

Python para finanzas

CURSO PRÁCTICO



www.ra-ma.com

Diríjete a www.ra-ma.com para obtener material adicional.

Carlos Mario Ramírez Gil



<https://dogramcode.com/bloglibros>



<https://dogramcode.com/programacion>

Python para finanzas

Curso práctico

Carlos Mario Ramírez Gil



Ramírez Gil, Carlos Mario

Python para finanzas/ Python para finanzas --. Bogotá: Ediciones de la U, 2021

390 p. ; 24 cm

ISBN 978-958-792-299-8 e-ISBN 978-958-792-300-1

1. Finanzas 2. Modelado financiero 3. Proceso de datos financieros 4.

Representación visual de datos Tít.

519.7 ed.

Edición original publicada por © Editorial Ra-ma (España)

Edición autorizada a Ediciones de la U para Colombia

Área: Finanzas

Primera edición: Bogotá, Colombia, septiembre de 2021

ISBN. 978-958-792-299-8

- © Carlos Mario Ramírez Gil
- © Ra-ma Editorial. Calle Jarama, 3-A (Polígono Industrial Igarsa) 28860 Paracuellos de Jarama
www.ra-ma.es y www.ra-ma.com / E-mail: editorial @ra-ma.com
Madrid, España
- © Ediciones de la U - Carrera 27 #27-43 -Tel. (+57-1) 3203510 -3203499
www.edicionesdelau.com - E-mail: editor@edicionesdelau.com
Bogotá, Colombia

Ediciones de la U es una empresa editorial que, con una visión moderna y estratégica de las tecnologías, desarrolla, promueve, distribuye y comercializa contenidos, herramientas de formación, libros técnicos y profesionales, e-books, e-learning o aprendizaje en línea, realizados por autores con amplia experiencia en las diferentes áreas profesionales e investigativas, para brindar a nuestros usuarios soluciones útiles y prácticas que contribuyan al dominio de sus campos de trabajo y a su mejor desempeño en un mundo global, cambiante y cada vez más competitivo.

Coordinación editorial: Adriana Gutiérrez M.

Carátula: Ediciones de la U

Impresión: DGP Editores SAS

Calle 63 #70D-34, Pbx (57+1) 3203510

Impreso y hecho en Colombia

Printed and made in Colombia

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro y otros medios, sin el permiso previo y por escrito de los titulares del Copyright.

A Adri, mi amada compañera de viaje.

A Lau, que me inicio en el mundo de Python.

A Andrés, por poner la vara tan alta.

*A Johny, por sus logros científicos.
El hombre del Renacimiento.*

*A Luchó, mi querido y brillante amigo.
El programador de Seda.*

ÍNDICE

CAPÍTULO 1. ¿POR QUÉ LOS USUARIOS DE EXCEL Y ESPECIALMENTE LOS PROFESIONALES DE FINANZAS NECESITAN APRENDER PYTHON?.....	13
1.1 VENTAJAS DE PYTHON SOBRE EXCEL	13
1.1.1 Python no está limitado por el tamaño	14
1.1.2 Python no está limitado por la memoria.....	14
1.1.3 Python no tiene limitaciones de integración	14
1.1.4 La automatización no es posible con Excel, lamentablemente	14
1.1.5 Capacidades multiplataforma	15
1.1.6 Comunidad y soporte de código abierto.....	15
1.1.7 Entonces, ¿debería deshacerme de Excel?	15
1.2 PYTHON-PANDAS VS EXCEL	15
1.2.1 Construido sobre Python	16
1.2.2 Maneja múltiples tipos de datos	17
1.2.3 Maneja grandes conjuntos de datos	17
1.2.4 Limpia archivos rápidamente y convierte archivos de datos	17
CAPÍTULO 2. APRENDIENDO PYTHON DESDE CERO.....	19
2.1 ¿QUÉ ES PYTHON?.....	19
2.2 CONFIGURANDO EL AMBIENTE PYTHON	20
2.2.1 Descarga e instalación de Python.....	21
2.2.2 Ejecutar una secuencia de comandos en la ventana de la terminal	22
2.2.3 La ejecución del entorno interactivo IDLE	23
2.2.4 Editores de texto para Python	25
2.2.5 Jupyter Notebook	27
2.2.6 Usando Python con Jupyter Notebook en Google Colaboratory	31
2.3 CONOCIENDO LO BÁSICO DE PYTHON.....	34
2.3.1 Los comentarios	34
2.3.2 Indentación	34
2.3.3 Variables	35
2.3.4 Operadores	37

2.3.5	Declaraciones condicionales	39
2.3.6	For loops (Para bucles).....	43
2.3.7	Bucle While (Mientras)	46
2.3.8	Input (Entrada) del usuario.....	48
2.3.9	Typecasting (Tipografía)	49
2.3.10	Diccionarios	50
2.3.11	Listas	55
2.3.12	Tuplas	58
2.3.13	Conjuntos	60
2.3.14	Funciones y argumentos	61
2.3.15	Ámbito.....	68
2.3.16	Declaración de devolución	70
2.3.17	Expresión Lambda.....	70
2.3.18	Comprensión de listas	71
2.4	CONCEPTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS (OOP).....	72
2.4.1	Ventajas de la programación orientada a objetos	73
2.4.2	Clases	74
2.4.3	Métodos	75
2.4.4	Objetos	75
2.4.5	Constructo	77
2.4.6	Atributos de instancia	78
2.4.7	Atributos de clase	78
2.4.8	Self	78
2.4.9	Herencia	79
2.4.10	Super.....	83
2.4.11	Herencia Múltiple.....	84
2.4.12	Polimorfismo	85
2.4.13	Encapsulación.....	86
2.4.14	Decorador	89
2.4.15	Excepciones.....	91
2.5	IMPORTACIÓN DE PAQUETES.....	93
2.6	GUÍA DE ESTILO PARA LA ESCRITURA DEL CÓDIGO PYTHON	96
2.6.1	Sangria (Indentation).....	96
2.6.2	Longitud máxima de la línea	97
2.6.3	Comillas simples o dobles.....	97
2.6.4	Líneas en blanco	98
2.6.5	Declaraciones de importación	99
2.6.6	Comentarios	99
2.6.7	Nombres de dunder a nivel de módulo.....	100
2.6.8	Convenciones de nombres.....	101
2.6.9	Espacios en blanco en expresiones / declaraciones.....	102
CAPÍTULO 3. LOS 10 MEJORES PAQUETES DE PYTHON PARA FINANZAS Y MODELADO FINANCIERO.....	103	
3.1	NUMPY	104
3.2	SCIPY	104
3.3	PANDAS	106

3.4	STATSMODELS.....	107
3.5	QUANDL.....	108
3.6	ZIPLINE	109
3.7	PYFOLIO.....	109
3.8	TA-LIB.....	111
3.9	QUANT-LIB	111
3.10	MATPLOTLIB.....	111
CAPÍTULO 4. OBTENIENDO Y PROCESANDO DATOS FINANCIEROS		113
4.1	OBTENIENDO DATOS DE YAHOO FINANCE	114
4.2	OBTENIENDO DATOS DE QUANDL.....	116
4.3	OBTENIENDO DATOS DE INTRINIO	117
4.4	TRANSFORMANDO PRECIOS DE ACTIVOS FINANCIEROS A RENDIMIENTOS.....	120
4.5	CAMBIANDO LA FRECUENCIA TEMPORAL DE LOS RENDIMIENTOS.....	125
4.6	VISUALIZANDO DATOS DE SERIES DE TIEMPO FINANCIERAS	128
4.7	IDENTIFICANDO VALORES ATÍPICOS EN LA SERIE TEMPORAL	135
4.8	OBTENCIÓN Y REMUESTREO DE DATOS SOBRE CRIPTOMONEDAS.....	138
4.8.1	Obteniendo los datos de Kraken (Exchanges).....	138
4.8.2	Remuestreo.....	143
CAPÍTULO 5. EL VALOR DEL DINERO EN EL TIEMPO.....		145
5.1	INTRODUCCIÓN AL VALOR TEMPORAL DEL DINERO. VALOR FUTURO Y VALOR PRESENTE.....	145
5.2	VALOR PRESENTE DE UNA PERPETUIDAD	149
5.3	VALOR PRESENTE DE UNA PERPETUIDAD CRECIENTE	151
5.4	VALOR PRESENTE Y FUTURO DE UNA ANUALIDAD	152
5.5	CÁLCULO DE LOS PAGOS DE UN PRÉSTAMO Y LA TABLA DE AMORTIZACIÓN EN PYTHON	153
5.6	DEFINICIÓN DE VPN Y REGLA DE VPN	157
5.7	DEFINICIÓN DE TIR Y REGLA DE LA TIR	159
CAPÍTULO 6. EXTRAER Y CALCULAR INDICADORES FINANCIEROS CLAVES CON PYTHON		163
6.1	PYTHON PARA FINANZAS ES NUESTRO MEJOR ALIADO	163
6.2	CONSEGUIR DATOS E INDICADORES FINANCIEROS CLAVES CON PYTHON	164
6.3	CONSOLIDAR INDICADORES FINANCIEROS EN UN DATAFRAME DE PANDAS.....	166
6.4	CALCULAR INDICADORES FINANCIEROS CON PYTHON.....	168
6.5	GRAFICAR INDICADORES FINANCIEROS CON PYTHON	172
6.6	EXPORTAR INDICADORES FINANCIEROS A UN ARCHIVO EN EXCEL	174

CAPÍTULO 7. VALORACIÓN DE BONOS Y ACCIONES	179
7.1 ESTRUCTURA TEMPORAL DE TASAS DE INTERÉS	179
7.2 RELACIÓN RIESGO Y RENTABILIDAD. LA DURACIÓN	183
7.3 EVALUACIÓN DE BONOS Y RENDIMIENTO HASTA EL VENCIMIENTO	188
7.4 VALORACIÓN DE ACCIONES	193
CAPÍTULO 8. COSTO PROMEDIO PONDERADO DE CAPITAL (WACC)	197
8.1 ¿QUÉ ES EL COSTO DE CAPITAL PROMEDIO PONDERADO (WACC)?	197
8.2 ¿CÓMO CALCULAR EL WACC?	198
8.3 COSTO DE LA DEUDA DE LAS EMPRESAS	198
8.4 COSTO DEL PATRIMONIO DE LA EMPRESA	199
8.5 CALCULANDO EL WACC CON PYTHON	200
8.5.1 Estimación del costo de la deuda con Python	200
8.5.2 Estimación del costo de patrimonio con Python	203
8.5.3 Costo de capital promedio ponderado (WACC) con Python	204
CAPÍTULO 9. FLUJO DE CAJA LIBRE DESCONTADO: VALORACIÓN DE UNA EMPRESA CON PYTHON	207
9.1 INTRODUCCIÓN AL MÉTODO DE FLUJO DE CAJA LIBRE DESCONTADO	207
9.2 VALOR DE LA EMPRESA CON EL MÉTODO DE FLUJO DE CAJA LIBRE DESCONTADO	208
9.3 PRONÓSTICO DEL FLUJO DE CAJA LIBRE DE LA EMPRESA	209
9.4 FLUJO DE CAJA LIBRE DESCONTADO CON PYTHON	210
9.4.1 Estimación del crecimiento futuro de los ingresos	211
9.4.2 Obtención del estado de resultados y el balance general	212
9.4.3 Proyección de los flujos de caja futuros de las operaciones	213
9.4.4 Convertir los flujos de caja libres proyectados a pandas	216
9.4.5 Estimación del costo de capital	217
9.4.6 Obtención del valor presente de los flujos de caja libre futuros	221
9.4.7 Cálculo del valor terminal	221
9.4.8 Calcular el precio objetivo de Google	222
CAPÍTULO 10. REPRESENTACIÓN GRÁFICA Y VISUAL DE DATOS FINANCIEROS EN PYTHON	223
10.1 IMPORTANDO LIBRERÍAS	223
10.2 EXTRAYENDO LOS PRECIOS HISTÓRICOS DE LAS ACCIONES	224
10.3 GRÁFICO DE ÁREA	224
10.4 GRÁFICO DE VELAS	227
10.5 GRÁFICO OHLC	229
10.6 GRÁFICO BULLET	231
10.7 GRÁFICO DE CALIBRE RADIAL	233

CAPÍTULO 11. ASIGNACIÓN DE ACTIVOS PARA UN PORTAFOLIO EFICIENTE EN PYTHON.....	235
11.1 EVALUAR EL RENDIMIENTO DE UN 1 / N PORTAFOLIO BÁSICO	237
11.2 ENCONTRAR LA FRONTERA EFICIENTE USANDO SIMULACIÓN MONTECARLO.....	243
11.3 ENCONTRAR LA FRONTERA EFICIENTE USANDO OPTIMIZACIÓN CON SCIPY.....	250
CAPÍTULO 12. SIMULACIÓN MONTE CARLO EN FINANZAS	259
12.1 SIMULANDO LA DINÁMICA DEL PRECIO DE LAS ACCIONES UTILIZANDO MOVIMIENTO BROWNIANO GEOMÉTRICO.....	260
12.2 PRECIOS DE OPCIONES EUROPEAS MEDIANTE SIMULACIONES	266
12.3 ESTIMACIÓN DEL VALOR EN RIESGO UTILIZANDO MONTE CARLO.....	272
CAPÍTULO 13. MODELADO DE SERIES DE TIEMPO FINANCIERAS	279
13.1 DESCOMPOSICIÓN DE SERIES DE TIEMPO.....	280
13.2 DESCOMPOSICIÓN DE SERIES DE TIEMPO USANDO PROPHET DE FACEBOOK	284
13.3 PRUEBA DE ESTACIONARIEDAD EN SERIES DE TIEMPO	289
13.4 CORRECCIÓN DE LA ESTACIONARIEDAD EN SERIES DE TIEMPO	294
13.5 MODELADO DE SERIES DE TIEMPO CON MÉTODOS DE SUAVIZADO EXPONENCIAL.....	302
13.6 MODELADO DE SERIES DE TIEMPO CON MODELOS DE CLASE ARIMA	311
CAPÍTULO 14. INTEGRACIÓN DE PYTHON CON EXCEL.....	323
14.1 PANDAS PARA LEER UN ARCHIVO DE EXCEL.....	324
14.1.1 Método y argumentos pd.read_excel ().....	324
14.1.2 Método y argumentos pd.read_csv ().....	327
14.2 LEER VARIAS HOJAS DE EXCEL CON PANDAS.....	328
14.2.1 método pd.read_excel ().....	328
14.2.2 Método pd.ExcelFile ()	330
14.3 LEER MÚLTIPLES ARCHIVOS DE EXCEL CON PYTHON	331
14.3.1 Método 1: Obtener archivos de la carpeta - estilo PowerQuery.....	331
14.3.2 Método 2: usar un archivo de entrada de Excel	332
14.4 GUARDANDO LOS DATOS EN UN ARCHIVO DE EXCEL USANDO PYTHON	334
14.5 GUARDAR EN ARCHIVO CSV.....	336
14.6 GUARDE VARIAS HOJAS EN UN ARCHIVO DE EXCEL CON PYTHON	336
14.6.1 Método 1	337
14.6.2 Método 2	337
14.7 OBTENGA VALORES, FILAS Y COLUMNAS EN DATAFRAME DE PANDAS.....	338

14.7.1	Obteniendo columnas con pandas	339
14.7.2	Obteniendo varias columnas	341
14.7.3	Obteniendo filas con Pandas	341
14.7.4	Obteniendo varias filas	342
14.7.5	Obteniendo valores de celda.....	343
14.8	OBTENIENDO DATOS DE LA TABLA DE UNA PÁGINA WEB USANDO PYTHON Y LA LIBRERÍA PANDAS	345
14.8.1	Obtener datos de un sitio web (web scraping)	345
14.8.2	Requisitos para usar pandas para web scraping	346
14.8.3	Web Scraping en acción	347
14.9	REPLICAR LAS FUNCIONES BUSCAR DE EXCEL EN PYTHON.....	348
CAPÍTULO 15. COMO CONSTRUIR UNA HERRAMIENTA DE ANÁLISIS DE SENTIMIENTOS PARA EL TRADING DE ACCIONES		355
15.1	¿QUÉ ES EL ANÁLISIS DE SENTIMIENTOS?	355
15.2	RECOPILACIÓN Y ANÁLISIS DE DATOS DE FINVIZ	356
15.3	APLICAR EL ANÁLISIS DE SENTIMIENTOS	358
15.4	VISUALIZACIÓN DE LOS RESULTADOS EN MATPLOTLIB	359
CAPÍTULO 16. WEB SCRAPING PARA ESTADOS FINANCIEROS CON PYTHON		361
16.1	¿QUÉ ES EL WEB SCRAPING?.....	361
16.2	IMPORTANDO LAS LIBRERÍAS URLIB Y BEAUTIFUL SOUP	362
16.3	PROCESANDO LOS DATOS	362
16.4	LEYENDO LA URL.....	363
16.5	MANIPULANDO LOS DATOS	364
16.6	LIMPIANDO LOS DATOS.....	366
CAPÍTULO 17. TRADING ALGORÍTMICO CON PYTHON.....		369
17.1	TABLERO DE TRADING CON YFINANCE & PYTHON	369
17.1.1	Extrayendo datos con la api Yfinance	369
17.1.2	Establecer las ventanas cortas y largas (medias móviles)	370
17.1.3	Generando señales de trading.....	371
17.1.4	Trazando puntos de entrada y salida	372
17.1.5	Realizar Backtest.....	376
17.1.6	Generar Dashboard o Tablero de Control.....	379
17.2	ESTRATEGIA DE NEGOCIACIÓN ALGORÍTMICA CON MACD	380
17.2.1	¿Qué es MACD Crossover?	380
17.2.2	Componentes del MACD	380
17.2.3	¿Cómo se calcula el MACD?	381
17.2.4	Implementación del MACD en Python	381
MATERIAL ADICIONAL.....		389

1

¿POR QUE LOS USUARIOS DE EXCEL Y ESPECIALMENTE LOS PROFESIONALES DE FINANZAS NECESITAN APRENDER PYTHON?

Amo Microsoft Excel. Es (casi) la herramienta más valiosa de mi arsenal para el análisis de datos y modelación financiera. Bueno, casi. Pero a pesar de lo poderoso que es Excel, muchas veces había deseado poder hacer más, especialmente cuando llegué a los ‘límites’ de Excel, y si trabaja con Excel todos los días, notará que hay bastantes limitaciones de Excel que fácilmente pueden convertirse en una tortura.

No estoy aquí para intentar que Excel se vea mal, no, Excel sigue siendo la herramienta más popular para la manipulación de datos, aunque a una escala limitada. Si usa Excel para análisis de datos simples e informes de tableros de control simples, puede arreglárselas fácilmente sin preocupaciones. Pero si ha llegado al punto en el que desea poder automatizar sus informes, manipular conjuntos de datos mucho más grandes con mejor velocidad y menos arrastre, o realizar cálculos más potentes, lo más recomendable es que siga leyendo este capítulo.

1.1 VENTAJAS DE PYTHON SOBRE EXCEL

A continuación, se relacionan algunos aspectos en los cuales Python supera ampliamente las capacidades de Excel.

1.1.1 Python no está limitado por el tamaño

Excel puede manejar más de 1 millón de filas (1.048.576 para ser específicos). Sin embargo, cuando supere las 10,000 filas, notará que el libro de trabajo comienza a ralentizarse significativamente. Pruebe esto con más hojas, y seguramente experimentará molestos y erráticos bloqueos de libros de trabajo. Por el contrario, Python puede manejar millones y millones de filas sin ningún problema, solo está limitado por la potencia informática de su PC.

1.1.2 Python no está limitado por la memoria

Los cálculos que consumen mucha memoria, como los cálculos con matrices (Ctrl + Shift + Enter) en Excel, pueden bloquear fácilmente su libro de trabajo, lamentablemente. Microsoft ha realizado muchas mejoras mediante las herramientas Power (PowerQuery, PowerPivot, PowerBI, etc.) pero las limitaciones de memoria no han desaparecido. Python, usando librerías como numpy y pandas, por otro lado, puede manejar cálculos muy complejos sin complicaciones.

1.1.3 Python no tiene limitaciones de integración

Ninguna herramienta de análisis de datos tiene más capacidades de integración que Python. Puede conectarse a archivos CSV, páginas web HTML, bases de datos SQL e incluso bases de datos NoSQL. Los desarrolladores escriben constantemente herramientas y librerías más eficientes para integrar sus aplicaciones con Python.

1.1.4 La automatización no es posible con Excel, lamentablemente

Excel no fue diseñado para automatizar tareas, por lo que pedir la automatización de Excel sería injusto. Sin embargo, utilizando herramientas de programación de tareas, los usuarios de Excel han podido realizar una automatización básica de sus tareas. Sin embargo, la capacidad de Python para automatizar tareas es simplemente ilimitada. En lugar de tener que abrir un archivo cada semana para ejecutar su análisis antes de adjuntar el resultado a un correo electrónico para enviarlo, puede automatizar este flujo de trabajo con Python y eliminar por completo la necesidad de interacción humana en el proceso, dejando como única acción necesaria el suministro del documento fuente para su análisis.

1.1.5 Capacidades multiplataforma

¿Se ha enfrentado a problemas en los que algunas fórmulas de su hoja no funcionan en Mac porque realizó su análisis en una máquina con Windows? Esto es muy común con Microsoft Excel, pero muy raro con Python. Las librerías pueden tener funciones obsoletas que se mueven de una versión a la siguiente (como con todas las actualizaciones de software, obviamente), pero la compatibilidad entre plataformas no afecta el análisis de datos en Python.

1.1.6 Comunidad y soporte de código abierto

Python es completamente gratuito y de código abierto. Sin tarifas de licencia, sin renovaciones, sin compromiso financiero necesario para mantener su proyecto de análisis de datos en funcionamiento. Y con el apoyo masivo de la comunidad, casi nunca puede quedarse atascado si se enfrenta a algún problema. Hay una gran cantidad de recursos que le permiten obtener fácilmente soluciones a los desafíos (la documentación de Python, Stack Overflow, Medium, GitHub, entre otros.).

1.1.7 Entonces, ¿debería deshacerme de Excel?

No, no es necesario. Las librerías de Python como pandas se han diseñado para que pueda importar fácilmente sus datos de Excel a Python y de regreso a Excel. Entonces, en lugar de intentar crear esa “hoja maestra” en Excel para contener sus cientos de libros de trabajo de origen, puede usar Python para conectar todos esos libros de trabajo de origen para el análisis y luego exportar los resultados de su análisis a Excel para presentarlos a la administración. Con el conocimiento de Python, puede potenciar sus análisis e informes de Excel.

1.2 PYTHON-PANDAS VS EXCEL

¿Por qué aprender a trabajar con Python cuando tenemos herramientas de Excel y BI?

Excel es la herramienta más utilizada en el mundo para el análisis y la manipulación de datos. Cada organización seguramente la está usando; analistas, contadores e incluso directores ejecutivos la utilizan para las estadísticas y el manejo de datos. Todos los usuarios de Excel saben cómo Excel puede realizar tareas desde funciones simples hasta fórmulas complejas, macros, Power Query, estadísticas complejas y luego codificación VBA, por lo que es nuestro mejor aliado para múltiples situaciones. Pero todos habremos notado que cada vez que abrimos

una nueva base de datos o hay un cambio en la base de datos existente, tenemos que limpiar los datos desde cero o tenemos que escribir una función o código VBA para resolver el problema. Sin entrar en detalles, las limitaciones de Excel son bien conocidas.

Al usar Python, puede disfrutar de la libertad adicional de jugar con datos de Excel y, además de eso, obtener una lista interminable de conexiones a su modelo de datos, es decir, enlaces a aplicaciones web en vivo, bases de datos SQL, etc. Existe una gran herramienta disponible en forma de librería de Python - **Pandas**. Pandas fue creado por Wes McKinney. Construyó Pandas para contribuir a trabajar de manera más eficiente con conjuntos de datos financieros. Pandas es una herramienta de manipulación y análisis de datos de código abierto rápida, potente, flexible y fácil de usar, construida sobre el lenguaje de programación Python. La perspectiva financiera de Wes Mckinney detrás de la construcción de pandas es un gran punto a favor para aprender nuevas habilidades, esto especialmente para los profesionales de Contabilidad y Finanzas que están en la confusión de si debiesen aprender o no un lenguaje de programación para trascender las herramientas de Excel y BI.

Aquí hay algunos beneficios por los cuales se recomienda comenzar a usar Pandas (una librería de Python)

1.2.1 Construido sobre Python

Pandas está construido sobre Python, eso es lo que lo hace más poderoso. El poder real de Pandas se disfruta cuando lo usa con otras librerías de Python disponibles. La lista de librerías de Python es muy larga y se puede utilizar cualquiera de ellas de forma gratuita para crear modelos específicos según sus necesidades. A continuación, se mencionan algunas de ellas, esto, para resaltar su valor.

- **Scikit-Learn** es una librería de aprendizaje automático en Python. Puede utilizar Pandas para introducir datos en modelos de aprendizaje automático de Scikit-Learn.
- **Flask y Django** son librerías de Python que proporcionan un marco de desarrollo web. Puede crear aplicaciones web con cualquiera de estas dos. Puede utilizar Pandas en el back-end de su aplicación web para el análisis de datos y la creación de informes.
- **Dash, Plotly y Plotly expresss**. Dash es la librería para crear aplicaciones web de ciencia de datos y aprendizaje automático. Estas librerías proporcionan el marco para crear paneles interactivos y visualizaciones de sus datos a través de aplicaciones web.

1.2.2 Maneja múltiples tipos de datos

El mayor obstáculo al que nos enfrentamos con el análisis de datos es el tipo de datos en sí. Se nos proporcionan datos en muchos formatos de datos, es decir, SQL y HTML, etc. Aunque podemos importarlos a Excel, siempre lleva mucho tiempo convertir estos conjuntos de datos en una forma utilizable de Excel. Aunque después de una conversión exitosa, el formateo a menudo se ve afectado y puede resultar en errores. Pandas proporciona una solución a esto al ayudar a importar más de 15 tipos de formatos de datos y la principal ventaja de importar todos estos tipos de datos es que se puede jugar con todos ellos de la misma manera. No importa si está importando desde SQL, HTML o Excel / CSV, Pandas proporciona una manera uniforme de tratarlos todos de la misma manera.

1.2.3 Maneja grandes conjuntos de datos

Todos sabemos que a medida que el tamaño de los datos comienza a aumentar, la velocidad de Excel comienza a disminuir y afecta nuestra eficiencia de trabajo. Cuando los números de las filas de Excel superan los 10,000, su velocidad disminuye. Pandas no tiene ninguna limitación en cuanto al número de filas y su capacidad informática gira en torno a la memoria del computador en la que se ejecuta. Puede crear cientos de cálculos con millones de datos en Pandas al instante.

1.2.4 Limpia archivos rápidamente y convierte archivos de datos

La primera etapa del análisis de datos es la limpieza de datos. No obtenemos datos en la forma deseada. Siempre hay algo para agregar, editar o eliminar. Pandas proporciona todas estas funciones para ayudar a los expertos en datos a limpiar los datos de la forma deseada en solo unas pocas líneas de código.

Al estar en el campo de las finanzas / contabilidad, animo a todos los profesionales en el campo a aprender habilidades de codificación y tratar de agregar Python-Pandas en su conjunto de habilidades. A medida que vaya avanzando con eso, aprenderá cuándo usar herramientas convencionales y cuándo usar herramientas de programación básicas. Cuando se trata de requisitos simples y una menor cantidad de datos, puede utilizar herramientas convencionales como Excel y herramientas de BI, pero si bien los datos serán más grandes o los requisitos de informes y análisis son más altos, es posible que deba tomar la ayuda de herramientas de programación centrales, es decir, Pandas. También puede importar y exportar desde / hacia Pandas en cualquier momento. Por ejemplo, se puede hacer un análisis simple en Excel y luego llevarlo a pandas para un análisis estadístico adicional, o se hace un análisis en Pandas y luego se exporta a Excel para su análisis.

2

APRENDIENDO PYTHON DESDE CERO

En este capítulo se describirán los elementos básicos o ladrillos que constituyen el lenguaje de programación Python. No se entrará en detalles conceptuales, dado que el propósito final será utilizar esta poderosa herramienta de programación para construir modelos financieros.

La primera pregunta que nos podríamos plantear es: ¿Qué es Python?

2.1 ¿QUÉ ES PYTHON?

Es un lenguaje de programación cuya filosofía hace hincapié en una sintaxis que favorezca un código legible (fue diseñado para ser leído con facilidad, una de sus características es el uso de palabras donde otros lenguajes utilizarían símbolos). Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado (no requiere de un compilador para ser ejecutado sino de un intérprete. Un intérprete, actúa de manera casi idéntica a un compilador, con la salvedad de que ejecuta el programa directamente, sin necesidad de generar previamente un ejecutable) usa tipado dinámico (las variables no requieren ser definidas asignando su tipo de datos, sino que éste, se auto-asigna en tiempo de ejecución, según el valor declarado) y es multiplataforma (Windows, Mac OS y Linux).

Python como lenguaje de programación de alto nivel (más cercano a las necesidades del programador o usuario, en contraste con los lenguajes de bajo nivel que son más cercanos a la arquitectura del hardware) y de propósito general permite al usuario centrarse en la resolución de problemas en múltiples ámbitos:

- ▶ Desarrollo web
- ▶ Ciencia de datos
- ▶ Machine learning e inteligencia artificial
- ▶ Programas de escritorio
- ▶ Internet de las cosas
- ▶ Web scraping y bots
- ▶ Criptografía
- ▶ Finanzas
- ▶ Biología computacional
- ▶ Ciencias exactas y academia

La figura 2.1 muestra la ubicación y el papel que juegan los lenguajes de programación de alto y bajo nivel con respecto al usuario y a la máquina.



Figura 2.1. Lenguajes de alto y bajo nivel

2.2 CONFIGURANDO EL AMBIENTE PYTHON

Para escribir programas en Python se requiere crear un entorno de desarrollo adecuado, esto implica realizar unas pocas instalaciones en nuestro computador y comprender el papel básico de las herramientas que utilizaremos. A continuación, abordaremos los aspectos relacionados con la descarga e instalación y los elementos que nos permitirán escribir los programas: la terminal (también se le denomina línea de comando), el entorno interactivo (IDLE por sus siglas en inglés), los editores

de texto y Jupyter Notebook (aplicación web de código abierto que permite crear y compartir documentos que contienen código, ecuaciones, visualizaciones y texto explicativo). La mayor parte de los scripts y programas presentados en este libro han sido escritos en Jupyter Notebook.

2.2.1 Descarga e instalación de Python

El software es administrado por la organización sin fines de lucro **Python Software Foundation** mediante una licencia de código abierto denominada Python Software Foundation License, compatible con la Licencia pública general GNU a partir de la versión 2.1.1.

El primer paso para descargar e instalar el programa es visitar el sitio web oficial <https://www.Python.org/downloads/>, descargar la versión del intérprete adecuado para el sistema operativo que vaya a utilizarse: Windows, Linux/Unix o Mac OS X, entre otros. En la figura 2.2. Zonas de descargas de Python.



Figura 2.2. Zona de descargas de Python

Todos los programas del libro se han escrito con la versión 3.8 de Python para Windows.

Una vez descargado el archivo ejecutable haga clic sobre él y continúe con las instrucciones del asistente.

2.2.2 Ejecutar una secuencia de comandos en la ventana de la terminal

Una secuencia de comandos o un programa python es un archivo de texto plano con la extensión .py. Se puede crear un programa de Python con cualquier editor de texto plano. En la figura 2.3 se puede observar un programa escrito en Notepad que imprime “Hola Mundo”



Figura 2.3. Programa Python en Notepad

A la ventana de la terminal también se le denomina línea de comandos. Para ejecutar un programa en una ventana de terminal, es necesario estar situado en el directorio en el que se encuentra el programa. Para abrir una ventana de terminal hay varias maneras, a continuación se presenta una de ellas (figura 2.4):

Señale en el Explorador de archivos de Windows la carpeta que contiene el programa, haga Shift + clic derecho y elija la opción “Abrir la ventana de PowerShell aquí” (en Windows 10) o “Abrir ventana de comandos aquí” (en Windows 7).

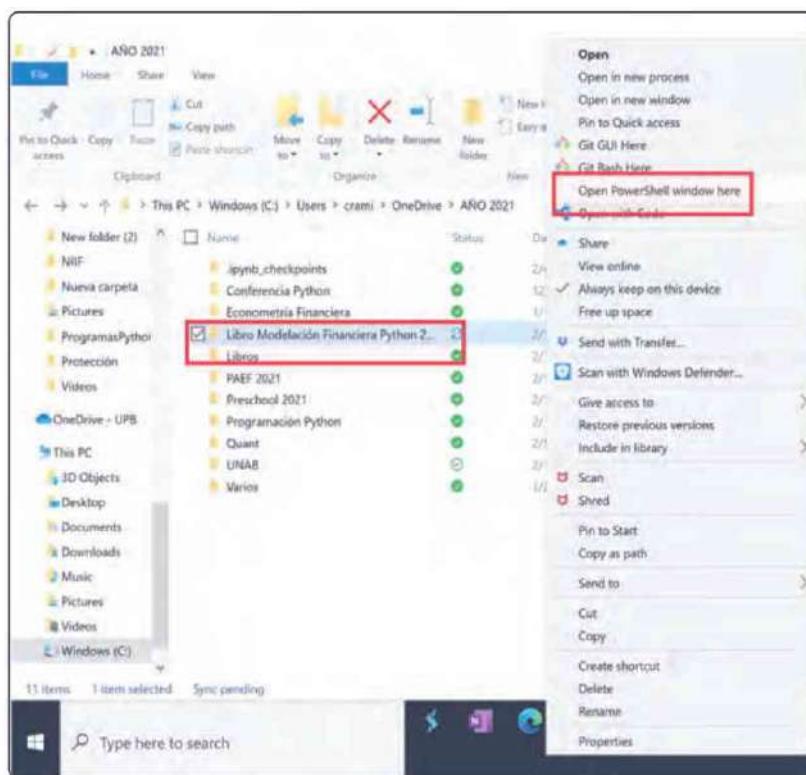
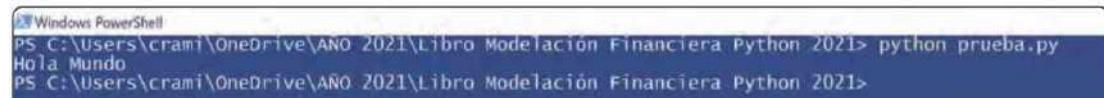


Figura 2.4. Pasos para abrir una ventana de terminal.

Una vez abierta la ventana de terminal en el directorio en el que se encuentra el programa, puede ejecutar el programa. En la figura 2.5 se ejecuta el programa *prueba.py* que escribe ¡Hola, mundo! en la pantalla.



```
Windows PowerShell
PS C:\Users\crami\OneDrive\AÑO 2021\Libro Modelación Financiera Python 2021> python prueba.py
Hola Mundo
PS C:\Users\crami\OneDrive\AÑO 2021\Libro Modelación Financiera Python 2021>
```

Figura 2.5. Ejecución programa Python en la ventana de la terminal

2.2.3 La ejecución del entorno interactivo IDLE

Python es el software que interpreta y ejecuta los programas escritos en dicho lenguaje. El IDLE (Integrated Development Environment por sus siglas en inglés) es el ambiente integrado de desarrollo en el que escribimos las líneas de código del programa. Si ya has instalado Python, abre una consola o terminal (presiona simultáneamente las teclas Windows + R) y ejecuta el comando **Python3**. Este comando lanzará el intérprete de Python correspondiente. Debes ver algo similar a la imagen de la figura 2.6.



```
C:\WINDOWS\system32\cmd.exe - python3
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

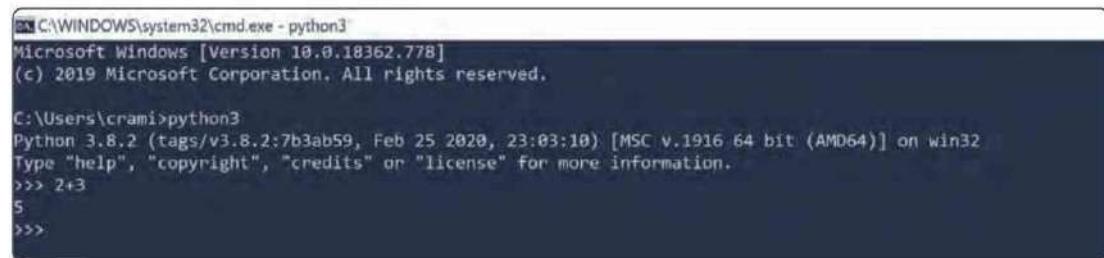
C:\Users\crami>python3
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Figura 2.6. IDLE de Python

Si te fijas bien, en la primera línea podemos ver la versión del intérprete de Python que tenemos instalado en el computador. En nuestro caso es la versión 3.8.2.

En el intérprete de Python podemos escribir expresiones e instrucciones que este interpretará y ejecutará.

Puedes probar, por ejemplo, a escribir $2 + 3$. El resultado debe ser el siguiente:

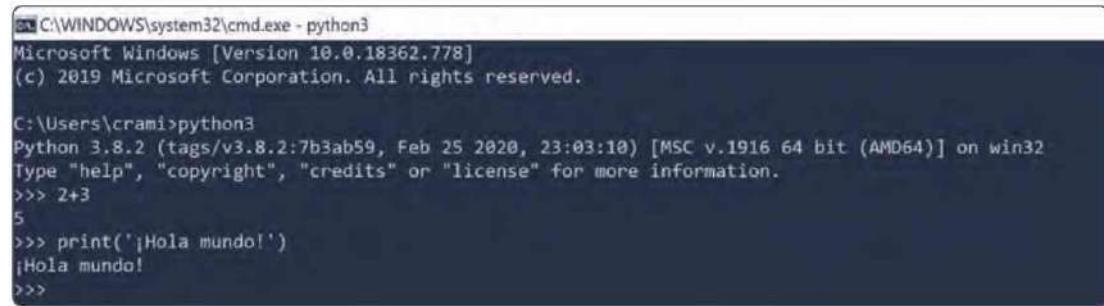


```
C:\WINDOWS\system32\cmd.exe - python3
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\crami>python3
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> 
```

Figura 2.7. Ejecución expresión suma

O ejecutar la instrucción `print('¡Hola mundo!')`:



```
C:\WINDOWS\system32\cmd.exe - python3
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\crami>python3
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> print('¡Hola mundo!')
¡Hola mundo!
>>>
```

Figura 2.8. Ejecución instrucción imprimir Hola mundo

Para salir del intérprete basta con ejecutar la instrucción `quit()`.

Ejemplo. Primer programa en Python.

Normalmente, los programas en Python se escriben en archivos con la extensión .py. Estos archivos se pasan al intérprete de Python para que los interprete y ejecute.

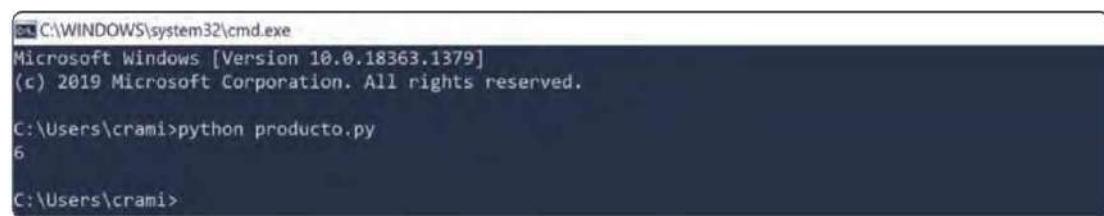
Vamos a verlo con un ejemplo. Crea con un editor de texto un archivo llamado `producto.py` con el siguiente contenido:

```
producto = 2 * 3
print (producto)
```

A continuación abre la terminal, sitúate en el directorio en el que creaste el archivo `producto.py` y ejecuta lo siguiente:

Python suma.py

En el terminal verás que aparece el número 6 como resultado de ejecutar el programa anterior. ¿Qué ha ocurrido aquí? Básicamente que el intérprete de Python ha leído y ejecutado las líneas de código que hemos escrito en el archivo `producto.py`.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\crami>python producto.py
6

C:\Users\crami>
```

Figura 2.9. Ejecución instrucción multiplicar 2*3

No obstante, aunque esta forma de escribir código puede ser útil para aprender y en casos muy puntuales, no es la habitual a la hora de escribir un script o programa en Python.

