficar(tickers, clave):
    dic = {}
    for ticker in tickers:
        print(ticker, end='.. ')
        dic[ticker] = keyStats(ticker, clave)

    media = sum(dic.values()) / len(dic.values())

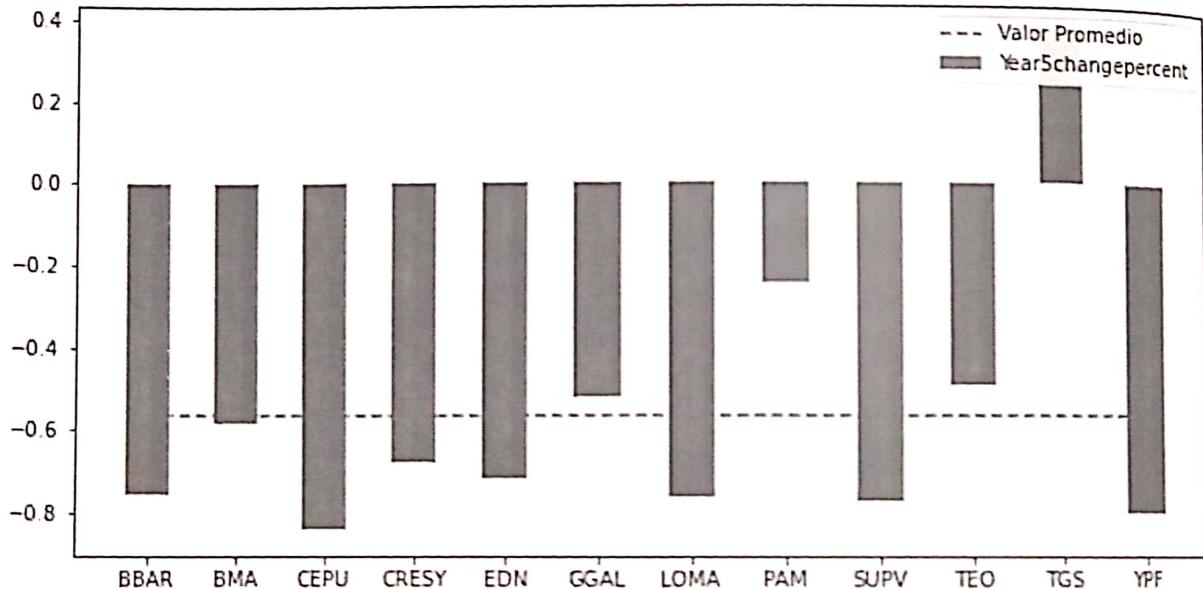
    fig, ax = plt.subplots(figsize=(10, 5))
    ax.bar(dic.keys(), dic.values(), label=clave.capitalize(), width=0.5)
    ax.plot([media] * len(dic), label='Valor Promedio', ls='--')
    ax.legend(loc='upper right')

    plt.show()

tickers = ['BBAR', 'BMA', 'CEPU', 'CRESY', 'EDN', 'GGAL', 'LOMA', 'PAM', 'SUPV', 'TEO', 'TGS', 'YPF..']
graficar(tickers, 'year5ChangePercent')

```

BBAR.. BMA.. CEPU.. CRESY.. EDN.. GGAL.. LOMA.. PAM.. SUPV.. TE O.. TGS.. YPF..



```

#-----#
# Rta Ejercicio 3  #
#-----#


def panel(tipo, nombre):
    url = 'https://cloud.iexapis.com/stable/stock/market/collection/' + tipo
    params = {'token': token, 'collectionName': nombre}
    r = requests.get(url, params)
    js = r.json()
    df = pd.DataFrame(js)
    return df

panel = panel('list', 'gainers')

panel = panel.loc[:, ['symbol', 'companyName', 'previousClose', 'latestPrice', 'latestTime']]
panel = panel.dropna().set_index('symbol').sort_index()
panel['Variación'] = (panel.latestPrice / panel.previousClose - 1) * 100
panel.round(2).sort_values('Variación', ascending=False)

```

		companyName	previousClose	latestPrice	latestTime	Variación
symbol						
GCI		Gannett Co., Inc.	1.39	2.81	June 5, 2020	102.16
CHK		Chesapeake Energy Corp.	14.05	24.80	June 5, 2020	76.51
HTZ		Hertz Global Holdings, Inc.	1.50	2.57	June 5, 2020	71.33
QEP		QEP Resources, Inc.	1.08	1.78	June 5, 2020	64.81
NBR		Nabors Industries Ltd.	43.43	65.62	June 5, 2020	51.09
RIG		Transocean Ltd.	1.66	2.50	June 5, 2020	50.60
OXY		Occidental Petroleum Corp.	15.55	20.79	June 5, 2020	33.70
SM		SM Energy Co.	4.20	5.49	June 5, 2020	30.71

```

#-----#
# Rta Ejercicio 4 #
#-----#

def panelIEX():
    url = 'https://cloud.iexapis.com/stable/tops/last'
    params = {'token':token}
    r = requests.get(url, params)
    js = r.json()
    df = pd.DataFrame(js)
    df.time = pd.to_datetime(df.time, unit='ms')
    df.set_index('symbol', inplace=True)
    return df

panelIEX = panelIEX()

tickers = ['BBAR', 'BMA', 'CEPU', 'CRESY', 'EDN', 'GGAL', 'LOMA', 'PAM', 'SUPV', 'TEO', 'TGS', 'YPF']
ADRs = panelIEX.loc[tickers]
ADRs

```

	price	size	time
<b>symbol</b>			
BBAR	4.085	100	2020-06-05 19:57:44.002
BMA	21.590	100	2020-06-05 19:59:33.025
CEPU	2.795	100	2020-06-05 19:56:21.180
CRESY	4.015	400	2020-06-05 19:52:39.323
EDN	4.100	200	2020-06-04 16:46:25.404
GGAL	10.360	100	2020-06-05 19:59:44.406
LOMA	5.090	400	2020-06-05 19:55:56.502
PAM	11.430	100	2020-06-05 19:59:51.796
SUPV	2.600	100	2020-06-05 19:59:50.466
TEO	10.030	100	2020-06-05 19:58:57.023
TGS	6.220	100	2020-06-05 19:58:34.532
YPF	5.740	100	2020-06-05 19:59:55.640

# API abierta de la reserva Federal

Bueno, les dejo a los que quieran investigar mas, esta API, básicamente es la api de info abierta de la reserva federal, informan de todo, características:

- # Es muy completa (datos económicos, financieros, de auditorías y mercado
  - \* Deuda
  - Presupuesto
  - Securities varios
    - \* Bills (letras corto plazo < 1yr) \* Notes (6meses a 10 años)
    - \* Bonds (30yr)
  - Hoja de balance de la reserva federal
  - Todo tipo de data financiera interna, hasta el rendimiento de fondo de desempleo
- \* Funciona espectacularmente bien
- \* No tiene límites de requests
- \* No tiene tokens ni autenticación
- \* Es 100% gratis

El link general de acceso a la API es:

- <https://www.transparency.treasury.gov/>

Para acceder a sus datos vamos a importar las librerías de siempre:

```
# import requests, pandas as pd
```

## Tipos de interés promedio:

Este endpoint nos trae datos de las tasas de interés de la reserva federal

Les dejo función de como llamar a la API básica:

```
1 def getRates(limit=100):  
2     base = 'https://www.transparency.treasury.gov/services/api/fiscal_service/v1/'  
3     url = base + 'accounting/od/avg_interest_rates'  
4     params = {'sort': '-reporting_date', 'limit': limit}  
5     r = requests.get(url, params=params)  
6     js = r.json()  
7     df = pd.DataFrame(js['data'])  
8     return df  
9 data = getRates()
```

Como pueden ver, acepta como parámetro el orden y la cantidad "limit" de datos a devolver

Con la siguiente consulta vamos a traer los ultimos valores:

```
r data = getRates()
2 ultimos = data.groupby('security_desc').avg_interest_rate_amt.first()
3 ultimos
```

security_desc	
Domestic Series	8.043
Federal Financing Bank	2.685
Foreign Series	7.312
Government Account Series	2.421
Government Account Series Inflation Securities	1.257
Special Purpose Vehicle	0.090
State and Local Government Series	1.630
Total Interest-bearing Debt	1.983
Total Marketable	1.843
Total Non-marketable	2.413
Treasury Bills	0.403
Treasury Bonds	3.716
Treasury Floating Rate Note (FRN)	0.287
Treasury Inflation-Protected Securities (TIPS)	0.741
Treasury Notes	2.045
United States Savings Inflation Securities	3.626
United States Savings Securities	2.967
Name: avg_interest_rate_amt, dtype: object	

Y con el siguiente script vamos a graficar una serie, en este caso la tasa de las letras cortas, para ello debemos hacer varias cosas, en primer lugar vamos a armar una tabla solo de las tasas:

```
# data = getRates(7000)
# letras = data.loc[data.security_desc=='Treasury Bills']
0 letras = letras.loc[:,['reporting_date','avg_interest_rate_amt']]
4 letras
```

	reporting_date	avg_interest_rate_amt
3	2020-05-31	0.403
20	2020-04-30	0.596
36	2020-03-31	1.216
52	2020-02-29	1.643
68	2020-01-31	1.683
...	...	...
3664	2001-05-31	4.560
3680	2001-04-30	5.076
3696	2001-03-31	5.369
3712	2001-02-28	5.755
3727	2001-01-31	6.059

233 rows x 2 columns

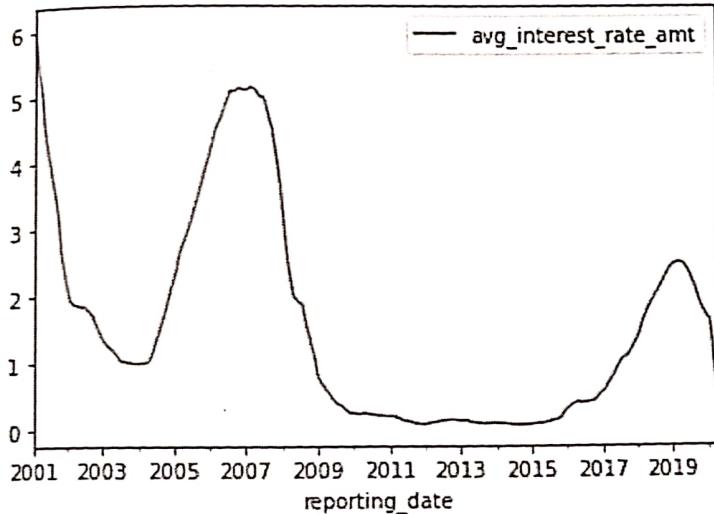
Bien, el tema ahora es que para graficarlo rápido vamos a querer que

La fecha esté en el índice

La fecha sea un datetime

El valor de la tasa sea numérico

```
1 # Seteamos el índice
2 letras.set_index('reporting_date', inplace=True)
3
4 # Pasamos el índice a datetime
5 letras.index = pd.to_datetime(letras.index)
6
7 # Nos aseguramos que la tasa sea numérica
8 letras.avg_interest_rate_amt = pd.to_numeric(letras.avg_interest_rate_amt)
9
10 letras.plot()
```



## Deuda externa

Entre varias de los endpoints nos ofrecen uno interesante con la deuda externa, se los muestro:

```
1 import requests, pandas as pd
2 def deuda(limit=10000):
3     base = 'https://www.transparency.treasury.gov/services/api/fiscal_service/v1/'
4     url = base + 'accounting/od/debt_to_penny'
5     params = {'sort': '-data_date', 'limit': limit}
6     r = requests.get(url, params=params)
7     js = r.json()
8     df = pd.DataFrame(js['data'])
9     return df
10
11 deuda = deuda(50)
12
13 # Muestro 5 filas y 5 columnas nomás
14 deuda.iloc[:, :6].head()
```

	data_date	debt_held_public_amt	intragov_hold_amt	tot_pub_debt_out_amt	reporting_calendar_year
0	2020-07-17	20612219680437.71	5920145726911.40	26532365407349.11	2020
1	2020-07-16	20611921079550.83	5917339138678.45	26529260218229.28	2020
2	2020-07-15	20574380830507.80	5911091609553.12	26485472440060.92	2020
3	2020-07-14	20539873335812.51	5909778434443.54	26449651770256.05	2020
4	2020-07-13	20583461928475.00	5904639271901.64	26488101200376.64	2020

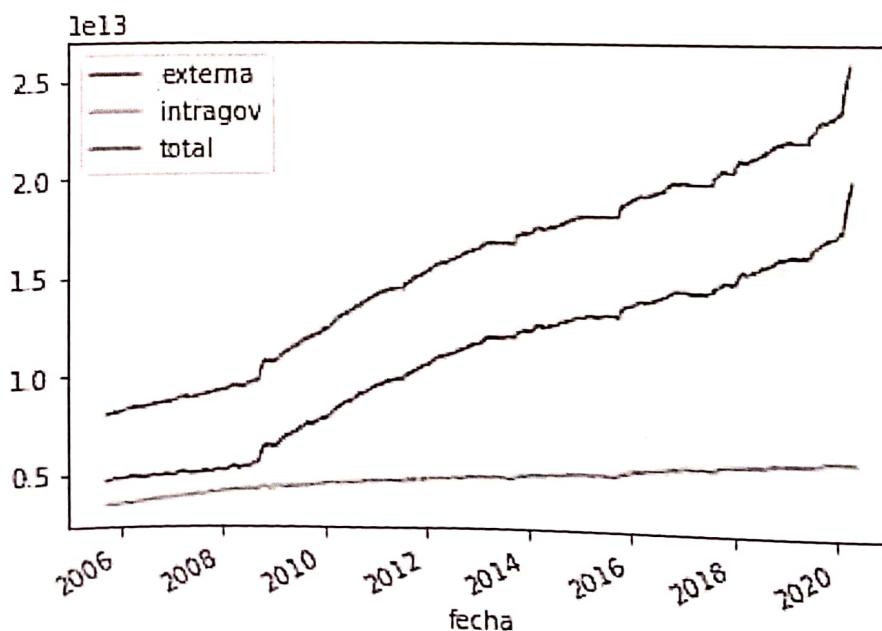
## Ejercicios

- 1- Hacer un script que tenga una función que me devuelva la deuda total intra y extra gubernamental, que este con el indice de las serie de tiempo y los datos en formato numperico y al final usando la función obtener los datos y graficar la deuda intra gov y total
- 2- Hacer un script que me grafique la serie historica de las notas y las letras del tesoro pero hacerlo de modo tal que me permita ingresar una lista de claves donde por ejemplo pueda ingresar 'Foreign Series' o cualquier otro y me lo agregue al gráfico sin cambiar nada mas