2.2.4 Editores de texto para Python

El editor de texto es considerado una herramienta fundamental para el programador. El editor de texto usado en el punto anterior (IDLE de Python), es el editor nativo de Python (la funcionalidad que ofrece es muy básica). Sin embargo, existen una gran variedad de editores de texto que incorporan múltiples funciones y facilitan el trabajo de programación. Elegir un editor de texto es una de las elecciones más personales que un programador puede hacer - Como un jugador de tenis escogiendo su raqueta, o un chef escogiendo su cuchillo favorito-. En la tabla 2.1 se presentan algunos editores de texto comúnmente usados por los programadores de Python.

Editor de texto	¿Dónde se puede obtener?
Sublime Text	https://www.sublimetext.com/3
Thonny	https://thonny.org/
Geany	https://www.geany.org/
LiClipse	http://www.liclipse.com/

Tabla 2.1 Editores de texto para Python

Para instalar cualquier editor de texto en el computador se debe descargar de su página oficial. Una vez instalado según corresponda con el sistema operativo, lo abrimos y ya podemos empezar a escribir nuestro código Python. Para efectos de mostrar un ejemplo utilizaremos el editor *Sublimetext*.

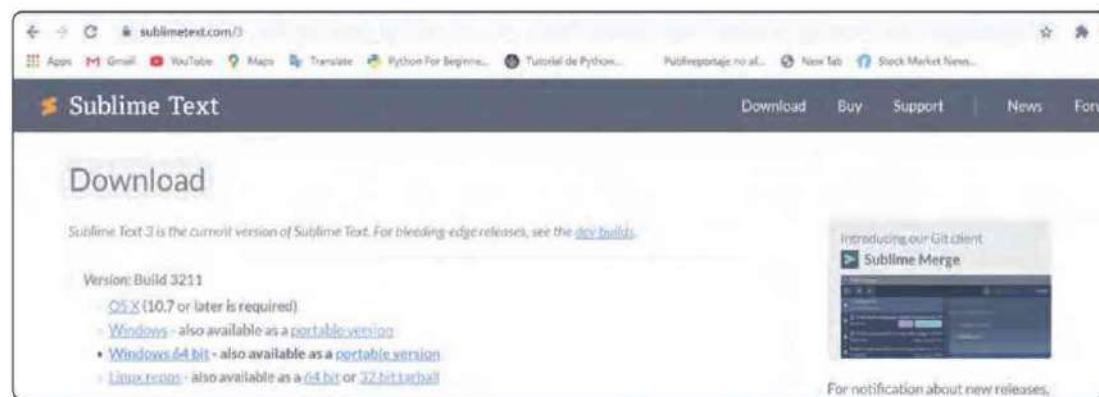


Figura 2.10. Sitio para descargar sublimetext

Ejecutar un script (secuencia de comandos) es tarea sencilla. Asumiendo que ya tenemos Python y Sublimetext instalados en el computador, a continuación seleccionamos el menú file y el archivo (programa) que nos interesa ejecutar (en nuestro caso el archivo *prueba.py*), mediante el atajo de teclado Ctrl+B en Windows y Linux o Cmd+B en macOS ejecutamos el programa, y en la parte inferior del editor nos aparecerá una sección con el resultado de nuestro código o los posibles errores sintácticos que hayamos cometido, esto lo podemos observar en la figura 2.11.

The screenshot shows a Sublime Text window with the title bar "C:\Users\crami\OneDrive\AÑO 2021\Libro Modelación Financiera Python 2021\prueba.py - Sublime Text (UNREGISTERED)". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu is a toolbar with icons for back, forward, and search. The main editor area contains a single line of Python code: "print('Hola Mundo')". In the bottom right corner of the editor, there is a status bar showing "Line 1, Column 1". At the bottom of the window is a terminal-like interface displaying the output of the program: "Hola Mundo [Finished in 0.8s]".

Figura 2.11. Ejecución archivo prueba.py con el editor Sublimetext

Para cualquier programador, y por extensión, para cualquier científico de datos, el entorno de desarrollo integrado (IDE) es una herramienta esencial. Los IDE están diseñados para maximizar la productividad del programador. Por lo tanto, a lo largo de los años, este software ha evolucionado para hacer que la tarea de codificación sea menos complicada. Elegir el IDE correcto para cada persona es crucial y, desafortunadamente, no hay un entorno de programa “único para todos”. La mejor solución es probar los IDE más populares entre la comunidad y mantener lo que mejor se adapte en cada caso.

2.2.5 Jupyter Notebook

Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos que contienen código (en Python y en otros lenguajes de programación), ecuaciones, visualizaciones y texto narrativo. Además, es una aplicación muy utilizada en el campo de la Ciencia de Datos (Data Science) para crear y compartir documentos que incluyen: limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización de datos, aprendizaje automático y mucho más. Permite editar y ejecutar documentos de notebook a través de cualquier navegador web (Explorer, Chrome, entre otros); y puede ejecutarse en un escritorio local que no requiere acceso a Internet o puede instalarse en un servidor remoto y acceder a través de Internet. También se puede ejecutar Jupyter Notebook sin ninguna instalación.

Recomendamos instalar Jupyter Notebook utilizando la plataforma de distribución libre y abierta Anaconda (incluye Python, Jupyter Notebook y otros paquetes de uso común para la computación científica y la ciencia de los datos). Anaconda es el estándar de la industria para desarrollar, probar y capacitar en una sola máquina), es una excelente opción. Si sigues el enlace (<https://www.anaconda.com/download/>) llegarás a la página de descarga de Anaconda.

Como se observa en la figura 2.12 puedes elegir entre los instaladores para Windows, macOS y Linux:

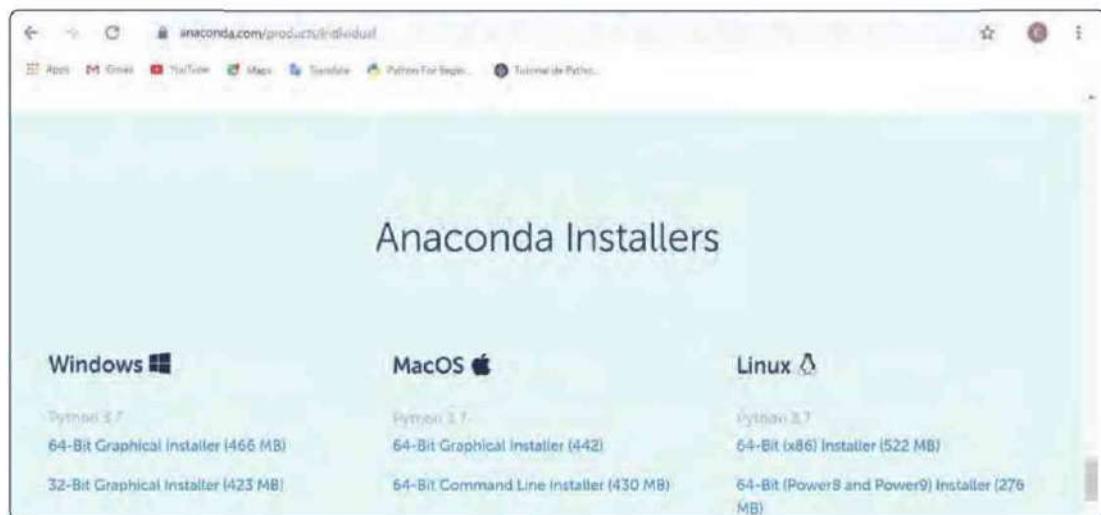


Figura 2.12. Instaladores Anaconda

Después de terminar el proceso de instalación, Anaconda nos brindará acceso a diversas herramientas relacionadas con el desarrollo en Python. Una forma fácil de acceder a todas estas herramientas instaladas es mediante el uso de Anaconda Navigator, que es una interfaz gráfica de usuario (GUI) de escritorio incluida en

la distribución de Anaconda® que nos permite iniciar aplicaciones y administrar fácilmente los paquetes y entornos sin usar la interfaz de línea de comandos. En la figura 2.13 se puede observar como ingresar a la interfaz de Anaconda:

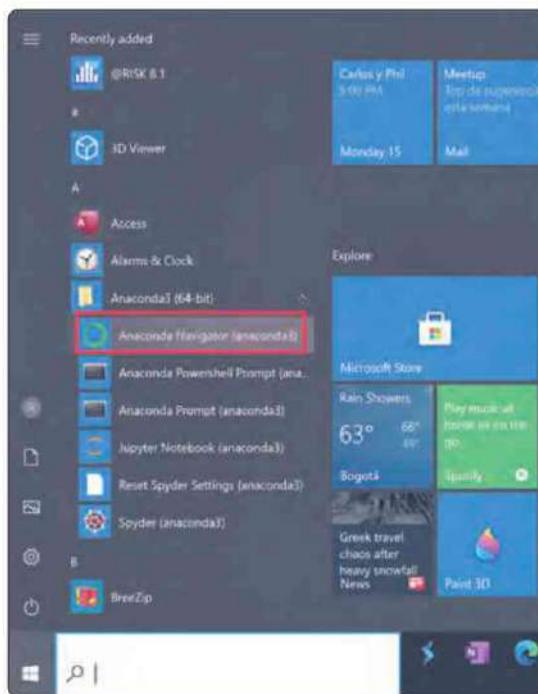


Figura 2.13. Acceso a la interfaz gráfica de Anaconda

Ahora en la figura 2.14 podemos observar la interfaz gráfica de usuario de Anaconda.

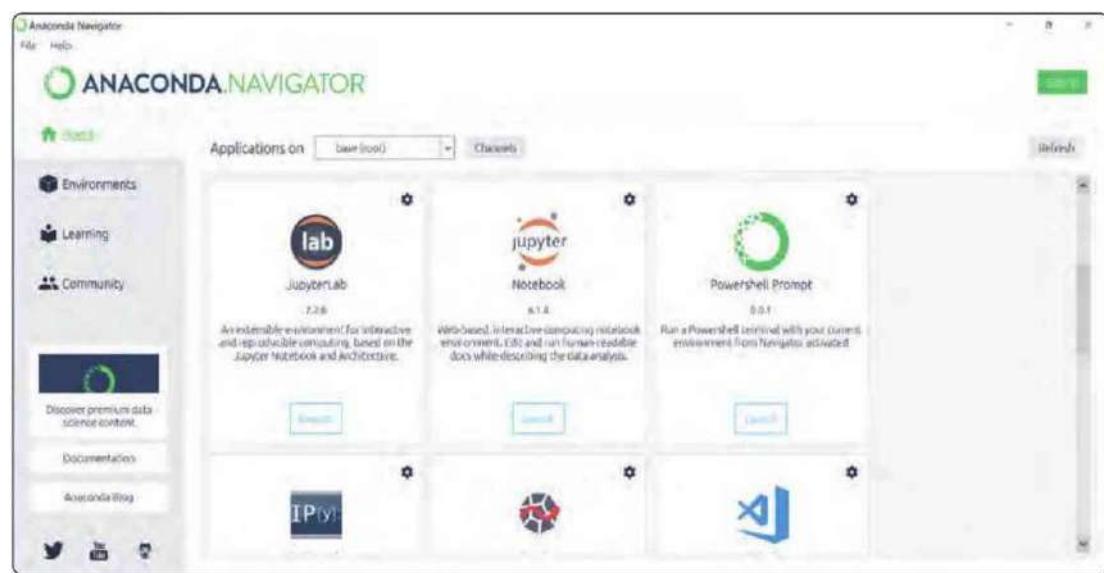


Figura 2.14. Interfaz gráfica de usuario de Anaconda.

Un simple clic en el ícono de Jupyter Notebook abrirá nuestro computador en el navegador web predeterminado (como Google Chrome, Mozilla Firefox, Safari y Windows IE / Edge) y se desplegará el cuaderno de Jupyter como se observa en la figura 2.15:



Figura 2.15. Cuaderno de Jupyter

Además, observamos un botón llamado “New” (figura 2.16) que, al hacer clic, nos permite seleccionar un kernel de Python (las opciones dependen de lo que esté instalado en nuestro servidor local) del menú desplegable.

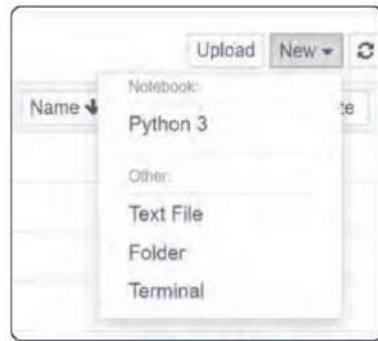


Figura 2.16. Kernel o interprete de líneas de comando de Python

Si le das clic a Python 3 se abrirá un nuevo cuaderno de Jupyter y se verá así (figura 2.17):

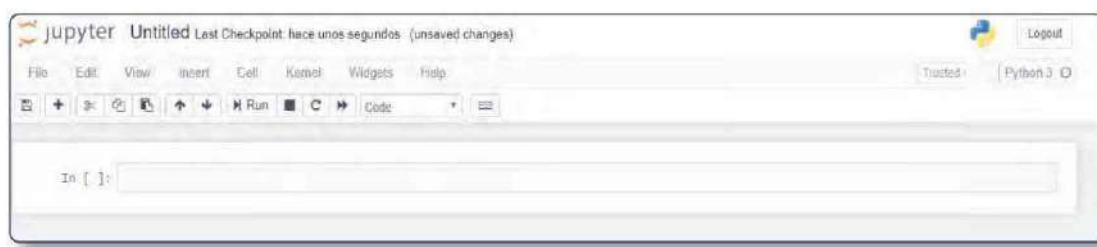


Figura 2.17. Cuaderno de Jupyter

Ya tenemos creado nuestro Jupyter Notebook y podemos empezar a usarlo para sumergirnos en el mundo de Python.

La forma tradicional de correr un programa en Python es con el comando Python *nombre.py*, donde *nombre.py* es un archivo con código fuente Python.

En lugar de eso, a lo largo de este libro utilizaremos un servidor de Jupyter Notebook con cuadernos de código. Estos *cuadernos (notebooks)* nos permiten combinar texto y código, organizados en *celdas*, lo cual es más cómodo para probar cosas nuevas y documentar lo que hacemos.

Los cuadernos tienen dos tipos de celdas, de texto y de código. Las celdas de texto se escriben con Markdown, un lenguaje de marcado parecido al que utiliza Wikipedia para sus páginas o al HTML. Las celdas de código son *ejecutables*, es decir, se pueden correr individualmente (con *ctrl+enter* o desde el menú Cell -> Run Cells). Veamos un ejemplo: en la figura 1 se puede observar una celda de texto:



Figura 2.18. Celda de texto escrita con Markdown

En la figura 2.19 se muestra una celda de código posterior a la celda de texto. En la celda de código se escriben dos instrucciones: Imprimir el texto “Hola Mundo” e imprimir el número 4. Para ambas instrucciones se muestran los resultados en pantalla.

A screenshot of a Jupyter Notebook interface. The title bar says 'jupyter Untitled Last Checkpoint: 3 hours ago (unsaved changes)'. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 kernel selection. Below the menu is a toolbar with icons for file operations like New, Open, Save, and Run, along with Cell type dropdowns for Code, Markdown, and others. The main content area shows a code cell with the following content:

```
In [2]: # Esta es una celda de código.

# La función print puede imprimir varias cosas
print("Hola Mundo") # impresión de un string
print(4) # impresión de un número
```

The output cell below it shows the results:

```
Hola Mundo
4
```

Figura 2.19. Celda que contiene código

2.2.6 Usando Python con Jupyter Notebook en Google Colaboratory

Google Colaboratory o Colab es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube.

Python trae consigo un modo interactivo que con un intérprete de línea de comandos permite lanzar sentencias y obtener los resultados, pero sus funcionalidades se quedaron cortas para los desarrolladores, por lo que surgió IPython, que más tarde evolucionaría en Jupyter.

Jupyter es un entorno interactivo que permite desarrollar código Python de manera dinámica. Jupyter se ejecuta en local como una aplicación cliente-servidor y posibilita tanto la ejecución de código como la escritura de texto, favoreciendo así la interactividad del entorno y que se pueda entender el código como la lectura de un documento.

Como se comentó al inicio, Colab es gratuito y forma parte de la suite de aplicaciones de Google en la nube. Por ello, para utilizarlo basta con acceder a una cuenta de Google y, o bien entrar directamente al enlace de Google Colab o ir a Google Drive, pulsar el botón de «Nuevo» y desplegar el menú de «Más» para seleccionar «Colaboratory», lo que creará un nuevo **cuaderno** (notebook). Ver la figura 2.20:

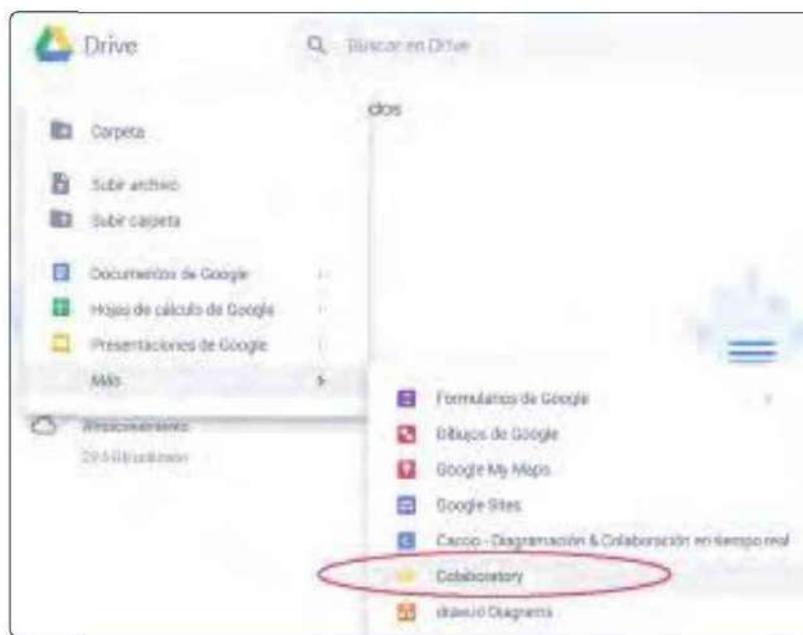


Figura 2.20. Acceder a un notebook en Colaboratory

El entorno de trabajo de un notebook en Colab tiene la siguiente apariencia al crearlo (figura 2.21):



Figura 2.21. Apariencia de un notebook en Colab

En la parte superior se encuentra el nombre del cuaderno (se puede cambiar) y su formato .ipynb, que viene de IPython Notebook y es un formato que nos permite ejecutar cuadernos tanto en Jupyter como en Colab. El archivo .ipynb contiene en formato JSON (formato de texto sencillo para el intercambio de datos) cada celda y su contenido.

Un cuaderno está compuesto por celdas. Una celda es la unidad mínima de ejecución dentro de un cuaderno, es decir, es donde incluimos nuestro código y lo ejecutamos. Para ejecutar una celda podemos pulsar el botón con el icono de play que se encuentra a la izquierda o pulsando Ctrl+Enter (ejecutar celda) o Shift+Enter (ejecutar celda y saltar a la siguiente). Tras la ejecución, debajo de la celda encontramos el resultado (si lo tiene).



Figura 2.22. Ejecución instrucción en cuaderno Colab

En la parte izquierda de una celda ejecutada se puede observar un número entre corchetes. Este número indica el orden en el que se ha ejecutado cada celda.

En la figura 2.22, por ejemplo, se puede ver un 1 porque se ha ejecutado 1 vez dicha celda. Si tuviésemos dos celdas y ejecutásemos la primera y después la segunda, en la primera aparecería un 1 y en la segunda un 2. Si pasamos el cursor por esta parte izquierda de la celda también podemos ver información sobre quién ejecutó la celda, en qué momento y cuánto tardó la ejecución.

Desde los botones de la parte superior o en el menú «Insertar» podemos añadir nuevas celdas, tanto específicas para código como para texto. La distinción que hace Colab sobre ellas es que las celdas de código son ejecutables, mientras que las de texto muestran directamente el texto que incluyamos y además contienen un pequeño editor de texto.

Cuando creamos un nuevo cuaderno, este es «estático», es decir, vemos su contenido, pero no estamos conectados a ningún entorno de ejecución. Nuestro cuaderno se conecta a una **VM de Google Compute Engine** (la infraestructura de máquinas virtuales de Google en la nube) cuando ejecutamos una celda o pulsamos sobre el botón de «Conectar». Al hacerlo, el cuaderno toma un momento en conectarse y después muestra, de ahí en adelante, el espacio de RAM y disco que estamos consumiendo (figura 2.23). La máquina en un inicio cuenta con 12 GB de RAM y 50 GB de almacenamiento en disco disponibles para el uso.



Figura 2.23. Espacio de RAM y disco que se está consumiendo

La duración de la máquina virtual a la que nos conectamos, es decir, el **tiempo máximo** que podemos estar conectados a una misma máquina desde un cuaderno es de **12 horas**. Aquí hay que tener cuidado, sobre todo si estamos llevando a cabo ejecuciones que toman mucho tiempo: si pasamos más de 90 minutos sin utilizar un cuaderno, el entorno se desconecta. Para que no se desconecte basta con dejar la ventana del navegador abierta o celdas ejecutándose. La ventaja que tiene para estos casos es que, si dejamos una celda ejecutándose y cerramos el navegador, la ejecución continuará y si posteriormente abrimos el navegador tendremos nuestro resultado.

2.3 CONOCIENDO LO BÁSICO DE PYTHON

Antes de iniciar a escribir programas para construir modelos financieros, primero es necesario aprender los conceptos básicos del lenguaje, nociones que se irán aplicando a lo largo del libro. Python se compone de una serie de elementos que constituyen su estructura fundamental, esto incluye variables, tipos de datos, operadores, comentarios, estructuras de control de flujo, módulos, paquetes y funciones entre otros. Veamos:

2.3.1 Los comentarios

```
# Esto es un comentario
```

Los comentarios facilitan la escritura de código, ya que nos ayudan a nosotros (y a otros) a comprender por qué se escribió un fragmento de código en particular. Otra cosa sobre los comentarios es que ayudan a mejorar la legibilidad del código.

Cuando agrega la sintaxis anterior, el intérprete de Python entiende que es un comentario. Todo después de # no se ejecuta.

Quizás se pregunte por qué debería usar los comentarios. Imagina que eres un desarrollador y te han asignado un gran proyecto. El proyecto tiene más de mil líneas de código. Para comprender cómo funciona todo, deberá ir línea por línea y leer todo el código.

¿Cuál sería una mejor solución? Ah-ha! Comentarios, los comentarios nos ayudan a entender por qué se escribió un fragmento de código en particular y qué devuelve o hace. Considérelo como documentación para cada pieza de código.

2.3.2 Indentación

Otra parte interesante de este lenguaje es la indentación (sangría). ¿Por qué? Bueno, la respuesta es simple: hace que el código sea legible y esté bien formateado. Es obligatorio en Python seguir las reglas de sangría. Si no se sigue la sangría adecuada, obtendrá el siguiente error:

```
IndentationError: unexpected indent
```

Mira, incluso los errores en Python son tan legibles y fáciles de entender. Al principio, puede ser molesto por la compulsión a la sangría. Pero con el tiempo comprenderás que la sangría es amiga de un desarrollador.

2.3.3 Variables

Como su nombre lo indica, una variable es algo que puede cambiar. Una variable es una forma de referirse a una ubicación de memoria utilizada por un programa de computador.

Bueno, en la mayoría de los lenguajes de programación necesitas asignar el tipo a una variable. Pero en Python, no es necesario. Por ejemplo, para declarar un número entero en C, se usa la siguiente sintaxis: int num = 5 , en Python es num = 5.

Vaya al Shell interactivo de Python y realice la operación paso a paso:

Entero: valores numéricos que pueden ser positivos, negativos o cero sin un punto decimal.

```
>>> num = 5
>>> print(num)
5
>>> type(num)
<class 'int'>
```

Como puede ver aquí, hemos declarado una variable num y hemos asignado 5 como valor. El método de tipo incorporado de Python se puede utilizar para verificar el tipo de variable. Cuando verificamos el tipo de num, vemos la salida <clase 'int'>. Por ahora, solo concéntrate en el int en esa salida. int representa un número entero.

Floating: similar a un entero pero con una ligera diferencia: los flotantes son un valor numérico con un decimal.

```
>>> num = 5.0
>>> print(num)
5.0
>>> type(num)
<class 'float'>
```

Aquí hemos asignado un número con un solo decimal al num. Cuando verificamos el tipo de num podemos ver que es flotante.

Cadena: una formación de caracteres o enteros. Se pueden representar con comillas dobles o simples.

```
>>> greet = "Hello user"
>>> print(greet)
Hello user
>>> type(greet)
<class 'str'>
```

Aquí hemos asignado una cadena para saludar. El tipo de saludo es una cadena como se puede ver en la salida.

Booleano: un operador binario con un valor verdadero o falso.

```
>>> is_available = True
>>> print(is_available)
True
>>> type(is_available)
<class 'bool'>
```

Aquí hemos asignado un valor verdadero a is_available. El tipo de esta variable es booleano. Solo puede asignar Verdadero o Falso. Recuerde que T y F deben ser mayúsculas o dará un error de la siguiente manera:

```
>>> is_available = true
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

NoneType: se usa cuando no tenemos el valor de la variable.

```
>>> num = None
>>> print(num)
None
>>> type(num)
<class 'NoneType'>
```

2.3.4 Operadores

En la tabla 2.2 se presentan todos los operadores aritméticos disponibles en Python:

OPERADOR	NOMBRE	EJEMPLO
+	Suma	a + b
-	Resta	a - b
*	Multiplicación	a * b
/	División	a / b
%	Módulo	a % b
**	Exponenciación	a ** b
//	División piso	a // b
=	Asignación	a=10
(+=)	Suma abreviada	a += 2 (es lo mismo que a = a + 2)
(-=)	Resta abreviada	a -= 2 (es lo mismo que a = a - 2)
*=	Multiplicación abreviada	a *= 2 (es lo mismo que a = a * 2)
/=	División abreviada	a /= 2 (es lo mismo que a = a / 2)
==	Igual	a == b
!=	No igual	a != b
>	Mayor que	a > b
<	Menor que	a < b
>=	Mayor que o igual a	a >= b
<=	Menor que o igual a	a <= b
and	Verdadero si ambas declaraciones son verdaderas	a < 5 and b > 10
or	Verdadera si cualquiera de las declaraciones es verdadera	a < 5 or b > 11
not	Invierte el resultado. Si el resultado es verdadero, entonces falso y viceversa.	not(a < 5)

Tabla 2.2. Operadores aritméticos disponibles en Python

Repasemos algunos de los operadores y su sintaxis:

Operadores aritméticos: incluyen suma, resta, multiplicación, exponenciación, módulo, división de piso entre otros.

Primero, declararemos dos variables, a y b.

```
>>> a = 6 # Asignación
>>> b = 2
```

Probemos las operaciones aritméticas básicas:

```
>>> a + b # Suma  
8  
>>> a - b # Resta  
4  
>>> a * b # Multiplicación  
12  
>>> a / b # División  
3.0  
>>> a ** b # Exponenciación  
36
```

Para probar otras operaciones aritméticas, cambiemos el valor de a y b.

```
>>> a = 7  
>>> b = 3  
>>> a % b # Módulo  
1  
>>> a // b # División  
2
```

Las operaciones aritméticas abreviadas también están disponibles en Python. Consulte la tabla 2.2 para probarlos. Para imprimir el resultado de las operaciones abreviadas, use la instrucción print.

Operadores de comparación: Estos incluyen igual, mayor que y menor que.

```
>>> a = 5  
>>> b = 2  
>>> a > b # Mayor que  
True  
>>> a < b # Menor que  
False  
>>> a == b # Igual a  
False  
>>> a >= 5 # Mayor que o igual a  
True  
>>> b <= 1 # Menor que o igual a  
False
```

Operadores lógicos: And, or y not

```
>>> a = 10
>>> b = 2
>>> a == 2 and b == 10 # and
False
>>> a == 10 or b == 10 # or
True
>>> not(a == 10) # not
False
>>> not(a == 2)
True
```

2.3.5 Declaraciones condicionales

Como su nombre indica, las declaraciones condicionales se utilizan para evaluar si una condición es verdadera o falsa. Muchas veces, cuando está desarrollando una aplicación, necesita verificar una determinada condición y hacer diferentes cosas según el resultado. En tales escenarios, las declaraciones condicionales son útiles. If, elif y else son las declaraciones condicionales utilizadas en Python.

Podemos comparar la variable, verificar si la variable tiene algún valor o si es booleana, luego verificar si es verdadera o falsa. Vaya al Shell (interprete) de Python y realice la operación paso a paso:

Condición número 1: tenemos un número entero y 3 condiciones aquí. El primero es la condición *if*. Está comprobando si el número es igual a 10. El segundo es la condición *elif*. Aquí estamos comprobando si el número es menor que 10. La última condición es otra. Esta condición se ejecuta cuando ninguna de las condiciones anteriores coincide.

```
>>> number = 5
>>> if number == 10:
...     print("Number is 10")
... elif number < 10:
...     print("Number is less than 10")
... else:
...     print("Number is more than 10")
...
```

Salida

```
number is less than 10
```

NOTA

No es obligatorio verificar que dos condiciones sean iguales en la condición if. Puedes hacerlo en el elif también.

Condición número 2: tenemos un booleano y 2 condiciones aquí. ¿Has notado cómo estamos verificando si la condición es verdadera? Si está disponible, imprima “Sí, está disponible”, de lo contrario imprima “No disponible”.

```
>>> is_available = True  
>>> if is_available:  
...     print("Yes it is available")  
... else:  
...     print("Not available")  
...
```

Salida

```
Yes it is available
```

Condición número 3: aquí hemos invertido la condición número 2 con la ayuda del operador no.

```
>>> is_available = True  
>>> if not is_available:  
...     print("Not available")  
... else:  
...     print("Yes it is available")  
...
```

Salida

```
Yes it is available
```

Condición número 4: Aquí declaramos los datos como None y verificamos si los datos están disponibles o no.

```
>>> data = None
>>> if data:
...     print("data is not none")
... else:
...     print("data is none")
...
```

Salida

```
data is none
```

Condición número 5: también puede usar una sola línea para verificar la condición. La sintaxis para lograr esto es la siguiente:

```
>>> num_a = 10
>>> num_b = 5
>>> if num_a > num_b: print("num_a is greater than num_b")
...
```

Salida

```
num_a is greater than num_b
```

Condición número 6: Otra forma de verificar la condición con una sola línea. La sintaxis para lograr esto es la siguiente:

```
expression_if_true if condition else expression_if_false
```

Veamos un ejemplo concreto:

```
>>> num = 5
>>> print("Number is five") if num == 5 else print("Number is not five")
```

Salida

```
Number is five
```

Condición número 7: también puede usar instrucciones anidadas if-else. La sintaxis para lograr esto es la siguiente:

```
>>> num = 25
>>> if num > 10:
...     print("Number is greater than 10")
...     if num > 20:
...         print("Number is greater than 20")
...     if num > 30:
...         print("Number is greater than 30")
... else:
...     print("Number is smaller than 10")
...
```

Salida

```
Number is greater than 10
Number is greater than 20
```

Condición Número 8: También puede usar el operador AND en una declaración condicional. Indica si la condición 1 y la condición 2 son ambas verdaderas, entonces ejecútelo.

```
>>> num = 10
>>> if num > 5 and num < 15:
...     print(num)
... else:
...     print("Number may be small than 5 or larger than 15")
...
```

Salida

```
10
```

Como el número está entre 5 y 15, obtenemos la salida de 10.

Condición Número 9: También puede usar el operador OR en una declaración condicional. Establece que si condición 1 o condición 2 es verdadera, ejecútelo.

```
>>> num = 10
>>> if num > 5 or num < 7:
...     print(num)
...
```

Salida

```
10
```

¿Estás confundido porque el valor de num es 10 y nuestra segunda condición indica que num es menor que 7? Entonces, ¿por qué obtenemos la salida como 10? Es por la condición OR. Cuando una de las condiciones coincida, la ejecutará.

2.3.6 For loops (Para bucles)

Otro método útil en cualquier lenguaje de programación es un iterador. Si tiene que implementar algo varias veces, ¿qué hará?

```
print("Hello")
print("Hello")
print("Hello")
```

Bueno, esa es una forma de hacerlo. Pero imagina que tienes que hacerlo cien o mil veces. Eso implicaría un montón de declaraciones impresas que tenemos que escribir. Hay una mejor manera denominada iteradores o bucles. Podemos usar un bucle for o while.

Aquí estamos usando el método de rango. Especifica el rango hasta el cual se debe repetir el ciclo. Por defecto, el punto de partida es 0.

```
>>> for i in range(3):
...     print("Hello")
...
```

Salida

```
Hello  
Hello  
Hello
```

También puede especificar el rango de esta manera (1,3).

```
>>> for i in range(1,3):  
...     print("Hello")  
...
```

Salida

```
Hello  
Hello
```

“Hola” solo se imprime dos veces, ya que hemos especificado el rango aquí. Piense en el rango como Número a la derecha - Número a la izquierda.

Bueno, también puede agregar una instrucción else en el ciclo for.

```
>>> for i in range(3):  
...     print("Hello")  
... else:  
...     print("Finished")
```

Salida

```
Hello  
Hello  
Hello  
Finished
```

Vea nuestro ciclo iterado 3 veces (3-0) y una vez hecho esto, ejecutó la instrucción else.

También podemos anidar un bucle for dentro de otro bucle for.

```
>>> for i in range(3):
...     for j in range(2):
...         print("Inner loop")
...     print("Outer loop")
...
...
```

Salida

```
Inner loop
Inner loop
Outer loop
Inner loop
Inner loop
Outer loop
Inner loop
Inner loop
Outer loop
```

Como puede ver, la declaración de impresión del bucle interno se ejecuta dos veces. Después de esa declaración de impresión el bucle externo se ejecuta. De nuevo, el bucle interno se ejecutó dos veces. Entonces, ¿Qué está pasando aquí? Si está confundido, considere esto para resolverlo:

Nuestro intérprete viene y ve que hay un bucle for. Vuelve a bajar y comprueba que hay otro bucle for.

Entonces ahora ejecutará el bucle interno dos veces y saldrá. Una vez terminado, sabe que el bucle for externo le ha indicado que repita dos veces más.

Comienza de nuevo y ve el bucle interno y se repite.

Bueno, también puedes elegir pasar una cierta condición de bucle. ¿Qué significa pasar aquí? Bueno, siempre que ocurra ese bucle for y el intérprete vea la declaración de aprobación, no la ejecutará y pasará a la siguiente línea.

```
>>> for i in range(3):
...     pass
...
...
```

No obtendrá ningún resultado en el Shell.

2.3.7 Bucle While (Mientras)

Otro bucle o iterador disponible en Python es *while loop*. Podemos lograr algunos resultados con la ayuda de un ciclo while como lo hacemos con el ciclo for.

```
>>> i = 0
>>> while i < 5:
...     print("Number", i)
...     i += 1
...
```

Salida

```
Number 0
Number 1
Number 2
Number 3
Number 4
```

Recuerde que siempre que use un ciclo while es importante que agregue una declaración de incremento o que una declaración finalice el ciclo while en algún momento. Si no, entonces el ciclo while se ejecutará para siempre.

Otra opción es agregar una declaración de interrupción en un ciclo while. Esto romperá el ciclo.

```
i=0
while i<5:
    if i==4:
        break
    print("Number",i)
    i+=1
```

Salida

```
Number 0
Number 1
Number 2
Number 3
```

Aquí estamos rompiendo el ciclo while si encontramos que el valor de i es 4.

Otra opción es agregar una instrucción else en el ciclo while. La declaración se ejecutará después de que se complete el ciclo while.

```
i = 0
while i < 5:
    print("Number", i)
    i += 1
else:
    print("Number is greater than 4")
```

Salida

```
Number 0
Number 1
Number 2
Number 3
Number 4
Number is greater than 4
```

La instrucción **continue** se puede usar para omitir la ejecución actual y pasar a la siguiente.

```
>>> i = 0
>>> while i < 6:
...     i += 1
...     if i == 2:
...         continue
...     print("number", i)
... 
```

Salida

```
Number 0
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
```

2.3.8 Input (Entrada) del usuario

Imagine que está creando una aplicación de línea de comandos. Ahora debe tomar la entrada del usuario y actuar en consecuencia. Para hacerlo, puede utilizar el método de entrada incorporado de Python.

La sintaxis para lograr esto es la siguiente:

```
variable = input("....")
```

Ejemplo

```
>>> name = input("Enter your name: ")
Enter your name: Carlos
```

Cuando use el método de entrada y presione enter, se le solicitará el texto que ingrese en el método input. Veamos si nuestra tarea funciona o no:

```
>>> print(name)
Carlos
```

¡¡Ahí está!! Funciona perfectamente. Aquí Carlos es del tipo string (cadena de texto).

```
>>> type(name)
<class 'str'>
```

Probemos un ejemplo más donde asignaremos un número entero en lugar de una cadena y verificaremos el tipo.

```
>>> date = input("Today's date: ")
Today's date: 12
>>> type(date)
<class 'str'>
```

¿Estás confundido? Ingresamos un número entero 12 y todavía nos da su tipo como una cadena. No es un error. Así es como se pretende que funcione la entrada. Para convertir la cadena a entero, utilizaremos la conversión de texto.

2.3.9 Typecasting (Tipografía)

Vimos que el método de entrada también devuelve una cadena para el entero. Ahora queremos comparar esta salida con otro número entero, entonces necesitamos una forma de convertirlo de nuevo a un número entero.

```
>>> date_to_int = int(date)
>>> type(date_to_int)
<class 'int'>
```

Aquí tomamos la fecha que habíamos declarado anteriormente en la sección de entrada del usuario y la convertimos en un entero usando el método `int` incorporado de Python. Esto se llama encasillamiento.

Básicamente, se pueden realizar los siguientes cambios con la ayuda de la conversión de texto:

Número entero a cadena: `str()`

Cadena a número entero: `int()`

Número entero a flotante: `float()`

NOTA

La conversión de flotante a entero también es posible.

```
>>> type(date)
<class 'str'>
# Convertir de tipo string a tipo flotante
>>> date_to_float = float(date)
>>> type(date_to_float)
<class 'float'>
# Convertir de tipo flotante a tipo string
>>> date_to_string = str(date_to_float)
>>> type(date_to_string)
```

```
<class 'str'>
# Convertir de tipo flotante a tipo entero
>>> date_to_int = int(date_to_float)
>>> type(date_to_int)
<class 'int'>
```

2.3.10 Diccionarios

Imagina que quieres almacenar algunos detalles del usuario. Entonces, ¿cómo puedes almacenar estos detalles? Sí, podemos usar variables para almacenarlas de la siguiente manera:

```
>>> fname = "Sharvin"
>>> lname = "Shah"
>>> profession = "Developer"
```

Para acceder a este valor podemos hacer lo siguiente:

```
>>> print(fname)
Sharvin
```

¿Pero es esta una forma elegante y optimizada de acceder? La respuesta es no. Para hacerlo más amigable, almacenemos los datos en un diccionario de valores clave.

¿Qué es un diccionario? Un diccionario es una colección desordenada y mutable (es decir, se puede actualizar).

El siguiente es el formato del diccionario:

```
data = {
    "key" : "value"
}
```

Comprendamos más el diccionario con un ejemplo:

```
>>> user_details = {
...     "fname": "Sharvin",
```

```
...     "lname": "Shah",
...     "profession": "Developer"
... }
```

Cómo acceder a un valor en un diccionario

Podemos acceder al valor dentro de un diccionario de dos maneras. Echaremos un vistazo a ambos y luego los depuraremos para descubrir cuál es mejor.

Método 1: para acceder al valor de la clave fname del diccionario user_details podemos usar la siguiente sintaxis:

```
>>> user_details["fname"]
'Sharvin'
```

Método 2: También podemos acceder al valor de la clave fname del diccionario user_details usando get.

```
>>> user_details.get("fname")
'Sharvin'
```

El método 1 parece más fácil de entender. El problema ocurre cuando intentamos acceder a los datos que no están disponibles en nuestro diccionario.