```

1  #-----#
2  # Rta Ejercicio 1 #
3  #-----#
4
5      # Importo Las Librerías
6      import requests, pandas as pd
7
8  def deuda(limit=10000):
9
10     # Preparo el request
11     base = 'https://www.transparency.treasury.gov/services/api/fiscal_service/v1/'
12     url = base + 'accounting/od/debt_to_penny'
13     params = {'sort': '-data_date', 'limit': limit}
14
15     # Hago el request y Lo paso a un DataFrame
16     r = requests.get(url, params=params)
17     js = r.json()
18     df = pd.DataFrame(js['data'])
19
20     # Tomo solo las columnas que necesito
21     df = df.iloc[:, [0, 1, 2, 3]]
22
23     # Renombro las columnas
24     df.columns = ['fecha', 'externa', 'intragov', 'total']
25
26     # Seteo el indice
27     df.set_index('fecha', inplace=True)
28
29     # Paso el indice a formato datetime
30     df.index = pd.to_datetime(df.index)
31
32     # Transformo a numerico los valores de las columnas de deuda
33     df = df.apply(pd.to_numeric)
34
35     return df
36
37 deuda().plot()

```



```

#-----#
# Rta Ejercicio 2 #
#-----#

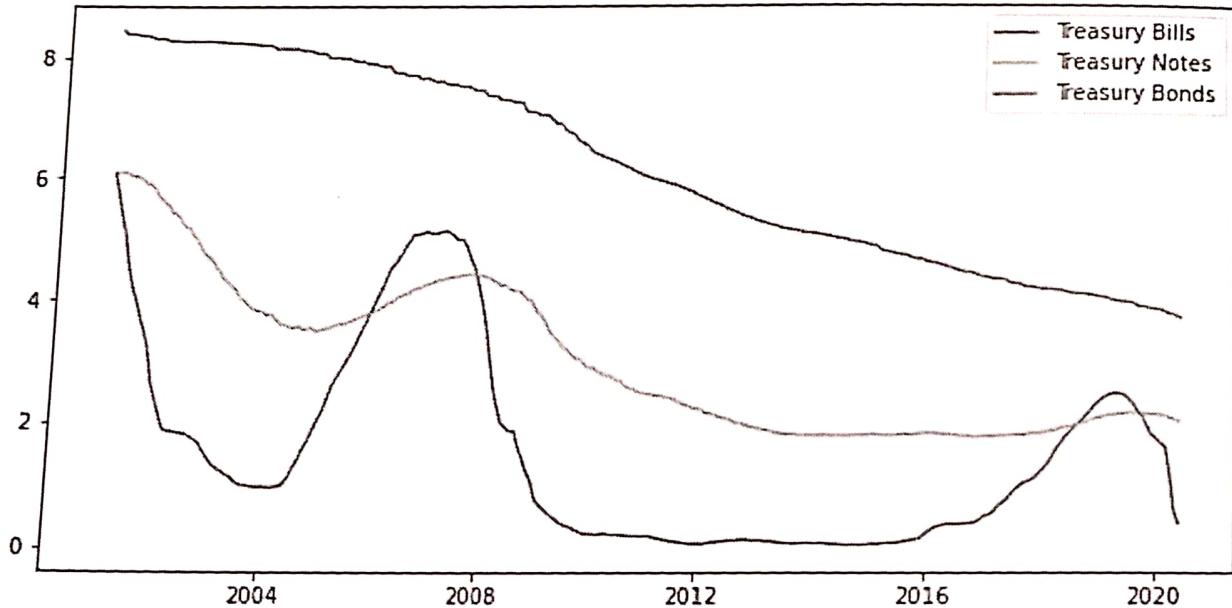
def getRates(limit=100):
    base = 'https://www.transparency.treasury.gov/services/api/fiscal_service/v1/'
    url = base + 'accounting/od/avg_interest_rates'
    params = {'sort': '-reporting_date', 'limit': limit}
    r = requests.get(url, params=params)
    js = r.json()
    df = pd.DataFrame(js['data'])
    return df

data = getRates(10000)
fig, ax = plt.subplots(figsize=(10,5))

tipos = ['Treasury Bills', 'Treasury Notes', 'Treasury Bonds']
for tipo in tipos:
    serie = data.loc[data.security_desc==tipo]
    serie = serie.loc[:,['reporting_date','avg_interest_rate_amt']]
    serie.set_index('reporting_date', inplace=True)
    serie.index = pd.to_datetime(serie.index)
    serie.avg_interest_rate_amt = pd.to_numeric(serie.avg_interest_rate_amt)
    ax.plot(serie, label=tipo)

ax.legend(loc='upper right')
plt.show()

```



## Lectura directa de CSVs

Bueno, antes de pasar al webScraping y ese tipo de cosas, vemos un paso previo interesante, muchas veces necesitamos automatizar datos o trabajar con series de datos que andan en la web pero que no tienen APIs desarrolladas, pero sin embargo a veces tenemos los links donde están los archivos CSV o excels, de ser ese el caso podríamos directamente bajar el data set con pandas y sus métodos "read\_csv" o "read\_excel", incluso con "read\_html" si lo tuviera en una tabla en la pagina web

Veamos por ejemplo la página datahub.io tiene muy buena data almacenada en archivos que podemos leer como si nos conectarnos a una API tan sencillo como eso

Para eso ni siquiera debemos importar la Librería requests porque la conexión la hace pandas directamente, vean que tan sencillo es el ejemplo que pongo a continuación, en este caso voy a leer un archivo en linea con los datos mensuales desde 1950 de los precios del ORO, vean:

```
3 import pandas as pd  
2 url = 'https://datahub.io/core/gold-prices/r/monthly.csv'  
3 data = pd.read_csv(url)  
4 data
```

	Date	Price
0	1950-01	34.730
1	1950-02	34.730
2	1950-03	34.730
3	1950-04	34.730
4	1950-05	34.730
...	...	...
840	2020-01	1560.668
841	2020-02	1598.818
842	2020-03	1593.764
843	2020-04	1680.030
844	2020-05	1715.697

845 rows × 2 columns

Lo mismo puedo leer por ejemplo en este caso una serie diaria del precio spot del WTI que tiene datos desde 1986

Muchas veces puede ser que tengamos que recurrir a este tipo de fuente de datos para series no actualizables día a día pero que sí necesitamos varios años hacia atrás y muchas APIs por cuestiones de seteo no tienen más de 20 años de data

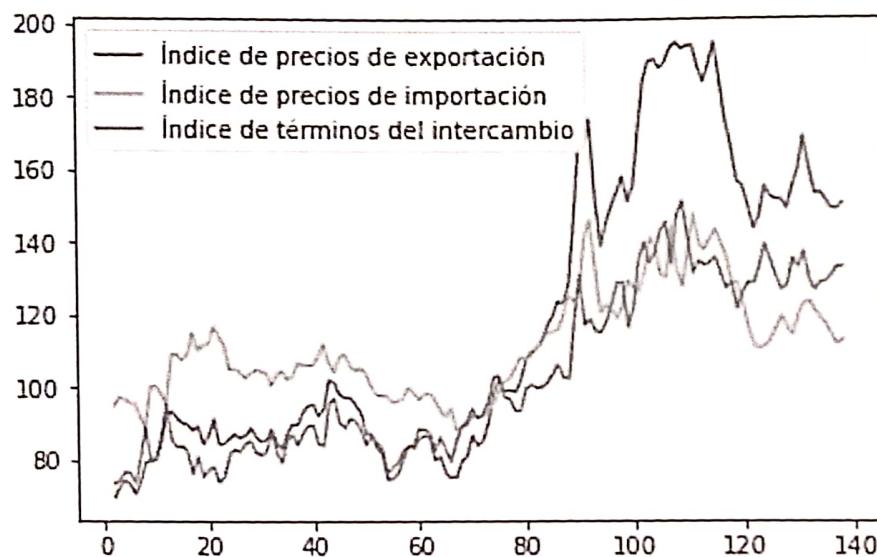
```
1 url = 'https://datahub.io/core/oil-prices/r/wti-daily.csv'  
2 data = pd.read_csv(url)  
3 data.head()
```

	Date	Price
0	1986-01-02	25.56
1	1986-01-03	26.00
2	1986-01-06	26.53
3	1986-01-07	25.85
4	1986-01-08	25.87

Pongo otro ejemplo de la base de datos del INDEC Argentina, términos de intercambio

En este caso la ploteamos directamente

```
# url = 'https://www.indec.gob.ar/ftp/cuadros/economia/sh_indextermint_04.xls'  
2 data = pd.read_excel(url, skiprows=3, skipfooter=6).dropna()  
3 data.plot()
```



O mostramos la serie, es trimestral en este caso, aranca de 1986

```
3 data.head()
```

	Periodo	Índice de precios de exportación	Índice de precios de importación	Índice de términos del intercambio
2	1986 1º trim.	69.8	94.9	73.551106
3	2º trim.	72.7	97.6	74.487705
4	3º trim.	74.5	97.0	76.804124
5	4º trim.	73.4	95.8	76.617954
6	1987 1º trim	70.6	95.8	73.695198

# Websockets ¶

Para cerrar un poco este tema del data feed por ahora antes de ver librerías, vamos a ver los websockets

Hasta ahora venimos haciendo peticiones http y solo traímos una vez lo que nos devolvía la petición y ya

Vamos a trabajar ahora con lo que se conoce como "WebScocket" en realidad esto es algo así como abrir una puerta y dejar que la info fluya hacia nosotros en un tiempo continuo, en lugar de hacer un llamado http como veníamos haciendo, lo que hacemos es abrir una puerta, por lo tanto en lugar de recibir de una tabla estanca, vamos a ir recibiendo datos continuamente segundo a segundo o mejor dicho milisegundo a milisegundo

## ¿Como funciona un Websocket?

Funcionan escuchando/enviando eventos, lo primero que debemos entender es que es un evento

Decimos que se produce un evento cuando hay alguna acción que genera un cambio de estado de alguna variable o inut/output de un proceso que llevado a nuestro campo de acción podría ser por ejemplo cuando cambia un precio de una acción, es decir cuando AAPL pasa de valer 320.44 a 320.45

Bien, pero solo eso?

Por ejemplo que pasa si se ejecutan dos trades seguidos a \$ 320.44 , ahí no cambió el precio, sin embargo está ocurriendo un evento, en este caso un trade, y si bien no cambia el precio, está cambiando el volumen operado, que si bien puede que no sea un output en mi set de datos, es una variable que está siendo modificada en algún lado

De la misma manera hay eventos mas silenciosos, fíjense que en el mismo ejemplo que dije antes, que se operen dos veces seguidas a \$320.44 acciones de AAPL, al suceder eso están modificándose los "sizes" de los bid y ask, es decir se está modificando el libro ya que si el comprador le pegó a la punta de venta, la cantidad de la punta del ask va a haber disminuido, y viceversa si el vendedor le daba a la punta de compra

Pero que pasa si cotizando en 320.44 se pone un comprador en la punta de compra a 320.43 y un vendedor en la punta de venta a 320.45, en ambos casos si no hay mas participantes no se produce ningún trade, pero si un evento, el evento es que se están modificando las puntas de compra y venta cuando entran las órdenes de comprador y vendedor, aunque ninguna de las dos se ejecute

No se si ya notaron el punto, la gran diferencia con consumir datos de una API rest, es que en el websocket vamos a estar "escuchando eventos" en forma continua

Esto es un gran cambio, porque antes descargábamos los datos hasta cierto momento y chau, ahí moría nuestra conexión, en cambio con el websocket vamos abrir una puerta de comunicación y vamos a estar escuchando todo el tiempo

## ¿Y para que sirve estar conectado a un websocket?

Pensemos lo siguiente: Si yo quisiera gatillar mi orden de trading cuando el precio toque tal valor, ¿que debo hacer?

Parece muy sencilla la pregunta, y lo primero que decimo es: Esperar que el valor sea ese que predefinimos y gatillar la orden

Genial pero ¿cuanto espero? ¿cada cuanto averiguo si toco o no tocó ese precio?

Claro, podríamos decir cada un minuto hago una consulta a una API que me de datos por minuto, fenómeno, pero y si toca ese precio en el medio entre un minuto y otro?

Ok, en ese caso me dirán, no, me conecto a una API con precios en tiempo real, y consulto el precio cada un segundo

Fenómeno, ¿y si entre un segundo y otro el precio tocó ese valor y se dispara por esperar mas de medio segundo hasta la siguiente llamada nuestra a la API?

Me dirás, no pasa nada, si se dispara no se opera y ya

¿Y si en lugar de gatillar una entrada es un gatillo de stop loss? Es decir, por esperar ese medio segundo hasta el siguiente llamado a la API mi stoploss en vez de ejecutarse en un valor se termina ejecutando en otro

Bueno, para eso sirven los websocket y para mucho mas obvio.. Es decir, en vez de nosotros cada "x" tiempo ur a buscar un precio, lo que hacemos es abrir una conexión y esperar que el precio venga a nosotros

Esto se usa para algoritmos de market making, para algoritmos de hedging para algoritmos de arbitrajes, y para partes de un bot de trading como puede ser el gatillo de un stopLoss

## Conectándose a un websocket

Bueno, basta de palabras, vamos a conectarnos y recibir información de un websocket

Para ello vamos a sacar una cuenta en la API tiingo visitando [api.tiingo.com](https://api.tiingo.com/) (Elijo este recurso porque no tiene limitaciones para websocket y es a fines didácticos, a fines de trading real, obviamente deberíamos buscar una fuente directa del broker con el que operemos, ya voy a explicar bien por qué)

- <https://api.tiingo.com/> (<https://api.tiingo.com/>)

Bueno, yo ya saqué un token de prueba que puedo poner aquí para fines didácticos