```
>>> user_details["age"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    KeyError: 'age'
```

Obtenemos un KeyError que indica que la clave no está disponible. Probemos el mismo escenario con el método 2.

```
>>> user_details.get("age")
```

No recibimos nada impreso en nuestra consola. Vamos a depurarlo más para saber por qué sucedió esto. Asigne una variable edad a nuestra operación get y la imprimiremos en nuestra consola.

```
>>> age = user_details.get("age")
>>> print(age)
None
```

Entonces, cuando get no encuentra la clave, establece el valor en None. Debido a esto, no recibimos ningún error. Ahora puede que se pregunte cuál es el correcto. La mayoría de las veces usar el método 2 tiene más sentido, pero para algunas condiciones de verificación estrictas, necesitamos usar el método 1.

¿Cómo verificar si existe una clave?

Tal vez se pregunte cómo verificar si el diccionario tiene una clave en particular o no. Python proporciona las teclas de método integradas () para resolver este problema.

```
>>> if "age" in user_details.keys():
...     print("Yes it is present")
... else:
...     print("Not present")
...
```

Obtendremos el siguiente resultado:

```
Not present
```

¿Qué pasa si queremos verificar si el diccionario está vacío o no? Para entender esto, declaremos un diccionario vacío de la siguiente manera:

```
>>> user_details = {}
```

Cuando usamos if-else en un diccionario directamente, devuelve verdadero si los datos están presentes o falso si está vacío.

```
>>> if user_details:
...     print("Not empty")
... else:
...     print("Empty")
...
```

Salida

Empty

También podemos usar el método incorporado `bool` de Python para verificar si el diccionario está vacío o no. Recuerde que `bool` devuelve `False` si el diccionario está vacío y `True` si está lleno.

```
>>> bool(user_details)
False
>>> user_details = {
...     "fname" : "Sharvin"
... }
>>> bool(user_details)
True
```

Cómo actualizar el valor de una clave existente

Entonces, ahora sabemos cómo obtener una clave particular y encontrar si existe, pero ¿cómo la actualiza en el diccionario?

Declare un diccionario de la siguiente manera:

```
>>> user_details = {
...     "fname": "Sharvin",
...     "lname": "Shah",
...     "profession": "Developer"
... }
```

Para actualizar el valor, use la siguiente sintaxis:

```
>>> user_details["profession"] = "Software Developer"
>>> print(user_details)
{'fname': 'Sharvin', 'lname': 'Shah', 'profession': 'Software Developer'}
```

Actualizar un valor de clave en el diccionario es lo mismo que asignar un valor a la variable.

Cómo agregar un par clave-valor

La siguiente pregunta es cómo agregar un nuevo valor al diccionario. Agreguemos una clave de edad con un valor de 100.

```
>>> user_details["age"] = "100"
>>> print(user_details)
{'fname': 'Sharvin', 'lname': 'Shah', 'profession': 'Software Developer', 'age': '100'}
```

Como puede ver, se agrega una nueva clave-valor en nuestro diccionario.

Cómo eliminar un par clave-valor

Para eliminar un valor-clave del diccionario, Python proporciona un método incorporado llamado pop.

```
>>> user_details.pop("age")
'100'
>>> print(user_details)
{'fname': 'Sharvin', 'lname': 'Shah', 'profession': 'Software Developer'}
```

Esto elimina el par clave-valor de edad del diccionario user_details. También podemos usar un operador del para eliminar el valor.

```
>>> del user_details["age"]
>>> print(user_details)
{'fname': 'Sharvin', 'lname': 'Shah', 'profession': 'Software Developer'}
```

El método del también se puede utilizar para eliminar el diccionario completo.

Cómo copiar un diccionario

Un diccionario no se puede copiar de forma tradicional. Por ejemplo, no puede copiar el valor de dictA a dictB de la siguiente manera:

```
dictA = dictB
```

Para copiar los valores, debe usar el método de copia.

```
>>> dictB = user_details.copy()  
>>> print(dictB)  
{'fname': 'Sharvin', 'lname': 'Shah', 'profession': 'Software Developer'}
```

2.3.11 Listas

Imagine que tiene un montón de datos que no están etiquetados. En otras palabras, cada dato no tiene una clave que lo defina. Entonces, ¿cómo lo guardarás? Listas al rescate. Se definen de la siguiente manera:

```
data = [ 1, 5, "xyz", True ]
```

Una lista es una colección de datos aleatorios, ordenados y mutables (es decir, se puede actualizar).

Cómo acceder a los elementos de la lista

Intentemos acceder al primer elemento:

```
>>> data[1]  
5
```

Espera, ¿qué pasó aquí? Estamos tratando de acceder al primer elemento pero estamos obteniendo el segundo elemento. ¿Por qué?

La indexación de la lista comienza desde cero. Entonces, ¿qué quiero decir con esto? La indexación de la posición de los elementos comienza desde cero. La sintaxis para acceder a un elemento es la siguiente:

```
list[position_in_list]
```

Para acceder al primer elemento necesitamos acceder a él de la siguiente manera:

```
>>> data[0]  
1
```

También puede especificar un rango para acceder al elemento entre esas posiciones.

```
>>> data[2:4]
['xyz', True]
```

Aquí, el primer valor representa el inicio, mientras que el último valor representa la posición hasta la que queremos el valor.

Cómo agregar un elemento a una lista

Para agregar un elemento en la lista, necesitamos usar el método de agregado (append) proporcionado por Python.

```
>>> data.append("Hello")
>>> data
[1, 5, 'abc', True, 'Hello']
```

Cómo cambiar el valor de un elemento

Para cambiar el valor de un elemento, use la siguiente sintaxis:

```
>>> data[2] = "abc"
>>> data
[1, 5, 'abc', True, 'Hello']
```

Cómo eliminar un elemento de una lista

Para eliminar un elemento de una lista, podemos usar el método de eliminación (remove) incorporado de Python.

```
>>> data.remove("Hello")
>>> data
[1, 5, 'abc', True]
```

Cómo recorrer una lista

También podemos recorrer la lista para encontrar un determinado elemento y operarlo.

```
>>> for i in data:  
...     print(i)  
...
```

Salida

```
1  
5  
abc  
True
```

Cómo verificar si un elemento existe o no

Para verificar si un elemento en particular existe o no en la lista, podemos usar el bucle if de la siguiente manera:

```
>>> if 'abc' in data:  
...     print("yess...")  
...  
yess..
```

Cómo copiar datos de la lista

Para copiar datos de una lista a otra, necesitamos usar el método de copia (copy).

```
>>> List2 = data.copy()  
>>> List2  
[1, 5, 'abc', True]
```

Cómo verificar la longitud de una lista

También podemos verificar la longitud de la lista utilizando el método de longitud (`len`) incorporado de Python.

```
>>> len(data)
4
```

Cómo unir dos listas

Para unir dos listas podemos usar el operador `+`.

```
>>> list1 = [1, 4, 6, "hello"]
>>> list2 = [2, 8, "bye"]

>>> list1 + list2
[1, 4, 6, 'hello', 2, 8, 'bye']
```

¿Qué sucede si intentamos acceder a una posición del elemento que no está disponible en la lista? Obtenemos un índice de lista de error fuera de rango en tal condición.

```
>>> list1[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

2.3.12 Tuplas

La tupla es un tipo de datos que está ordenado e inmutable (es decir, los datos no se pueden cambiar).

Vamos a crear una tupla:

```
>>> data = ( 1, 3 , 5, "bye")
>>> data
(1, 3, 5, 'bye')
```

Cómo acceder a un elemento tupla

Podemos acceder a elementos en la tupla de la misma manera que accedemos a ellos en una lista:

```
>>> data[3]
'bye'
```

Podemos acceder al rango de índice de la siguiente manera:

```
>>> data[2:4]
(5, 'bye')
```

Cómo cambiar el valor de una tupla

No podemos cambiar el valor de la tupla ya que es inmutable. Obtenemos el siguiente error si intentamos cambiar el valor de una tupla:

```
>>> data[1] = 8
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Hay una solución disponible para cambiar el valor de una tupla:

```
>>> data = ( 1, 3 , 5, "bye")
>>> data_two = list(data) # Convertir data a lista
>>> data_two[1] = 8 # Actualizar el valor ya que la lista es mutable
>>> data = tuple(data_two) # Convertir de nuevo a tupla
>>> data
(1, 8, 5, 'bye')
```

Todos los demás métodos que hemos visto en la lista también son aplicables para la tupla.

NOTA

Una vez que se crea una tupla, no se puede agregar un nuevo valor.

Cómo acceder a un elemento tupla

Podemos acceder a elementos en la tupla de la misma manera que accedemos a ellos en una lista:

```
>>> data[3]
'bye'
```

Podemos acceder al rango de índice de la siguiente manera:

```
>>> data[2:4]
(5, 'bye')
```

Cómo cambiar el valor de una tupla

No podemos cambiar el valor de la tupla ya que es inmutable. Obtenemos el siguiente error si intentamos cambiar el valor de una tupla:

```
>>> data[1] = 8
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Hay una solución disponible para cambiar el valor de una tupla:

```
>>> data = ( 1, 3 , 5, "bye")
>>> data_two = list(data) # Convertir data a lista
>>> data_two[1] = 8 # Actualizar el valor ya que la lista es mutable
>>> data = tuple(data_two) # Convertir de nuevo a tupla
>>> data
(1, 8, 5, 'bye')
```

Todos los demás métodos que hemos visto en la lista también son aplicables para la tupla.

NOTA

Una vez que se crea una tupla, no se puede agregar un nuevo valor.

2.3.13 Conjuntos

Los conjuntos son otro tipo de datos en Python que no están ordenados ni indexados. Los conjuntos se declaran de la siguiente manera:

```
>>> data = { "hello", "bye", 10, 15 }
>>> data
{10, 15, 'hello', 'bye'}
```

Cómo acceder a un valor

Como los conjuntos no están indexados, no podemos acceder directamente al valor de un conjunto. Por lo tanto, para acceder al valor en el conjunto necesita usar un bucle for.

```
>>> for i in data:
...     print(i)
...
10
15
hello
bye
```

Cómo cambiar un valor

Una vez que se crea el conjunto, los valores no se pueden cambiar.

Cómo agregar un elemento

Para agregar un elemento al conjunto, Python proporciona un método incorporado llamado agregar (add).

```
>>> data.add("test")
>>> data
{10, 'bye', 'hello', 15, 'test'}
```

Cómo verificar la longitud

Para verificar la longitud del conjunto usamos el método len.

```
>>> len(data)
5
```

Cómo quitar un elemento

Para eliminar un elemento, use el método remove:

```
>>> data.remove("test")
>>> data
{10, 'bye', 'hello', 15}
```

2.3.14 Funciones y argumentos

Las funciones son una forma práctica de declarar una operación que queremos realizar. Con la ayuda de funciones, puede separar la lógica de acuerdo con la operación.

Las funciones son un bloque de código que nos ayuda en la reutilización de la lógica repetitiva. Las funciones pueden ser tanto integradas como definidas por el usuario.

Para declarar una función usamos la palabra clave *def*. La siguiente es la sintaxis de las funciones:

```
>>> def hello_world():
...     print("Hello world")
...
```

Aquí estamos declarando una función que, cuando se llama, imprime una declaración de “Hello world”. Para llamar a una función usamos la siguiente sintaxis:

```
>>> hello_world()
```

Obtendremos el siguiente resultado:

```
.....Hello world.....
```

Recuerde que los corchetes () en una llamada de función significa ejecutarlo. Quite esos corchetes e intente la llamada nuevamente.

```
.....>>> hello_world.....
```

Obtendrá el siguiente resultado:

```
.....<function hello_world at 0x1083eb510>.....
```

Cuando eliminamos los corchetes de la llamada a la función, nos da una referencia de función. Aquí arriba, como puede ver, la referencia de la función hello_world apunta a esta dirección de memoria 0x1083eb510.

Considere que tiene que realizar una operación de suma. Puede hacerlo declarando a y b y luego realizando la suma.

```
.....>>> a = 5  
>>> b = 10  
>>> a + b  
15.....
```

Este es un camino a seguir. Pero ahora considere que el valor de a y b ha cambiado y debe hacerlo nuevamente.

```
.....>>> a = 5  
>>> b = 10  
>>> a + b  
15  
>>> a = 2  
>>> b = 11  
>>> a + b  
13.....
```

Esto todavía parece factible. Ahora imagine que necesitamos agregar un conjunto de dos números cien veces. Los números dentro del conjunto son diferentes para cada cálculo. Eso es mucho por hacer. No se preocupe, tenemos una función a nuestra disposición para resolver este problema.

```
>>> def add(a,b):  
...     print(a+b)  
...
```

Aquí estamos agregando a y b como argumento obligatorio para la función add. Para llamar a esta función usaremos la siguiente sintaxis:

```
>>> add(10,5)
```

Salida

15

¿Ves lo fácil que es definir una función y usarla? Entonces, ¿qué pasa si no pasamos un argumento?

Python lanza un `TypeError` y nos informa que la función requiere dos argumentos.

```
>>> add()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: add() missing 2 required positional arguments: 'a' and 'b'
```

¿Puedes adivinar qué sucederá si pasamos un tercer argumento?

```
>>> add(10,5,1)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: add() takes 2 positional arguments but 3 were given
```

Bueno, Python nos informará que hemos pasado 3 argumentos, pero solo hay 2 argumentos posicionales.

Entonces, ¿qué podemos hacer cuando no sabemos cuántos argumentos puede tomar una función? Para resolver este problema, usamos args y kwargs.

Args

Cuando no sepa cuántos argumentos se pasarán a la función, use args y kwargs (los kwargs se analizan a continuación).

Para pasar un número n de argumentos a una función, usamos args. Agregamos un * delante del argumento.

Recuerde que cuando adjunte un * delante, recibirá una tupla de argumentos.

```
>>> def add(*num):
...     print(num)
...
```

Aquí * num es una instancia de args. Ahora, cuando llamamos a la función add, podemos pasar un número n de argumentos y no arrojará un TypeError.

```
>>> add(1,2,3)
(1, 2, 3)

>>> add(1,2,3,4)
(1, 2, 3, 4)
```

Ahora, para realizar la operación de suma, utilizaremos la suma de la función incorporada de Python.

```
>>> def add(*num):
...     print(sum(num))
...
```

Ahora, cuando llamemos a la función add, obtendremos el siguiente resultado:

```
>>> add(1,2,3) # Llamada de función  
6  
>>> add(1,2,3,4) # Llamada de función  
10
```

Argumentos de palabras clave

Hay momentos en que no conocemos el orden de los argumentos que se pasarán a nuestra función cuando se llame. En tal escenario, usamos argumentos de palabras clave porque puede pasarlos en cualquier orden en su llamada y nuestra función sabrá el valor. Analicemos el siguiente ejemplo:

```
>>> def user_details(username, age):  
...     print("Username is", username)  
...     print("Age is", age)  
...
```

Llamemos a esta función de la siguiente manera:

```
>>> user_details("Sharvin", 100)
```

Obtendremos el siguiente resultado:

```
Username is Sharvin  
Age is 100
```

Bueno, esto parece correcto, pero imagina si llamamos a nuestra función de esta manera:

```
>>> user_details(100, "Sharvin")
```

Obtendremos el siguiente resultado:

```
Username is 100  
Age is Sharvin
```

Esto no se ve bien. Lo que sucedió es que Username tomó el valor de 100, mientras que Age tomó el valor de “Sharvin”. En escenarios como este donde no conocemos el orden de los argumentos, podemos usar argumentos de palabras clave cuando llamamos a la función:

```
>>> user_details(age=100, username="Sharvin")
```

Salida

```
Username is 100  
Age is Sharvin
```

Argumento predeterminado

Supongamos que hay una condición en la que no estamos seguros si un argumento en particular obtendrá un valor o no cuando se llama a la función. En tal escenario, podemos usar argumentos predeterminados de la siguiente manera:

```
>>> def user_details(username, age = None):  
...     print("Username is", username)  
...     print("Age is", age)  
...
```

Aquí estamos asignando un `None` a nuestro argumento `age`. Si no pasamos un segundo argumento mientras llamamos a la función, tomará `None` como valor predeterminado.

Llamemos a la función:

```
>>> user_details("Sharvin")
```

Salida

```
Username is Sharvin  
Age is None
```

Si pasamos el segundo argumento, anulará `None` y lo usará como valor.

```
>>> user_details("Sharvin", 200)
Username is Sharvin
Age is 200
```

Pero, ¿qué pasará si asignamos el primer argumento en nuestra función como predeterminado y el segundo como un argumento obligatorio? Vaya al shell de Python y pruebe esto:

```
>>> def user_details(username=None, age):
...     print("Username is", username)
...     print("Age is", age)
...
```

Obtendrás el siguiente error:

```
File "<stdin>", line 1
SyntaxError: non-default argument follows default argument
```

RECUERDE

Todos los argumentos obligatorios deben declararse primero y luego debe declararse el argumento predeterminado.

kwargs

Puede haber una situación en la que no sepa cuántos argumentos de palabras clave se pasarán a la función. En tal escenario, podemos usar Kwargs.

Para usar kwargs ponemos ** delante del argumento.

RECUERDE

Cuando adjunte un ** delante, recibirá un diccionario de argumentos.

Comprendamos esto con un ejemplo. Declararemos una función que acepte el nombre de usuario como argumento con ** delante de él.

```
>>> def user(**username):
...     print(username)
...
```

Cuando llamamos a la función de usuario de la siguiente manera, recibiremos un diccionario.

```
>>> user(username1="xyz",username2="abc")
```

Salida

```
{'username1': 'xyz', 'username2': 'abc'}
```

Entonces, ¿qué está pasando aquí? Se ve igual que los args, ¿verdad?

No, no lo es. En args, no puede acceder a un valor particular por su nombre, ya que tiene la forma de una tupla. Aquí obtenemos los datos en forma de diccionario para que podamos acceder fácilmente al valor.

Considere este ejemplo:

```
>>> def user(**user_details):
...     print(user_details['username'])
...
```

Llamemos a nuestra función:

```
>>> user(username="Sharvin",age="1000")
```

Y obtendrá el siguiente resultado:

```
Sharvin
```

2.3.15 Ámbito <https://dogramcode.com/programacion>

Un ámbito define dónde está disponible una variable o función. Hay dos tipos de alcance en Python: Global y Local.

Alcance global

Una variable o función creada en el cuerpo principal del código de Python se denomina variable o función global y es parte del alcance global. Por ejemplo:

```
>>> greet = "Hello world"
>>> def testing():
...     print(greet)
...
>>> testing()
Hello world
```

Aquí la variable `greet` está disponible globalmente porque se declara en el cuerpo del programa.

Alcance local

Una variable o función creada dentro de una función se llama variable o función local y es parte del ámbito local:

```
>>> def testing():
...     greet = "Hello world"
...     print(greet)
...
>>> testing()
Hello world
```

Aquí se crea `greet` dentro de la función de prueba y solo está disponible allí. Intentemos acceder a él en nuestro cuerpo principal y veamos qué sucede:

```
>>> nt(greet)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'greet' is not defined
```

RECUERDE

Reinic peace la consola de Python presionando `ctrl + d` e iniciando el Shell nuevamente usando el comando `Python3` antes de probar el código anterior. El primer ejemplo hace que declare la variable `greet` en el ámbito global, lo que significa que todavía estará disponible en la memoria cuando ejecute el segundo ejemplo.

Como greet no está disponible globalmente, obtenemos el error de que no está definido.

2.3.16 Declaración de devolución

Hasta ahora nuestras funciones son bastante simples. Están recibiendo datos, procesándolos e imprimiéndolos. Pero en el mundo real, se necesita una función para devolver la salida para que pueda usarse en diferentes operaciones.

Para lograr esto, se utilizan declaraciones de devolución. Recuerde, las declaraciones de devolución son solo parte de funciones y métodos. La sintaxis para la declaración de devolución es bastante fácil.

```
>>> def add(a, b):
...     return a + b
...
>>> add(1,3)
4
```

En lugar de imprimir la suma, estamos devolviendo la salida. El valor de la salida devuelta también se puede almacenar en una variable.

```
>>> sum = add(5,10)
>>> print(sum)
15
```

2.3.17 Expresión Lambda

Considere una situación en la que no desea realizar muchos cálculos en una función. En tal situación, escribir una función completa no tiene sentido. Para resolver esto, usamos una expresión lambda o una función lambda.

Entonces, ¿qué es una expresión lambda? Es una función anónima y están restringidas a una sola expresión. La expresión lambda puede tomar n número de argumentos.

La sintaxis para la expresión lambda es:

```
variable = lambda arguments: operation
```

Veamos un ejemplo:

```
>>> sum = lambda a: a + 10
```

Aquí hemos declarado una suma variable que estamos utilizando para llamar a la función lambda. a representa el argumento que se pasa a esa función.

Llamemos a nuestra función:

```
>>> sum(5)  
15
```

2.3.18 Comprensión de listas

Considere una situación en la que desea una lista de cuadrados. Normalmente declararás una lista de cuadrados y luego, en un ciclo for, elevarás los números al cuadrado.

```
>>> squares = []  
>>> for x in range(10):  
...     squares.append(x**2)  
  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Bueno, esto es factible, pero podemos lograrlo en una sola línea con la ayuda de la comprensión de la lista.

Hay dos formas de lograr esto. Veamos:

```
>>> squares = list(map(lambda x: x**2, range(10)))  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

En la primera forma estamos usando el constructor de listas para construir una lista y dentro de esa una función lambda que eleva al cuadrado el número. La segunda forma de lograr el mismo resultado es la siguiente:

```
>>> squares = list(x**2 for x in range(10))
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Prefiero de esta manera porque es más conciso y fácil de entender.

¿Qué pasa cuando tenemos una condición en la que queremos un conjunto de dos números que sean iguales? Bueno, necesitamos escribir dos *for loops* y un *if loop*.

Veamos cómo se verá eso:

```
>>> num_list = []
>>> for i in range(10):
...     for j in range(10):
...         if i == j:
...             num_list.append((i,j))
...
>>> num_list
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9)]
```

Eso implica mucho trabajo. Y en términos de legibilidad es difícil de entender.

Usemos la comprensión de la lista para lograr el mismo resultado.

```
>>> num_list = list((i,j) for i in range(10) for j in range(10) if i == j)
>>> num_list
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9)]
```

¡Ves lo fácil que es obtener el mismo resultado en una sola expresión? Bueno, ese es el poder de la comprensión de listas.

2.4 CONCEPTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS (OOP)

Python es un lenguaje de programación multiparadigma. Significa que Python puede usar diferentes enfoques para resolver un problema. Uno de los paradigmas es la programación procesal o funcional. Estructura el código como una receta: un conjunto de pasos en forma de funciones y bloques de código.

Otro enfoque para resolver el problema es mediante la creación de clases y objetos. Esto se conoce como programación orientada a objetos. Un objeto es una colección de datos (variables) y métodos que actúan sobre esos datos. Y las clases son un plano para cada objeto.

Lo importante a entender en la programación orientada a objetos es que los objetos están en el centro del paradigma: no solo representan los datos, sino también la estructura del programa.

Puede elegir el paradigma que mejor se adapte al problema en cuestión, mezclar diferentes paradigmas en un programa y / o cambiar de un paradigma a otro a medida que su programa evoluciona.

2.4.1 Ventajas de la programación orientada a objetos

Herencia: este es uno de los conceptos más útiles en POO (Programación Orientada a Objetos). Especifica que el objeto secundario tendrá todas las propiedades y el comportamiento del objeto primario. Por lo tanto, la herencia nos permite definir una clase que hereda todos los métodos y propiedades de otra clase.

Polimorfismo: para comprender el polimorfismo, dividamos la palabra en dos partes. La primera parte “poli” significa muchas y “morfismo” significa formar o dar forma. Por lo tanto, el polimorfismo significa que una tarea se puede realizar de muchas maneras diferentes.

Por ejemplo, tienes un animal de clase y todos los animales hablan. Pero ellos hablan diferente. Aquí, el comportamiento de “hablar” es polimórfico y depende del animal. Entonces, el concepto abstracto de “animal” en realidad no “habla”, pero los animales específicos (como perros y gatos) tienen una implementación concreta de la acción “hablar”.

Polimorfismo significa el mismo nombre de función o nombre de método que se utiliza para diferentes tipos.

Encapsulación: en la programación orientada a objetos puede restringir el acceso a métodos y variables; podemos hacer que los métodos y variables sean privados. Esto puede evitar que los datos se modifiquen por accidente y se conoce como encapsulación.

Primero, entenderemos clases, objetos y constructores. Luego, después de eso, analizaremos nuevamente las propiedades anteriores.

2.4.2 Clases

Hay estructuras de datos primitivas disponibles en Python, por ejemplo, números, cadenas y listas. Todos estos pueden usarse para representaciones simples como nombre, lugar, costo, entre otros.

Pero, ¿y si tenemos datos más complejos? Si hay un patrón en la repetición de las propiedades de esos datos, ¿qué podemos hacer?

Supongamos que tenemos 100 animales diferentes. Cada animal tiene un nombre, edad, patas, etc. ¿Qué sucede si queremos agregar otras propiedades a cada animal, o si un animal más se agrega a esa lista? Para gestionar un escenario tan complejo necesitamos clases.

De acuerdo con la documentación oficial de Python:

Las clases proporcionan un medio de agrupar datos y funcionalidad juntos. La creación de una nueva clase crea un nuevo tipo de objeto, lo que permite crear nuevas instancias de ese tipo.

Cada instancia de clase puede tener atributos adjuntos para mantener su estado. Las instancias de clase también pueden tener métodos (definidos por su clase) para modificar su estado.

Sintaxis de clase:

```
class ClassName:  
    <expression-1>  
    .  
    .  
    .  
    <expression-N>
```

Usamos la palabra clave `class` para definir una clase. Definiremos una clase `Car`.

```
class Car:  
    pass
```

2.4.3 Métodos

Los métodos se parecen a las funciones. La única diferencia es que los métodos dependen de un objeto. Una función se puede invocar por nombre, mientras que los métodos se deben invocar utilizando su referencia de clase. Se definen dentro de la clase.

En nuestro ejemplo, creemos dos métodos. Uno es un motor (Engine) y otro es una rueda (Wheel). Estos dos métodos definen las piezas disponibles en nuestro automóvil.

El siguiente programa nos dará una mejor idea de las clases:

```
>>> class Car:  
...     def engine(self):  
...         print("Engine")  
  
>>> Car().engine()  
Engine
```

Aquí llamamos al método engine utilizando la referencia Car () .

Para resumir, la clase proporciona un modelo de lo que debe definirse, pero no proporciona ningún contenido real. La clase Car anterior define el motor (engine), pero no indicará cuál es el motor de un automóvil específico. Está especificado por el objeto.

2.4.4 Objetos

El objeto es una instancia de la clase. Consideremos el ejemplo anterior de un automóvil. Aquí el automóvil es nuestra clase y Toyota es el objeto del automóvil. Podemos crear múltiples copias del objeto. Cada objeto debe definirse utilizando la clase.

La sintaxis para crear un objeto es:

```
toyota = Car()
```

Consideremos nuestro ejemplo de Auto para comprender los objetos un poco mejor:

```
class Car:  
    def engine(self):  
        print("Engine")  
    def wheel(self):  
        print("Wheel")  
toyota = Car()
```

El toyota = Car () anterior es un objeto de clase. Los objetos de clase admiten dos tipos de operaciones: referencias de atributos e instanciación.

La instanciación de clase utiliza la notación de función. La operación de instanciación (“llamar” a un objeto de clase) crea un objeto vacío.

Ahora podemos llamar a diferentes métodos de nuestra clase Car usando el objeto toyota que hemos creado. Llamemos al método motor y al método rueda.

Abra su editor de texto y cree un archivo llamado mycar.py. En ese archivo, copie el siguiente código:

```
class Car:  
    def engine(self):  
        print("Engine")  
    def wheel(self):  
        print("Wheel")  
if __name__ == "__main__":  
    toyota = Car()  
    toyota.engine()  
    toyota.wheel()
```

Guarda el código anterior. Ahora echemos un vistazo más de cerca a al programa.

Aquí estamos creando un objeto toyota con la ayuda de la clase Car. Toyota. engine () es un método del objeto. ¿Qué sucede exactamente cuando se llama a un método del objeto?

En la llamada toyota.engine () no toma ningún argumento, pero si ve la declaración del método, podemos ver que toma un argumento propio.

Puede estar confundido acerca de por qué no arroja un error. Bueno, siempre que usamos un método del objeto, la llamada `toyota.engine()` se convierte en `Car.engine(toyota)`.

Ejecute el programa con el siguiente comando.

```
Python mycar.py
```

Obtendrá el siguiente resultado:

```
Engine  
Wheel
```

2.4.5 Constructo

El método `__init__` es el método constructor en Python. El método del constructor se usa para inicializar los datos.

Vaya al shell de Python e ingrese este ejemplo:

```
>>> class Car():  
...     def __init__(self):  
...         print("Hello I am the constructor method.")  
...
```

Cuando llamemos a nuestra clase obtendremos el siguiente resultado:

```
>>> toyota = Car()  
Hello I am the constructor method.
```

NOTA

Nunca tendrá que llamar al método `init()`; se llama automáticamente cuando crea una instancia de clase.

2.4.6 Atributos de instancia

Todas las clases tienen objetos y todos los objetos tienen atributos. Los atributos son las propiedades. Utilizamos el método `__init__()` para especificar el atributo inicial de un objeto.

Consideremos nuestro ejemplo de automóvil:

```
class Car():
    def __init__(self, model):
        self.model = model # Atributo de instancia
```

En nuestro ejemplo, cada `Car()` tiene un modelo específico. Por lo tanto, los atributos de instancia son datos únicos para cada instancia.

2.4.7 Atributos de clase

Vimos que los atributos de instancia son específicos para cada objeto, pero los atributos de clase son los mismos para todas las instancias. Veamos el ejemplo del automóvil con la ayuda de los atributos de clase.

```
class Car():
    no_of_wheels = 4 # Atributos de clase
```

2.4.8 Self

Ahora comprendamos qué significa `self` y cómo lo usamos en la programación orientada a objetos. `Self` representa la instancia de una clase. Al usar la palabra clave `self` podemos acceder a los datos inicializados en el constructor y los métodos de una clase.

Veamos un ejemplo de cómo se puede usar `self`. Creemos un método llamado `marca` bajo nuestra clase de automóviles.

Dentro de ese método `__init__`, pasaremos un modelo pasando el nombre del modelo de nuestro automóvil cuando creamos una instancia de nuestro objeto. Se puede acceder a este nombre en cualquier lugar de la clase, por ejemplo `self.model` en nuestro caso.

Vaya al archivo llamado mycar.py y reemplace el código antiguo con este código:

```
.....  
class Car():  
    def __init__(self, model):  
        self.model = model  
  
    def brand(self):  
        print("The brand is", self.model)  
  
if __name__ == "__main__":  
    car = Car("Bmw")  
    car.brand()  
.....
```

Ahora cuando ejecutamos nuestro programa anterior usando el siguiente comando:

```
.....  
Python mycar.py  
.....
```

Obtendremos el siguiente resultado:

```
.....  
The brand is Bmw  
.....
```

ⓘ NOTA

Self es una convención y no una palabra clave de Python real. Self es un argumento en un método y podemos usar otro nombre en su lugar. Pero se recomienda usar self porque aumenta la legibilidad de su código.

2.4.9 Herencia

La herencia se refiere a cuando una clase hereda la propiedad de otra clase.

La clase de la que se heredan las propiedades se llama clase base. La clase que hereda la propiedad de otra clase se llama clase derivada.

La herencia se puede definir como una relación padre e hijo. El hijo hereda las propiedades del padre. Por lo tanto, hacer que el niño sea una clase derivada

mientras que el padre es una clase base. Aquí el término propiedad se refiere a atributos y métodos.

La sintaxis para una definición de clase derivada se ve así:

```
.....  
class DerivedClassName(BaseClassName):  
    <statement-1>  
    .  
    .  
    .  
    <statement-N>  
.....
```

Es importante tener en cuenta que las clases secundarias anulan o amplían los atributos y comportamientos de los métodos de la clase principal. Esto quiere decir que las clases secundarias heredan todos los atributos y comportamientos de sus padres, pero también pueden especificar diferentes comportamientos a seguir.

El tipo de clase más básico es un objeto, que generalmente todas las demás clases heredan como parente. Modifiquemos nuestro ejemplo anterior para comprender cómo funciona la herencia.

Crearemos una clase base llamada vehículo:

```
.....  
class Vehicle:  
    def __init__(self, name):  
        self.name = name  
  
    def getName(self):  
        return self.name  
.....
```

Hemos creado una clase Vehículo e instanciamos un constructor con self. name que estamos usando en el método getName. Siempre que se llame a este método, devolverá el nombre que se ha pasado cuando se crea una instancia de un objeto para esa clase.

Ahora creemos una clase Car hijo.

```
.....  
class Vehicle:  
    def __init__(self, name):  
        self.name = name
```

```
def getName(self):
    return self.name

class Car(Vehicle):
    pass
```

Car es una clase hijo de Vehicle. Hereda todos los métodos y atributos de la clase padre.

Ahora usemos métodos y atributos de la clase Vehicle en nuestra clase secundaria Car.

```
class Vehicle:

    def __init__(self, name, color='silver'):
        self.name = name
        self.color = color

    def get_name(self):
        return self.name

    def get_color(self):
        return self.color

class Car(Vehicle):

    pass
audi = Car("Audi r8")
print("The name of our car is", audi.get_name(), "and color is", audi.get_color())
```

Comprendamos lo que hemos hecho aquí.

Hemos declarado una clase llamada Vehículo con un constructor que toma el nombre como argumento, mientras que el color tiene un argumento predeterminado.

Tenemos dos métodos en su interior. get_name devuelve el nombre, mientras que get_color devuelve el color. Hemos instanciado un objeto y hemos pasado el nombre del auto.

Una cosa que notará aquí es que estamos utilizando métodos de clase base en nuestra declaración de clase secundaria.

Ejecute el programa anterior con el siguiente comando:

```
.....  
Python mycar.py  
.....
```

Salida

```
.....  
The name of our car is Audi r8 and color is silver  
.....
```

También podemos anular un método o atributo principal. En el ejemplo anterior, hemos definido que el color de nuestro vehículo es plateado. Pero, ¿y si el color de nuestro auto es negro?

Ahora, para cada clase secundaria, no podemos hacer cambios en la clase primaria. Llega la funcionalidad primordial.

```
.....  
class Vehicle:  
    def __init__(self, name, color='silver'):  
        self.name = name  
        self.color = color  
    def get_name(self):  
        return self.name  
    def get_color(self):  
        return self.color  
class Car(Vehicle):  
    def get_color(self):  
        self.color = 'black'  
        return self.color  
audi = Car("Audi r8")  
print("The name of our car is", audi.get_name(), "and color is", audi.get_color())  
.....
```

Como puede ver en el programa anterior, no se ha instanciado un constructor. La razón detrás de esto es que nuestra clase secundaria Car solo usa atributos de la clase Vehicle y ya los está heredando. Entonces, en tal escenario, no hay necesidad de volver a instanciar estos atributos.

Ahora, cuando ejecutamos el programa anterior, obtendremos el siguiente resultado:

```
.....  
The name of our car is Audi r8 and color is black  
.....
```

2.4.10 Super

Super() devuelve un objeto temporal de la superclase que luego nos permite llamar a los métodos de esa superclase.

Llamar a los métodos creados anteriormente con super() nos ahorra la necesidad de reescribir esos métodos en nuestra subclase, y nos permite intercambiar superclases con cambios mínimos de código. Por lo tanto, super() extiende la funcionalidad del método heredado.

Extendamos nuestro ejemplo de auto usando super(). Instanciaremos un constructor con brand_name y color en la clase padre, Vehicle. Ahora llamaremos a este constructor desde nuestra clase secundaria (Car) usando super. Crearemos un método get_description que devuelve self.model de la clase Car y self.brand_name, self.color de la clase Vehicle.

```
class Vehicle:

    def __init__(self, brand_name, color):
        self.brand_name = brand_name
        self.color = color

    def get_brand_name(self):
        return self.brand_name

class Car(Vehicle):

    def __init__(self, brand_name, model, color):
        super().__init__(brand_name, color)
        self.model = model

    def get_description(self):
        return "Car Name: " + self.get_brand_name() + self.model + " Color:" +
               self.color
c = Car("Audi ", "r8", " Red")
print("Car description:", c.get_description())
print("Brand name:", c.get_brand_name())
```

Cuando ejecutamos el programa anterior, obtenemos el siguiente resultado:

```
Car description: Car Name: Audi r8 Color: Red
Brand name: Audi
```

2.4.11 Herencia Múltiple

Cuando una clase hereda el método y los atributos de la clase principal múltiple, se llama herencia múltiple. Esto nos permite usar la propiedad de múltiples clases base o clases primarias en una clase derivada o secundaria.

La sintaxis general de herencia múltiple es la siguiente:

```
.....  
class DerivedClassName(Base1, Base2, Base3):  
    <statement-1>  
    .  
    .  
    .  
    <statement-N>  
.....
```

Extendamos nuestro ejemplo de vehículo usando la propiedad de herencia múltiple. Aquí en este ejemplo, crearemos 3 clases, es decir, vehículo, costo y automóvil.

Las clases Vehículo y Costo serán la clase de Padres. Una clase de vehículo representa la propiedad general, mientras que la clase de costo representa su precio.

Como Car tiene una propiedad general y el costo tendrá dos clases para padres. Por lo tanto, heredaremos varias clases primarias.

```
.....  
class Vehicle:  
    def __init__(self, brand_name):  
        self.brand_name = brand_name  
  
    def get_brand_name(self):  
        return self.brand_name  
  
class Cost:  
    def __init__(self, cost):  
        self.cost = cost  
  
    def get_cost(self):  
        return self.cost  
  
class Car(Vehicle, Cost):  
  
    def __init__(self, brand_name, model, cost):  
        self.model = model  
        Vehicle.__init__(self, brand_name)  
.....
```

```
Cost.__init__(self, cost)

def get_description(self):
    return self.get_brand_name() + self.model + " is the car " + "and it's
cost is " + self.get_cost()

c = Car("Audi ", "r8", "2 cr")
print("Car description:", c.get_description())
```

Aquí encontrará una cosa en el programa anterior que es diferente de todos los otros programas presentados hasta ahora. Se ha usado `Vehicle .__ init __(self, brand_name)` en el constructor de la clase `Car`. Esta es una forma de llamar a los atributos de la clase padre. Otro fue `super`, que se ha explicado anteriormente.

Cuando ejecutamos el programa anterior, obtendremos el siguiente resultado:

```
Car description: Audi r8 is the car and it's cost is 2 cr
```

Aunque se puede usar de manera efectiva, la herencia múltiple se debe hacer con cuidado para que nuestros programas no se vuelvan ambiguos y difíciles de entender para otros programadores.

2.4.12 Polimorfismo

La palabra polimorfismo significa tener muchas formas. En programación, el polimorfismo significa el mismo nombre de función (pero diferentes firmas) que se utiliza para diferentes tipos.

Extendamos nuestro programa de autos usando polimorfismo. Crearemos dos clases, `Car` y `Bike`. Ambas clases tienen un método o función común, pero están imprimiendo datos diferentes. El programa se explica por sí mismo:

```
class Car:
    def company(self):
        print("Car belongs to Audi company.")

    def model(self):
        print("The Model is R8.")

    def color(self):
        print("The color is silver.")
```

```
class Bike:

    def company(self):
        print("Bike belongs to pulsar company.")

    def model(self):
        print("The Model is dominar.")

    def color(self):
        print("The color is black.")

def func(obj):
    obj.company()
    obj.model()
    obj.color()

car = Car()
bike = Bike()

func(car)
func(bike)
```

Cuando ejecutamos el código anterior, obtendremos el siguiente resultado:

```
Car belongs to Audi company.
The Model is R8.
The color is silver.
Bike belongs to pulsar company.
The Model is dominar.
The color is black.
```

2.4.13 Encapsulación

En la mayoría de la programación orientada a objetos, podemos restringir el acceso a métodos y variables. Esto puede evitar que los datos se modifiquen por accidente y se conoce como encapsulación.

Usemos la encapsulación en nuestro ejemplo de automóvil. Ahora imagine que tenemos un motor súper secreto. En el primer ejemplo, ocultaremos nuestro motor usando una variable privada. En el segundo ejemplo, ocultaremos nuestro motor usando un método privado.

Ejemplo 1:

```
class Car:

    def __init__(self):
        self.brand_name = 'Audi'
        self.model = 'r8'
        self.__engine = '5.2 L V10'

    def get_description(self):
        return self.brand_name + self.model + " is the car"

c = Car()
print(c.get_description)
print(c.__engine)
```

En este ejemplo, el motor `self.__` es un atributo privado. Cuando ejecutamos este programa obtendremos el siguiente resultado.