```
1 token = 'a5e16af3958b5c8d88d63c84fb9fc9f0d1be4434'
```

Luego lo que vamos a hacer es importar la librería websocket

```
1 import websocket as ws
```

Y ahora si nos conectamos, explico los 4 pasos que hay que dar:

Creamos una conexión nueva al webSocket, la guardamos en la variable en nuestro ejemplo la vamos a llamar "conn"

Creamos un JSON que contiene 3 cosas:

El evento que vamos a producir en la comunicación al websocket: que es una suscripción a datos

La autorización, que sera nuestro token (lo recibimos cuando nos creamos la cuenta en tiigo)

Los datos del evento, (suscripción), para esto tenemos dos opciones:

thresholdLevel = 2 (Datos de ultimo trade operado)

thresholdLevel = 5 (Datos de la caja bid-ask, ultimo cambio)

Enviamos la solicitud de suscripción

Empezamos a recibir datos

```
1 # Creamos La conexión
2 conn = ws.create_connection("wss://api.tiingo.com/fx")
3
4 # Creamos el JSON con los datos de suscripcion
5 subscribe = '{"eventName":"subscribe","authorization":"' +token+ '","eventData"
6
7 # Nos suscribimos al websocket
8 conn.send(subscribe)
9
10 # Recibimos datos
11 conn.recv()
```

```
'{"messageType": "I", "response": {"code": 200, "message": "Success"}, "data": {"subscriptionId": 86917}}'
```

Bien, ya está, ese mensaje que vemos es el primer mensaje que leemos cuando nos conectamos  
Y si volvemos a pedir nos vuelve a mandar, tantas veces como solicitemos info

```
1 conn.recv()
```

```
'{"data": ["Q", "cadchf", "2020-06-09T01:11:03.481784+00:00", 364000.0, 0.715575, 364000.0, 0.7157], "service": "fx", "messageType": "A"}'
```

```
1 conn.recv()
```

```
'{"data": ["Q", "cadjpy", "2020-06-09T01:11:03.496324+00:00", 910000.0, 80.745, 910000.0, 80.76], "service": "fx", "messageType": "A"}'
```

Ahí ejecuté dos veces seguidas un pedido

- En la primera me mandó datos de un evento de Dolar Canadiense vs Franco Suizo a las 1:11:03 con 481 milisegundos
- En el segundo dato recibo un evento en Dolar Canadiense vs Yen Japones a 80.76 a las 1:11:03 con 496 milisegundos

Ojo que no se si los eventos fueron trades realizados o cambios en las putnas bid-ask

## Otros web sockets de esta misma API

Bueno, esta API esta muy piola porque me brinda un websocket no solo para Forex, sino tambien para cryptos y para equity es decir acciones cotizadas en IEX (international excanges)

- Pares de divisas: fx
- Pares de cryptos: crypto
- Equity: iex

Esos son los "endpoints" de los websockets, que serían en forma completa los siguientes:

- wss://api.tiingo.com/fx
- wss://api.tiingo.com/crypto
- wss://api.tiingo.com/iex

Cabe destacar que es mucho mas líquido el de cryptos por los bajos costos transaccionales, y además podemos probarlo a cualquier hora ya que el mercado crypto no para nunca

## Ciclos abiertos con websockets

Bueno, hasta ahora vimos como recibir un dato, pero vamos a meternos al menos un poco en el flujo de un algo que trabaje con websockets

Algo muy típico que vamos a ver siempre en los algoritmos de websockets es un **WHILE**, lo cual es lógico, porque vamos a seguir recibiendo info mientras pase algo por ejemplo mientras no se dispare el gatillo de compra o venta o stoploss etc..