```
Audi r8 is the car
AttributeError: 'Car' object has no attribute '__engine'
```

Recibimos un error de que el objeto `Car` no tiene `_engine` porque es un objeto privado.

Ejemplo 2:

También podemos definir un método privado agregando `__` delante del nombre del método. El siguiente es el ejemplo de cómo podemos definir un método privado.

```
class Car:

    def __init__(self):
        self.brand_name = 'Audi'
        self.model = 'r8'

    def __engine(self):
        return '5.2 L V10'

    def get_description(self):
```

```

        return self.brand_name + self.model + " is the car"

c = Car()
print(c.get_description())
print(c.__engine())

```

En este ejemplo, def __engine (self) es un método privado. Cuando ejecutamos este programa obtendremos el siguiente resultado.

```

Audi r8 is the car
AttributeError: 'Car' object has no attribute '__engine'

```

Ahora supongamos que queremos acceder al atributo o método privado, podemos hacerlo de la siguiente manera:

```

class Car:

    def __init__(self):
        self.brand_name = 'Audi '
        self.model = 'r8'
        self.__engine_name = '5.2 L V10'

    def __engine(self):
        return '5.2 L V10'

    def get_description(self):
        return self.brand_name + self.model + " is the car"

c = Car()
print(c.get_description())
print("Accessing Private Method: ", c.__Car__engine())
print("Accessing Private variable: ", c.__Car__engine_name)

```

La salida del programa es:

```

Audi r8 is the car
Accessing Private Method:  5.2 L V10
Accessing Private variable:  5.2 L V10

```

La encapsulación le brinda más control sobre el grado de acoplamiento en su código. Permite que una clase cambie su implementación sin afectar otras partes del código.

2.4.14 Decorador

Imagine que tiene que extender la funcionalidad de múltiples funciones. ¿Cómo lo hará?

Bueno, una forma es que puedes hacer llamadas funcionales y en esa función, puedes manejarlo. Hacer cambios en 30 a 40 llamadas de función y recordar dónde realizar la llamada es una tarea complicada. Pero la forma más elegante que proporciona Python es con decoradores.

¿Qué es un decorador? Un decorador es una función que toma una función y extiende su funcionalidad sin modificarla explícitamente.

Probemos un ejemplo para entender al decorador. Hay dos formas de escribir un decorador.

Método 1

Declaramos una función decoradora y en los argumentos de la función esperamos que la función se pase como argumento. Dentro de eso, escribimos una función de contenedor donde las operaciones se llevan a cabo y se devuelve.

```
>>> def my_decorator(func):
...     def wrapper():
...         print("Line Number 1")
...         func()
...         print("Line Number 3")
...     return wrapper
...
>>> def say_hello():
...     print("Hello I am line Number 2")
...
```

Para llamar a la función asignamos el decorador con `say_hello` como argumento.

```
>>> say_hello = my_decorator(say_hello)
```

También podemos verificar la referencia usando `say_hello`. Obtendremos la salida que nos dice que ha sido envuelta por la función `my_decorator`.

```
<function my_decorator.<locals>.wrapper at 0x10dc84598>
```

Llamemos a nuestra función `say_hello`:

```
>>> say_hello()
Line Number 1
Hello I am line Number 2
Line Number 3
```

Vea la magia que la línea “Hello, I am line Number 2” se imprime entre la línea número 1 y 3 porque la llamada a la función se ejecuta allí.

El método 1 es torpe, y por eso muchas personas prefieren un enfoque diferente.

Método 2

Aquí nuestra declaración de decorador permanece igual, pero cambiamos cómo se asigna la llamada a ese decorador. Cualquier función que requiera que el decorador se envuelva con `@decorator_name`.

```
>>> def my_decorator(func):
...     def wrapper():
...         print("Line Number 1")
...         func()
...         print("Line Number 3")
...     return wrapper
...
>>> @my_decorator
... def say_hello():
...     print("Hello I am line Number 2")
...
>>> say_hello()
```

La salida es la misma:

```
Line Number 1  
Hello I am line Number 2  
Line Number 3
```

Un decorador es una herramienta poderosa y se usa en los siguientes escenarios de desarrollo de una aplicación:

- ▶ Configurar el registrador
- ▶ Configuración de la instalación
- ▶ Error al configurar la captura
- ▶ Extender la funcionalidad común para todas las funciones y clases.

2.4.15 Excepciones

Cuando estábamos aprendiendo varias sintaxis, encontramos varios errores. Esos errores ocurrieron debido a la sintaxis. Pero en una aplicación del mundo real, los errores no solo ocurren debido a problemas de sintaxis, sino también debido a errores de red u otra causa.

Para manejar estos problemas usamos Try - Except. En el bloque try, escribimos la expresión que queremos que se ejecute, mientras que en el bloque except captamos el error. El bloque Try-Except tiene el siguiente aspecto:

```
try:  
    expression  
except:  
    catch error
```

Comprendamos esto con un ejemplo:

```
>>> try:  
...     print(value)  
... except:  
...     print("Something went wrong")  
...
```

Aquí estamos tratando de imprimir la variable de valor pero no está definida. Entonces obtenemos el siguiente resultado:

```
.....  
Something went wrong  
.....
```

Puede estar pensando que la línea “algo salió mal” no es tan útil. Entonces, ¿cómo podemos saber qué salió mal aquí?

Podemos imprimir la excepción y usarla para averiguar qué salió mal. Probemos esto en nuestro ejemplo:

```
.....  
>>> try:  
...     print(value)  
... except Exception as e:  
...     print(e)  
...  
.....
```

Y el resultado es:

```
.....  
name 'value' is not defined  
.....
```

Whoa! Eso es magia. Me está notificando que el “valor” no está definido.

Python también proporciona una herramienta llamada *raise*. Suponga que no desea que se produzca una determinada condición y, si se produce, desea aumentarla. En tal condición, puede usar raise. Considere el siguiente ejemplo:

```
.....  
>>> i = 5  
>>> if i < 6:  
...     raise Exception("Number below 6 are not allowed")  
...  
.....
```

La salida que obtenemos es la siguiente:

```
.....  
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
Exception: Number below 6 are not allowed  
.....
```

Hay muchos subtipos de excepciones, por lo que se recomienda revisar la documentación de Python para comprenderlas.

2.5 IMPORTACIÓN DE PAQUETES

Has aprendido los conceptos básicos de Python y ahora estás listo para crear aplicaciones increíbles. Pero espere, todavía nos faltan algunos temas importantes.

Sin la importación de paquetes, se verá obligado a escribir todo en un solo archivo. Imagine qué tragedia sería.

Cree dos archivos llamados main.py y hello.py. Recuerde que ambos archivos deben estar en el mismo directorio.

En hello.py copie y pegue el siguiente código:

```
def say_hello():
    print("Hello world")
```

En main.py copie y pegue el siguiente código:

```
import hello

if __name__ == "__main__":
    hello.say_hello()
```

En hello.py hemos declarado una función say_hello () que imprime “Hello world”. En main.py observas una declaración de importación. Estamos importando el módulo hello y llamando a la función say_hello () desde ese módulo.

Ejecute el programa usando el siguiente comando:

```
→ Python main.py
```

Salida

```
Hello world
```

Ahora analicemos cómo importar un módulo que está en otro directorio.

Vamos a crear un directorio llamado “data” y mover nuestro hello.py dentro de ese directorio.

Vaya a main.py y cambie la declaración de importación anterior.

```
from data import hello

if __name__ == "__main__":
    hello.say_hello()
```

Hay dos formas de importar desde un directorio.

- ▶ Método 1: from data import hello
- ▶ Método 2: import data.hello

Prefiero el método 1 debido a su legibilidad. Puedes elegir el método que te parezca mejor.

Ejecutemos nuestra aplicación usando el siguiente comando:

```
→ Python main.py
```

Y se produce un error. Espera, ¿por qué sucedió esto? Hicimos todo bien. Veamos el error:

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    from data import hello
ImportError: No module named data
```

Bueno, Python nos dice que no reconoce un módulo llamado data. Para resolver este problema, cree un `__init__.py` dentro del directorio de data. Deje el archivo en blanco y ejecute el programa nuevamente y obtendrá el siguiente resultado:

```
Hello world
```

Bueno, Python por defecto no trata un directorio como un módulo. Para informar a Python que trate un directorio como un módulo, se requiere `__init__.py`.

Manejo de JSON

Si ha trabajado anteriormente con desarrollo web o desarrollo de aplicaciones, puede saber que todas las llamadas a la API se realizan en formato JSON. Si bien JSON se parece a un diccionario en Python, recuerde que es muy diferente.

Para manejar JSON, Python proporciona un paquete json incorporado. Para usar este paquete necesitamos importarlo de la siguiente manera:

```
.....  
Import json  
.....
```

Esta librería proporciona dos métodos que nos ayudan a manejar el JSON. Vamos a entenderlos uno por uno.

JSON loads

Si tiene una cadena JSON y desea volver a convertirla en un diccionario, debe usar el método loads. Vaya al shell de Python y copie y pegue el siguiente código:

```
.....  
>>> import json  
>>> json_string = '{ "user_name": "Sharvin", "age":1000}'  
>>> type(json_string)  
<class 'str'>  
>>> data = json.loads(json_string)  
>>> type(data)  
<class 'dict'>  
>>> data  
{'user_name': 'Sharvin', 'age': 1000}  
.....
```

JSON dumps

Ahora volvamos nuestros datos al formato de cadena JSON usando el método dumps.

```
>>> jsonString = json.dumps(data)
>>> type(jsonString)
<class 'str'>
>>> jsonString
'{"user_name": "Sharvin", "age": 1000}'
```

2.6 GUÍA DE ESTILO PARA LA ESCRITURA DEL CÓDIGO PYTHON

Python tiene una excelente guía de estilo llamada PEP8 (<https://www.python.org/dev/peps/pep-0008/>). Cubre la mayoría de las situaciones que se puedan presentar al escribir Python, su propósito es brindar pautas para que su código sea organizado y legible. Por otro lado, PEP8 puede considerarse una guía genérica de Python en lugar de reglas estrictas, ya que permite diferentes enfoques para lograr objetivos similares.

El objetivo final de la guía es tener un código limpio, coherente y eficiente. Recuerde: el código se lee con más frecuencia de lo que se escribe y solo incidentalmente para que las máquinas lo ejecuten. A continuación se presentan los aspectos claves de la guía PEP8:

2.6.1 Sangria (Indentation)

Es mejor dejar siempre cuatro espacios para la sangría.

```
if True:
    print("If works")
```

Cuando escriba una expresión grande, es mejor mantener la expresión alineada verticalmente. Cuando haga esto, creará una “sangría francesa”.

```
value = square_of_numbers(num1, num2, num3, num4)

list_of_people = [
    "Rama",
    "John",
    "Shiva"
]
```

Python 3 no permite mezclar tabulaciones y espacios para la sangría. Es por eso que debes elegir uno de los dos y seguir con él.

2.6.2 Longitud máxima de la línea

Límite todas las líneas a un máximo de 79 caracteres.

```
.....with open('/path/to/some/file/you/want/to/read') as file_1, \
open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
.....
```

Para bloques de texto largos (cadenas de texto o comentarios), la longitud de la línea debe limitarse a 72 caracteres.

Mientras usa cualquier operador (+, -, etc.), se recomienda aplicar un salto de línea, esto hace que su código sea más fácil de entender:

```
.....total = (A +
             B +
             C)
.....
```

Separar las cadenas de texto a través de múltiples líneas. La sugerencia es usar barras \:

```
.....my_string = 'This is a very long string, \
'so long that it will not fit into just one line ' \
'so it must be split across multiple lines.'
.....
```

2.6.3 Comillas simples o dobles

Python le permite usar comillas simples o dobles. PEP8 dice que puede usar cualquiera, siempre que sea consistente.

Intente ceñirse al uso de comillas simples, excepto en los casos en que el uso de comillas dobles sea más legible.

```
print('This doesn\'t look so nice.')
print("Doesn't this look nicer?")
```

2.6.4 Líneas en blanco

Las funciones y clases de nivel superior están separadas por dos líneas en blanco.

Las definiciones de métodos dentro de las clases deben estar separadas por una línea en blanco.

Se pueden usar líneas en blanco adicionales (con moderación) para separar grupos de funciones relacionadas

```
class SwapTestSuite(unittest.TestCase):
    """
    Swap Operation Test Case
    """

    def setUp(self):
        self.a = 1
        self.b = 2

    def test_swap_operations(self):
        instance = Swap(self.a, self.b)
        value1, value2 = instance.get_swap_values()
        self.assertEqual(self.a, value2)
        self.assertEqual(self.b, value1)

class OddOrEvenTestSuite(unittest.TestCase):
    """
```

```
This is the Odd or Even Test case Suite

"""

def test_odd_even_operations(self):

    instance1 = OddOrEven(self.value1)

    instance2 = OddOrEven(self.value2)

    message1 = instance1.get_odd_or_even()

    message2 = instance2.get_odd_or_even()

    self.assertEqual(message1, 'Odd')

    self.assertEqual(message2, 'Even')
```

2.6.5 Declaraciones de importación

Siempre debe importar librerías al comienzo de su script.

Si necesita importar muchos nombres de un módulo o paquete, y no caben todos en una línea (sin hacer la línea demasiado larga), utilice paréntesis para distribuir los nombres en varias líneas.

```
from Tkinter import (

    Tk, Frame, Button, Entry, Canvas, Text,

    LEFT, DISABLED, NORMAL, RIDGE, END,

)
```

2.6.6 Comentarios

Utilice comentarios de bloque para explicar el código que es más complejo o desconocido para los demás. Por lo general, estos son comentarios de formato más largo y se aplican a parte o a todo el código que sigue. Los comentarios de bloque se sangran al mismo nivel que el código. Cada línea de un comentario de bloque comienza con el hashtag # y un solo espacio. Si necesita usar más de un párrafo, deben estar separados por una línea que contenga un solo #.

```
if Gram is None or Gram is False:  
  
    Gram = None  
  
    if copy_X:  
  
        # force copy, setting the array to be fortran-ordered  
  
        # speeds up the calculation of the (partial) Gram matrix  
  
        # and allows to easily swap columns  
  
        X = X.copy('F')
```

Debe usar los comentarios en línea con moderación, aunque pueden ser efectivos cuando necesita explicar algunas partes de su código.

```
counter = 0 # Inicializar el contador
```

Escriba cadenas de documentación o cadenas de comentarios al comienzo de los módulos, archivos, clases y métodos públicos. Este tipo de comentarios comienzan con “” “y terminan con” “”

```
"""  
Este módulo está destinado a proporcionar funciones para computación  
científica
```

2.6.7 Nombres de dunder a nivel de módulo

Los métodos dunder o mágicos en Python son los métodos que tienen dos subrayados de prefijo y sufijo en el nombre del método. Dunder aquí significa “Double Under (Guiones bajos)”. Estos se utilizan comúnmente para la sobrecarga del operador. Algunos ejemplos de métodos mágicos son: `_init_`, `_add_`, `_len_`, `_repr_` etc.

Un dunder de nivel de módulo como (`__all__`, `__author__`, `__version__`) debe colocarse en la cadena de documentos principal del módulo y debe estar antes de todas las declaraciones de importación. Debe definir las importaciones de `__future__` antes de cualquier otro código, excepto las cadenas de texto.

“””

El módulo Algos consta de todos los algoritmos básicos y su implementación

“””

```
from __future__ import print_
all__ = ['searching', 'sorting']__
version__ = '0.0.1'__
author__ = 'Chitrang Dixit'
import os
import sys
```

2.6.8 Convenciones de nombres

- ▶ No utilice “l”, “O” o “I” como nombre de variable única: estos caracteres se parecen a cero (0) y (1) en algunas fuentes.
- ▶ Generalmente, es bueno usar nombres cortos si es posible. En algunos casos, puede utilizar guiones bajos para mejorar la legibilidad.
- ▶ “Interno” significa interno a un módulo o protegido o privado dentro de una clase.
- ▶ Anteponer un guion bajo (`_`) tiene cierto soporte para proteger las variables y funciones del módulo (no incluido con la importación `*` desde).
- ▶ Anteponer un guion bajo doble (`__`) a una variable de instancia o método sirve efectivamente para hacer que la variable o método sea privado para su clase (usando la modificación de nombres).

2.6.9 Espacios en blanco en expresiones / declaraciones

Evite los espacios en blanco inmediatamente entre paréntesis, corchetes o llaves

```
spam(ham[1], {eggs: 2}) # Válido  
spam( ham[ 1 ], { eggs: 2 } ) # Evitar esto
```

Entre una coma al final y un paréntesis cercano siguiente

```
foo = (0,) # Válido  
bar = (0, ) # Evitar esto
```

Inmediatamente antes de una coma, punto y coma o dos puntos

```
if x == 4: print x, y; x, y = y, x # Válido  
if x == 4 : print x , y ; x , y = y , x # Evitar esto
```

Inmediatamente antes del paréntesis abierto que inicia la lista de argumentos de una llamada a función

```
Yes: dct['key'] = lst[index] # Válido  
No: dct ['key'] = lst [index] # Evitar esto
```

Más de un espacio alrededor de un operador de asignación (u otro) para alinearlo con otro

Válido:

```
x = 1  
y = 2  
long_variable = 3
```

Evite esto:

```
x = 1  
y = 2  
long_variable = 3
```

3

LOS 10 MEJORES PAQUETES DE PYTHON PARA FINANZAS Y MODELADO FINANCIERO

La popularidad del lenguaje de programación Python se debe, al menos en parte, a la versatilidad que ofrece. Además de la gran cantidad de casos de uso en el desarrollo web y de aplicaciones, Python proporciona las herramientas para construir e implementar cualquier tipo de modelo científico o matemático, independientemente del origen o tipo de datos. Esta versatilidad está habilitada por la extensa librería estándar que ofrece una gama de facilidades destinadas a mejorar la funcionalidad y portabilidad del lenguaje. Para aplicaciones más específicas, Python Package Index (PyPI) proporciona paquetes adicionales que amplían las capacidades de Python para adaptarse a las necesidades de cada dominio.

Por estas razones, Python ha demostrado ser una herramienta formidable en el desarrollo de nuevas tecnologías financieras. Desde la preparación de datos sin procesar hasta la creación de interfaces gráficas de usuario (GUI) estéticamente agradables e intuitivas, existe una gran variedad de paquetes para ayudar a los usuarios a construir sus propios modelos financieros.

El campo de las tecnologías financieras es amplio y abarca todo, desde seguros, préstamos y comercio hasta banca electrónica y otros servicios de pago. Este capítulo se centra en aplicaciones específicas de las finanzas cuantitativas, que requieren tareas de programación como la importación y transformación de datos, series de tiempo y análisis de riesgos, negociación y backtesting, integración de Excel y visualización de datos.

A continuación, se destacan 10 paquetes importantes de finanzas y modelos financieros con algunos ejemplos básicos.

3.1 NUMPY

NumPy es el paquete más esencial para la computación científica y matemática en Python. No solo introduce matrices y matrices n-dimensionales en Python, sino que también contiene algunas funciones matemáticas básicas para manipular estas estructuras de datos. La mayoría de los paquetes de Python de nivel superior para finanzas que se mencionan más adelante en esta lista dependen de NumPy.

Por ejemplo, para crear dos matrices complejas de 2×2 e imprimir la suma:

```
import numpy as np
a = np.array([[1+2j, 2+1j], [3, 4]])
b = np.array([[5, 6+6j], [7, 8+4j]])
print(a+b)
```

Salida

```
[[ 6.+2.j  8.+7.j]
 [10.+0.j 12.+4.j]]
```

Y para tomar el complejo conjugado de una de ellas:

```
np.conj(a)

array([[1.-2.j, 2.-1.j],
 [3.-0.j, 4.-0.j]])
```

3.2 SCIPY

El paquete NumPy proporciona estructuras matemáticas básicas para manipular y almacenar datos. Pero para construir modelos sofisticados basados en estos datos, se necesita un repositorio de herramientas y operaciones estadísticas más avanzadas. Entra en escena SciPy. Este paquete proporciona funciones y algoritmos críticos para los cálculos científicos avanzados necesarios para construir cualquier modelo estadístico. Estos incluyen algoritmos para la interpolación, optimización, agrupación, transformación e integración de datos. Estas operaciones son fundamentales a la hora de realizar cualquier tipo de análisis de datos, o construir cualquier tipo de modelo predictivo.

Para demostrar la interpolación, primero uso NumPy para crear algunos puntos de datos con una función arbitraria, luego comparo diferentes métodos de interpolación:

```
import numpy as np
from scipy.interpolate import interp1d
import pylab

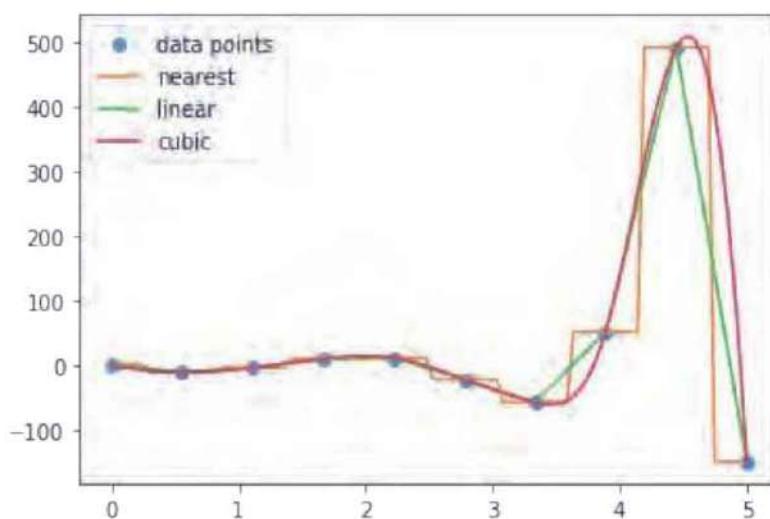
x = np.linspace(0, 5, 10)
y = np.exp(x) / np.cos(np.pi * x)

f_nearest = interp1d(x, y, kind='nearest')
f_linear = interp1d(x, y)
f_cubic = interp1d(x, y, kind='cubic')

x2 = np.linspace(0, 5, 100)

pylab.plot(x, y, 'o', label='data points')
pylab.plot(x2, f_nearest(x2), label='nearest')
pylab.plot(x2, f_linear(x2), label='linear')
pylab.plot(x2, f_cubic(x2), label='cubic')
pylab.legend()
pylab.show()
```

Salida



3.3 PANDAS

NumPy y SciPy sientan las bases matemáticas. El paquete Pandas, por otro lado, establece una estructura de datos intuitiva y fácil de usar, un DataFrame, diseñado específicamente para el análisis y la construcción de modelos. Se basa en las matrices que presenta NumPy y está optimizado para datos tabulares, multidimensionales y heterogéneos. Las manipulaciones más comunes, como agrupar, unir, fusionar o completar, reemplazar e imputar valores nulos, se pueden ejecutar en una sola línea. Además, el paquete proporciona funciones para importar datos de una variedad de formatos estándar y otros para graficar rápidamente, recuperar estadísticas básicas o generar datos.

Para crear un DataFrame:

```
import pandas as pd

df_1 = pd.DataFrame({'col1': [1,2], 'col2': [3,4]})
```

Y para concatenar dos DataFrame de datos juntos:

```
df_2 = pd.DataFrame({'col3': [5,6], 'col4': [7,8]})

df = pd.concat([df_1,df_2], axis = 1)

df
```

Salida

	col1	col2	col3	col4
0	1	3	5	7
1	2	4	6	8

Para realizar una operación de filtrado simple, extrayendo la fila que cumple la condición lógica:

```
df[df.col3 == 5]
```

	col1	col2	col3	col4
0	1	3	5	7

3.4 STATSMODELS

SciPy proporciona una librería de herramientas estadísticas que permiten a los usuarios construir un modelo, y pandas facilita su implementación. El paquete Statsmodels se basa en estos paquetes implementando pruebas más avanzadas de diferentes modelos estadísticos. Una lista extensa de diagnósticos y estadísticas de resultados para cada estimador está disponible para cualquier modelo dado, con el objetivo de brindar al usuario una imagen completa del rendimiento del modelo. Los resultados se comparan con paquetes estadísticos existentes para garantizar que sean correctos.

Como ejemplo, a continuación se importa un conjunto de datos integrado:

```
import numpy as np
import statsmodels.api as sm

rand_data = sm.datasets.randhie.load(as_pandas=False)
rand_exog = rand_data.exog.view(float).reshape(len(rand_data.exog), -1)
rand_exog = sm.add_constant(rand_exog, prepend=False)

poisson_mod = sm.Poisson(rand_data.endog, rand_exog)
poisson_res = poisson_mod.fit(method="newton")
print(poisson_res.summary())
```

Salida

```
Optimization terminated successfully.
    Current function value: 3.091609
    Iterations 6
                    Poisson Regression Results
=====
Dep. Variable:                      y   No. Observations:      20190
Model:                          Poisson   Df Residuals:          20180
Method:                         MLE   Df Model:                  9
Date:              Tue, 13 Oct 2020   Pseudo R-squ.:     0.06343
Time:                17:44:56   Log-Likelihood:   -62420.
converged:                     True   LL-Null:        -66647.
Covariance Type:            nonrobust   LLR p-value:     0.000
=====
                           coef      std err          z      P>|z|      [0.025      0.975]
=====
```

x1	-0.0525	0.003	-18.216	0.000	-0.058	-0.047
x2	-0.2471	0.011	-23.272	0.000	-0.268	-0.226
x3	0.0353	0.002	19.302	0.000	0.032	0.039
x4	-0.0346	0.002	-21.439	0.000	-0.038	-0.031
x5	0.2717	0.012	22.200	0.000	0.248	0.296
x6	0.0339	0.001	60.098	0.000	0.033	0.035
x7	-0.0126	0.009	-1.366	0.172	-0.031	0.005
x8	0.0541	0.015	3.531	0.000	0.024	0.084
x9	0.2061	0.026	7.843	0.000	0.155	0.258
const	0.7004	0.011	62.741	0.000	0.678	0.722

3.5 QUANDL

Hasta ahora, los paquetes que se han mencionado son imparciales con respecto al tipo de datos que se están considerando. Por supuesto, al considerar los modelos financieros, necesitamos datos financieros. Aquí es donde Quandl viene al rescate. El módulo Quandl Python brinda a los usuarios acceso a la vasta colección de datos económicos, financieros y de mercado recopilados de bancos centrales, gobiernos, organizaciones multinacionales y muchas otras fuentes. La mayoría de los conjuntos de datos sin procesar son de acceso gratuito al registrarse (necesita una clave API), para conjuntos de datos más avanzados y detallados se debe asumir un costo.

Obtener datos de Quandl en Python es bastante simple. Supongamos que estamos interesados en ABN Amro Group de la Bolsa de Valores de Euronext. El símbolo de cotización en Quandl es EURONEXT / ABN. En una celda de cuaderno Jupyter, ejecute el siguiente código:

```
import quandl

# Reemplace por su propio API key de Quandl

QUANDL_API_KEY = 'API key'
quandl.ApiConfig.api_key = QUANDL_API_KEY
df = quandl.get('EURONEXT/ABN')

df.head()
```

Salida

Date	Open	High	Low	Last	Volume	Turnover
2015-11-20	18.18	18.43	18.000	18.35	38392898.0	7.003281e+08
2015-11-23	18.45	18.70	18.215	18.61	3352514.0	6.186446e+07
2015-11-24	18.70	18.80	18.370	18.80	4871901.0	8.994087e+07
2015-11-25	18.85	19.50	18.770	19.45	4802607.0	9.153862e+07
2015-11-26	19.48	19.67	19.410	19.43	1648481.0	3.220713e+07

3.6 Zipline

Zipline es un paquete que une las estadísticas, las estructuras de datos y las fuentes de datos. Es una formidable librería de negociación algorítmica para Python, evidentemente porque cuenta con el apoyo de Quantopian, una plataforma gratuita para crear y ejecutar estrategias comerciales. Los datos de Quandl se importan fácilmente y los algoritmos personalizados se diseñan, prueban e implementan fácilmente. Esto incluye backtesting de algoritmos y operaciones en vivo. Un algoritmo básico se ve así:

```
from zipline.api import order, record, symbol

def initialize(context):
    pass

def handle_data(context, data):
    order(symbol('AAPL'), 10)
    record(AAPL=data.current(symbol('AAPL'), 'price'))
```

Importamos las funciones de orden, registro y símbolo de zipline, para construir un algoritmo que registra el precio de las acciones de Apple.

3.7 PYFOLIO

Después de diseñar y probar un algoritmo en zipline, el paquete pyfolio proporciona una manera fácil de generar un informe que contiene estadísticas de rendimiento. Estas estadísticas incluyen rendimientos anuales / mensuales, cuantiles de rendimiento, ratio de Sharpe, rotación de la cartera y algunos más. Para generar el informe procedemos con el siguiente código:

```
import pyfolio as pf

stock_rets = pf.utils.get_symbol_rets('FB')
pf.create_returns_tear_sheet(stock_rets, live_start_date='2015-12-1')
```

El resultado será una serie de tablas y gráficos que contienen las métricas de rendimiento.

Salida

Entire data start date: 2012-05-21
 Entire data end date: 2017-08-22
 In-sample months: 42
 Out-of-sample months: 20

	All	In-sample	Out-of-sample
Annual return	32.8%	32.9%	32.6%
Cumulative returns	343.7%	172.7%	62.7%
Annual volatility	38.0%	43.2%	24.3%
Sharpe ratio	0.93	0.87	1.28
Calmar ratio	0.61	0.61	2.38
Stability	0.92	0.89	0.88
Max drawdown	-53.6%	-53.6%	-13.7%
Omega ratio	1.20	1.18	1.29
Sortino ratio	1.50	1.39	2.09
Skew	1.89	1.73	2.12
Kurtosis	23.62	19.44	24.74
Tail ratio	1.11	1.04	1.02
Daily value at risk	-4.7%	-5.3%	-2.9%
Alpha	0.20	0.21	0.18
Beta	1.04	1.01	1.11

Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	47.90	2012-05-21	2012-09-04	2013-07-25	309
1	22.06	2014-03-10	2014-04-28	2014-07-24	99
2	17.34	2013-10-18	2013-11-25	2013-12-17	43
3	16.57	2015-07-21	2015-08-24	2015-10-19	65
4	13.68	2016-10-24	2016-12-30	2017-02-08	78

3.8 TA-LIB

TA-Lib es una alternativa al uso de zipline y pyfolio. El proyecto está escrito en C++, pero existe un contenedor para Python. Como zipline, TA-Lib proporciona herramientas financieras comunes como estudios de superposición, indicadores de impulso, indicadores de volumen, indicadores de volatilidad, transformaciones de precios, indicadores de ciclo, reconocimiento de patrones y funciones estadísticas puras.

3.9 QUANT-LIB

Otra alternativa a zipline y pyfolio es QuantLib. Similar a TA-Lib, QuantLib se escribe en C++ y luego se exporta a Python. El proyecto QuantLib tiene como objetivo crear una librería gratuita de código abierto para modelar, comercializar y gestionar riesgos. El paquete contiene herramientas para diseñar e implementar algoritmos avanzados que incluyen características tales como convenciones de mercado, modelos de curva de rendimiento, solucionadores, PDE, Monte Carlo y otros.

3.10 MATPLOTLIB

Los paquetes de Python para finanzas antes mencionados establecen fuentes de datos financieros, estructuras de datos óptimas para los datos financieros, así como modelos estadísticos y mecanismos de evaluación. Pero ninguna proporciona una de las herramientas de Python más importantes para el modelado financiero: visualización de datos.

La visualización no solo es importante para comprender las tendencias dentro de los datos financieros, sino también para transmitir conocimientos al personal no técnico. Hay múltiples paquetes de visualización de datos dentro de Python, cada uno con aspectos positivos y negativos, pero el más fácil de implementar para el modelado financiero es matplotlib. Esto se debe principalmente al hecho de que muchos de los paquetes de esta lista ya se basan en matplotlib. Además, la documentación es abundante y la sintaxis simple y directa. En los ejercicios que realizaremos más adelante, utilizaremos profusamente la librería Matplotlib.

4

OBteniendo y PROCESANDO DATOS FINANCIEROS

El capítulo cuarto está dedicado a una parte muy importante (si no la más importante) de cualquier proyecto de ciencia de datos / finanzas cuantitativas: recopilar y trabajar con datos. En concordancia con la máxima de “basura adentro, basura afuera”, debemos esforzarnos por tener datos de la más alta calidad y procesarlos correctamente para su uso posterior con algoritmos. La razón de esto es simple: los resultados de nuestros análisis dependen en gran medida de los datos de entrada, y ningún modelo sofisticado podrá compensar esto.

Aquí se cubrirá todo el proceso de recopilación de datos financieros y su preprocesamiento en la forma que se usa comúnmente en proyectos de la vida real. Se comenzará presentando algunas fuentes de datos de alta calidad, se muestra como transformar los precios en rendimientos y como reescalar dichos rendimientos, por ejemplo de un período diario a mensual o anual. Por último, aprenderemos cómo visualizar los datos gráficamente y como identificamos valores atípicos.

Una cosa a tener en cuenta al leer este capítulo es que los datos difieren entre las fuentes, por lo que los precios que vemos, por ejemplo, en Yahoo Finance y Quandl probablemente diferirán, ya que los sitios respectivos también obtienen sus datos de diferentes fuentes y podrían usar otros métodos para ajustar los precios de las acciones corporativas. La mejor práctica es encontrar una fuente en la que confiemos más con respecto a un tipo particular de datos y luego usarla para descargar datos. En este capítulo, abordaremos específicamente los siguientes temas:

- Obteniendo datos de Yahoo Finance
- Obteniendo datos de Quandl
- Obteniendo datos de Intrinio

- ▶ Transformando precios de activos financieros a rendimientos
- ▶ Cambiando la frecuencia temporal de los rendimientos
- ▶ Visualizando datos de series de tiempo financieras
- ▶ Identificando valores atípicos en una serie temporal
- ▶ Obtención y remuestreo de datos sobre criptomonedas

4.1 OBTENIENDO DATOS DE YAHOO FINANCE

Una de las fuentes más populares de datos financieros gratuitos es Yahoo Finance. Contiene no solo los precios de las acciones históricas y actuales en diferentes frecuencias (diarias, semanales, mensuales), sino también indicadores calculados, como la beta (una medida de la volatilidad de un activo individual en comparación con la volatilidad de todo el mercado) y mucho más. A continuación, nos enfocamos en obtener los precios históricos de las acciones.

Durante un largo período de tiempo, la herramienta de acceso para descargar datos de Yahoo Finance fue la librería pandas-datareader. El objetivo de la librería era extraer datos de una variedad de fuentes y almacenarlos en forma de un DataFrame (estructura de datos de dos dimensiones, similar a una tabla) de pandas. Sin embargo, después de algunos cambios a la API (Las siglas **API** provienen del inglés “Application Programming Interface”, que en español sería “Interfaz de Programación de Aplicaciones”) de Yahoo Finance, esta funcionalidad quedó en desuso.

A partir de ahora, la forma más fácil y rápida de descargar los precios históricos de las acciones es usar la librería `yfinance` (anteriormente conocida como `fix_yahoo_finance`), que se puede usar además de `pandas-datareader` o como una librería independiente para descargar precios de acciones de Yahoo Finance. Nos centraremos en el último caso de uso.

Ejemplo: estamos interesados en obtener los precios de las acciones de Apple en el período comprendido entre los años 2010 y 2020:

Implementación en Python

#1. Importar librerías

```
import pandas as pd  
import yfinance as yf
```

#2. Descargar los datos

```
df_yahoo = yf.download('AAPL',
                      start='2010-01-01',
                      end='2020-11-30',
                      progress=False)
```

#3. Mostrar los resultados

```
df_yahoo
```

Salida

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.539882	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656428	6.551187	601904800
2010-01-06	7.656428	7.686786	7.526786	7.534843	6.446983	552160000
2010-01-07	7.562500	7.571429	7.466072	7.520714	6.435065	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.477847	447610800
...
2020-11-20	118.639999	118.769997	117.290001	117.339996	117.339996	73391400
2020-11-23	117.180000	117.620003	113.750000	113.849998	113.849998	127959300
2020-11-24	113.910004	115.849998	112.589996	115.169998	115.169998	113874200
2020-11-25	115.550003	116.750000	115.169998	116.029999	116.029999	76499200
2020-11-27	116.570000	117.489998	116.220001	116.589996	116.589996	46691300

2746 rows × 6 columns

El resultado de la solicitud es un DataFrame (2.746 filas x 6 columnas) que contiene los precios diarios de apertura, el precio más alto, el precio más bajo, el precio de cierre, así como el precio de cierre ajustado y el volumen.

¿Cómo funciona el algoritmo?

La función de descarga es muy intuitiva; en el caso más básico, solo necesitamos proporcionar el ticker (símbolo) y descargar todos los datos requeridos, en nuestro caso, descargamos datos de un rango específico (enero 2010 a noviembre 2020). También podríamos pasar una lista de múltiples tickers, como ['AAPL', 'MSFT']. *progress = False* deshabilita la barra de progreso.

4.2 OBTENIENDO DATOS DE QUANDL

Quandl es un proveedor de productos de datos alternativos para profesionales de las finanzas y ofrece una forma sencilla de descargar datos, también a través de una librería de Python.

Un buen punto de partida para los datos financieros sería la base de datos de precios WIKI, que contiene precios de acciones, dividendos y splits de 3.000 empresas estadounidenses que cotizan en bolsa. El inconveniente de esta base de datos es que, a partir de abril de 2018, ya no se actualiza (lo que significa que no hay datos recientes). Sin embargo, para obtener datos históricos o aprender a acceder a las bases de datos, es más que suficiente.

Utilizaremos el mismo ejemplo que usamos en la sección anterior: descargamos los precios de las acciones de Apple para los años 2010-2020.

Antes de descargar los datos, necesitamos crear una cuenta en Quandl (<https://www.quandl.com>) y luego podemos encontrar nuestra clave API personal en nuestro perfil (<https://www.quandl.com/cuenta/perfil>). Podemos buscar datos de interés utilizando la función de búsqueda (<https://www.quandl.com/search>).

Ejecute los siguientes pasos para descargar datos de Quandl.

Implementación en Python

#1. Importar librerías

```
import pandas as pd  
import quandl
```

#2. Autentíquese con la clave API personal:

```
QUANDL_API_KEY = '{key}'  
quandl.ApiConfig.api_key = QUANDL_API_KEY  
#Necesita reemplazar {key} con su propia clave API.
```

#3. Descargar los datos:

```
df_quandl = quandl.get(dataset='WIKI/AAPL',  
                      start_date='2010-01-01',  
                      end_date='2020-11-30')
```

Podemos inspeccionar los datos descargados:

Salida

Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	Adj. Close	Adj. Volume
2000-01-03	104.8700	112.50	101.80	111.94	4783900.0	0.0	1.0	3.368014	3.814454	3.287148	3.596463	133949200.0
2000-01-04	108.2500	110.62	101.10	102.50	4574800.0	0.0	1.0	3.477908	3.554053	3.251081	3.293170	128084400.0
2000-01-05	103.7500	110.56	103.00	104.00	6949300.0	0.0	1.0	3.333330	3.652125	3.309234	3.341362	194580400.0
2000-01-06	106.1200	107.00	95.00	95.00	6856900.0	0.0	1.0	3.409475	3.437748	3.052206	3.052206	191990200.0
2000-01-07	96.5000	101.00	95.50	99.50	4113700.0	0.0	1.0	3.100399	3.244977	3.068270	3.196784	115183600.0
...
2010-12-27	322.8519	325.44	321.52	324.68	8922000.0	0.0	1.0	41.480969	41.823578	41.319801	41.725905	82454000.0
2010-12-28	326.9102	326.66	325.06	325.47	6283000.0	0.0	1.0	41.884003	41.980383	41.774741	41.827431	43981000.0
2010-12-29	326.2200	326.45	325.10	325.29	5826400.0	0.0	1.0	41.923817	41.953375	41.779861	41.804299	40784800.0
2010-12-30	325.4800	325.51	323.05	323.66	5824800.0	0.0	1.0	41.828716	41.832572	41.516426	41.594821	39373600.0
2010-12-31	322.9500	323.48	321.31	322.56	6911900.0	0.0	1.0	41.503576	41.571689	41.292813	41.450456	46377000.0

2767 rows × 12 columns

El resultado de la solicitud es un DataFrame (2767 filas x 12 columnas) que contiene los precios diarios de OHLC (open,hight,low,close), los precios ajustados, los dividendos y las posibles split de acciones.