```
1 import websocket as ws
2
3 # Creamos la conexión
4 conn = ws.create_connection("wss://api.tiingo.com/crypto")
5
6 # Creamos el JSON con los datos de suscripción
7 subscribe = '{"eventName":"subscribe","authorization":"'"+token+"","eventData"
8
9 # Nos suscribimos al websocket
10 conn.send(subscribe)
11
12 # Vamos a recibir 5 datos, y cuando eso este ok, cortamos
13 i = 0
14 while i<5:
15     result = json.loads(conn.recv())
16     print(result, '\n')
17     i = i+1
```

```
{"messageType": "I", "response": {"code": 200, "message": "Success"}, "data": {"subscriptionId": 86964}}
```

```
{"messageType": "H", "response": {"code": 200, "message": "HeartBeat"}}
```

```
{"data": ["T", "btcusd", "2020-06-09T01:55:50.425000+00:00", "binance", 0.060252, 9698.37], "service": "crypto_data", "messageType": "A"}
```

```
{"data": ["T", "iotabtc", "2020-06-09T01:55:50.491000+00:00", "binance", 21.0, 2.502e-05], "service": "crypto_data", "messageType": "A"}
```

```
{"data": ["T", "ambbtc", "2020-06-09T01:55:50.588000+00:00", "binance", 2669.0, 1.44e-06], "service": "crypto_data", "messageType": "A"}
```

Si observamos bien los dos primeros mensajes tienen una estructura y los otros 3 otra

Bien, los dos primeros mensajes son como de bienvenida, no de datos, es decir, nos informan el ID de nuestra suscripción y nos dan un mensaje de que se empiezan a recibir datos, nos interesa a partir del 3er mensaje

Pero eso solo pasa cuando inicializamos la suscripción, si yo ahora ejecutara de nuevo a partir de que la suscripción al websocket fue creada, ya no me aparecen más esos dos mensajes iniciales

```

1 i = 0
2 while i<5:
3     result = json.loads(conn.recv())
4     print(result,'\\n')
5     i = i+1

{'data': ['T', 'ethtrx', '2020-06-09T01:55:50+00:00', 'poloniex', 0.02477835, 13793.99117348], 'service': 'crypto_data', 'messageType': 'A'}

{'data': ['T', 'btcusd', '2020-06-09T01:55:50.720000+00:00', 'binance', 0.058473, 9698.37], 'service': 'crypto_data', 'messageType': 'A'}
```

{'data': ['T', 'maidusd', '2020-06-09T01:55:50.071000+00:00', 'hitbtc', 0.3, 0.123486], 'service': 'crypto\_data', 'messageType': 'A'}

{'data': ['T', 'dataeth', '2020-06-09T01:55:50.777000+00:00', 'binance', 12.0, 0.00024535], 'service': 'crypto\_data', 'messageType': 'A'}

{'data': ['T', 'smartbtc', '2020-06-09T01:55:50.597000+00:00', 'hitbtc', 405.0, 3.382e-07], 'service': 'crypto\_data', 'messageType': 'A'}

## Suscribiendo a otro websocket de la misma API

Bien, pero si ahora quisiera recibir info de un websocket de equity, por ejemplo de IEX, tenemos que conectar a otro websocket, y por lo tanto vamos a tener que recibir de nuevo esos mensajes de bienvenida. Para ello vamos a sobreescibir la variable "conn" que es la conexión al websocket, en este caso cambiamos crypto por iex

La variable "suscribe" la vamos a dejar igual por el momento

Nota: El mercado crypto opera 24x7, si prueban un websocket fuera de horario de equity no prueben el de iex porque no les va a devolver nada, prueben el de crypto

```

import websocket as ws

# Creamos la conexion
conn = ws.create_connection("wss://api.tiingo.com/iex")

# Creamos el JSON con los datos de suscripcion
subscribe = '{"eventName":"subscribe","authorization":"'"+token+"","eventData":'

# Nos suscribimos al websocket
conn.send(subscribe)

# Vamos a recibir 5 datos, y cuando eso este ok, cortamos
i = 0
while i<5:
    result = json.loads(conn.recv())
    print(result,'\\n')
    i = i+1
```

- output Código página anterior -

```
{'messageType': 'I', 'response': {'code': 200, 'message': 'Success'}, 'data': {'subscriptionId': 87183}}
```

```
{'messageType': 'H', 'response': {'code': 200, 'message': 'HeartBeat'}}
```

```
{'data': ['Q', '2020-06-11T09:00:53.845696379-04:00', 1591880453845 696379, 'iau', 20000, 16.53, 16.54, 16.55, 20000, None, None, 0, 1, None, None, None], 'service': 'iex', 'messageType': 'A'}
```

```
{'data': ['Q', '2020-06-11T09:00:54.875388800-04:00', 1591880454875 388800, 'ung', 900, 10.96, 10.97, 10.98, 800, None, None, 0, 1, Non e, None, None], 'service': 'iex', 'messageType': 'A'}
```

```
{'data': ['T', '2020-06-11T09:00:55.337622088-04:00', 1591880455337 622088, 'ags', None, None, None, None, None, 4.31, 300, None, 1, 0, 0, 0], 'service': 'iex', 'messageType': 'A'}
```

## Filtrando data de un websocket

Bueno, vamos a ir un poco mas allá, en lugar de mostrar en pantalla todo lo que me va trayendo el websocket, mostremos solo un ticker por ejemplo "SPY"

Si se fijan, le tuve que poner un bloque TRY/EXCEPT porque cada tanto el websocket me devuelve un mensaje informativo "HeartBeat" indicando que esta todo ok.. pude haberlo resuelto con algo mas especifico por ejemplo preguntar si el resultado "result" tiene o no la clave "data"

```
i = 0
while i<500:
    result = json.loads(conn.recv())
    try:
        if(result['data'][3]=='spy'):
            print('\n',result,'\n')
        else:
            print('.',end='')
    except:
        print('\n',result,'\n')
    i = i+1
```

```
...
{'data': ['Q', '2020-06-11T09:04:54.912146213-04:00', 159188069491 2146213, 'spy', 500, 310.52, 310.56, 310.6, 500, None, None, 0, 1, None, None, None], 'service': 'iex', 'messageType': 'A'}
.....
{'messageType': 'H', 'response': {'code': 200, 'message': 'HeartBe at'}}}
.....
{'messageType': 'H', 'response': {'code': 200, 'message': 'HeartBe at'}}}
.....
```

Obviamente podría meter todo lo que me va trayendo el websocket en un DataFrame, no tendría sentido desde el punto de vista operativo en si, ya que la idea del websocket y la info en tiempo real es más que nada tomar decisiones en tiempo real con un bot operando en función de si se cumple o no tal condición

Ahora si tuviera sentido en el caso que quieran guardar toda la data que van recolectando del websocket para luego backtestear por ejemplo o simplemente para dejar un registro de lo que fue viendo el bot

## Guardando la data de un websocket

Bueno, vamos a volver al websocket de crypto para mostrar esto porque es mas liquido y práctico para mostrarlo pero pueden

Vamos a agrupar los trades, los que tienen la letra "T" en la posición "0" del response en el dataframe "df\_t" y los cambios en el libro que son los que tienen la letra "Q" en la posición "0" del response en el dataframe "df\_q"

Nota: en el websocket de crypto en realidad no hace falta separar en dataframe de trades y dataframes de quotes, porque lo tiene separado en la configuración del thresholdLevel, pero por ejemplo al día de hoy, el websocket de IEX para equity no lo tiene separado, y te manda mezclado ambos feed, así que me pareció piola dejarles el Código suponiendo que viene siempre mezclado para que no se traben con esa dificultad si se les ocurría probar con el otro websocket