¿Cómo funciona el algoritmo?

El primer paso después de importar las librerías necesarias fue la autenticación con la clave API (péguela en lugar de {clave}). Al proporcionar el argumento del conjunto de datos, usamos la siguiente estructura: DATASET / TICKER.

Algunas características adicionales de la función *get* son:

- ▶ Podemos especificar varios conjuntos de datos a la vez usando una lista como ['WIKI / AAPL', 'WIKI / MSFT'].
- ▶ El parámetro *collapse* se puede utilizar para definir la frecuencia (opciones disponibles: diario, semanal, mensual, trimestral o anual).

4.3 OBTENIENDO DATOS DE INTRINIO

Otra fuente de datos financieros es Intrinio, que ofrece acceso gratuito a su base de datos (con algunos límites). Una característica adicional es que también podemos descargar indicadores para análisis técnico ya calculados como la divergencia convergencia de la media móvil (MACD), índice de fuerza relativa (RSI), medias móviles y bandas de Bollinger entre otros. Consulte <https://github.com/intrinio/Python-sdk> para ver la lista completa de Indicadores disponibles para descargar.

A pesar de que la base de datos no solo se limita a los precios de las acciones, aquí continuaremos con el ejemplo anterior de descargar los precios de las acciones de Apple para los años 2010-2020. Antes de descargar los datos, debemos registrarnos en <https://intrinio.com> para obtener la clave API.

Implementación en Python

#1. Importar librerías

```
import intrinio_sdk  
import pandas as pd
```

```
#2. Autenticar usando la clave API personal. Debe reemplazar ['api_key']  
por su propia clave API.
```

```
intrinio_sdk.ApiClient().configuration.api_key['api_key'] = '{key}'  
security_api = intrinio_sdk.SecurityApi()
```

#3. Solicitar los datos

```
r = security_api.get_security_stock_prices(identifier='AAPL',  
                                         start_date='2010-01-01',  
                                         end_date='2020-11-30',  
                                         frequency='daily',  
                                         page_size=10000)
```

```
#4. Convertir los resultados en un DataFrame:
```

```
response_list = [x.to_dict() for x in r.stock_prices]  
df_intrinio = pd.DataFrame(response_list).sort_values('date')  
df_intrinio.set_index('date', inplace=True)
```

#5. Mostrar los resultados

```
df_intrinio
```

Salida

Out[6]:													
	date	intraperiod	frequency	open	high	low	close	volume	adj_open	adj_high	adj_low	adj_close	adj_volume
2010-01-04		False	daily	213.50	214.50	212.3800	214.01	17633150.0	6.589059	6.619921	6.554494	6.604799	4.937282e+08
2010-01-05		False	daily	214.79	215.59	213.2500	214.38	2149572.0	6.628871	6.653561	6.581344	6.616216	8.019040e+08
2010-01-06		False	daily	214.38	215.23	210.7500	210.97	19719942.0	6.616218	6.642451	6.504189	6.510978	5.521584e+08
2010-01-07		False	daily	211.88	212.00	209.0500	210.58	17640332.0	6.532890	6.542768	6.451723	6.498942	4.771203e+08
2010-01-08		False	daily	210.40	212.00	200.0800	211.98	15095583.0	6.493387	6.542768	6.452032	6.542140	4.478763e+08
...
2020-10-06		False	daily	115.70	116.12	112.2500	113.16	161488212.0	115.700000	116.120000	112.250000	113.160000	1.814982e+08
2020-10-07		False	daily	114.82	115.55	114.1300	115.08	95848985.0	114.620000	115.550000	114.130000	115.080000	9.684898e+07
2020-10-08		False	daily	116.25	116.40	114.5801	114.97	83477153.0	116.250000	116.400000	114.589100	114.970000	8.347715e+07
2020-10-09		False	daily	115.28	117.00	114.9200	116.97	100598865.0	115.280000	117.000000	114.920000	116.970000	1.005969e+08
2020-10-12		False	daily	120.06	125.18	119.2845	124.40	237940124.0	120.060000	125.180000	119.284500	124.400000	2.379401e+08

2721 rows × 12 columns

El DataFrame resultante (2,721 filas x 12 columnas) contiene los precios OHLC y el volumen, así como sus contrapartes ajustadas.

¿Cómo funciona el algoritmo?

El primer paso después de importar las librerías requeridas fue la autenticación usando la clave API (pegarla en lugar de ['api_key']). Luego, seleccionamos la API que necesitamos usar para obtener los precios de las acciones; en este caso es SecurityApi.

Para descargar los datos, utilizamos el método `get_security_stock_prices` de la clase `SecurityApi`. Los parámetros que podemos especificar son los siguientes:

- ▶ identificador: Stock ticker u otro identificador aceptable.
- ▶ fecha_inicio / fecha_final: Esto se explica por sí mismo.
- ▶ frecuencia: qué frecuencia de datos nos interesa (opciones disponibles: diaria, semanal, mensual, trimestral o anual).
- ▶ tamaño_página: define el número de observaciones que se devolverán en una página; lo configuramos en un número alto para recopilar todos los datos en una solicitud sin necesidad del token `next_page`.

La API devuelve un objeto similar a JSON (*JavaScript Object Notation*, JSON es un formato para el intercambio de datos basado en texto), que luego se transforma en un DataFrame y se establece la fecha como un índice utilizando el método `set_index` de un DataFrame de pandas.

En esta sección, se han cubierto algunas fuentes de datos financieros: Yahoo Finance, Quandl e Intrinio. Existen fuentes de datos adicionales potencialmente interesantes como:

- ▀ iexfinance: una librería que se puede usar para descargar datos de IEX Cloud
- ▀ tiingo: una librería que se puede usar para descargar datos de Tiingo
- ▀ alpha_vantage: una librería que es un contenedor para la API de Alpha Vantage

4.4 TRANSFORMANDO PRECIOS DE ACTIVOS FINANCIEROS A RENDIMIENTOS

Los precios de los activos financieros generalmente no son estacionarios, es decir, sus estadísticas, como la media y la varianza (momentos matemáticos) cambian con el tiempo. Esto podría evitar observar algunas tendencias o estacionalidades en la serie de precios. Al transformar los precios en rendimientos, intentamos que la serie temporal sea estacionaria, que es una propiedad deseada en el modelado estadístico.

Existen dos tipos de rendimientos:

Rendimientos simples: se calculan sobre los precios de los activos y se componen (capitalizan) sobre subperíodos de tiempo. El rendimiento simple de un portafolio es la suma ponderada de los rendimientos de los activos individuales del portafolio.

Los rendimientos simples se definen como:

$$R_t = \frac{(P_t - P_{t-1})}{P_{t-1}}$$

Rendimientos logarítmicos: se componen (capitalizan) en forma continua en lugar de subperíodos. Los rendimientos logarítmicos de un mes determinado son la suma de los rendimientos logarítmicos de los días de ese mes. Los rendimientos logarítmicos se definen como:

$$R_t = \log\left(\frac{P_t}{P_{t-1}}\right)$$

P_t es el precio de un activo en el tiempo t . En el caso anterior, no se consideran los dividendos, que obviamente afectan los rendimientos y requieren una pequeña modificación de las fórmulas.

La mejor práctica al trabajar con precios de acciones es usar precios ajustados, ya que tienen en cuenta posibles decisiones corporativas, como divisiones (split) de acciones.

La diferencia entre los rendimientos simples y logarítmicos para datos diarios / intradía será muy pequeña, sin embargo, la regla general es que los rendimientos logarítmicos tienen un valor menor que los rendimientos simples. A continuación, se muestra cómo calcular ambos tipos de rendimiento utilizando los precios de las acciones de Apple.

Ejecute los siguientes pasos para descargar los precios de las acciones y calcular los rendimientos simples y logarítmicos.

Implementación en Python

#1. Importar librerías

```
import pandas as pd
import numpy as np
import yfinance as yf
```

#2. Descargue los datos y mantenga solo los precios de cierre ajustados:

```
df = yf.download('AAPL',
                 start='2010-01-01',
                 end='2020-11-30',
                 progress=False)
df = df.loc[:, ['Adj Close']]
df.rename(columns={'Adj Close':'adj_close'}, inplace=True)
```

#3. Calcule los rendimientos simples y log utilizando los precios de cierre ajustados:

```
df['simple rtn'] = df.adj_close.pct_change()
df['log rtn'] = np.log(df.adj_close/df.adj_close.shift(1))
```

#4. Mostrar los resultados

```
df
```

Salida

Date	adj_close	simple_rtn	log_rtn
2010-01-04	6.539882	NaN	NaN
2010-01-05	6.551187	0.001729	0.001727
2010-01-06	6.446983	-0.015906	-0.016034
2010-01-07	6.435065	-0.001849	-0.001850
2010-01-08	6.477847	0.006648	0.006626
...
2020-11-20	117.339996	-0.010958	-0.011018
2020-11-23	113.849998	-0.029743	-0.030194
2020-11-24	115.169998	0.011594	0.011528
2020-11-25	116.029999	0.007467	0.007439
2020-11-27	116.589996	0.004826	0.004815

2746 rows × 3 columns

La primera fila siempre contendrá un valor que no es un número (NaN), ya que no hay un precio anterior para calcular los rendimientos.

¿Cómo funciona el algoritmo?

En el Paso 2, descargamos datos de precios de Yahoo Finance y solo conservamos el precio de cierre ajustado para el cálculo de los rendimientos.

En el paso 3 para calcular los rendimientos simples, utilizamos el método *pct_change* de pandas Series / DataFrame, que calcula el cambio porcentual entre el elemento actual y el anterior (podemos especificar el número de rezagos, pero para este caso específico, el valor predeterminado de 1 es suficiente) .

Para calcular los rendimientos logarítmicos, seguimos la fórmula dada en la introducción de esta sección. Al dividir cada elemento de la serie por su valor previo, utilizamos el método de cambio con un valor de 1 para acceder al elemento anterior. Al final, tomamos el logaritmo natural de los valores divididos usando *np.log*.

Incorporando la inflación en la serie de rendimientos

La inflación es una variable que afecta el rendimiento de los activos financieros, usualmente de forma negativa. A continuación, se muestra como incorporar dicha variable al cálculo de los rendimientos. Se continuará utilizando como ejemplo las acciones de la empresa Apple.

Ejecute los siguientes pasos para contabilizar la inflación en la serie de rendimientos.

Implementación en Python

#1. Importar librerías y autenticarse

```
import pandas as pd
import numpy as np
import yfinance as yf
import quandl
```

#2. Reemplace {key} con su propia API key de Quandl

```
QUANDL_API_KEY = '{key}'
quandl.ApiConfig.api_key = QUANDL_API_KEY
```

#3. Descargar los precios de cierre ajustados para el período 2010-2020

```
df = yf.download('AAPL',
                 start='2010-01-01',
                 end='2020-11-30',
                 progress=False)
df = df.loc[:, ['Adj Close']]
df.rename(columns={'Adj Close':'adj_close'}, inplace=True)
```

#4. Calcula los rendimientos simples y logarítmicos

```
df['simple rtn'] = df.adj_close.pct_change()
df['log rtn'] = np.log(df.adj_close / df.adj_close.shift(1))
```

#5. Crear un DataFrame con la unión de las fechas (a la izquierda) y los precios de cierre a la derecha

```
df_all_dates = pd.DataFrame(index=pd.date_range(start='2009-12-31',
                                                 end='2020-11-30'))
df = df_all_dates.join(df[['adj_close']], how='left') \
    .fillna(method='ffill') \
    .asfreq('M')
```

En el paso 5 se usa una combinación izquierda, que es un tipo de combinación (utilizada para combinar DataFrames) que devuelve todas las filas de la tabla izquierda y las filas coincidentes de la tabla derecha mientras deja vacías las filas no coincidentes. En caso de que el último día del mes no fuera un día de negociación, utilizamos el último precio conocido de ese mes (fillna (method = 'ffill')). Por último, seleccionamos las filas de fin de mes solo aplicando asfreq ('M').

En el paso 6 del algoritmo, se descargan los valores mensuales del Índice de Precios al Consumidor (IPC) de Quandl y se calcula el cambio porcentual (rendimiento simple) en el precio. Luego se fusionan los datos de inflación con los rendimientos de las acciones de Apple, y finalmente en el paso 9 se contabiliza la inflación utilizando la siguiente fórmula:

$$R_R = \frac{1+R_t}{1+\pi_t} - 1$$

Aquí, R_t es un rendimiento simple, π_t es la tasa de inflación y R_R es rendimiento real.

Continúa implementación en Python

#6. Descargue los datos de inflación de Quandl:

```
df_cpi = quandl.get(dataset='RATEINF/CPI_USA',
                     start_date='2009-12-01',
                     end_date='2020-11-30')
df_cpi.rename(columns={'Value':'cpi'}, inplace=True)
```

#7. Combinar los datos de inflación con los precios:

```
df_merged = df.join(df_cpi, how='left')
```

#8. Calcule los rendimientos simples y la tasa de inflación:

```
df_merged['simple_rtn'] = df_merged.adj_close.pct_change()
df_merged['inflation_rate'] = df_merged.cpi.pct_change()
```

#9. Ajuste los rendimientos por inflación:

```
df_merged['real_rtn'] = (df_merged.simple_rtn + 1) / (df_merged.inflation_rate + 1) - 1
```

#10. Mostrar los resultados

```
df_merged
```

Salida

	adj_close	cpi	simple_rtn	inflation_rate	real_rtn
2009-12-31	NaN	215.949	NaN	NaN	NaN
2010-01-31	5.869116	216.687	NaN	0.003417	NaN
2010-02-28	6.252933	216.741	0.065396	0.000249	0.065131
2010-03-31	7.181309	217.631	0.148471	0.004106	0.143774
2010-04-30	7.978589	218.009	0.111022	0.001737	0.109095
...
2020-07-31	9.857034	259.101	0.000000	0.005058	-0.005033
2020-08-31	9.857034	259.918	0.000000	0.003153	-0.003143
2020-09-30	9.857034	260.280	0.000000	0.001393	-0.001391
2020-10-31	9.857034	260.388	0.000000	0.000415	-0.000415
2020-11-30	9.857034	260.229	0.000000	-0.000611	0.000611

132 rows × 5 columns

El DataFrame contiene todos los resultados intermedios, y la columna real_rtn contiene los rendimientos ajustados por inflación (rendimientos reales).

4.5 CAMBIANDO LA FRECUENCIA TEMPORAL DE LOS RENDIMIENTOS

La regla general para cambiar la frecuencia de los rendimientos se puede dividir de la siguiente forma:

- ▀ Multiplicar / dividir los rendimientos logarítmicos por el número de períodos de tiempo.
- ▀ Multiplicar / dividir la volatilidad por la raíz cuadrada del número de períodos de tiempo.

A continuación, se presenta un ejemplo de cómo calcular las volatilidades históricas mensuales para Apple utilizando rendimientos diarios y luego anualizándolos.

La fórmula para la volatilidad histórica es la siguiente:

$$RV = \sqrt{\sum_{t=1}^T r_t^2}$$

Para calcular la volatilidad histórica diaria se usan con frecuencia los rendimientos intradía.

Los pasos que se deben seguir son los siguientes:

- ▀ Descargue los datos y calcule los rendimientos logarítmicos
- ▀ Calcule la volatilidad histórica a lo largo de los meses
- ▀ Analice los valores multiplicando por $\sqrt{12}$, esto considerando que estamos convirtiendo valores mensuales

Ejecute los siguientes pasos para calcular y anualizar la volatilidad mensual histórica

Implementación en Python

```
#1. Importar las librerías requeridas

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import pandas as pd
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
import warnings
```

Continúa implementación en Python

```
#2. Definir estilo del gráfico y control mensajes de advertencia

plt.style.use('seaborn')
# plt.style.use('seaborn-colorblind') # Alternativa
# plt.rcParams['figure.figsize'] = [16, 9]
plt.rcParams['figure.dpi'] = 300
warnings.simplefilter(action='ignore', category=FutureWarning)
```

#3. Descargar los datos

```
df = yf.download('AAPL',
                 start='2010-01-01',
```

```
        end='2020-11-30',
        auto_adjust=False,
        progress=False)

# 4. Guardar solamente el precio de cierre ajustado

df = df.loc[:, ['Adj Close']]
df.rename(columns={'Adj Close': 'adj_close'}, inplace=True)

# 5. Calcular los rendimientos simples

df['log_rtn'] = np.log(df.adj_close/df.adj_close.shift(1))

#6. Eliminar datos redundantes

df.drop('adj_close', axis=1, inplace=True)
df.dropna(axis=0, inplace=True)

#7. Definir la función para calcular la volatilidad histórica

def realized_volatility(x):
    return np.sqrt(np.sum(x**2))

# 8. Calcular la volatilidad histórica mensual

df_rv = df.groupby(pd.Grouper(freq='M')).apply(realized_volatility)
df_rv.rename(columns={'log_rtn': 'rv'}, inplace=True)

#9. Anualizar los valores

df_rv.rv = df_rv.rv * np.sqrt(12)
```

Continúa implementación en Python

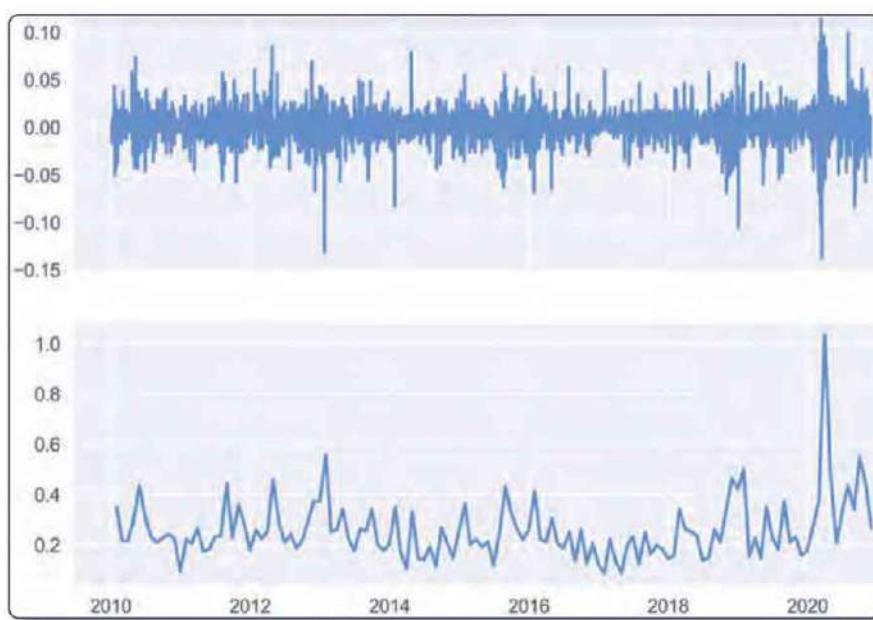
```
# 10. Graficar los resultados

fig, ax = plt.subplots(2, 1, sharex=True)
ax[0].plot(df)
ax[1].plot(df_rv)

# plt.tight_layout()
# plt.savefig('images/ch1_im6.png')
plt.show()
```

La ejecución del código anterior da como resultado los siguientes gráficos:

Salida



Podemos ver que los picos en la volatilidad histórica coinciden con algunos rendimientos extremos (que podrían ser atípicos).

¿Cómo funciona el algoritmo?

Normalmente, podríamos usar el método de remuestreo de un DataFrame de pandas. Suponiendo que quisiéramos calcular el rendimiento mensual promedio, podríamos ejecutar `df.log_rtn.resample('M').Mean()`.

Para el método de remuestreo, podemos usar cualquier función agregada incorporada de pandas, como media, suma, min y max. Sin embargo, nuestro caso es un poco más complejo, por lo que definimos en el paso 7 una función auxiliar llamada `realized_volatility(x)` y replicamos el comportamiento de remuestreo usando una combinación de groupby, Grouper y apply.

4.6 VISUALIZANDO DATOS DE SERIES DE TIEMPO FINANCIERAS

Después de aprender a descargar y preprocesar datos financieros, es hora de aprender cómo graficarlos de una manera visualmente atractiva. Se cubrirán dos enfoques usando lo siguiente:

-
- ▀ El método predeterminado *plot* de un DataFrame de pandas.
 - ▀ Una combinación de las librerías plotly y cufflinks.

La librería plotly está construida sobre d3.js (una librería de JavaScript utilizada para crear visualizaciones interactivas en navegadores web) y es conocida por crear gráficos de alta calidad con un grado significativo de interactividad (inspección de valores de observaciones, visualización de información sobre herramientas de un determinado punto, acercamiento, entre otros). Plotly es también la compañía responsable de desarrollar esta librería y brinda alojamiento para las visualizaciones. Es posible crear un número infinito de visualizaciones sin conexión y hasta 25 gratuitas para compartir en línea (con un número limitado de vistas por día).

La librería cufflinks también facilita el proceso, ya que permite crear visualizaciones gráficas directamente sobre los DataFrames de pandas.

A continuación, se grafican los precios de las acciones de Microsoft y los rendimientos. Para detalles sobre cómo descargar y procesar los datos, consulte los numerales 4.1, 4.2 y 4.3.

Para este ejemplo, suponemos que ya tenemos un DataFrame llamado df con tres columnas (adj_close, simple rtn y log rtn) y fechas establecidas como índice.

En esta sección, presentamos cómo graficar datos de series de tiempo. Comenzamos utilizando el método predeterminado *plot* de un DataFrame / serie de pandas, y luego presentamos la alternativa interactiva que ofrece la combinación con cufflinks.

El método gráfico de Pandas

Ejecute el siguiente código para graficar los precios de las acciones de Microsoft junto con los rendimientos simples y logarítmicos.

Implementación en Python

#1. Importar librerías

```
import pandas as pd  
import numpy as np  
import yfinance as yf
```

#2. Descargar los datos y mantener solo los precios de cierre ajustados:

```
df = yf.download('MSFT',
                 start='1988-01-01',
                 end='2020-12-31',
                 progress=False)
df = df.loc[:, ['Adj Close']]
df.rename(columns={'Adj Close':'adj_close'}, inplace=True)
```

#3. Calcular los rendimientos simples y logarítmico utilizando los precios de cierre ajustados:

```
df['simple_rtn'] = df.adj_close.pct_change()
df['log_rtn'] = np.log(df.adj_close/df.adj_close.shift(1))
```

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

#4. Importar librerías

```
import matplotlib.pyplot as plt
import warnings
```

```
fig, ax = plt.subplots(3, 1, figsize=(12, 10), sharex=True)
```

#5. Agregar precios

```
df.adj_close.plot(ax=ax[0])
ax[0].set(title = 'MSFT time series',
          ylabel = 'Stock price ($)')
```

#6. Agregar rendimientos simples

```
df.simple_rtn.plot(ax=ax[1])
ax[1].set(ylabel = 'Simple returns (%)')
```

Continúa implementación en Python

#7. Agregar rendimientos logarítmicos y graficar

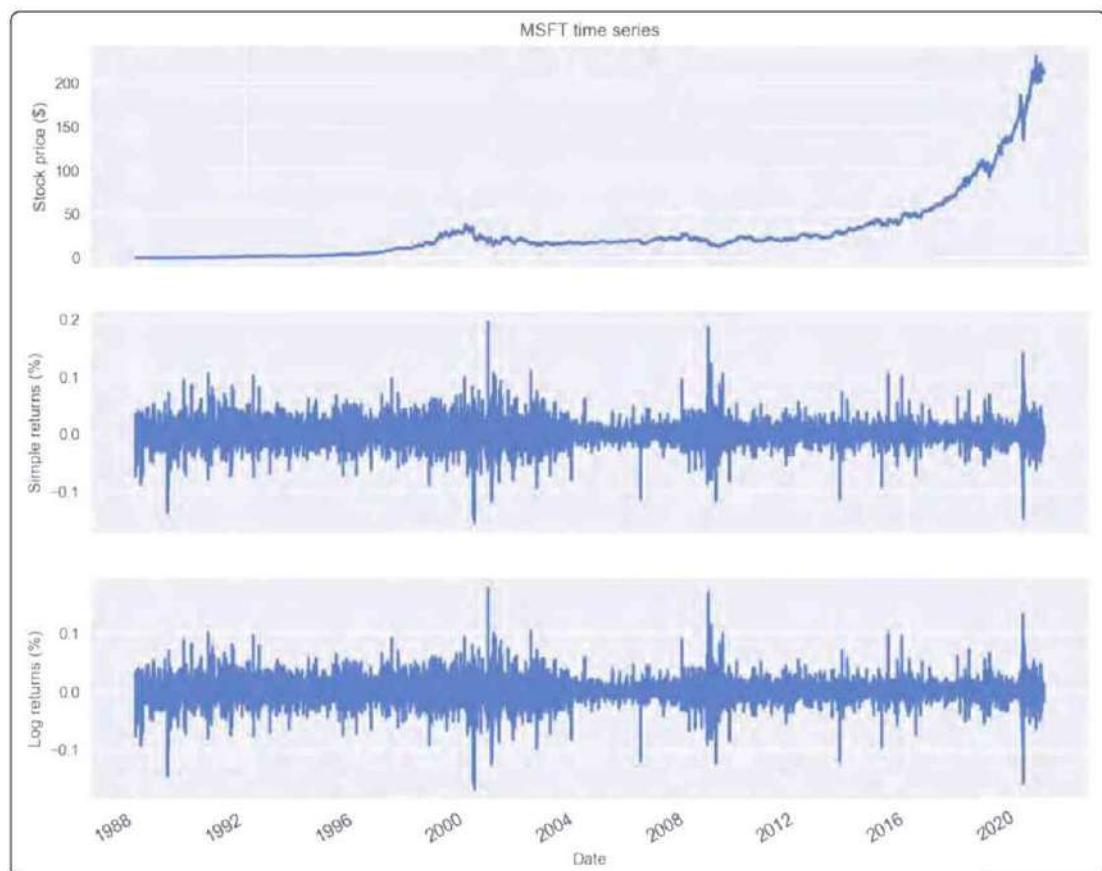
```
df.log_rtn.plot(ax=ax[2])
ax[2].set(xlabel = 'Date',
          ylabel = 'Log returns (%)')
```

```
ax[2].tick_params(axis='x',
                   which='major',
                   labelsize=12)

# plt.tight_layout()
# plt.savefig('images/ch1_im7.png')
plt.show()
```

La ejecución del código anterior da como resultado el siguiente diagrama:

Salida



La gráfica resultante contiene tres ejes. Cada uno de ellos presenta una serie diferente: precios de la acción, rendimientos simples y rendimientos logarítmicos. Al revisar la trama en tal entorno nos permite ver los períodos de mayor volatilidad y lo que sucedía al mismo tiempo con el precio de las acciones de Microsoft. Además, vemos cuán similares son los rendimientos simples y los rendimientos logarítmicos.

Plotly y cufflinks

Ejecute el siguiente código para graficar los precios de las acciones de Microsoft junto con los rendimientos simples y logarítmicos.

Implementación en Python

```
#1. Importar librerías
```

```
import pandas as pd
import numpy as np
import yfinance as yf
```

```
#2. Descargue los datos y mantenga solo los precios de cierre ajustados:
```

```
df = yf.download('MSFT',
                 start='1988-01-01',
                 end='2020-12-31',
                 progress=False)
df = df.loc[:, ['Adj Close']]
df.rename(columns={'Adj Close':'adj_close'}, inplace=True)
```

```
#3. Calcule los rendimientos simples y log utilizando los precios de cierre ajustados:
```

```
df['simple rtn'] = df.adj_close.pct_change()
df['log rtn'] = np.log(df.adj_close / df.adj_close.shift(1))
```

```
#4. Importar las librerías y manejar la configuración:
```

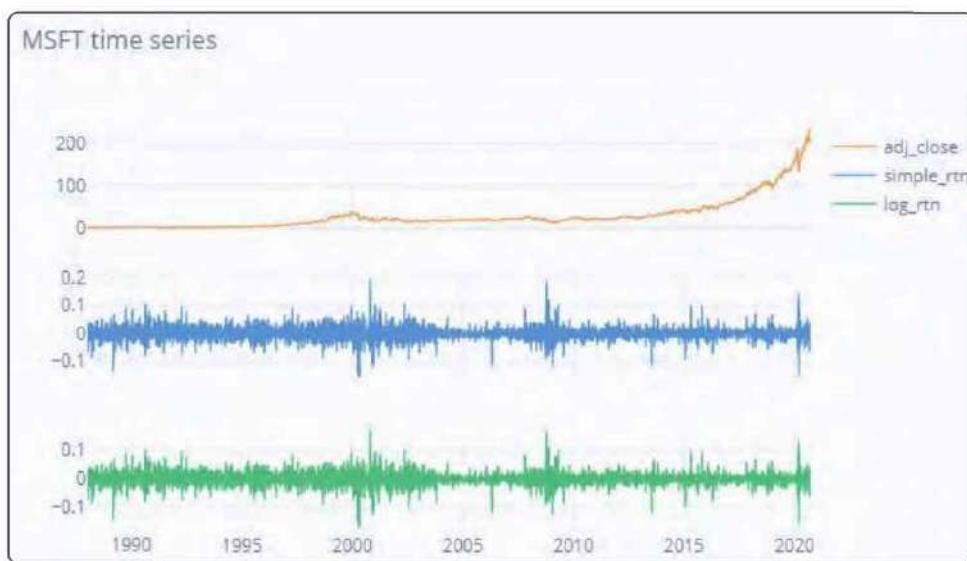
```
%matplotlib inline
import plotly.graph_objs as go
from plotly.offline import plot
import chart_studio.plotly as py
import cufflinks as cf
cf.go_offline()
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,iplot
init_notebook_mode(connected='true')
```

```
#5. Crear el gráfico
```

```
df.iplot(subplots=True, shape=(3,1), shared_xaxes=True, title='MSFT time series')
```

Podemos observar las series temporales en la siguiente gráfica:

Salida



La principal ventaja de usar *plotly* con *cufflinks* es la interactividad del gráfico anterior, que lamentablemente solo se puede demostrar en el Notebook.

¿Cómo funciona el algoritmo?

A continuación, vamos a recorrer los detalles de los dos métodos seleccionados de graficación de series de tiempo en Python.

El método gráfico de pandas

El objetivo era visualizar las tres series en la misma trama (compartiendo el eje x) para permitir una comparación visual rápida. Para lograr esto, se completaron los siguientes pasos:

- ▶ Crear una subtrama, que luego se pobló con tramas individuales. Especificamos que queríamos tres parcelas verticalmente (indicando `plt.subplots(3,1)`). También especificamos el tamaño de la figura configurando el parámetro `figsize`.
- ▶ Se agregan los gráficos individuales usando el método de diagrama en una sola Serie (columna) y especificando el eje en el que queríamos colocar el diagrama.

- ▶ Utilizar el método set para especificar las etiquetas de título y eje en cada una de las parcelas.
- ▶ Recomendación: Cuando trabaje en Jupyter Notebook, la mejor práctica es ejecutar %matplotlib inline magic (una vez por núcleo) para mostrar el trazado directamente debajo de la celda de código que lo ha producido. Además, si está trabajando en un MacBook con una pantalla Retina, ejecute el IPython magic% config extra InlineBackend.figure_format= 'retina' , esto duplicará la resolución de los gráficos que realice.

El método gráfico de Plotly y cufflinks

Al usar **cufflinks**, podemos usar el método iplot directamente en un DataFrame de pandas. Para crear la gráfica anterior, utilizamos subtramas (subplots = True), especificamos la forma de la figura (shape = (3,1)), indicamos que las gráficas comparten el eje x (shared_xaxes = True), y agregamos el título (título = MSFT time series ').

Una nota sobre el uso de plotly en Jupyter: para compartir un cuaderno con la opción de ver el diagrama (sin volver a ejecutar el script), debe usar nbviewer o renderizar el cuaderno como un archivo HTML y luego compartirlo.

Existen otras formas de crear gráficos en Python. A continuación se mencionan algunas de las librerías:

- ▶ Matplotlib
- ▶ Seaborn
- ▶ Plotly
- ▶ Plotly_express
- ▶ Altair
- ▶ Plotnine

Se han presentado las dos seleccionadas por su simplicidad, sin embargo, un caso de uso específico podría requerir la utilización de algunas de las librerías mencionadas anteriormente, ya que pueden ofrecer más flexibilidad al crear la visualización. También se debería mencionar que el método de graficación de un DataFrame de pandas en realidad usa matplotlib para el gráfico, la API de pandas facilita el proceso.

4.7 IDENTIFICANDO VALORES ATÍPICOS EN LA SERIE TEMPORAL

Mientras se trabaja con cualquier tipo de datos, a menudo se encuentran observaciones que son significativamente diferentes de la mayoría, es decir, valores atípicos. Pueden ser el resultado de un precio incorrecto o un error en el procesamiento de los datos, y así sucesivamente. Muchos algoritmos de aprendizaje automático y enfoques estadísticos pueden verse influenciados por valores atípicos, lo que conduce a resultados incorrectos / sesgados. Es por eso que debemos manejar los valores atípicos antes de crear cualquier modelo.

En este ejemplo, buscamos detectar valores atípicos utilizando el enfoque 3σ .

Ejecute los siguientes pasos para detectar valores atípicos utilizando el enfoque 3σ e identifíquelos en un gráfico:

Implementación en Python

#1. Importe las librerías requeridas

```
import pandas as pd
import yfinance as yf
```

#2. Descargue los datos y mantenga solo los precios de cierre ajustados:

```
df = yf.download('AAPL',
                 start='2000-01-01',
                 end='2010-12-31',
                 progress=False)
```

```
df = df.loc[:, ['Adj Close']]
df.rename(columns={'Adj Close':'adj_close'}, inplace=True)
```

3. Convierta los precios de cierre ajustados a rendimientos simples

```
df['simple rtn'] = df.adj_close.pct_change()
```

#4. Calcule la media móvil y la desviación estandar:

```
df_rolling = df[['simple rtn']].rolling(window=21) \
              .agg(['mean', 'std'])
df_rolling.columns = df_rolling.columns.droplevel()
```

#5. Unir las medias móviles a los datos originales:

```
df_outliers = df.join(df_rolling)
```

Continúa implementación en Python

```
#6.Defina una función para detectar valores atípicos:  
def identify_outliers(row, n_sigmas=3):  
    """  
        Función para identificar los valores atípicos utilizando la regla 3 sigma.  
        La fila debe contener las siguientes columnas / índices: simple_rtn, mean, std.
```

Parámetros

row : pd.Series

Una fila de un pd.DataFrame, sobre la cual se puede aplicar la función.

n_sigmas : int

*El número de desviaciones estándar por encima / por debajo de la media,
utilizado para detectar valores atípicos*

Returns

0/1 : int

Un número entero con 1 que indica un valor atípico y 0 en caso contrario.

"""

x = row['simple_rtn']

mu = row['mean']

sigma = row['std']

```
if (x > mu + 3 * sigma) | (x < mu - 3 * sigma):
```

```
    return 1
```

```
else:
```

```
    return 0
```

```
#7.Identifique los valores atípicos y extraiga sus valores para su uso posterior:
```

```
df_outliers['outlier'] = df_outliers.apply(identify_outliers,  
                                         axis=1)  
outliers = df_outliers.loc[df_outliers['outlier'] == 1,  
                           ['simple_rtn']]
```

Continúa implementación en Python

```
#8.Grafique los resultados:
```

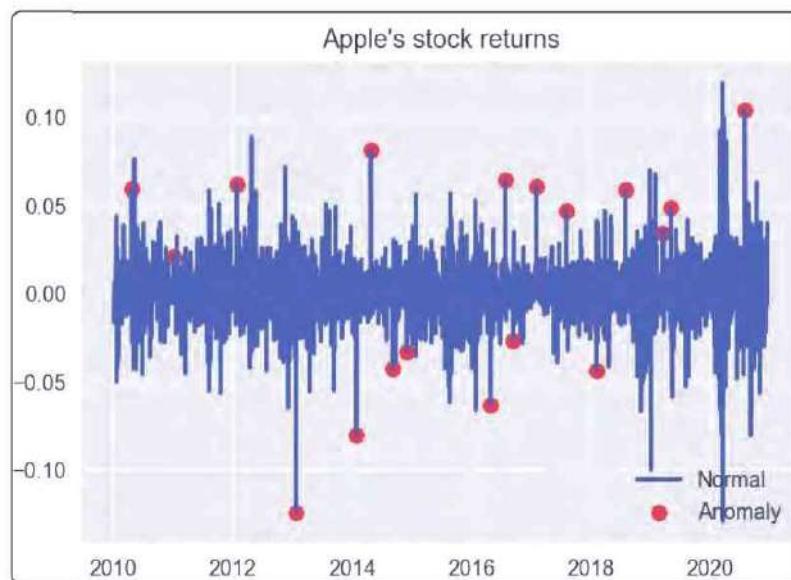
```
fig, ax = plt.subplots()
```

```
ax.plot(df_outliers.index, df_outliers.simple rtn,
        color='blue', label='Normal')
ax.scatter(outliers.index, outliers.simple rtn,
           color='red', label='Anomaly')
ax.set_title("Apple's stock returns")
ax.legend(loc='lower right')

# plt.tight_layout()
# plt.savefig('images/ch1_im9.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:

Salida



En la trama, podemos observar valores atípicos marcados con un punto rojo.

¿Cómo funciona el algoritmo?

En el enfoque 3σ , para cada punto de tiempo, se calcula la media móvil (μ) y la desviación estándar (σ) utilizando los últimos 21 días (sin incluir ese día). Usamos 21, ya que este es el número promedio de días de negociación en un mes y trabajamos con datos diarios. Sin embargo, se pueden elegir diferentes valores, y luego la media móvil reaccionará más rápido / más lento a los cambios. También podemos usar la media móvil ponderada (exponencialmente) si la encontramos más significativa en nuestro caso particular.

La condición para que una observación x dada se califique como un valor atípico es $x > \mu + 3\sigma$ ó

$$x < \mu - 3\sigma.$$

Se calculan las medias móviles (paso 4) utilizando el método *rolling* de un DataFrame de pandas. Especificamos el tamaño de la ventana y las medias que nos gustaría calcular. En el quinto paso, unimos los dos DataFrames.

En el paso 6, definimos una función que devuelve 1 si la observación se considera un valor atípico, de acuerdo con la regla 3σ (parametrizamos el número de desviaciones estándar) y 0 en caso contrario. Luego, en el séptimo paso, aplicamos la función a todas las filas del DataFrame usando el método *apply*.

En el último paso, graficamos la serie de rendimientos y marcamos los valores atípicos con un punto rojo. En casos de la vida real, no solo debemos identificar los valores atípicos, sino también tratarlos, por ejemplo, limitándolos al valor máximo / mínimo aceptable, reemplazándolos por valores interpolados, o siguiendo otros enfoques posibles.

4.8 OBTENCIÓN Y REMUESTREO DE DATOS SOBRE CRIPTOMONEDAS

El comercio de criptomonedas ha ganado una enorme popularidad en los últimos años. Concebir estrategias comerciales rentables requiere estrategias bien pensadas y backtesting adecuado con datos confiables. Afortunadamente, obtener estos datos es significativamente más fácil que para las clases de activos tradicionales, ya que muchas casas de cambio (Exchange) ofrecen APIs para descargar los datos de forma gratuita. En esta sección usaremos una de estas API para extraer datos de negociaciones y volver a muestreálos para obtener datos OHLC (Open-High-Low-Close) para una frecuencia determinada.

4.8.1 Obteniendo los datos de Kraken (Exchanges)

Para obtener nuestros datos de negociaciones de criptomonedas, usaremos la API de Kraken (otros Exchanges también brindan excelentes API, por ejemplo, Binance o Bitfinex, por mencionar algunos). Kraken proporciona acceso gratuito a los datos del mercado sin exigir la apertura de una cuenta real. Para obtener más información sobre la API, consulte su documentación. A continuación la implementación en Python:

Kraken proporciona una API REST fácil de usar que podemos usar convenientemente con solo unas pocas líneas de código. Como primer paso,

importaremos las librerías que necesitamos. Usaremos la librería *request* para enviar nuestra consulta a la API y *Pandas* para almacenar el resultado en un DataFrame.