```
import pandas as pd

# Creamos La conexion
conn = ws.create_connection("wss://api.tiingo.com/crypto")

# Creamos el JSON con Los datos de suscripcion
subscribe = '{"eventName":"subscribe","authorization":"'"+token+"','eventData":'

# Nos suscribimos al websocket
conn.send(subscribe)

i = 0
datos_t, datos_q = [], []
while i<1000:
    result = json.loads(conn.recv())
    if result['messageType'] == "A":
        if result['data'][0]=='T':
            datos_t.append(result['data'])
        if result['data'][0]=='Q':
            datos_q.append(result['data'])
    i = i+1

df_t = pd.DataFrame(datos_t)
df_t.columns = ['tipo','par','fecha','exchange','cantidad','precio']
df_q = pd.DataFrame(datos_q)
df_t.head()
```

	tipo	par	fecha	exchange	cantidad	precio
0	T	ethusd	2020-06-11T14:05:26.768025+00:00	gdax	5.125639	240.61
1	T	ethusd	2020-06-11T14:05:26.768025+00:00	gdax	5.125639	240.61
2	T	ethusd	2020-06-11T14:05:26.768025+00:00	gdax	30.562551	240.61
3	T	btcusd	2020-06-11T14:05:26.815000+00:00	binance	0.019751	9625.00
4	T	btcusd	2020-06-11T14:05:26.815000+00:00	binance	0.040756	9627.41

```
df_t = df_t.sort_values('fecha')
df_t.head()
```

	tipo	par	fecha	exchange	cantidad	precio
330	T	thetausd	2020-06-11T14:05:24.040000+00:00	binance	0.9	0.229530
339	T	btlusdc	2020-06-11T14:05:24.044000+00:00	binance	50794.0	0.000308
388	T	thetabnb	2020-06-11T14:05:24.048000+00:00	binance	58.0	0.013428
529	T	thetabnb	2020-06-11T14:05:24.063000+00:00	binance	1.0	0.013427
484	T	thetabnb	2020-06-11T14:05:24.063000+00:00	binance	17.0	0.013428

Vemos que la diferencia entre fechas es de milisegundos, calculemos de hecho la diferencia entre el primero y el numero 50 de la tabla:

```
df_t.iloc[0]
```

```
tipo                                T
par                                 thetausd
fecha      2020-06-11T14:05:24.040000+00:00
exchange                           binance
cantidad                            0.9
precio                               0.22953
Name: 330, dtype: object
```

```
1df_t.iloc[len(df_t)-1]
```

```
tipo                                T
par                                 xrpusd
fecha      2020-06-11T14:05:39.407603+00:00
exchange                           gdax
cantidad                            506
precio                               0.1966
Name: 995, dtype: object
```

Como vemos en este caso, en apenas 15 segundos, recibimos 1000 datos y los tenemos en nuestro datframe ya guardados que lo podríamos pasar a un excel con solamente un `df_t.to_excel("nombre_a_guardar.xlsx")`

Ademas de poder guardarlo en un excel o una base de datos, o lo que fuera para llevar registros de lo que fuimos recolectando o trabajando, podríamos sacar todo tipo de filtro o agrupamiento

En este caso les saco una impresión de conteo como para mostrar un ejemplo de los pares más operados en ese lapso

```
conteo = df_t.groupby('par').par.count()
conteo = conteo.sort_values(0, ascending=False).head()
```

```
par
btcusd      155
ethusd       107
astbtc        62
dgtxbtc       45
btcbusd       45
Name: par, dtype: int64
```

o bien en este ejemplo un conteo de cantidad de trades por exchange

```
1 conteo_exchanges = df_t.groupby('exchange').par.count()
2 conteo_exchanges = conteo_exchanges.sort_values(0, ascending=False).head()
3 conteo_exchanges
```

```
exchange
binance      813
hitbtc        85
gdax          71
kucoin         21
poloniex       8
Name: par, dtype: int64
```

FIELD NAME	ARRAY INDEX	DATA TYPE	DESCRIPTION
Update Message Type	0	char	Communicates what type of price update this is. Will always be "T" for last trade message, "Q" for top-of-book update message, and "B" for trade book messages.
Date	1	datetime	A string representing the datetime this quote or trade came in. This is the timestamp reported by iEX in ISO8601 UTC Format.
Nanoseconds	2	int64	An integer representing the number of nanoseconds since POSIX (Epoch) time UTC.
Ticker	3	string	Ticker related to the asset.
Bid Size	4	int32	The number shares of the bid price. Only available for Quote updates, null otherwise.
Bid Price	5	float	The current highest bid price. Only available for Quote updates, null otherwise.
Mid Price	6	float	The mid price of the current timestamp when both "bidPrice" and "askPrice" are non-null. In mathematical terms: $\frac{\text{bid} + \text{ask}}{2}$ This value is calculated by Tingo and not provided by iEX.
Ask Price	7	float	Only available for Quote updates, null otherwise.
Ask Size	8	int32	The current lowest ask price. Only available for Quote updates, null otherwise.
Last Price	9	float	The last price the last trade was executed at. Only available for Trade and Break updates, null otherwise.
Last Size	10	int32	The amount of shares/volume traded at the last price. Only available for Trade and Break updates, null otherwise.
Halted	11	int32	1 if the security asset is halted, 0 if it is not halted (this comes from iEX).
After Hours	12	int32	1 if the data is after hours, 0 if the update was during market hours (this comes from iEX).
Intermarket Sweep Order (ISO)	13	int32	1 if the order is an intermarket sweep order (ISO) sweeping order, 0 if its a non-ISO order (this comes from iEX).
Oddlot	14	int32	1 if the trade is an odd lot, 0 if the trade is a round or mixed lot (this comes from iEX). Only available for Trade updates, null otherwise.
NMS Rule 611	15	int32	1 if the trade is not subject to NMS Rule 611 (trade through), 0 if the trade is subject to Rule NMS 611 (this comes from iEX). Only available for Trade updates, null otherwise.

# Ejercicios

Siendo que la siguiente captura muestra el response que me da el panel del webSocket de IEX (se puede ver en <https://api.tiingo.com/documentation/websockets/iex> (<https://api.tiingo.com/documentation/websockets/iex>)), se pide

1- Armar la conexión al websocket y recibir en horario de mercado 1000 datos y mostrar un panel de los del tipo "Q" e decir que hayan sufrido un cambio en el book, y que muestre solo:

- \* Ticker
- \* bidSize
- \* bidPrice
- \* askPrice

2- Armar un panel similar al anterior, pero con la conexión al websocket que reciba datos durante 30 segundos y devolver el panel de los últimos trades (Tipo "T") solo:

- \* Ticker
- \* Hora (H:M:S.milisegundos)
- \* lastPrice
- \* lastSize

3- Mostrar con los datos recibidos en 30 segundos ejecutando el script del ejercicio 2, un resumen de cuantos datos fueron del tipo "T", cuantos del tipo "Q" y cuantos del tipo "B"

4- Mostrar con los datos del ejercicio 2, el top10 de la cantidad de trades (Tipo T) de los tickers más operados en ese lapso

5- Con los datos recolectados en el ejercicio 2, armar un panel con los trades que tenga los siguientes campos, pero solo tomando el último valor recibido de cada ticker:

- \* Ticker
- \* Hora (H:M:S.milisegundos)
- \* bidSize
- \* bidPrice
- \* askPrice
- \* askSize
- \* lastPrice

# Respuestas