```
import requests
import pandas as pd
```

Para el ejemplo utilizaremos la criptomoneda Ethereum (ETH) en dólares estadounidenses. En Kraken, este par cripto-fiduciario tiene el símbolo XETHZUSD. Para devolver las 1000 operaciones más recientes, solo se necesitan un par de líneas de código:

```
endpoint = 'https://api.kraken.com/0/public/Trades'
payLoad = {'pair': 'XETHZUSD'}
response = requests.get(url=endpoint, params=payLoad)
tradeData = response.json()
trades = tradeData['result']['XETHZUSD']
```

Entonces, ¿qué está pasando aquí? En primer lugar, definimos el punto final de la API al que enviamos nuestra consulta. Luego configuramos un diccionario con los parámetros para enviar nuestra consulta, en este caso el par. Solicitamos nuestro resultado usando el método GET y nos queda un objeto json que convertimos en un diccionario. El diccionario tiene dos claves, error y resultado. Si todo funcionó, el error es solo una lista vacía y el resultado es nuevamente un diccionario con dos claves, el nombre de par (XETHZUSD en nuestro ejemplo).

El valor del nombre del par de nuestro diccionario de resultados es una lista de listas con cada entrada que contiene la información de una sola operación que se produjo en el intercambio. Podemos imprimir las primeras 5 operaciones e inspeccionar los datos resultantes.

```
trades[:5]
```

Salida

```
[[481.79000, '1.0000000', 1605635544.883, 's', 'm'],
 [481.86000, '0.1000000', 1605635545.9436, 'b', 'm'],
 [481.74000, '0.02925041', 1605635551.0994, 's', 'l'],
 [481.71000, '0.5000000', 1605635551.1095, 's', 'l'],
 [481.69000, '0.51274959', 1605635551.1138, 's', 'l']]
```

Según la documentación de la API de Kraken, los valores de cada elemento de la lista corresponden a Precio, Volumen, Tiempo, Compra o Venta, Mercado u Orden Límite y Varios (que podemos ignorar). Para convertir esto en un DataFrame de Pandas, podemos usar el método `from_records`.

```
tradesDF = pd.DataFrame.from_records(trades,
columns=['Price','Volume','Time','BuySell','MarketLimit','Misc'])
tradesDF
```

Salida

	Price	Volume	Time	BuySell	MarketLimit	Misc
0	481.79000	1.0000000	1.605636e+09	s	m	
1	481.86000	0.1000000	1.605636e+09	b	m	
2	481.74000	0.02925041	1.605636e+09	s	l	
3	481.71000	0.50000000	1.605636e+09	s	l	
4	481.69000	0.51274959	1.605636e+09	s	l	
...
995	477.76000	0.14965576	1.605639e+09	s	l	
996	477.75000	9.49528721	1.605639e+09	s	l	
997	477.99000	0.02000000	1.605639e+09	s	l	
998	478.00000	0.98019602	1.605639e+09	b	m	
999	478.24000	1.51980398	1.605639e+09	b	m	

1000 rows × 6 columns

Probablemente haya notado que la columna Tiempo no está en formato de fecha y hora. De hecho, Kraken devuelve las marcas de tiempo en segundos con 4 dígitos de precisión. Podemos convertir fácilmente estas marcas de tiempo en objetos pandas de fecha y hora y usarlos como índice para nuestro DataFrame. Un punto a tener en cuenta es que las marcas de tiempo que devuelve Kraken utilizan UTC como zona horaria.

```
tradesDF['Time'] = pd.to_datetime(tradesDF['Time'], unit='s')
tradesDF.set_index('Time', inplace=True)
```

```
tradesDF
```

Salida

Price	Volume	BuySell	MarketLimit	Misc
Time				
2020-11-17 17:52:24.882999897	481.79000	1.00000000	s	m
2020-11-17 17:52:25.943599939	481.86000	0.10000000	b	m
2020-11-17 17:52:31.099400043	481.74000	0.02925041	s	l
2020-11-17 17:52:31.109499931	481.71000	0.50000000	s	l
2020-11-17 17:52:31.113800049	481.69000	0.51274959	s	l
...
2020-11-17 18:48:13.031599997	477.76000	0.14965576	s	l
2020-11-17 18:48:13.035399914	477.75000	9.49528721	s	l
2020-11-17 18:48:42.795000076	477.99000	0.02000000	s	l
2020-11-17 18:48:49.018599987	478.00000	0.98019602	b	m
2020-11-17 18:48:49.033600092	478.24000	1.51980398	b	m

1000 rows × 5 columns

Ahora que sabemos cómo obtener las últimas 1000 operaciones, la pregunta obvia es ¿cómo obtenemos más? Como se mencionó anteriormente, nuestro diccionario de resultados también contiene una clave llamada last. Esta entrada contiene la última marca de tiempo (en nanosegundos) para la que consultamos datos.

```
tradeData["result"]["last"]
```

Salida

```
'1605638929033600676'  
pd.to_datetime(int(tradeData["result"]["last"]),unit="ns")
```

Salida

```
Timestamp('2020-11-17 18:48:49.033600676')
```

Cuando enviamos nuestra consulta a Kraken antes, solo teníamos el nombre de par como parámetro. También podemos enviar un valor para since, que toma una marca de tiempo en nanosegundos como entrada. Entonces, para consultar datos de un ticket específico para un período de tiempo determinado, simplemente necesitamos convertir una fecha de inicio en nanosegundos, usarla como el primer

valor desde y luego volver a consultar con la última marca de tiempo que se devuelve hasta que alcancemos nuestra fecha de finalización. A continuación, encontrará una función que hace esto por usted, para cualquier par dado que figure en la casa de cambio (Exchange) Kraken:

```
import requests
import pandas as pd
import datetime
from datetime import timezone
import time

def getKrakenTradeData(pair, startDate, endDate):
    endpoint = 'https://api.kraken.com/0/public/Trades'

    startTime = int(datetime.datetime.strptime(startDate, '%Y-%m-%d').
    replace(tzinfo=timezone.utc).timestamp())*1000000000
    endTime = int(datetime.datetime.strptime(endDate, '%Y-%m-%d').
    replace(tzinfo=timezone.utc).timestamp())*1000000000

    timeLoaded = startTime

    result = pd.DataFrame()

    while timeLoaded < endTime:
        print(pd.to_datetime(timeLoaded, unit='ns').strftime('%Y-%m-%d %H:%M:%S'))
        payLoad = {'pair': pair,
                   'since': timeLoaded}

        response = requests.get(url=endpoint, params=payLoad)
        data = response.json()['result']
        tradesRaw = data[pair]
        timeLoaded = int(data["last"])

        tradeData = pd.DataFrame.from_records(tradesRaw,
                                               columns=['Price', 'Volume', 'Time', 'BuySell',
                                               'MarketLimit', 'Misc'])
        tradeData['Time'] = pd.to_datetime(tradeData['Time'], unit='s')

        result = result.append(tradeData)

        time.sleep(3)

    result.set_index("Time", inplace = True)
    result = result.loc[startDate:endDate+' 00:00:00']

    return result
```

Salida

```
2020-01-01 00:00:00
2020-01-01 11:23:20
2020-01-01 19:56:56
trades
```

Salida

Price	Volume	BuySell	MarketLimit	Misc
Time				
2020-01-01 00:00:03.751399994	128.66000	0.02000000	b	m
2020-01-01 00:01:53.191600084	128.56000	0.05700488	b	m
2020-01-01 00:03:20.194700003	128.55000	5.40752000	s	l
2020-01-01 00:04:01.168299913	128.55000	0.61468754	s	m
2020-01-01 00:04:07.520699978	128.55000	0.89984228	s	l
...
2020-01-01 23:54:42.613500118	130.30000	11.26862000	b	l
2020-01-01 23:56:11.696500063	130.38999	6.03000000	b	m
2020-01-01 23:56:11.703900099	130.40000	43.97000000	b	m
2020-01-01 23:56:42.356400013	130.37000	1.47302839	s	l
2020-01-01 23:56:42.366400003	130.33000	3.57000000	s	l

2529 rows × 5 columns

4.8.2 Remuestreo

Después de esta explicación relativamente larga sobre cómo obtener los datos, el remuestreo es más sencillo. Pandas nos proporciona el método `dataframe.resample`. Proporciona conversión de frecuencia y remuestreo de nuestros datos. El método trabaja junto con una función de agregación para convertir nuestros datos. Esto es más fácil de lo que parece:

```
trades.resample('1H')[‘Price’].agg([‘first’])
```

En este ejemplo, primero volvemos a muestrear el DataFrame a datos por hora (“1H”), seleccionamos la columna Precio y luego agregamos utilizando el primer método.

Salida

Time	first
2020-01-01 00:00:00	128.66000
2020-01-01 01:00:00	128.33000
2020-01-01 02:00:00	130.19999
2020-01-01 03:00:00	130.47999
2020-01-01 04:00:00	129.72999
2020-01-01 05:00:00	129.66999
2020-01-01 06:00:00	129.82000
2020-01-01 07:00:00	130.05000

Finalmente, podemos extender fácilmente esto para obtener nuestros datos OHLC para datos horarios de ETH.

```
ohlc=trades.resample("1H",label="left")["Price"].agg(["first","max","min","last"]).set_axis(["Open","High","Low","Close"],1)
```

```
ohlc
```

Salida

Open	High	Low	Close	
Time				
2020-01-01 00:00:00	128.66000	128.66000	128.19000	128.41999
2020-01-01 01:00:00	128.33000	130.05000	128.30000	130.05000
2020-01-01 02:00:00	130.19999	130.58000	130.05000	130.44000
2020-01-01 03:00:00	130.47999	130.47999	129.61000	129.63000
2020-01-01 04:00:00	129.72999	130.13000	129.66999	129.66999
2020-01-01 05:00:00	129.66999	129.90000	129.66999	129.82000
2020-01-01 06:00:00	129.82000	130.25000	129.81000	130.04000
2020-01-01 07:00:00	130.05000	130.25000	129.74000	129.74000
2020-01-01 08:00:00	129.82000	129.86000	129.60000	129.81000
2020-01-01 09:00:00	130.03000	130.15000	129.97999	130.02000

En esta sección, hemos visto cómo descargar fácilmente datos de negociación de criptomonedas de la casa de cambio (Exchange) Kraken a través de su API y volver a muestrearlos para obtener datos OHLC para una frecuencia deseada. Ahora puede usar esto para analizar los mercados de criptomonedas y tal vez idear algunas estrategias comerciales rentables.

5

EL VALOR DEL DINERO EN EL TIEMPO

En este capítulo, se presentarán y discutirán en detalle varios conceptos y fórmulas asociados con las finanzas. Dado que esos conceptos y fórmulas son tan básicos, los lectores que han tomado un curso de finanzas, o los profesionales con algunos años de experiencia laboral en el área financiera, podrían revisar este capítulo rápidamente. Una vez más, una característica de este libro, bastante diferente de un libro de texto de finanzas típico, es que Python se utiliza como herramienta computacional. En particular, se cubrirán los siguientes temas:

- ▀ Valor temporal del dinero, valor futuro y valor presente
- ▀ Valor presente de una perpetuidad
- ▀ Valor presente de una perpetuidad creciente
- ▀ Valor presente y futuro de una anualidad
- ▀ Cálculo de la cuota de un préstamo y la tabla de amortización
- ▀ Definición del VPN y regla del VPN
- ▀ Definición de la TIR y regla de la TIR

5.1 INTRODUCCIÓN AL VALOR TEMPORAL DEL DINERO, VALOR FUTURO Y VALOR PRESENTE

Usemos un ejemplo simple para ilustrar la situación. Suponga que hoy se depositan € 100 en un banco con una tasa de interés anual del 10%. ¿Cuál es el valor del depósito un año después? En la figura 5.1 se puede observar la línea de tiempo y los flujos de efectivo:



Figura 5.1. Diagrama flujo de efectivo a un año

Obviamente, nuestro pago anual de intereses será de € 10, es decir, $100 * 0.1 = 10$. Por lo tanto, el valor total será 110, es decir, $100 + 10$. Los € 100 originales son el principal. Alternativamente, tenemos el siguiente resultado:

$$100 + 100 * 0.10 = 100 * (1 + 0.10)$$

Suponga que se mantendrán € 100 en el banco durante dos años con la misma tasa de interés anual del 10% durante dos años. ¿Cuál será el valor futuro al final del segundo año?

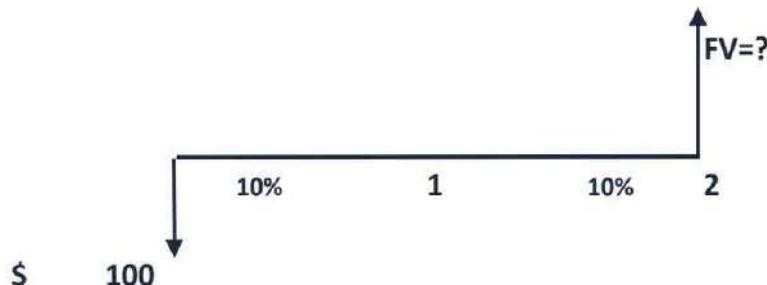


Figura 5.2. Diagrama flujo de efectivo a dos años

Como al final del primer año, tenemos € 110 y aplicando la misma lógica, el valor futuro al final del segundo año debería ser:

$$110 + 110 * 0.10 = 110 * (1 + 0.10) = 121$$

Como $110 = 100 * (1 + 0.1)$, tenemos la siguiente expresión:

$$FV(\text{al final del año 2}) = 100 * (1 + 0.10)^2$$

Si se depositan € 100 por cinco años con una tasa de interés anual del 10%, ¿cuál es el valor futuro al final del quinto año? Según la lógica anterior, podríamos tener la siguiente fórmula:

$$FV(\text{al final del año } 5) = 100 * (1 + 0.10)^5$$

La generalización lleva a nuestra primera fórmula para estimar el valor futuro de un valor presente dado:

$$FV = PV * (1 + R)^n \quad \dots \dots \dots (1)$$

Aquí, FV es el valor futuro, PV es el valor presente, R es la tasa del período y n es el número de períodos. En el ejemplo anterior, R es la tasa de interés anual y n es el número de años. Las frecuencias de R y n deberían ser las mismas. Esto significa que si R es la tasa anual (mensual / trimestral / diaria), entonces n debe ser el número de años (meses / trimestres / días). La función correspondiente, llamada fv () en el módulo numpy_financial, podría usarse para estimar el valor futuro. Para estimar el valor futuro al final del año dos con una tasa de interés anual del 10%, tenemos el siguiente código:

```
>>>import numpy_financial as npf #Se importa la librería
>>> npf.fv(0.1,2,0,100)
-121.0000000000001 #Resultado
```

Para la función, el formato de entrada es npf.fv(rate, nper, pmt, pv[, when]). Por el momento, simplemente ignore el último parámetro llamado when. Para la ecuación (1), no hay pmt, por lo tanto, la tercera entrada debe ser cero. Por favor, preste atención al signo negativo del resultado anterior. La razón es que la función npf.fv () sigue la convención de signos de Excel: un valor futuro positivo conduce a un valor presente negativo, y viceversa.

De la ecuación (1), podríamos derivar fácilmente nuestra segunda fórmula:

$$PV = \frac{FV}{(1+R)^n} \quad \dots \dots \dots (2)$$

Las notaciones de PV, FV, R y n siguen siendo las mismas que las de la ecuación (1). Si planeamos tener € 234 al final del quinto año y la tasa de interés es de 1.45% por año, ¿cuánto tenemos que depositar hoy? El resultado se muestra a continuación: primero después de aplicar la ecuación (2) manualmente y segundo se podría usar la función npf.pv():

```
>>> 234/(1+0.0145)**5  
217.74871488824184  
  
>>> npf.pv (0.0145,5,0,234)  
-217.74871488824184
```

Para encontrar más información sobre esta función, tecleamos help(npf.fv):

```
>>> import numpy_financial as npf  
>>> help(npf.fv)
```

Salida

```
Help on function pv in module numpy_financial._financial:  
  
pv(rate, nper, pmt, fv=0, when='end')  
    Compute the present value.  
  
    Given:  
        * a future value, `fv`  
        * an interest `rate` compounded once per period, of which  
            there are  
        * `nper` total  
        * a (fixed) payment, `pmt`, paid either  
            * at the beginning (`when` = {'begin', 1}) or the end  
                (`when` = {'end', 0}) of each period  
  
    Return:  
        the value now  
  
    Parameters  
    -----  
    rate : array_like  
        Rate of interest (per period)  
    nper : array_like
```

```

Number of compounding periods
pmt : array_like
    Payment
fv : array_like, optional
    Future value
when : {{'begin', 1}, {'end', 0}}, {string, int}, optional
    When payments are due ('begin' (1) or 'end' (0))

Returns
-----
out : ndarray, float
    Present value of a series of payments or investments.

```

5.2 VALOR PRESENTE DE UNA PERPETUIDAD

El siguiente concepto es la perpetuidad, que se define como los mismos flujos de efectivo constantes, en los mismos intervalos para siempre. Aquí está la línea de tiempo y esos flujos de efectivo constantes:

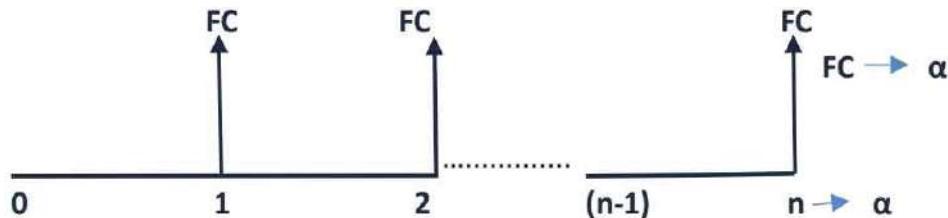


Figura 5.3. Diagrama flujo de efectivo a perpetuidad

Tenga en cuenta que en el caso anterior, el primer flujo de efectivo ocurre al final del primer período. ¿Cuál es el valor presente de tal perpetuidad cuando la tasa de descuento del período es R ? Primero, la Ecuación (2) podría aplicarse a cada uno de esos flujos de efectivo futuros. Por lo tanto, la suma de todos esos valores presentes será la solución:

$$PV(\text{perpetuidad}) = \frac{FC}{(1+R)^1} + \frac{FC}{(1+R)^2} + \frac{FC}{(1+R)^3} + \dots$$

Para simplificar nuestra derivación, PV (perpetuidad) se reemplaza por PV. Llamémoslo Ecuación (I):

$$PV = \frac{FC}{(1+R)^1} + \frac{FC}{(1+R)^2} + \frac{FC}{(1+R)^3} + \dots \quad (1)$$

Para derivar la fórmula, ambos lados de la ecuación (I) se multiplican por $1 / (1 + R)$; ver la siguiente ecuación. Llamémoslo Ecuación (2):

$$PV \frac{1}{(1+R)^1} = \frac{C}{(1+R)^2} + \frac{C}{(1+R)^3} + \dots \quad (2)$$

La ecuación (I) menos la ecuación (II) conduce a la siguiente ecuación:

$$PV - PV \frac{1}{(1+R)^1} = \frac{C}{(1+R)^1}$$

Multiplicando a ambos lados por $(1 + R)$, tenemos:

$$PV(1 + R) - PV = C$$

Reorganizando el resultado anterior, finalmente tenemos la fórmula para estimar el valor presente de la perpetuidad:

$$PV(\text{perpetuidad}) = \frac{C}{R} \dots \quad (3)$$

Aquí hay un ejemplo. John planea donar € 3,000 por año a su alma mater para tener una fiesta de bienvenida para los próximos estudiantes de MBA al final del año para siempre. Si la tasa de descuento anual es del 2.5% y la primera fiesta se realizará al final del primer año, ¿cuánto debería donar hoy? Al aplicar la fórmula anterior, la respuesta es € 120,000:

```
>>> 3000/0.025
120000.0
```

5.3 VALOR PRESENTE DE UNA PERPETUIDAD CRECIENTE

Suponga que el primer flujo de efectivo es C y que los siguientes flujos de efectivo disfrutan de una tasa de crecimiento constante de g ; vea la siguiente línea de tiempo y flujos de efectivo:

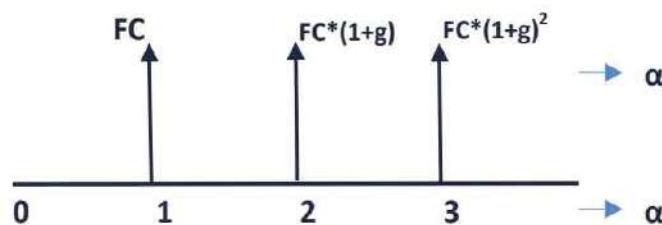


Figura 5.4. Diagrama flujo de efectivo de una perpetuidad creciente

Si la tasa de descuento es R , entonces la fórmula para estimar el valor presente de una perpetuidad creciente tiene la siguiente forma:

$$PV(\text{perpetuidad creciente}) = \frac{C}{R-g} \dots \dots \quad (4)$$

Nuevamente, las frecuencias de C , R y g deben ser consistentes, es decir, tener las mismas frecuencias. Para el ejemplo anterior de la donación a la fiesta de bienvenida de John a los estudiantes del MBA, el costo de € 3,000 necesarios cada año se basa en una inflación cero. Suponga que la inflación anual es del 1%, ¿cuánto tiene que donar hoy? La cantidad necesaria es:

```
>>> 3000/(0.025-0.01)
199999.9999999997
```

Hoy necesita donar € 200 mil.

Para una perpetuidad, si el primer flujo de efectivo ocurre al final del período k , tenemos la siguiente fórmula:

$$PV(\text{perpetuidad, 1er flujo al final del kesimo periodo}) = \frac{1}{(1+R)^{k-1}} \frac{C}{R-g} \quad (5)$$

Obviamente, cuando el primer flujo de efectivo ocurre al final del primer período, la ecuación (5) se convierte en la ecuación (3).

5.4 VALOR PRESENTE Y FUTURO DE UNA ANUALIDAD

Una anualidad se define como los mismos flujos de efectivo en los mismos intervalos durante n períodos. Si el primer flujo de efectivo ocurre al final del primer período, el valor presente de una anualidad se estima mediante la siguiente fórmula:

$$PV(\text{anualidad}) = \frac{C}{R} \left[1 - \frac{1}{(1+R)^n} \right] \quad (6)$$

Aquí, C es un flujo de efectivo recursivo que ocurre al final de cada período, R es la tasa de descuento del período y n es el número de períodos. La ecuación (5) es más compleja que las otras ecuaciones. Sin embargo, con un poco de imaginación, la ecuación (6) podría derivarse combinando las ecuaciones (2) y (3).

Para estimar el valor futuro de la anualidad, tenemos la siguiente fórmula:

$$FV(\text{anualidad}) = \frac{C}{R} [(1 + R)^n - 1] J \quad (7)$$

Conceptualmente, podríamos ver la ecuación (7) como la combinación de las ecuaciones (6) y (1). En las fórmulas anteriores relacionadas con la perpetuidad o anualidad, se supone que todos los flujos de efectivo ocurren al final de los períodos. En una anualidad o perpetuidad, cuando los flujos de efectivo ocurren al comienzo de cada período de tiempo, se denominan anualidad o perpetuidad anticipada. Hay dos formas de calcular sus valores presentes.

Para el primer método, el último valor de entrada en `npf.pv(rate, nper, pmt[, fv, when])` tomará un valor de uno. Suponga que la tasa de descuento es del 1% anual. El flujo de caja anual es de € 20 por los próximos 10 años. El primer flujo de efectivo se pagará hoy. ¿Cuál es el valor presente de esos flujos de efectivo? El resultado se muestra aquí:

```
import numpy as np
import numpy_financial as npf
npf.pv(0.01, 10, 20, 0, 1)
```

Salida

-191.32035152017377

Tenga en cuenta que el formato de entrada para la función `npf.pv(rate, nper, pmt[, fv, when])`. El valor predeterminado de la última variable llamada `when` es cero, es decir, al final del período. Cuando la variable llamada `when` toma un valor de uno, significa que es una anualidad anticipada.

Para el segundo método, se podrían aplicar las siguientes fórmulas:

$$PV(\text{anualidad vencida}) = \frac{C}{R} [1 - \frac{1}{(1+R)^n}] * (1+R)^n \quad (8)$$

$$FV(\text{anualidad}) = \frac{C}{R} [(1+R)^n - 1] * (1+R)^n \quad (9)$$

Aquí está la metodología: trate la anualidad vencida como anualidad normal, luego multiplique el resultado por $(1 + R)$. La aplicación se muestra aquí:

```
>>>import numpy.lib.financial as fin
>>> fin.pv(0.01,10,20,0)*(1+0.01)
-191.3203515201738
```

5.5 CÁLCULO DE LOS PAGOS DE UN PRÉSTAMO Y LA TABLA DE AMORTIZACIÓN EN PYTHON

Supongamos que deseas comprar un automóvil por € 15000. Solo tienes € 5000. Un banco te ofrece € 10000 al 7% de interés efectivo anual que debes devolver en 5 años. ¿Cuál es tu pago anual por el automóvil? Y ¿la tabla de amortización?

Esto es lo que sabemos

Precio del automóvil € 15000

Monto del préstamo € 10000

Tasas de interés 7% anual

Duración 5 años

Pagos - Necesitamos resolver esto

Antes de construir la tabla de amortización, debemos calcular el pago anual de nuestro préstamo. Para calcular esto, necesitamos seguir el siguiente proceso:

Importar librerías

```
import pandas as pd
import numpy as np
import numpy_financial as npf
```

A continuación, configuramos las variables conocidas y calculamos la cuota anual.

```
car_loan = 10000
interest = 0.07
years = 5
car_payments = npf.pmt(rate = interest, nper = years, pv = -car_loan)
print(car_payments)
```

Salida

```
2438.9069444137394
```

Nuestro préstamo de automóvil nos costará € 2438 en pagos anuales. Podemos crear una tabla de programación de pagos y ver como el saldo del préstamo se reducirá a cero al final del quinto año.

A continuación, creamos un DataFrame vacío para contener nuestros valores anuales. Llenaremos esta tabla usando un bucle for.

```
# Crear una tabla de 5 filas y 6 columnas llenas de ceros
loan_table = np.zeros((5,6))

# Convertirlo en un DataFrame
loan_table = pd.DataFrame(loan_table)

# Asignamos los nombres de las columnas
loan_table.columns = ["Año", "Saldo_Inicial", "Pago", "Intereses","Principal",
"Saldo_Final"]

print(loan_table)
```

Salida

Año	Saldo_Inicial	Pago	Intereses	Principal	Saldo_Final
0 0.0	0.0	0.0	0.0	0.0	0.0
1 0.0	0.0	0.0	0.0	0.0	0.0
2 0.0	0.0	0.0	0.0	0.0	0.0
3 0.0	0.0	0.0	0.0	0.0	0.0
4 0.0	0.0	0.0	0.0	0.0	0.0

Ahora que tenemos una tabla vacía, la debemos llenar con los respectivos cálculos. Necesitamos calcular la primera fila manualmente y luego podemos llenar el resto usando un bucle for usando los primeros valores. Esto es similar a Excel, donde necesitamos calcular la primera fila manualmente y luego arrastrar hacia abajo para copiar la fórmula y llenar las columnas restantes.

```
# La fila 0 y la columna 0 es nuestro año 1.

# use iloc[] para localizarlo
loan_table.iloc[0,0] = 1

# Saldo inicial es el monto del préstamo del automóvil
loan_table.iloc[0,1] = car_loan

# Los pagos del automóvil son los mismos que calculamos anteriormente.
loan_table.iloc[0,2] = car_payments

# El pago inicial es el monto del préstamo por los intereses
loan_table.iloc[0,3] = car_loan * interest

# El principal es el pago de la cuota menos los intereses
loan_table.iloc[0,4] = car_payments - (car_loan * interest)

# El saldo final es el saldo inicial menos el principal
loan_table.iloc[0,5] = car_loan - (car_payments - (car_loan * interest))
```

Una vez que tenemos la primera fila llena, ahora podemos ejecutar el ciclo for para calcular todos los demás valores.

```

# Nuestro bucle se ejecutará desde la fila 1 a la 4

for i in range(1,5):

    # La primera fila es el año
    loan_table.iloc[i,0] = i + 1

    # El saldo inicial es el saldo final de años anteriores
    loan_table.iloc[i,1] = loan_table.iloc[(i-1), 5]

    # Los pagos corresponden a la cuota.
    loan_table.iloc[i,2] = car_payments

    # El interés anual es el saldo inicial * tasa de interes

    loan_table.iloc[i,3] = loan_table.iloc[i,1] * interest

    # El principal es la cuota menos los intereses
    loan_table.iloc[i,4] = car_payments-(loan_table.iloc[i,1] * interest)

    # El saldo final es el saldo inicial menos el principal
    loan_table.iloc[i,5] = loan_table.iloc[i,1] - (car_payments - (loan_table.
    iloc[i,1] * interest))

    # Queremos redondear todos los valores a 2 posiciones decimales.
    loan_table = loan_table.round(2)

print(loan_table)

```

Salida

	Año	Saldo_Inicial	Pago	Intereses	Principal	Saldo_Final
0	1.0	10000.00	2438.91	700.00	1738.91	8261.09
1	2.0	8261.09	2438.91	578.28	1860.63	6400.46
2	3.0	6400.46	2438.91	448.03	1990.87	4409.58
3	4.0	4409.58	2438.91	308.67	2130.24	2279.35
4	5.0	2279.35	2438.91	159.55	2279.35	-0.00

Y ahí lo tiene, acabamos de crear nuestra tabla de amortización en Python. Puede parecer demasiada escritura cuando lo comparas con Excel (y lo es). Pero el verdadero poder de la programación no está en un cálculo tan simple (porque en este Excel puede ser mejor). Sin embargo, esto es muy útil cuando intenta ejecutar un bucle for sobre millones de elementos. Python tiene la capacidad de realizar millones de cálculos en una fracción de segundo.

5.6 DEFINICIÓN DE VPN Y REGLA DE VPN

El valor presente neto (VPN) se define mediante la siguiente fórmula:

$$VPN = VP(Ingresos) - VP(Egresos) \quad (10)$$

A continuación se presenta un ejemplo. La inversión inicial es de € 100. Las entradas de efectivo en los próximos cinco años son € 50, € 60, € 70, € 100 y € 20, a partir del primer año. Si la tasa de descuento es 11.2%, ¿cuál es el valor de VPN del proyecto? Dado que solo hay seis flujos de efectivo involucrados, nosotros podríamos hacer el cálculo manualmente:

```
>>> r=0.112
>>> -100+50/(1+r)+60/(1+r)**2+70/(1+r)**3+100/(1+r)**4+20/(1+r)**5
121.55722687966407
```

Usando la función `npv(rate, values)`, el proceso de estimación podría ser simplificado dramáticamente:

```
import numpy as np
import numpy_financial as npf
cashflows=[-100,50,60,70,100,20]
npf.npv(0.112, cashflows)
121.55722687966407
```

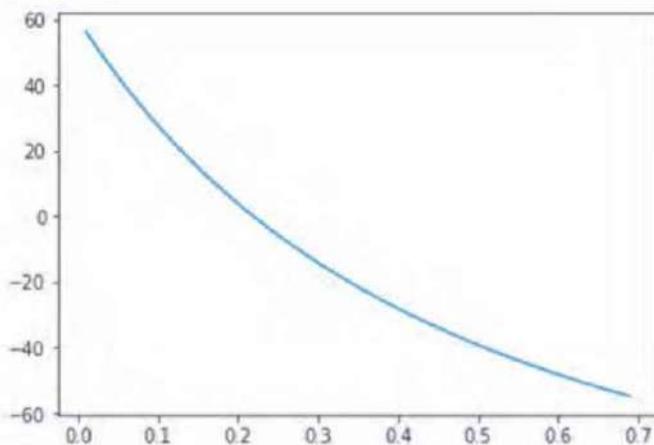
Según el resultado anterior, el VPN de este proyecto es de € 121.56. El VPN está correlacionado negativamente con la tasa de descuento. La razón es que cuando la tasa de descuento aumenta, el valor presente de los flujos de efectivo futuros disminuirá más que los flujos de efectivo actuales o los primeros. El perfil de VPN describe la relación entre el VPN y la tasa de descuento como se muestra en el siguiente gráfico. El eje y representa el VPN, mientras que el eje x representa la tasa de descuento:

```
import numpy as np
import numpy_financial as npf
from matplotlib.pyplot import *
cashflows=[-120,50,60,70]
```

```
rate=[]
npv =[]

for i in range(1,70):
    rate.append(0.01*i)
    npv.append(npf.npv(0.01*i,cashflows))
plot(rate,npv)
show()
```

Salida



Para estimar el VPN de un proyecto, podríamos llamar a la función npv () contenida en Numpy Financial; ver el siguiente código:

```
import numpy_financial as npf
cashflows=[-100,50,60,70]
rate=0.1
npv=npf.npv(rate,cashflows)
round(npv,2)
```

Salida

```
47.63
```

La función npf.npv () estima los valores actuales para un conjunto dado de flujos de efectivo. La primera variable de entrada es la tasa de descuento, mientras que

la segunda entrada es una matriz de flujos de efectivo. Tenga en cuenta que el primer flujo de efectivo en esta matriz de flujo de efectivo ocurre en el momento cero.

La regla del VPN se da aquí:

SI VPN > 0 Se acepta

SI VPN < 0 Se rechaza

5.7 DEFINICIÓN DE TIR Y REGLA DE LA TIR

La tasa interna de rendimiento (TIR) se define como la tasa de descuento que hace que el VPN sea igual a cero. Supongamos que invertimos € 100 hoy y los flujos de efectivo futuros serán de € 30, € 40, € 40 y € 50 por los próximos cuatro años. Suponiendo que todos los flujos de efectivo ocurran al final del año, ¿cuál es la TIR de esta inversión? En el siguiente programa, se aplica la función npf.irr (values):

```
import numpy_financial as npf  
  
cashflows=[-100,30,40,40,50]  
npf.irr(cashflows)  
0.2001879105140867
```

Podríamos verificar si tal tasa hace que el VPN sea igual a cero. Si el VPN es cero, el 20.02% es de hecho una TIR:

```
>>> r=npf.irr(cashflows)  
>>> npf.npv(r,cashflows)  
1.7763568394002505e-14      #Este resultado es prácticamente cero
```

Para un proyecto, la regla TIR se da aquí:

SI TIR > R_c Se acepta

SI TIR < R_c Se rechaza

Aquí, R_c es el costo de capital.

Veamos la siguiente oportunidad de inversión. La inversión inicial es de € 100 hoy y € 50 el próximo año. Las entradas de efectivo para los próximos cinco años serán de € 50, € 70, € 100, € 90 y € 20. Si el costo de capital es del 10%, ¿deberíamos tomar el proyecto? La línea de tiempo y los flujos de efectivo correspondientes se muestran en la figura 5.5:

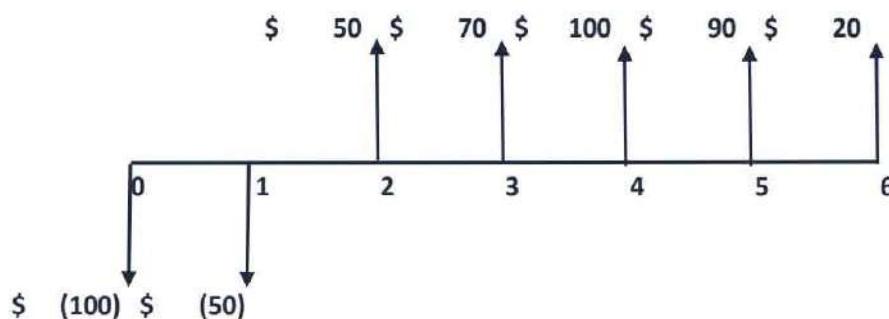


Figura 5.5. Diagrama flujo de efectivo proyecto de inversión

El código en Python se presenta a continuación:

```
import numpy_financial as npf

cashflows=[-100,-50,50,70,100,90,20]
npf.irr(cashflows)
```

Salida

```
0.25949919326073245
```

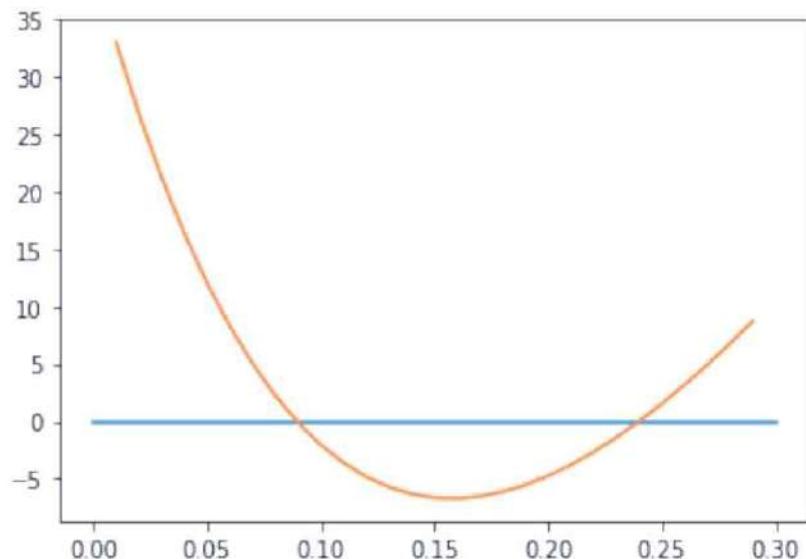
Dado que la TIR es del 25.9%, más alta que el costo de capital del 10%, debemos aceptar el proyecto según la regla de la TIR. Para proyectos con múltiples TIR, no podemos aplicar la regla de la TIR. Cuando los flujos de efectivo cambian de dirección más de una vez, podríamos tener TIR múltiples. Suponga que nuestros flujos de efectivo fueran 504, -432, -432, -432 y 832, a partir de hoy. El siguiente código nos permitirá graficar la presencia de dos TIR, veamos:

```
import matplotlib.pyplot as plt
cashflows=[504,-432,-432,-432,832]
rate=[]
npv=[]
```

```
x=[0,0.3]
y=[0,0]
for i in range(1,30):
    rate.append(0.01*i)
    npv.append(npf.npv(0.01*i,cashflows))

plt.plot(x,y),plt.plot(rate,npv)
plt.show()
```

Salida



En el gráfico anterior se puede observar que el valor presente neto es igual a cero en dos puntos, estos corresponde a dos TIR. Esto se presenta porque la dirección de los flujos de efectivo cambian dos veces. Aquí no se podría aplicar el criterio de decisión relacionado con la TIR.

6

EXTRAER Y CALCULAR INDICADORES FINANCIEROS CLAVES CON PYTHON

¿Se imagina presionar un botón y en segundos tener indicadores financieros claves como el ROE y el ROA para múltiples empresas? En este capítulo, vamos a hacer exactamente eso. Construiremos un script de Python que nos permitirá comparar indicadores financieros para un grupo de empresas.

El resultado será un maravilloso Pandas DataFrame que nos brindará información muy valiosa sobre diferentes indicadores financieros y otras métricas claves para un conjunto de empresas.

Este capítulo se dividirá en tres partes diferentes. En la primera, obtendremos todos los indicadores financieros de una API (Interfaz de Programación de Aplicaciones) y aprenderemos por qué puede ser mejor construir nuestros propios indicadores en lugar de confiar en los proporcionados por la API. En la segunda parte, construiremos cada uno de los indicadores por nosotros mismos obteniendo los datos del Balance General y del Estado de Resultados a través de una API. También graficaremos los indicadores usando Matplotlib. En la tercera parte calcularemos diversos indicadores y los exportaremos a un archivo en Excel.

6.1 PYTHON PARA FINANZAS ES NUESTRO MEJOR ALIADO

Los analistas financieros que buscan oportunidades de inversión en acciones pueden pasar horas y horas analizando estados financieros y comparando empresas. Para cada empresa, hay algunas métricas clave que vale la pena analizar. Por ejemplo:

- ▀ Al analizar el precio y las ganancias, podemos tener una idea de si la empresa cotiza con descuento en comparación con empresas comparables
- ▀ Podemos ver cómo las empresas están gestionando sus niveles de deuda en comparación con empresas similares
- ▀ O, en cambio, podríamos estar interesados en identificar empresas con mayores rendimientos por dividendos

Obtener todos estos indicadores para una sola empresa puede llevar algunas horas de trabajo e investigación. Imagínese tener que hacerlo para un grupo de empresas. Afortunadamente tenemos Python que hace el trabajo sucio por nosotros.

6.2 CONSEGUIR DATOS E INDICADORES FINANCIEROS CLAVES CON PYTHON

Para obtener indicadores financieros, usaremos la API financiera *fmcloud.io*. Ofrece algunas llamadas gratuitas solo por registrarse.

Uno de los aspectos clave a tener en cuenta al realizar un análisis financiero es la importancia de contar con datos comparables. Un número, es solo un número. No nos da ninguna idea a menos que podamos compararlo con otra cosa. Es por eso que siempre debemos analizar las cifras financieras de una empresa y compararlas con períodos anteriores o empresas comparables.

Las empresas comparables son empresas que operan en el mismo sector y tienen una capitalización de mercado muy similar. Para nuestro análisis de razones financieras, trabajaremos con empresas tecnológicas que tengan una capitalización de mercado superior a € 100 mil millones.

Realizaremos una solicitud a la API *fmcloud.io* para obtener los tickers de las empresas que operan en el sector tecnológico. Tenga en cuenta que en el siguiente código, pasamos el nombre del sector que nos interesa como parámetro de URL. Luego, analizamos la respuesta para extraer cada uno de los tickers de la empresa. Finalmente, guardamos el ticker de cada una de las empresas en una lista de Python:

```
import requests
import pandas as pd

import requests

demo= 'digite aquí su api key'

companies = requests.get(f'https://fmcloud.io/api/v3/stock-screener?sector=tech')
```

```
nology&marketCapMoreThan=100000000000&limit=100&apikey={demo}'')
companies = companies.json()

technological_companies = []

for item in companies:
    technological_companies.append(item['symbol'])
print(technological_companies)
```

Salida

```
['AAPL', 'MSF.BR', 'MSFT', 'TSM', 'NVDA', 'INTC', 'ASML', 'INCO.BR', 'ADBE', 'CRM', 'SHOP.TO', 'ASML.AS', 'CIS.BR', 'AVGO', 'CSCO', 'ORCL', 'SHOP', 'ACN', 'QCOM', 'TXN', 'SAP', 'SNE', 'SQ', 'INTU', 'NOW', 'UBER', 'AMD', 'AMAT', 'IBM', 'IBMA.BR']
```

Ahora que tenemos todos los tickers en una lista de Python, podemos pasar al siguiente paso y recuperar los indicadores financieros. Primero, recorremos cada una de las empresas de nuestra lista. Luego, extraemos y almacenamos cada uno de los indicadores de la empresa en un diccionario de Python:

```
metrics = {}
for item in technological_companies:
    try:
        metrics[item] = {}
        keymetrics = requests.get(f'https://fmpcloud.io/api/v3/ratios/{item}?apikey={demo}')
        keymetrics = keymetrics.json()
        keymetrics[0]

        metrics[item]['date'] = keymetrics[0]['date']
        metrics[item]['currentratio'] = float(keymetrics[0]['currentRatio'])

        metrics[item]['debtToAssets'] = float(keymetrics[0]['debtRatio'])
        metrics[item]['debtToEquity'] = float(keymetrics[0]['debtEquityRatio'])
        metrics[item]['dividendYield'] = float(keymetrics[0]['dividendYield'])

        metrics[item]['interestCoverage'] = float(keymetrics[0]['interestCoverage'])

        metrics[item]['Gross_Profit_Margin'] = float(keymetrics[0]['grossProfitMargin'])

        metrics[item]['roe'] = float(keymetrics[0]['returnOnEquity'])

        metrics[item]['priceToSalesRatio'] = float(keymetrics[0]['priceSalesRatio'])
```

```

metrics[item]['price_to_book_Ratio'] = float(keymetrics[0]['priceToBookRatio'])
metrics[item]['priceEarningsRatio'] = float(keymetrics[0]['priceEarningsRatio'])
metrics[item]['return_on_assets'] = float(keymetrics[0]['returnOnAssets'])
except:
    pass

print(metrics)

```

Salida

```

('AAPL': {'date': '2020-09-26', 'currentratio': 1.163684481554577, 'debtToAssets': 0.798266842799239, 'debtToEquity': 3.957039440456609, 'dividendyield': 0.007053312328582797}, 'MSFT': {'date': '2020-06-30', 'currentratio': 2.51576545425480112, 'debtToAssets': 0.58736913132342898, 'debtToEquity': 1.54653145983976194, 'dividendyield': 0.0057782871818958, 'interestCoverage': 20.4693118006074873, 'gross_Profit_Margin': 0.6778100199279796, 'roe': 0.37429841763592103, 'priceToSalesRatio': 16.823604662930462, 'price_to_book_Ratio': 13.08440625818484987, 'priceEarningsRatio': 34.957410543725874, 'return_on_assets': 6.14696111326835065}, 'MSF.BR': {}, 'TSLA': {'date': '2018-12-31', 'currentratio': 2.666986428414248, 'debtToAssets': 0.20522471548147888, 'debtToEquity': 0.253224887458958, 'dividendyield': 0.20055818412152005, 'interestCoverage': 131.29073807829653, 'gross_Profit_Margin': 0.4827900313432763, 'roe': 0.21868211819190485, 'priceToSalesRatio': 0.9710390890219328, 'price_to_book_Ratio': 0.604832995035056, 'priceEarningsRatio': 2.765812412877124, 'return_on_assets': 0.1737324304058234}, 'NVDA': {'date': '2020-01-26', 'currentratio': 7.673766816143497, 'debtToAssets': 0.19517739168351143, 'debtToEquity': 0.4187971156927795, 'dividendyield': 0.000141145884968851, 'interestCoverage': 57.11538481538481, 'gross_Profit_Margin': 0.619893753034595, 'roe': 0.12291052114060935, 'priceToSalesRatio': 14.812958498566638, 'price_to_book_Ratio': 13.252838741395264, 'priceEarningsRatio': 57.842589689574862, 'return_on_assets': 0.16147848686110388}, 'CRM': {'date': '2020-01-31', 'currentratio': 1.0733115527113506, 'debtToAssets': 0.38531727315604253, 'debtToEquity': 0.8260555497997639}, 'ADBE': {'date': '2019-11-29', 'currentratio': 0.7949483201440421, 'debtToAssets': 0.4938257738559877, 'debtToEquity': 0.9717088684828426}, 'INCO.BR': {}, 'INTC': {'date': '2019-12-20', 'currentratio': 1.406224147467502, 'debtToAssets': 0.4311696112034909, 'debtToEquity': 0.7579932783064423, 'dividendyield': 0.01955310352343531, 'interestCoverage': 49.19834400817996, 'gross_Profit_Margin': 0.5855624261793926, 'roe': 0.27189104588307987, 'priceToSalesRatio': 3.9626501882059746, 'price_to_book_Ratio': 3.6723065272537627, 'priceEarningsRatio': 13.548656442417832, 'return_on_assets': 0.15417850526237145}, 'ASML': {'date': '2018-12-31', 'currentratio': 2.095899431307992, 'debtToAssets': 0.4132776434381183, 'debtToEquity': 0.70438364293743, 'dividendyield': 0.90002528956293052575}, 'CIS.BR': {}, 'TSCO': {'date': '2020-07-25', 'currentratio': 1.7201457265388285, 'debtToAssets': 0.6002213503737167, 'debtToEquity': 0.5018976793242844, 'dividendyield': 0.033678169349318946, 'interestCoverage': 23.00034188343188, 'gross_Profit_Margin': 0.642644165837498, 'roe': 0.29572784810126584, 'priceToSalesRatio': 3.62413529877286, 'price_to_book_Ratio': 4.71187046839443, 'priceEarningsRatio': 15.9331621187880956, 'return_on_assets': 0.118225042958121367}, 'ORCL': {'date': '2020-05-31', 'currentratio': 3.081385348837289, 'debtToAssets': 0.895487058334344, 'debtToEquity': 8.560074686592078, 'dividendyield': 0.0129352610723897, 'interestCoverage': 0.00651651531341865, 'roe': 0.0394869082259339, 'priceToSalesRatio': 4.648444800284771, 'price_to_book_Ratio': 15.015145453838588, 'priceEarningsRatio': 17.887881382417353, 'return_on_assets': 0.98779604636254059}, 'QCOM': {'date': '2020-09-27', 'currentratio': 2.1354935424354244, 'debtToAssets': 0.8292689779176265, 'debtToEquity': 4.857056384982722, 'dividendyield': 0.02016584113533556, 'interestCoverage': 3.5, 'gross_Profit_Margin': 0.8660205048839227, 'roe': 0.8353562613121479, 'priceToSalesRatio': 8.073173878713182, 'price_to_book_Ratio': 78.516184719117931, 'priceEarningsRatio': 77.492851895885307, 'return_on_assets': 0.146815848738}

```

Simple, ¿verdad? En unas pocas líneas de código hemos ahorrado horas y horas de investigación y recopilación de datos.

6.3 CONSOLIDAR INDICADORES FINANCIEROS EN UN DATAFRAME DE PANDAS

Ahora que hemos extraído los indicadores para cada una de las empresas, podemos llevarlos a un DataFrame (es una colección ordenada de columnas con nombres y tipos, parecido a una tabla de base de datos, donde una sola fila representa un único caso (ejemplo) y las columnas representan atributos particulares) de Pandas. Esto será particularmente útil para comparar los números y para realizar operaciones adicionales con los datos, como el cálculo de la media o su representación gráfica.

Para convertir un diccionario en un DataFrame de pandas, usamos el método de Pandas from_dict. Luego, transponemos el DataFrame.

```
metrics_df = pd.DataFrame.from_dict(metrics, orient='index')
metrics_df = metrics_df.T
metrics_df
```

Salida

	AAPL	MSFT	TSM	NVDA	CRM	ADBE	INTC	ASML	CSCO	ORCL	QCOM	ASMLAS
date	2020-09-26	2020-06-30	2018-12-31	2020-01-26	2020-01-31	2019-11-29	2019-12-28	2018-12-31	2020-07-25	2020-05-31	2020-09-27	2019-12-31
currentratio	1.3636	2.51577	2.66699	7.67377	1.07531	0.79294	1.40022	2.6959	1.72015	3.0314	2.13549	2.55041
debtToAssets	0.798267	0.607369	0.205225	0.298178	0.385317	0.492826	0.43117	0.413278	0.600224	0.895407	0.829269	0.42264
debtToEquity	3.96794	1.54692	0.258322	0.418797	0.626856	0.971709	0.757993	0.704384	1.5014	8.56087	4.85717	0.732022
dividendYield	0.00705333	0.00977882	0.208558	0.00241146	NaN	NaN	0.0195531	-0.0002529	0.0336702	0.0169339	0.0201868	0.0118658
interestCoverage	NaN	20.4693	131.291	57.1154	NaN	NaN	49.1984	NaN	23.8803	6.04662	9.5	-107.668
Gross_Profit_Margin	NaN	0.67781	0.482791	0.619894	NaN	NaN	0.585562	NaN	0.642644	0.796816	0.606689	0.41456
roe	NaN	0.374298	0.216682	0.229105	NaN	NaN	0.271031	NaN	0.295728	0.839407	0.855356	0.186
priceToSalesRatio	NaN	10.8236	0.97364	14.813	NaN	NaN	3.96265	NaN	3.62416	4.84044	6.07317	9.45212
price_to_book_Ratio	NaN	13.0644	0.004834	13.252	NaN	NaN	3.67211	NaN	4.71188	15.0151	23.5162	8.05108
priceEarningsRatio	NaN	34.9571	2.76581	57.8426	NaN	NaN	13.5487	NaN	15.9332	17.8678	27.4929	43.2854
return_on_assets	NaN	0.146981	0.173732	0.161478	NaN	NaN	0.154171	NaN	0.118225	0.087796	0.146036	0.107389

Con unas pocas líneas de código Python, tenemos un increíble DataFrame de Pandas que nos permite comparar las empresas entre sí. Desafortunadamente, la API no devuelve todos los indicadores para cada una de las empresas. Por ejemplo, aparecen algunos valores de NaN para la empresa Salesforce (CRM). Esa es una de las razones por las que se recomienda crear el propio código para calcular las diferentes razones financieras.

La otra razón es estar seguros de que tenemos los datos más actualizados. En los datos que obtuvimos de la API, algunas de las razones utilizan la capitalización de mercado del día en que se publicaron los resultados financieros. Por lo tanto, las razones que utilizan la capitalización de mercado pueden estar desactualizadas si el precio ha cambiado de manera significativa recientemente.

A continuación, se mostrará cómo calcular las mismas razones utilizando el balance general, el estado de resultados y los datos del mercado. De esta forma, tendremos las métricas más actualizadas. Además, calcularemos la media del sector para cada uno de los indicadores y las graficaremos.

6.4 CALCULAR INDICADORES FINANCIEROS CON PYTHON

En la sección anterior, obtuvimos indicadores financieros de diferentes empresas utilizando una API. Como vimos, los datos que obtuvimos estaban incompletos o no estaban actualizados para la mayoría de las empresas. Para corregir esto, vamos a calcular las métricas financieras más actualizadas usando Python.

A continuación, se muestran algunos de los indicadores que calcularemos:

- Razón circulante = Activo corriente / Pasivo corriente
- Deuda a activos = deuda total / activos totales
- Deuda a capital = Pasivo total / Capital contable total
- Rendimiento de dividendos = Dividendo anual / Precio de la acción
- Cobertura de intereses = EBIT / Gastos por intereses
- Margen de utilidad bruta = (Ventas netas – Costo de ventas) / Ventas netas
- ROE = Utilidad neta / Valor contable promedio del capital ordinario
- Precio a ventas = Capitalización de mercado / Ventas
- Precio a ganancias = Capitalización de mercado / Utilidad neta
- ROA = Utilidad neta / Activos totales

Como en la sección anterior, vamos a calcular métricas financieras para un grupo de empresas comparables. Es decir, empresas que operan en el mismo sector y de tamaño similar. En esta sección, calcularemos métricas financieras para empresas que operan en el sector tecnológico.

Lo primero que vamos a hacer es obtener una lista de estas empresas. Para hacer eso, usaremos la API financiera llamada API fmpcloud.io.

```
import requests
import pandas as pd
import requests

demo= 'aquí digite su api key'

# Pasar sector y capitalización de mercado como argumentos para limitar los resultados
companies = requests.get(f'https://fmpcloud.io/api/v3/stock-screener?sector=technology&marketCapMoreThan=100000000000&limit=100&apikey={demo}')
companies = companies.json()

technological_companies = []

for item in companies:
    technological_companies.append(item['symbol'])
print(technological_companies)
```

Salida

```
[‘AAPL’, ‘MSF.BR’, ‘MSFT’, ‘TSM’, ‘NVDA’, ‘INTC’, ‘ASML’, ‘INCO.BR’, ‘ADBE’, ‘CRM’, ‘SHOP.TO’, ‘ASML.AS’, ‘CIS.BR’, ‘AVGO’, ‘CSCO’, ‘ORCL’, ‘SHOP’, ‘ACN’, ‘QCOM’, ‘TXN’, ‘SAP’, ‘SNE’, ‘SQ’, ‘INTU’, ‘NOW’, ‘UBER’, ‘AMD’, ‘AMAT’, ‘IBM’, ‘IBMA.BR’]
```

A continuación, vamos a extraer el último balance y estado de resultados anual de cada una de las empresas. Estas son las cifras que usaremos para calcular los indicadores financieros más actualizados.

NOTA

Alternativamente, se pueden obtener los estados financieros con corte trimestral, estos puede estar más actualizados que el último informe anual de la empresa. Sin embargo, para obtener proporciones anualizadas, es necesario multiplicar los elementos del estado de resultados (es decir, los ingresos) por 4. Es posible que este enfoque no siempre le brinde los datos financieros más precisos. Por ejemplo, si una empresa tiene un trimestre en el que las ventas son significativamente más altas que en otros trimestres. En este ejemplo, trabajaremos con datos anuales.

Obtendremos la fecha del estado financiero de la misma API *fmpcloud.io* que usamos para recuperar nuestra lista de acciones. Si observa el siguiente código, verá que podemos obtener datos financieros para cada una de las empresas simplemente cambiando el argumento de la URL.

Además del estado de resultados y los datos del balance, también extraemos la capitalización de mercado más reciente de cada empresa. Esto garantizará que nuestros cálculos reflejen los datos de precios más actualizados. Esto es importante en las razones que utilizan la capitalización de mercado.

Como se muestra en el siguiente código, extraemos cada uno de los datos requeridos. Luego, calculamos los indicadores financieros aplicando la fórmula mostrada al inicio de esta sección. Finalmente, guardamos todos los indicadores en un diccionario de Python:

```
metrics= {}

for item in technological_companies:
    try:
        balancesheet = requests.get(f’https://fmpcloud.io/api/v3/balance-sheet-statement/{item}?&apikey={demo}’).json()
        incomestatement = requests.get(f’https://fmpcloud.io/api/v3/income-statement/{item}?&apikey={demo}’).json()
```

```
marketcap = requests.get(f'https://fmpcloud.io/api/v3/market-capitalization/{item}?apikey={demo}').json()
marketcap = marketcap[0]['marketCap']
companydata = requests.get(f'https://fmpcloud.io/api/v3/profile/{item}?apikey={demo}').json()
latest_Annual_Dividend = companydata[0]['lastDiv']
price = companydata[0]['price']

balancesheet[0]
current_ratio = balancesheet[0]['totalCurrentAssets'] / balancesheet[0]['totalCurrentLiabilities']
debt_to_assets = balancesheet[0]['totalDebt'] / balancesheet[0]['totalAssets']
debt_to_equity = balancesheet[0]['totalDebt'] / balancesheet[0]['totalStockholdersEquity']
dividend_yield = latest_Annual_Dividend / price
interest_coverage = incomestatemnt[0]['operatingIncome'] / incomestatemnt[0]['interestExpense']
gross_profit_margin = (incomestatemnt[0]['revenue'] - incomestatemnt[0]['costOfRevenue'])/incomestatemnt[0]['revenue']
ROE = (incomestatemnt[0]['netIncome'] / ((balancesheet[0]['totalStockholdersEquity']+ balancesheet[1]['totalStockholdersEquity'])/2))
price_to_sales = marketcap / (incomestatemnt[0]['revenue'])
price_to_earnings = marketcap / (incomestatemnt[0]['netIncome'])
price_to_book = marketcap / balancesheet[0]['totalStockholdersEquity']
ROA = (incomestatemnt[0]['netIncome']/balancesheet[0]['totalAssets'])

metrics[item] = {}
metrics[item]['ROA'] = ROA

metrics[item]['ROE'] = ROE
metrics[item]['Current Ratio'] = current_ratio
metrics[item]['Debt to Assets'] = debt_to_assets
metrics[item]['Debt to Equity'] = debt_to_equity
metrics[item]['Dividend Yield'] = (dividend_yield*100)
metrics[item]['Interest Coverage'] = interest_coverage
metrics[item]['Gross Profit Margin'] = gross_profit_margin
metrics[item]['Price to Sales'] = price_to_sales
metrics[item]['Price to Earnings'] = price_to_earnings
metrics[item]['Price to Book'] = price_to_book
except:
    pass

print(metrics)
```

Salida

```
{'MSFT': {'ROA': 0.1496111326835965, 'ROE': 0.48139779088510895, 'Current Ratio': 2.5157654542948115, 'Debt to Assets': 0.4243671684422092, 'Debt to Equity': 0.615558221883419, 'Dividend Yield': 1.2249128427408355, 'Interest Coverage': 20.33825707834813, 'Gross Profit Margin': 0.6778108199279796, 'Price to Sales': 11.452162598437618, 'Price to Earnings': 36.98721844593456, 'Price to Book': 11.844257337638824}, 'NVDA': {'ROA': 0.16147848586118308, 'ROE': 0.25853775322194373, 'Current Ratio': 0.7873766816143477, 'Debt to Assets': 0.11497008548657234, 'Debt to Equity': 0.1631432312720752, 'Dividend Yield': 0.116365466679144378, 'Interest Coverage': 54.73076918076923, 'Gross Profit Margin': 0.6158937530346595, 'Price to Sales': 30.78330488252427, 'Price to Earnings': 128.20498569384836, 'Price to Book': 27.529586659455916}, 'CRM': {'ROA': 0.0022856728222617278, 'ROE': 0.00509183776520592, 'Current Ratio': 1.0753115327113505, 'Debt to Assets': 0.05547291650108225, 'Debt to Equity': 0.6992464217295253, 'Dividend Yield': 0.0, 'Interest Coverage': 0.71565626506924096, 'Gross Profit Margin': 0.752318211720669, 'Price to Sales': 19.813822699318848, 'Price to Earnings': 1874.5134695238894, 'Price to Book': 6.97830367419212}, 'ADBE': {'ROA': 0.142153300790482797, 'ROE': 0.29674422761928265, 'Current Ratio': 0.7920001101010101, 'Debt to Assets': 0.13910542711825222, 'Debt to Equity': 0.3629712781858357, 'Dividend Yield': 0.0, 'Interest Coverage': 38.78772246746479, 'Gross Profit Margin': 0.6591266272024565, 'Price to Sales': 20.11597108409395, 'Price to Earnings': 76.8901475991552, 'Price to Book': 31.55293535565341}, 'INTC': {'ROA': 0.1541789159526297149, 'ROE': 0.273784356798669, 'Current Ratio': 1.0882241147407982, 'Debt to Assets': 0.2124241891535547, 'Debt to Equity': 0.3734402966816464, 'Dividend Yield': 1.9198623617868293, 'Interest Coverage': 45.061349893251536, 'Gross Profit Margin': 0.58556242161703928, 'Price to Sales': 2.708062592685431, 'Price to Earnings': 9.234491153855871, 'Price to Book': 2.582813799814541}, 'CSCO': {'ROA': 0.11822584296121357, 'ROE': 0.3137178106533956, 'Current Ratio': 1.716145270536155, 'Debt to Assets': 0.15374315684693785, 'Debt to Equity': 0.3645727848101246, 'Dividend Yield': 0.48275862068966, 'Interest Coverage': 23.38285128205128, 'Gross Profit Margin': 0.6426441054327498, 'Price to Sales': 9.80475791564894, 'Price to Earnings': 15.84224719101126, 'Price to Book': 4.684993678986076}, 'ORCL': {'ROA': 0.087756644636254959, 'ROE': 0.5986591452789589, 'Current Ratio': 0.031805348837209, 'Debt to Assets': 0.6200731128397928, 'Debt to Equity': 0.928441270788851, 'Dividend Yield': 1.784691594923227, 'Interest Coverage': 6.905419333838586, 'Gross Profit Margin': 0.7964158083341665, 'Price to Sales': 4.34263950055835, 'Price to Earnings': 17.510788357178897, 'Price to Book': 14.698278151399703}, 'QCOM': {'ROA': 0.14003584873886144, 'ROE': 0.9862952849808638, 'Current Ratio': 2.135493542454234, 'Debt to Assets': 0.441109836318631, 'Debt to Equity': 2.5877986279743295, 'Dividend Yield': 1.7228178365547645, 'Interest Coverage': 10.359385448504983, 'Gross Profit Margin': 0.6066894848485227, 'Price to Sales': 7.19964557392376, 'Price to Earnings': 3.25923116275480575, 'Price to Book': 27.87894179893928}, 'AVGO': {'ROA': 0.04035974181017803, 'ROE': 0.18558540078056053, 'Current Ratio': 1.437454783588229, 'Debt to Assets': 0.4863763849563658, 'Debt to Equity': 1.7161861994396563, 'Dividend Yield': 0.2973161202649913, 'Interest Coverage': 2.38584155124663375, 'Gross Profit Margin': 0.5550359946610552, 'Price to Sales': 7.10772264017347, 'Price to Earnings': 58.962261380323055, 'Price to Book': 6.439725752776551}, 'ACN': {'ROA': 0.1377588803199823, 'ROE': 0.3252276282626807, 'Current Ratio': 1.4027476677362115, 'Debt to Assets': 0.0016668717319643503, 'Debt
```

Genial, tenemos nuestros indicadores en un diccionario Python. Los datos están listos para llevarse a un DataFrame de Pandas. Esto nos permitirá comparar visualmente los indicadores financieros de cada una de las empresas con mucha facilidad.

Teniendo los indicadores listos en un diccionario Pandas, podemos llevarlos fácilmente a un DataFrame de Pandas usando el método Pandas from_dict. Además, también transpondremos el Pandas DataFrame y calcularemos la media:

```
metrics_df = pd.DataFrame.from_dict(metrics, orient='index')
metrics_df = metrics_df.T
metrics_df['mean'] = metrics_df.mean(axis=1)

metrics_df
```

A continuación se muestra el DataFrame de Pandas que contiene algunos indicadores financieros para empresas comparables. Al analizarlo, podemos comparar fácilmente cada una de las empresas entre sí y ver cuál puede ser una buena oportunidad de inversión.

Salida

	MSFT	NVDA	CRM	ADBE	INTC	CSCO	ORCL	QCOM	AVGO	ACN	TXN	SHOP
ROA	0.146961	0.161478	-0.002266	0.142154	0.154171	0.118225	0.087796	0.148036	0.040360	0.137757	0.278444	-0.035777
ROE	0.401398	0.259538	0.005092	0.296744	0.275784	0.313718	0.598659	0.946295	0.105565	0.325227	0.560527	-0.048895
Current Ratio	2.515765	7.673767	1.075312	0.792940	1.400224	1.720145	3.031395	2.135494	1.437455	1.401748	4.126707	8.675361
Debt to Assets	0.241687	0.114987	0.055473	0.199305	0.212424	0.153743	0.620073	0.441816	0.486376	0.001669	0.322067	0.000000
Debt to Equity	0.615558	0.163143	0.090246	0.392972	0.373440	0.384573	5.928441	2.587790	1.316188	0.003639	0.651510	0.000000
Dividend Yield	1.224913	0.118368	0.000000	0.000000	2.919062	3.448276	1.704092	1.722618	3.397518	1.366439	2.410732	0.000000
Interest Coverage	20.388267	54.730769	0.715863	20.787722	45.061350	23.282051	6.965414	10.390365	2.385042	179.637170	33.664706	-3.113628
Gross Profit Margin	0.677810	0.619694	0.752310	0.850266	0.585562	0.842644	0.796816	0.606689	0.855826	0.315296	0.637141	0.548510
Price to Sales	11.452163	30.783398	13.813823	20.315971	2.700883	3.603478	4.542640	7.199846	7.107722	3.563581	10.087012	71.092293
Price to Earnings	36.987218	120.204986	1874.513810	76.896146	9.234491	15.842247	17.510788	32.592316	58.962261	30.925604	28.917979	-698.703455
Price to Book	13.844267	27.539589	6.970304	21.552935	2.502834	4.664904	14.698878	27.878042	6.439726	9.290902	16.288481	37.203525

En ocasiones, es recomendable graficar los datos para obtener una mejor visualización. En la siguiente sección procederemos con la representación gráfica de los datos.

6.5 GRAFICAR INDICADORES FINANCIEROS CON PYTHON

Para visualizar nuestros datos, usaremos Matplotlib, que es muy fácil de usar para graficar en Python. Tenga en cuenta que no vamos a representar todos los indicadores juntos en un solo gráfico, ya que es posible que no se ajusten bien debido a la diferente escala de las métricas. En este ejemplo, graficaremos juntos las relaciones Precio / Ventas, Precio / Ganancias y Precio / Valor en libros:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
x1 = metrics_df.columns
# En la fila 8 tenemos datos de la relación precio a ventas
y1 = metrics_df.iloc[8,:]
plt.plot(x1, y1, label = "Price to Sales")

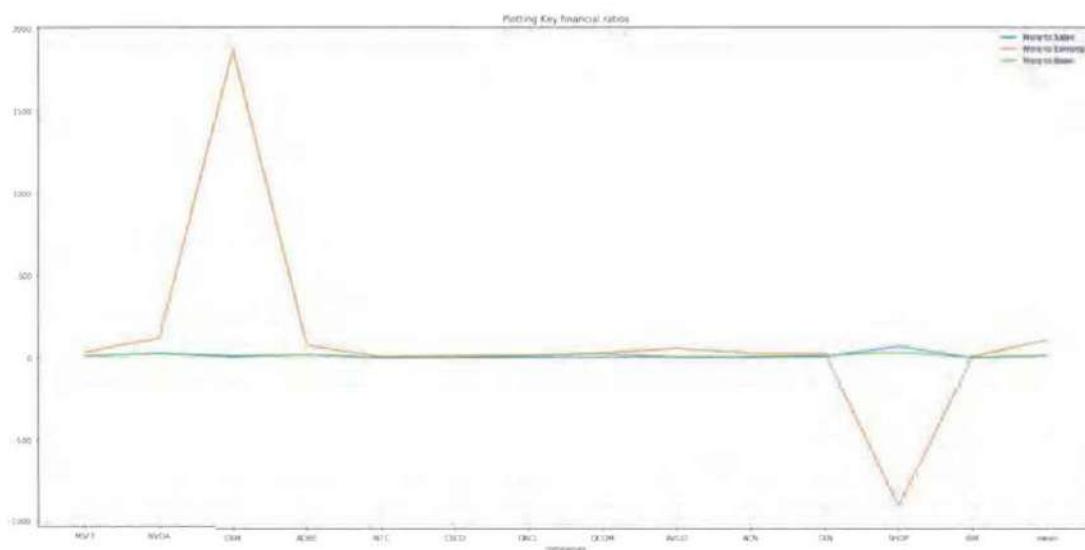
# Línea 2 puntos
x2 = metrics_df.columns
# En la fila 9 tenemos datos de la relación precio ganancia
y2 = metrics_df.iloc[9,:]
# Graficando la línea 2 puntos
```

```
plt.plot(x2, y2, label = "Price to Earnings")

# Línea 3 puntos
x3 = metrics_df.columns
# En la fila 10 tenemos datos de la relación precio valor en libros
y3 = metrics_df.iloc[10,:]
# Graficando la línea de 3 puntos
plt.plot(x3, y3, label = "Price to Book")

plt.xlabel('companies')
# Establecer la etiqueta del eje y
plt.ylabel('')
# Establecer el título de los ejes actuales
plt.title('Plotting Key financial ratios')
# Mostrar una leyenda en el gráfico
plt.legend()
# Mostrar una figura
plt.tight_layout()
plt.show()
```

Salida



Al observar el gráfico, vemos que la empresa Salesforce (**CRM**) tiene una relación precio / ganancia súper alta en comparación con otras compañías. Esto puede verse como una señal de que la empresa está sobrevalorada. A los precios mostrados, Salesforce se está negociando a más de 100 veces sus ganancias.

Además de todo lo que hemos hecho hasta ahora, también podríamos aplicar algunos filtros a nuestro DataFrame de Pandas para filtrar las empresas que cumplen con ciertos criterios. Por ejemplo, nos pueden interesar empresas con un ROE superior a 0,2 y un ratio de liquidez superior a 1,09. Podemos hacerlo muy fácilmente en Pandas aplicando las siguientes condiciones a nuestro DataFrame:

```
metrics_filter = metrics_df.T
metrics_filter[(metrics_filter['ROE'] > 0.2) & (metrics_filter['Current Ratio'] > 1.09)]
```

Salida

	ROA	ROE	Current Ratio	Debt to Assets	Debt to Equity	Dividend Yield	Interest Coverage	Gross Profit Margin	Price to Sales	Price to Earnings	Price to Book
MSFT	0.146961	0.401398	2.515765	0.241687	0.815558	1.224913	20.388267	0.677810	11.452163	36.987218	13.844257
NVDA	0.161478	0.259538	7.673767	0.114987	0.163143	0.118359	54.730769	0.619894	30.783398	120.204986	27.539589
INTC	0.154171	0.275784	1.400224	0.212424	0.373440	2.919062	45.081350	0.585562	2.700863	9.234491	2.502834
CSCO	0.118225	0.313718	1.720145	0.153743	0.384573	3.448278	23.282051	0.642644	3.603476	15.842247	4.684994
ORCL	0.087786	0.598659	3.031395	0.620073	5.928441	1.704002	8.985414	0.796816	4.542640	17.510788	14.698678
QCOM	0.146036	0.946295	2.135494	0.441816	2.587790	1.722618	10.390385	0.608889	7.199646	32.592316	27.878042
ACN	0.137757	0.325227	1.401748	0.001869	0.003639	1.366439	179.637176	0.315296	3.563581	30.925604	9.290902
TXN	0.276444	0.580527	4.126707	0.322087	0.651510	2.410732	33.664706	0.637141	10.087012	28.917979	16.288481
mean	0.110912	0.349296	2.846568	0.250994	1.194272	1.837049	30.958376	0.620133	14.432920	108.850257	14.919112

Como se puede observar, con solo unas pocas líneas de código Python, hemos podido calcular una cantidad considerable de métricas financieras para empresas comparables en el sector tecnológico. Para hacer esto manualmente, se habrían requerido días de trabajo.

6.6 EXPORTAR INDICADORES FINANCIEROS A UN ARCHIVO EN EXCEL

En esta sección, vamos a crear una herramienta de Python para obtener indicadores financieros de una empresa y exportarlos a un archivo en Excel. Pasaremos el ticker de la empresa que nos interesa y Python hará su magia para mostrarnos a continuación un Data Frame de Pandas convertido en un archivo de Excel.

El objetivo principal del código es obtener datos financieros de los últimos 5 años de una empresa. Luego, calcular algunos indicadores y finalmente exportarlos a un archivo de Excel. A continuación, se indican los pasos más importantes para lograrlo:

1. Solicite datos de la API financiera gratuita *financialmodelingprep*.
2. Cargue en una variable de Python los principales datos financieros solicitados a la API. Es decir, IS (Income Statement), BS (Balance Sheet), CF(Cash Flow) y principales ratios financieros
3. Cree un diccionario vacío para almacenar los valores utilizando el año respectivo como clave
4. Transforma el diccionario en un DataFrame de Pandas
5. Calcule algunos indicadores de crecimiento como nuevas columnas
6. Exportar a Excel

Algoritmo de Python para obtener indicadores financieros de la empresa:

```
import requests
import pandas as pd

pd.options.display.float_format = '{:,.2f}'.format

# Pasar el ticket de la empresa
company = 'AAPL'

api = 'digite su api key'

# Solicite datos financieros de la API y cargue en variables

IS = requests.get(f'https://financialmodelingprep.com/api/v3/income-statement/{company}?apikey={api}').json()
BS = requests.get(f'https://financialmodelingprep.com/api/v3/balance-sheet-statement/{company}?apikey={api}').json()
CF = requests.get(f'https://financialmodelingprep.com/api/v3/cash-flow-statement/{company}?apikey={api}').json()

Ratios = requests.get(f'https://financialmodelingprep.com/api/v3/ratios/{company}?apikey={api}').json()
key_Metrics = requests.get(f'https://financialmodelingprep.com/api/v3/key-metrics/{company}?apikey={api}').json()

profile = requests.get(f'https://financialmodelingprep.com/api/v3/profile/{company}?apikey={api}').json()

millions = 1000000
```

```
# Cree un diccionario vacío y agréguele las cifras financieras
financials = {}

dates = [2020,2019,2018,2017,2016]
for item in range(5):
    #print(dates[item])
    financials[dates[item]] = {}

#Indicadores claves

financials[dates[item]]['Mkt Cap'] = key_Metrics[item]['marketCap'] / millions
financials[dates[item]]['Debt to Equity'] = key_Metrics[item]['debtToEquity']
financials[dates[item]]['Debt to Assets'] = key_Metrics[item]['debtToAssets']
financials[dates[item]]['Revenue per Share'] = key_Metrics[item]['revenuePerShare']
financials[dates[item]]['NI per Share'] = key_Metrics[item]['netIncomePerShare']

financials[dates[item]]['Revenue'] = IS[item]['revenue'] / millions
financials[dates[item]]['Gross Profit'] = IS[item]['grossProfit'] / millions
financials[dates[item]]['R&D Expenses'] = IS[item]['researchAndDevelopmentExpenses'] / millions
financials[dates[item]]['Op Expenses'] = IS[item]['operatingExpenses'] / millions
financials[dates[item]]['Op Income'] = IS[item]['operatingIncome'] / millions
financials[dates[item]]['Net Income'] = IS[item]['netIncome'] / millions

financials[dates[item]]['Cash'] = BS[item]['cashAndCashEquivalents'] / millions
financials[dates[item]]['Inventory'] = BS[item]['inventory'] / millions
financials[dates[item]]['Cur Assets'] = BS[item]['totalCurrentAssets'] / millions
financials[dates[item]]['LT Assets'] = BS[item]['totalNonCurrentAssets'] / millions
financials[dates[item]]['Int Assets'] = BS[item]['intangibleAssets'] / millions
financials[dates[item]]['Total Assets'] = BS[item]['totalAssets'] / millions

financials[dates[item]]['Cur Liab'] = BS[item]['totalCurrentLiabilities'] / millions
financials[dates[item]]['LT Debt'] = BS[item]['longTermDebt'] / millions
financials[dates[item]]['LT Liab'] = BS[item]['totalNonCurrentLiabilities'] / millions
financials[dates[item]]['Total Liab'] = BS[item]['totalLiabilities'] / millions
financials[dates[item]]['SH Equity'] = BS[item]['totalStockholdersEquity'] / millions
```

```
financials[dates[item]]['CF Operations'] = CF[item]['netCashProvidedByOperatingActivities'] / millions
financials[dates[item]]['CF Investing'] = CF[item]['netCashUsedForInvestingActivities'] / millions
financials[dates[item]]['CF Financing'] = CF[item]['netCashUsedProvidedByFinancingActivities'] / millions
financials[dates[item]]['CAPEX'] = CF[item]['capitalExpenditure'] / millions
financials[dates[item]]['FCF'] = CF[item]['freeCashFlow'] / millions
financials[dates[item]]['Dividends Paid'] = CF[item]['dividendsPaid'] / millions

#Indicadores del estado de resultados
financials[dates[item]]['Gross Profit Margin'] = Ratios[item]['grossProfitMargin']
financials[dates[item]]['Op Margin'] = Ratios[item]['operatingProfitMargin']
financials[dates[item]]['Int Coverage'] = Ratios[item]['interestCoverage']
financials[dates[item]]['Net Profit Margin'] = Ratios[item]['netProfitMargin']
financials[dates[item]]['Dividend Yield'] = Ratios[item]['dividendYield']

#Indicadores del estado de situación financiera
financials[dates[item]]['Current Ratio'] = Ratios[item]['currentRatio']
financials[dates[item]]['Operating Cycle'] = Ratios[item]['operatingCycle']
financials[dates[item]]['Days of AP Outstanding'] = Ratios[item]['daysOfPayablesOutstanding']
financials[dates[item]]['Cash Conversion Cycle'] = Ratios[item]['cashConversionCycle']

#Indicadores de rentabilidad
financials[dates[item]]['ROA'] = Ratios[item]['returnOnAssets']
financials[dates[item]]['ROE'] = Ratios[item]['returnOnEquity']
financials[dates[item]]['ROCE'] = Ratios[item]['returnOnCapitalEmployed']
financials[dates[item]]['Dividend Yield'] = Ratios[item]['dividendYield']

#Indicadores de precio
financials[dates[item]]['PE'] = Ratios[item]['priceEarningsRatio']
financials[dates[item]]['PS'] = Ratios[item]['priceToSalesRatio']
financials[dates[item]]['PB'] = Ratios[item]['priceToBookRatio']
financials[dates[item]]['Price To FCF'] = Ratios[item]['priceToFreeCashFlowsRatio']
financials[dates[item]]['PEG'] = Ratios[item]['priceEarningsToGrowthRatio']
financials[dates[item]]['EPS'] = IS[item]['eps']
financials[dates[item]]['EPS'] = IS[item]['eps']
```

```
#Transforma el diccionario en un DataFrame de pandas
fundamentals = pd.DataFrame.from_dict(financials,orient='columns')

fundamentals['CAGR'] = (fundamentals[2020]/fundamentals[2016])**(1/5) - 1
fundamentals['2020 growth'] = (fundamentals[2020] - fundamentals[2019]) / fundamentals[2019]
fundamentals['2019 growth'] = (fundamentals[2019] - fundamentals[2018]) / fundamentals[2018]
fundamentals['2018 growth'] = (fundamentals[2018] - fundamentals[2017]) / fundamentals[2017]
fundamentals['2017 growth'] = (fundamentals[2017] - fundamentals[2016]) / fundamentals[2016]

#Exportar a Excel
fundamentals.to_excel('fundamentals.xlsx')

print(fundamentals)
```

Salida

	2020	2019	2018	2017	2016	CAGR	2020 growth	2019 growth	2018 growth	2017 growth
Mkt Cap	1,157,811.80	1,061,223.15	875,562.09	647,108.06	667,824.99	0.12	0.09	0.21	0.35	-0.03
Debt to Equity	2.74	2.41	1.80	1.51	1.43	0.14	0.14	0.34	0.19	0.05
Debt to Assets	0.73	0.71	0.64	0.60	0.59	0.04	0.04	0.10	0.07	0.02
Revenue per Share	13.99	13.28	10.91	9.80	10.09	0.07	0.05	0.22	0.11	-0.03
NI per Share	2.97	2.98	2.30	2.08	2.30	0.05	-0.00	0.29	0.11	-0.10
Revenue	260,174.00	265,595.00	229,234.00	215,639.00	233,715.00	0.02	-0.02	0.16	0.06	-0.08
Gross Profit	98,392.00	101,839.00	88,186.00	84,263.00	93,626.00	0.01	-0.03	0.15	0.05	-0.10
R&D Expenses	16,217.00	14,236.00	11,581.00	10,045.00	8,067.00	0.15	0.14	0.23	0.15	0.25
Op Expenses	34,462.00	30,941.00	26,842.00	24,239.00	22,396.00	0.09	0.11	0.15	0.11	0.08
Op Income	63,930.00	70,898.00	61,344.00	60,024.00	71,230.00	-0.02	-0.10	0.16	0.02	-0.16
Net Income	55,256.00	59,531.00	48,351.00	45,667.00	53,394.00	0.01	-0.07	0.23	0.06	-0.14
Cash	48,844.00	25,913.00	20,289.00	20,484.00	21,120.00	0.18	0.88	0.28	-0.01	-0.03
Inventory	4,106.00	3,956.00	4,855.00	2,132.00	2,349.00	0.12	0.04	-0.19	1.28	-0.09

Una vez que se ejecuta el script, Python crea un archivo de Excel que incluye todos los indicadores calculados. Para el ejemplo en cuestión el archivo se denomina *fundamentals*.