```
1 #-----#
2 # Ejercicio 1 #
3 #-----#
4
5 import pandas as pd
6 import datetime as dt
7
8 # Creamos la conexión
9 conn = ws.create_connection("wss://api.tiingo.com/iex")
10
11 # Creamos el JSON con los datos de suscripción
12 subscribe = '{"eventName":"subscribe","authorization":"'"+token+"'"
13             , "eventData": {"thresholdLevel":5}}'
14
15 # Nos suscribimos al websocket
16 conn.send(subscribe)
17
18 i = 0
19 datos = []
20 while True:
21     result = json.loads(conn.recv())
22     if result['messageType'] == "A":
23         if result['data'][0]=='Q':
24             datos.append(result['data'])
25
26     # Comando para salir cuando supere los 1000 datos recibidos
27     if i > 1000:
28         break
29
30     # Contador de ciclos
31     i += 1
32
33 df = pd.DataFrame(datos)
34 df.columns = ['tipo', 'fecha', 'nanosegundos', 'ticker', 'bidSize', 'bidPrice',
35               'midPrice', 'askPrice', 'askSize', 'lastPrice', 'lastSize', 'halted',
36               'afterHours', 'swepp', 'oddlot', 'nms']
37 df = df.loc[ : , ['ticker', 'bidSize', 'bidPrice', 'askPrice',
38                   'askSize', 'lastPrice']]
39 df
```

	ticker	bidSize	bidPrice	askPrice	askSize	lastPrice
0	iemg	1100	47.10	47.13	1600	None
1	hewg	100	26.29	26.33	4000	None
2	sbs	400	10.36	10.38	100	None
3	pfe	200	33.24	33.25	100	None
4	kim	200	13.34	13.35	200	None
...	...	...	...	...	...	...
109	dbo	2000	6.60	6.61	100	None
110	amu	1000	8.89	8.92	1000	None
111	amub	1000	8.89	8.93	2000	None
112	vod	100	15.72	15.73	100	None
113	htz	100	2.95	2.96	300	None

```

#-----#
# Rta Ejercicio 2 #
#-----#
import pandas as pd
import datetime as dt

# Creamos la conexion
conn = ws.create_connection("wss://api.tiingo.com/1ex")

# Creamos el JSON con los datos de suscripcion
subscribe = '{"eventName":"subscribe","authorization":"'${token}'"
              , "eventData": {"thresholdLevel":5}}'

# Nos suscribimos al websocket
conn.send(subscribe)

# Seteamos momento del inicio del ciclo
inicio = dt.datetime.now().timestamp()

datos = []
while True:
    result = json.loads(conn.recv())
    if result['messageType'] == "A":
        datos.append(result['data'])

    # Vamos viendo si no supera los 30 segundos desde que inicio
    ahora = dt.datetime.now().timestamp()
    tiempo = ahora - inicio if
    tiempo > 30:
        break

df = pd.DataFrame(datos)

df.columns = ['tipo','fecha','hora','ticker','bidSize','bidPrice',
              'midPrice','askPrice','askSize','lastPrice','lastSize','halted'
              'afterHours','swepp','oddlot','nms']

df['fecha'] = pd.to_datetime(df.fecha)
panel = df.loc[ df.tipo=='T' , ['fecha','ticker','lastPrice','lastSize']]
```

panel.head()

	fecha	ticker	lastPrice	lastSize
0	2020-06-12 15:04:33.467290797-04:00	jrvr	41.105	10.0
1	2020-06-12 15:04:33.474933109-04:00	nclh	19.930	75.0
2	2020-06-12 15:04:33.479194994-04:00	amzn	2537.590	100.0
3	2020-06-12 15:04:33.476419831-04:00	exel	22.300	100.0
4	2020-06-12 15:04:33.476746094-04:00	swi	18.260	4.0

```
#-----#
#      Ejercicio 3      #
#-----#
```

```
df.groupby('tipo').tipo.count()
```

```
tipo
```

```
Q    445  
T    1911
```

```
Name: tipo, dtype: int64
```

```
#-----#
#      Ejercicio 4      #
#-----#
```

```
df.groupby('ticker').ticker.count().sort_values(0, ascending=False).head(10)
```

```
ticker
```

```
gps    29  
htz    23  
rig    21  
xlu    20  
wu     19  
ntco   17  
ctlt   16  
mat    15  
snap   15  
bp     15
```

```
Name: ticker, dtype: int64
```

```
1 #-----#
2 #  Rta Ejercicio 5  #
3 #-----#
4
5
6 panel = df.loc[ df.tipo=='Q' , ['fecha','ticker','bidSize','bidPrice',
7                           'midPrice','askPrice','askSize']]  
8 panel
9 # panel.groupby('ticker').last()
```

El presente libro tiene como objetivo hacer dar los primeros pasos en programación a gente que nunca antes programó. Tiene la particularidad de estar enfocado en temas de mercado de capitales ya que es parte de una serie de libros que van desde los primeros scripts hasta terminar en temas mas complejos como el análisis cuantitativo y research de sistemas de inversión, backtestings, screeners, bots de trading entre otros ejemplos

Me enfoqué en la parte práctica ya que entiendo que la programación es una disciplina que se aprende y se asimila con horas y horas de sentarse a resolver problemas. Esto es 80% práctica y 20% teoría, aprender a programar es como aprender a manejar, la cantidad de horas al teclado es lo que hará la diferencia.

Creo que tanto la programación como las finanzas son dos campos importantes en la formación profesional y absolutamente imprescindibles para la planificación personal laboral y patrimonial en un mundo cada vez mas interesante, complejo y competitivo. Por eso espero que sea de utilidad y sea la puerta de entrada que termine acercando a mucha gente a este mundo tan fascinante y desafiante de la programación y de las finanzas cuantitativas