<input checked="" type="checkbox"/> fundamentals.xlsx	2/16/2021 7:34 PM	Microsoft Excel W...	7 KB
 Hola.py	2/12/2021 9:09 PM	Python File	1 KB

7

VALORACIÓN DE BONOS Y ACCIONES

Los bonos y las acciones son dos vehículos de inversión ampliamente utilizados. Por lo tanto, merecen un análisis sobre su modelación financiera en Python. Este capítulo cubrirá los siguientes temas:

- ▀ La estructura temporal de las tasas de interés
- ▀ Relación riesgo y rentabilidad. La duración.
- ▀ Evaluación de bonos y rendimiento hasta el vencimiento.
- ▀ Valoración de acciones

7.1 ESTRUCTURA TEMPORAL DE TASAS DE INTERÉS

La estructura temporal de las tasas de interés se define como la relación entre la tasa libre de riesgo y el tiempo. Una tasa libre de riesgo generalmente se define como la tasa de tesorería libre de incumplimiento. De diversas fuentes, podríamos obtener la estructura temporal actual de las tasas de interés. Por ejemplo, el 16/02/2021, del Departamento del Tesoro de los Estados Unidos: <https://www.treasury.gov/resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield> podríamos obtener la siguiente información:

Date	1 Mo	2 Mo	3 Mo	6 Mo	1 Yr	2 Yr	3 Yr	5 Yr	7 Yr	10 Yr	20 Yr	30 Yr
02/01/21	0.06	0.07	0.07	0.08	0.08	0.11	0.17	0.42	0.76	1.09	1.66	1.84
02/02/21	0.04	0.05	0.07	0.08	0.08	0.11	0.18	0.45	0.79	1.12	1.69	1.87
02/03/21	0.03	0.04	0.04	0.06	0.08	0.11	0.19	0.46	0.81	1.15	1.73	1.92
02/04/21	0.03	0.04	0.04	0.06	0.07	0.11	0.18	0.46	0.81	1.15	1.75	1.93
02/05/21	0.02	0.03	0.03	0.05	0.06	0.09	0.19	0.47	0.83	1.19	1.79	1.97
02/08/21	0.04	0.03	0.05	0.05	0.07	0.11	0.20	0.48	0.83	1.19	1.78	1.96
02/09/21	0.04	0.04	0.04	0.06	0.07	0.11	0.19	0.48	0.83	1.18	1.78	1.95
02/10/21	0.05	0.04	0.05	0.06	0.07	0.11	0.19	0.46	0.80	1.15	1.75	1.92
02/11/21	0.05	0.05	0.05	0.06	0.07	0.11	0.19	0.46	0.81	1.16	1.77	1.94
02/12/21	0.03	0.04	0.04	0.05	0.06	0.11	0.20	0.50	0.85	1.20	1.83	2.01
02/16/21	0.03	0.04	0.04	0.06	0.08	0.13	0.23	0.57	0.94	1.30	1.92	2.08

Tuesday Feb 16, 2021

Tabla 7.1. Tasas de interés, curva de rendimiento bonos del tesoro EEUU. Actualizado el 16 de febrero de 2021.

Según la información proporcionada en la tabla 7.1, tenemos el siguiente código para graficar la denominada curva de rendimiento:

```
from matplotlib.pyplot import *
time=[3/12,6/12,2,3,5,10,30]
rate=[0.04,0.06,0.13,0.23,0.57,1.30,2.08]
title("Estructura temporal de la tasa de interés ")
xlabel("Time ")
ylabel("Risk-free rate (%)")
plot(time,rate)
show()
```

Salida

El gráfico relacionado se da en la siguiente imagen:



La estructura de plazos de la pendiente ascendente significa que las tasas a largo plazo son más altas que las tasas a corto plazo.

En el mercado de los EEUU, hay una gran cantidad de emisiones que permiten diseñar una curva de rendimientos utilizando bonos cupón cero y emisiones con cupón. El resultado es una curva de tasas cero “teórica”. Es teórica puesto que adolece de dos imperfecciones: i) no existen bonos cupón cero para todos los plazos, ii) no existen bonos con cupón para todos los plazos. La falta de bonos con cupón se soluciona con un procedimiento de interpolación para generar la tasa cupón del plazo que falta y la falta de bonos cupón cero con la técnica del bootstrapping.

En Python, el módulo Pandas tiene una función llamada `.interpolate()` que nos permite interpolar esos valores que faltan para los bonos con cupón, veamos el siguiente ejemplo donde tenemos dos valores faltantes entre 2 y 6:

```
import pandas as pd
import numpy as np
x=pd.Series([1,2,np.nan,np.nan,6])
x.interpolate()
```

La salida relacionada se muestra aquí

```
0    1.000000
1    2.000000
2    3.333333
3    4.666667
4    6.000000
dtype: float64
```

Podríamos calcular manualmente esos valores faltantes. Primero, se estima un Δ :

$$\Delta = \frac{V2 - V1}{n}$$

Aquí, Δ es el valor incremental entre $v2$ (el valor final) y $v1$ (el valor inicial), y n es el número de elementos internos entre esos dos valores. El Δ para el caso anterior es $(6-2)/3=1.33333$. Por lo tanto, el siguiente valor será $v1 + \Delta = 2 + 1.33333 = 3.33333$.

Para el ejemplo anterior, relacionado con la estructura de plazos de las tasas de interés, de los años 6 a 9, no hay datos. El código y la salida se muestran aquí:

```
import pandas as pd
import numpy as np
nan=np.nan
x=pd.Series([0.57,nan,nan,nan,nan,2.08])
x.interpolate()
```

Salida

```
Out[25]: 0    0.57
         1    0.87
         2    1.17
         3    1.48
         4    1.78
         5    2.08
         dtype: float64
```

La estructura temporal de las tasas de interés es muy importante ya que sirve como punto de referencia para estimar el rendimiento al vencimiento (YTM) de los bonos corporativos. YTM es la rentabilidad que genera un bono si se conserva hasta su fecha de vencimiento. Técnicamente hablando, YTM es lo mismo que la Tasa Interna de Retorno (TIR). En la industria financiera, el diferencial, definido como la diferencia entre YTM de un bono corporativo sobre la tasa libre de riesgo, se usa para estimar la tasa de descuento para los bonos corporativos. El spread es una medida del riesgo de incumplimiento. Por lo tanto, debe estar estrechamente relacionado con la calificación crediticia de la empresa y del bono.

A continuación se utiliza un conjunto de datos que se encuentran en el archivo SpreadBasedOnCreditRating.pkl, esto para explicar la relación entre el diferencial predeterminado y la calificación crediticia. El siguiente programa recupera e imprime los datos. Se supone que el conjunto de datos está en el directorio c:/temp/:

```
import pandas as pd
spread=pd.read_pickle("c:/Users/spreadBasedOnCreditRating.pkl")
spread
```

Salida

	1	2	3	5	7	10	30
Rating							
Aaa/AAA	5.00	8.00	12.00	18.00	28.00	42.00	65.00
Aa1/AA+	10.00	18.00	25.00	34.00	42.00	54.00	77.00
Aa2/AA	14.00	29.00	38.00	50.00	57.00	65.00	89.00
Aa3/AA-	19.00	34.00	43.00	54.00	61.00	69.00	92.00
A1/A+	23.00	39.00	47.00	58.00	65.00	72.00	95.00
A2/A	24.00	39.00	49.00	61.00	69.00	77.00	103.00
A3/A-	32.00	49.00	59.00	72.00	80.00	89.00	117.00
Baa1/BBB+	38.00	61.00	75.00	92.00	103.00	115.00	151.00
Baa2/BBB	47.00	75.00	89.00	107.00	119.00	132.00	170.00
Baa3/BBB-	83.00	108.00	122.00	140.00	152.00	165.00	204.00
Ba1/BB+	157.00	182.00	198.00	217.00	232.00	248.00	286.00
Ba2/BB	231.00	256.00	274.00	295.00	312.00	330.00	367.00
B2/B	452.00	478.00	502.00	527.00	552.00	578.00	612.00
B3/B-	526.00	552.00	578.00	604.00	632.00	660.00	693.00
US Treasury Yield	0.13	0.45	0.93	1.74	2.31	2.73	3.55

La columna de índice es la calificación crediticia basada en las escalas de calificación crediticia de Moody's y Standard & Poor's. A excepción de la última fila, el rendimiento del Tesoro de los Estados Unidos, los valores en el conjunto de datos tienen una unidad de punto base que vale una centésima parte del 1%. En otras palabras, cada valor debe dividirse por 100 dos veces. Por ejemplo, para una calificación AA- bono, su spread en el año 5 es de 54 puntos básicos, es decir, 0.0054 (= 54/10000). Si la tasa libre de riesgo para un bono de cupón cero a 5 años es 1.74%, la tasa correspondiente para un bono corporativo, calificado como AA-, sería 2.28% (1.74% + 0.54%).

7.2 RELACIÓN RIESGO Y RENTABILIDAD. LA DURACIÓN

Los precios de los bonos suelen cambiar por varias causas: un cambio en el riesgo de crédito percibido sobre el emisor, un premio o descuento a medida que se aproxima el vencimiento, pero fundamentalmente, por los cambios en las tasas de interés. A continuación se analizará el concepto "Duración", éste nos brindará una medida del cambio en el precio del bono para un cambio en la tasa de interés.

La duración es un concepto muy importante para el análisis de riesgos y la cobertura. La duración se define como: la cantidad de años necesarios para recuperar la inversión inicial. Veamos el caso simple: un bono cupón cero. Hoy compramos un bono de cupón cero a 1 año. Un año después, recibiríamos su valor nominal de € 100. Su línea de tiempo y flujo de caja se pueden observar en la figura 7.1



Figura 7.1. Flujo de caja de un bono cupón cero

Obviamente, tenemos que esperar un año para recuperar nuestra inversión inicial. Por lo tanto, la duración de este bono de 1 año es 1. Para un bono cero cupones, la duración del bono es la misma que su vencimiento:

$$D = T$$

Aquí, D es la duración y T es el vencimiento de un bono cero cupones (en años). Veamos nuestro segundo ejemplo: tendríamos dos flujos de efectivo iguales de € 100 al final de los primeros dos años:

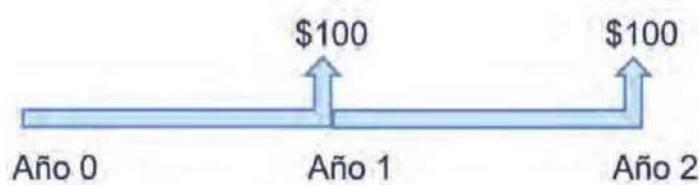


Figura 7.2. Flujo de caja de un bono cupón cero

¿Cuántos años tenemos que esperar para recuperar nuestra inversión inicial? El hecho es que tenemos que esperar un año para recibir los primeros € 100 y esperar dos años para recibir los segundos € 100. Por lo tanto, la primera suposición sería de 1,5 años. Sin embargo, después de leer el capítulo 5, el valor del dinero en el tiempo,

sabemos que € 100 recibidos en el año 2 no son equivalentes a € 100 recibidos en el año 1. Si utilizamos el final del año 1 como nuestro punto de referencia, se muestra el valor equivalente de los segundos € 100 aquí:

```
>>> 100/(1+0.05)
```

Salida

```
95.23809523809524
```

Ahora, diríamos que tenemos que esperar 1 año para recibir € 100 y esperar dos años para recibir € 95.24. En promedio, ¿cuántos años esperaríamos? La solución debe ser un promedio ponderado. Las participaciones porcentuales de esos dos € 100 se dan aquí:

```
pv2=100/(1+0.05)  
w1=100/(100+pv2)  
w1
```

Salida

```
0.5121951219512195
```

```
w2= pv2/(100+pv2)  
w2
```

Salida

```
0.4878048780487805
```

```
w1*1 + w2*2
```

Salida

```
1.4878048780487805
```

Finalmente, tenemos $D = w1 * T1 + w2 * T2 = w1 * 1 + w2 * 2 = 0.5122 * 1 + 0.487805 * 2 = 1.487$. La respuesta es que tenemos que esperar 1.487 años para recuperar nuestra inversión inicial. En el razonamiento anterior, descontamos los segundos € 100 al final del año 1 para obtener nuestra respuesta.

Alternativamente, podríamos aumentar los primeros € 100 hasta el final del año 2, luego comparar, ver el siguiente código:

```
fv=100*(1+0.05)  
fv
```

Salida

```
105.0
```

Las participaciones porcentuales correspondientes se dan aquí:

```
w1=105/(100+105)  
w1
```

Salida

```
0.5121951219512195
```

```
w2=100/(100+105)  
w2
```

Salida

```
0.4878048780487805
```

La solución debe ser la misma ya que las participaciones porcentuales son las mismas que antes. Esto sugiere que podríamos usar cualquier punto de tiempo para estimar las participaciones de esos flujos de efectivo que ocurren en diferentes momentos. Convencionalmente, el valor presente se utiliza como punto de referencia, analice el siguiente código:

```
pv1=100/(1+0.05)
pv2=100/(1+0.05)**2
w1= pv1/(pv1+pv2)
w1
```

Salida

```
0.5121951219512195
```

```
1- w1
```

Salida

```
0.4878048780487805
```

Nuevamente, ambas participaciones permanecen iguales. Otra ventaja de usar el valor presente como nuestro punto de referencia es que también podríamos estimar el valor presente total. El valor total se da aquí. Podríamos argumentar que si invirtiéramos € 185.94 hoy, recuperaríamos 51.2% en el año 1 y el resto al final del año 2. Por lo tanto, en promedio tenemos que esperar 1.487 años:

```
pv1+pv2
```

Salida

```
185.94104308390024
```

La fórmula general para estimar la duración de los flujos de efectivo futuros dados se da en la siguiente fórmula:

$$D = \sum_{i=1}^n w_i * T_i$$

D es la duración, n es el número de flujos de efectivo, wi es el peso del i-ésimo flujo de efectivo, y wi se define como el valor presente del i-ésimo flujo de efectivo sobre los valores actuales de todos los flujos de efectivo, Ti es el momento (en años) del i-ésimo flujo de caja. A continuación, una función de Python llamada duración:

```

import math as m
def duration(t,valor_nominal,y,tasacupon):
    n=list(range(1,t+1))
    cupon=valor_nominal*tasacupon
    ultimoflujo=valor_nominal*tasacupon+valor_nominal
    flujoxperiodo=[cupon]*(len(n)-1)+[ultimoflujo]

    B=0
    for k in range(0,t):
        B=B+flujoxperiodo[i]*(m.exp(-y*n[i]))

    D=0
    for i in range(0,t):
        D+=n[i]*flujoxperiodo[i]*m.exp(-y*n[i])/B

    return (B,D)
duration(2,100,0.05,0.10)

```

Salida

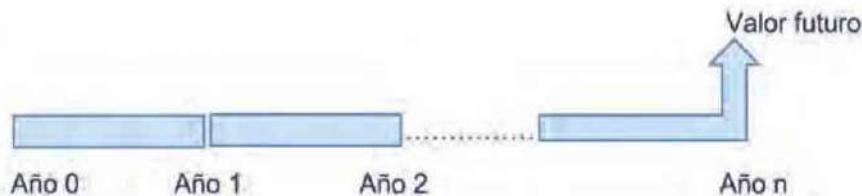
(109.04441022896269, 1.9127667871738314)

7.3 EVALUACIÓN DE BONOS Y RENDIMIENTO HASTA EL VENCIMIENTO

Los bonos también se denominan valores de renta fija. Existen diferentes tipos de categorías. Según el vencimiento, los bonos se pueden clasificar a corto plazo, mediano plazo y largo plazo. Para los títulos del Tesoro de los EE. UU., Las letras T son valores emitidos por el Departamento del Tesoro con un vencimiento inferior a 1 año, las notas T son para bonos del gobierno después de 1 año pero menos de 10 años. Los bonos T son títulos de tesorería con un vencimiento superior a 10 años. Según los pagos de cupones, hay bonos cero cupón y bonos con cupón. Cuando se trata de un bono del gobierno central, usualmente se denominan bonos libres de riesgo.

Si un tenedor de bonos puede convertir su bono en acciones comunes subyacentes con un número predeterminado de acciones antes del vencimiento, se llama un bono convertible. Si un emisor de bonos puede retirarse o recomprar un bono antes de su vencimiento, se llama un bono exigible. Por otro lado, si los compradores de bonos pudieran vender el bono a los emisores originales antes del vencimiento, se considera un bono con opción de venta.

El flujo de caja para un bono cero cupón se muestra aquí:

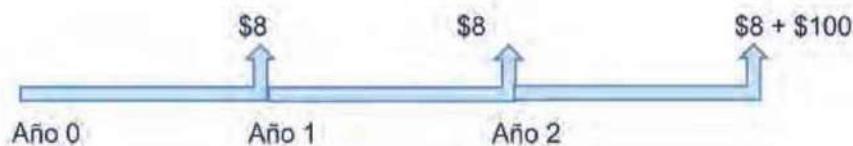


Aquí, valor futuro es el valor nominal y n es el vencimiento (en años). Para estimar el precio de dicho bono cupón cero, podríamos aplicar fácilmente el valor presente de un flujo de caja futuro. En otras palabras, podríamos aplicar la función npf.pv () .

Para un bono con cupón, esperamos un conjunto de pagos regulares del cupón. El pago periódico de cupones se estima mediante la siguiente fórmula:

$$\text{Pago cupón} = \frac{\text{Tasa cupón} * FV}{\text{Frecuencia}}$$

Aquí, FV es el valor nominal del bono y la frecuencia es el número de pagos de cupones cada año. Veamos un bono con cupón a 3 años. El valor nominal es de € 100 con una tasa de cupón anual del 8%. El pago del cupón es anual. El pago del cupón anual es de € 8 por los próximos tres años y los inversores también recibirían el valor nominal de € 100 en la fecha de vencimiento. La línea de tiempo de este bono con cupón y los flujos de efectivo futuros relacionados se muestran aquí:



Recuerde que para el valor presente de un flujo de caja futuro y el valor presente de la anualidad, tenemos las siguientes dos fórmulas:

$$VP = \frac{VF}{(1 + R)^n}$$

$$VP(\text{anualidad}) = \frac{C}{R} \left[1 - \frac{1}{(1 + R)^n} \right]$$

Aquí, C es un flujo de caja constante y n es el número de períodos. El precio de un bono cupón es la combinación de estos dos tipos de pagos:

$$VP(\text{bono}) = \frac{C}{R} \left[1 - \frac{1}{(1+R)^n} \right] + \frac{VF}{(1+R)^n}$$

La función `npf.pv()` podría usarse para calcular el precio del bono. Suponga que la tasa anual efectiva es 2.4%:

```
import numpy as np
import numpy_financial as npf
npf.pv(0.024, 3, 0.08*100, 100)
```

Salida

```
-116.0247325897217
```

Según el resultado anterior, el precio de este bono con cupón a 3 años es de € 116.02.

Dado que el precio de un bono es el valor presente de todos sus flujos de efectivo futuros, su precio debe estar correlacionado negativamente con la tasa de descuento. En otras palabras, si la tasa de interés aumentara, el precio de los bonos caería, y viceversa.

Rendimiento hasta el vencimiento (YTM) es el mismo concepto que la Tasa Interna de Retorno (TIR). Supongamos que compramos un bono de cupón cero por € 717.25. El valor nominal del bono es de € 1,000 y vencería en 10 años. ¿Cuál es su YTM? Para un bono de cupón cero, tenemos la siguiente fórmula para YTM:

$$YTM = \left(\frac{VF}{VP} \right)^{1/n}$$

Aquí, VF es el valor nominal, PV es el precio del bono cero cupón y n es el número de años (vencimiento). Al aplicar la fórmula, tenemos $717.25 * (1 + YTM)^{-10} = 1000$. Por lo tanto, tenemos el siguiente resultado:

```
(1000/717.25)**(1/10)-1
```

Salida

0.033791469771228044

Suponga que hoy compramos un bono por € 818. Tiene un plazo de vencimiento de 5 años. La tasa de cupón es del 3% y los pagos de cupones son anuales. Si el valor nominal es de € 1,000, ¿cuál es el YTM? La función npf.rate () podría usarse para estimar el YTM:

```
import numpy as np
import numpy_financial as npf
npf.rate(5, 0.03*1000, -818, 1000, when='end')
```

Salida

0.07498180431487073

Según este resultado, el YTM es del 7,498%. La relación entre el precio del bono, la tasa del cupón y el valor nominal se muestra en la tabla 7.2

Condición	Precio del bono vs valor facial	Con prima, a la par, con descuento
Tasa cupón > YTM	Precio del bono > Valor facial	Con prima
Tasa cupón = YTM	Precio del bono = Valor facial	A la par
Tasa cupón < YTM	Precio del bono < Valor facial	Con descuento

Tabla 7.2. Relación entre el precio del bono, la tasa cupón y el valor facial

Obviamente, para dos bonos cero cupón, cuanto más largo sea el vencimiento, más riesgoso será el bono. La razón es que para un bono cero cupón con un vencimiento más largo, tenemos que esperar más para recuperar nuestra inversión inicial. Para dos bonos con cupón con el mismo vencimiento, cuanto más altas sean las tasas de cupón, más seguro será el bono ya que podríamos recibir más pagos anticipados por el bono con una tasa de cupón más alta.

Aquí hay un ejemplo, tenemos un bono cero cupón a 15 años con un valor nominal de € 100 y un bono con cupón de 30 años. La tasa cupón es del 9% con un pago de cupón anual. ¿Qué bono es más arriesgado? Si el rendimiento actual salta del 4% al 5%, ¿cuáles son los porcentajes para ambos?

```
# Para el bono cero cupón
import numpy as np
import numpy_financial as npf
p0=npf.pv(0.04,15, 0, -100)
p1=npf.pv(0.05,15, 0, -100)
(p1-p0)/p0
```

Salida

```
-0.1337153811552842
```

El bono más riesgoso tendría un cambio porcentual mucho mayor cuando la rentabilidad se incrementa o decrece:

La salidas relacionadas se muestra aquí:

```
p0
```

Salida

```
55.526450271327484
```

```
p1
```

Salida

```
48.101709809096995
```

Para el bono de cupón, tenemos el siguiente resultado:

```
# Para el bono con cupón
p0=npf.pv(0.04,30, -0.09*100, -100)
p1=npf.pv(0.05,30, -0.09*100, -100)
(p1-p0)/p0
```

Salida

0.13391794539315816

p0

Salida

186.46016650332245

p1

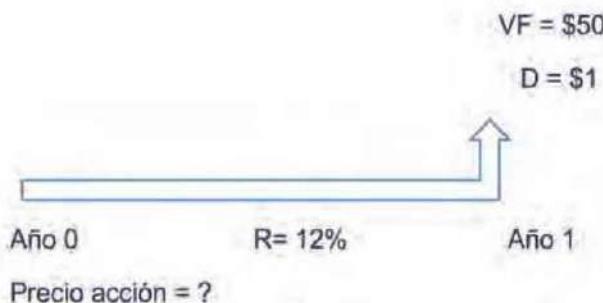
Salida

161.48980410753134

Con base en los resultados anteriores, el bono de cupón a 30 años es más riesgoso que el bono cero cupones a 15 años ya que tiene un mayor cambio porcentual.

7.4 VALORACIÓN DE ACCIONES

Hay varias formas de estimar el precio de una acción. Un método se llama modelo de descuento de dividendos. La lógica es que el precio de una acción hoy es simplemente la suma del valor presente de todos sus dividendos futuros. Usemos el modelo más simple de un período para ilustrar la situación. Esperamos un dividendo de € 1 al final de un año y se espera que el precio de venta sea de € 50. Si el costo de capital es del 12%, ¿cuál es el precio actual de la acción? La línea de tiempo y los flujos de efectivo futuros se muestran aquí:



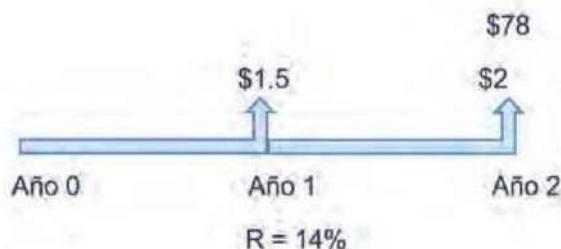
El precio de las acciones es simplemente el valor presente de esos dos flujos de efectivo futuros, € 45.54:

```
import numpy as np
import numpy_financial as npf
npf.pv(0.12, 1, 0, 50+1)
```

Salida

```
-45.535714285714285
```

Veamos un modelo de dos períodos. Esperamos dos dividendos de € 1.5 y € 2 al final de los próximos 2 años. Además, se espera que el precio de venta sea de € 78. ¿Cuál es el precio hoy?



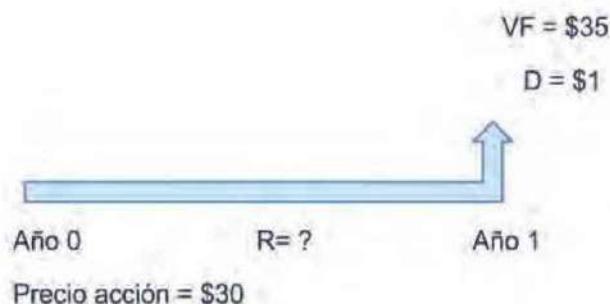
Suponga que para esta acción, la tasa de descuento apropiada es del 14%. Entonces el valor presente de la acción es € 62.87:

```
1.5/(1+0.14)+(2+78)/(1+0.14)**2
```

Salida

```
62.873191751308084
```

En la misma línea, podríamos estimar el costo del patrimonio si se dan tanto el valor presente como los valores futuros. Si el precio actual es de € 30 y el precio de venta esperado al final de un año es de € 35, con un dividendo de € 1:



Entonces podríamos estimar el rendimiento total:

$$(35-30+1)/30$$

Salida

$$0.2$$

El rendimiento total, el costo de capital (R_e), tiene dos componentes: rendimiento de ganancia de capital y rendimiento de dividendos:

$$R_e = \frac{P_1 - P_0 + D_1}{P_0} = \frac{P_1 - P_0}{P_0} + \frac{D_1}{P_0}$$

El primer término del lado derecho de la ecuación se refiere a la rentabilidad del capital y el segundo término a la rentabilidad del dividendo.

El rendimiento de la ganancia de capital es de 16.667%, mientras que el rendimiento de dividendos es de 3.333%. Otro posible escenario es que una acción pueda disfrutar de una tasa constante de crecimiento de dividendos. Se espera que la compañía A emita un dividendo de € 4 el próximo año y luego disfrute de una tasa de crecimiento constante del 2%. Si el costo del patrimonio es del 18%, ¿cuál será el precio de la acción hoy? Del Capítulo 5, el valor del dinero en el tiempo, sabemos que el valor actual de la fórmula de perpetuidad creciente podría aplicarse:

$$VP \text{ (crecimiento a perpetuidad)} \frac{C}{(R - g)}$$

Al usar la notación correcta, es decir, P_0 como precio de las acciones de hoy, D_1 como el primer dividendo esperado, podríamos tener la siguiente fórmula de precios equivalente:

$$P_0 = \frac{D_1}{(R - g)}$$

De los siguientes resultados, sabemos que el precio de hoy debería ser de € 25:

.....
4/(0.18-0.02)
.....

Salida

.....
25.0
.....

8

COSTO PROMEDIO PONDERADO DE CAPITAL (WACC)

En este capítulo, aprenderemos cómo construir un modelo para calcular el costo de capital promedio ponderado (WACC) de una empresa con Python. Aprenderemos qué es el WACC y cómo calcularlo paso a paso. El WACC puede utilizarse para estimar el valor de una empresa aplicando el método de flujo de caja libre descontado.

8.1 ¿QUÉ ES EL COSTO DE CAPITAL PROMEDIO PONDERADO (WACC)?

El WACC es el costo de capital de la empresa. Es la tasa que se espera que una empresa pague en promedio a todos sus proveedores de capital. Como veremos en la fórmula de la siguiente sección, el WACC usa el costo de la deuda, el costo del patrimonio, la estructura de capital de la empresa y la tasa impositiva.

La estructura de capital de una empresa es importante para calcular el WACC, ya que utiliza el costo de capital combinado en todas las fuentes de capital (es decir, costo de deuda y costo de patrimonio).

La tasa de impuestos se utiliza en el WACC debido a los efectos fiscales de los pagos de intereses. Los pagos de intereses se pueden deducir del impuesto de renta y por ende obtener un ahorro en el pago de impuestos.

El WACC se utiliza comúnmente como tasa de descuento para calcular el valor de una empresa. Por ejemplo, al valorar una empresa utilizando el modelo de flujo de caja libre descontado, deberíamos usar el WACC para descontar los flujos de caja libres futuros.

8.2 ¿CÓMO CALCULAR EL WACC?

Podemos calcular el costo de capital promedio ponderado de una empresa aplicando la siguiente fórmula:

$$\text{WACC} = K_d * (1 - T_c) * (D / (D + E)) + K_e * (E / (D + E))$$

Dónde:

WACC = Costo de capital promedio ponderado

Kd = Costo de la deuda

Tc = Tasa impositiva de la empresa

Ke = Costo de patrimonio

D / (D + E) = Proporción de deuda en la estructura de capital de la empresa

E / (D + E) = Proporción de patrimonio en la estructura de capital de la empresa

Veamos cómo podemos obtener cada uno de estos elementos:

8.3 COSTO DE LA DEUDA DE LAS EMPRESAS

Para estimar el costo de la deuda, podríamos seguir uno de los dos enfoques siguientes:

1. Calcular los rendimientos de los bonos hasta el vencimiento de los bonos negociables existentes de una empresa. El rendimiento al vencimiento es un buen indicador del costo de la deuda. El problema es que este enfoque requeriría examinar a cada empresa y ver cuál es el bono más representativo para usar como proxy del costo de la deuda. Algunas empresas pueden tener más de 10 bonos disponibles. Por lo tanto, es difícil automatizar este enfoque con Python y puede llevar mucho tiempo hacerlo manualmente.
2. Segundo. Por lo tanto, vamos a utilizar un enfoque diferente. Usaremos la calificación crediticia de las empresas para calcular el diferencial crediticio. Luego, aplicaremos la siguiente fórmula para obtener nuestro costo de la deuda (Kd).

$$K_d = R_f + \text{diferencial de crédito.}$$

Es posible que no todas las empresas tengan una calificación crediticia disponible. Incluso si tienen una calificación crediticia publicada, tendríamos que hacer un trabajo manual para encontrar en línea la calificación de una empresa en

particular. Como solución alternativa, vamos a calcular una calificación crediticia sintética para las empresas en función de su índice de cobertura de intereses.

Luego, basándonos en el sistema de calificación de sitios como el de Damodaran (<http://pages.stern.nyu.edu/~adamodar/>), podemos calcular el margen crediticio según la tabla siguiente.

For developed market firms with market cap > \$5 billion			
If interest coverage ratio is			
>	\leq to	Rating is	Spread is
8.50	100000	Aaa/AAA	0.63%
6.5	8.499999	Aa2/AA	0.78%
5.5	6.499999	A1/A+	0.98%
4.25	5.499999	A2/A	1.08%
3	4.249999	A3/A-	1.22%
2.5	2.999999	Baa2/BBB	1.56%
2.25	2.249999	Ba1/BB+	2.00%
2	2.2499999	Ba2/BB	2.40%
1.75	1.999999	B1/B+	3.51%
1.5	1.749999	B2/B	4.21%
1.25	1.499999	B3/B-	5.15%
0.8	1.249999	Caa/CCC	8.20%
0.65	0.799999	Ca2/CC	8.64%
0.2	0.649999	C2/C	11.34%
-100000	0.199999	D2/D	15.12%

Tabla 8.1. Calificación crediticia Damodaran

Para la tasa libre de riesgo R_f , usaremos la tasa de interés ofrecida por una letra del tesoro estadounidense a 1 año, para la fecha de escribir esto es de alrededor del 0,15%.

8.4 COSTO DEL PATRIMONIO DE LA EMPRESA

Para estimar el costo del patrimonio de una empresa, podemos utilizar dos enfoques diferentes; el modelo de valoración de activos de capital (CAPM) o el modelo de descuento de dividendos. Aquí usaremos el modelo CAPM.

Para obtener el WACC requerimos conocer la estructura de capital de la empresa y su tasa impositiva, esta información la podemos obtener al examinar los estados financieros de la respectiva empresa. En la siguiente sección, aprenderemos cómo extraer esta información con Python de manera automática.

8.5 CALCULANDO EL WACC CON PYTHON

Es hora de pasar a la parte de programación. Vamos a construir diferentes funciones en Python con el fin de obtener los elementos necesarios para calcular el WACC de una empresa. En este ejemplo, calcularemos el WACC de Microsoft.

Primero, estimaremos el costo de la deuda, luego calcularemos el costo del patrimonio usando el método CAPM. Finalmente, recuperaremos la tasa impositiva de la empresa y la estructura de capital para calcular el WACC de la empresa.

Tenga en cuenta que vamos a obtener todos los datos financieros necesarios para nuestro cálculo de la API financialmodelingprep. Es una API financiera que ofrece hasta 250 solicitudes gratuitas por mes.

La forma en que va a funcionar el código es muy simple. Tendremos tres funciones. Una para estimar el costo de la deuda, otra para estimar el costo del patrimonio y una tercera para obtener la tasa de impuestos de la empresa, la estructura de capital y calcular el WACC.

Comencemos por crear la función para calcular el costo de la deuda.

8.5.1 Estimación del costo de la deuda con Python

En primer lugar, importaremos las librerías necesarias para calcular el WACC con Python. Luego, necesitamos crear una función para calcular el índice de cobertura de intereses y la tasa libre de riesgo. Recuerde que usaremos la cobertura de intereses para crear una calificación crediticia sintética y por lo tanto estimar el costo de la deuda con ella.

La tasa libre de riesgo se puede recuperar utilizando pandas_datareader y extrayendo las letras del tesoro de 1 año. Luego, tomamos el dato más reciente. Esa será nuestra tasa libre de riesgo requerida para estimar el costo de la deuda.

```
import pandas_datareader.data as web
import datetime
import requests
```

```
company = 'MSFT'
demo = 'API key' #aquí debe colocar su clave
#Indicador cobertura de intereses = EBIT / interest expenses

def interest_coverage_and_RF(company):
    IS= requests.get(f'https://financialmodelingprep.com/api/v3/income-statement/{company}?apikey={demo}').json()
    EBIT= IS[0]['ebitda'] - IS[0]['depreciationAndAmortization']
    interest_expense = IS[0]['interestExpense']
    interest_coverage_ratio = EBIT / interest_expense

#RF o Tasa Libre de Riesgo
start = datetime.datetime(2019, 7, 10)

end= datetime.datetime.today().strftime('%Y-%m-%d')
#end = datetime.datetime(2020, 7, 10)

Treasury = web.DataReader(['TB1YR'], 'fred', start, end)
RF = float(Treasury.iloc[-1])
RF = RF/100
print(RF,interest_coverage_ratio)
return [RF,interest_coverage_ratio]
```

Salida

```
RF is 0.0013
Interest Coverage Ratio is 25.362794287919723
```

Habiendo calculado el índice de cobertura de intereses y la tasa libre de riesgo, podemos pasar a estimar el costo de la deuda. Tendremos una función de costo de deuda que tomará como argumentos el nombre de la empresa, la tasa libre de riesgo (R_f) y el índice de cobertura de intereses.

Tenga en cuenta que lo que estamos haciendo es replicar el sistema de calificación crediticia propuesto en el sitio de Damodaran con Python. Con base en el índice de cobertura de intereses calculado anteriormente, estimaremos el margen de crédito para calcular el costo de la deuda.

```
#Costo de deuda
def cost_of_debt(company, RF,interest_coverage_ratio):
    if interest_coverage_ratio > 8.5:
        #Rating is AAA
        credit_spread = 0.0063
    if (interest_coverage_ratio > 6.5) & (interest_coverage_ratio <= 8.5):
        #Rating is AA
        credit_spread = 0.0078
    if (interest_coverage_ratio > 5.5) & (interest_coverage_ratio <= 6.5):
        #Rating is A+
        credit_spread = 0.0098
    if (interest_coverage_ratio > 4.25) & (interest_coverage_ratio <= 5.49):
        #Rating is A
        credit_spread = 0.0108
    if (interest_coverage_ratio > 3) & (interest_coverage_ratio <= 4.25):
        #Rating is A-
        credit_spread = 0.0122
    if (interest_coverage_ratio > 2.5) & (interest_coverage_ratio <= 3):
        #Rating is BBB
        credit_spread = 0.0156
    if (interest_coverage_ratio > 2.25) & (interest_coverage_ratio <= 2.5):
        #Rating is BB+
        credit_spread = 0.02
    if (interest_coverage_ratio > 2) & (interest_coverage_ratio <= 2.25):
        #Rating is BB
        credit_spread = 0.0240
    if (interest_coverage_ratio > 1.75) & (interest_coverage_ratio <= 2):
        #Rating is B+
        credit_spread = 0.0351
    if (interest_coverage_ratio > 1.5) & (interest_coverage_ratio <= 1.75):
        #Rating is B
        credit_spread = 0.0421
    if (interest_coverage_ratio > 1.25) & (interest_coverage_ratio <= 1.5):
        #Rating is B-
        credit_spread = 0.0515
    if (interest_coverage_ratio > 0.8) & (interest_coverage_ratio <= 1.25):
        #Rating is CCC
        credit_spread = 0.0820
    if (interest_coverage_ratio > 0.65) & (interest_coverage_ratio <= 0.8):
        #Rating is CC
        credit_spread = 0.0864
    if (interest_coverage_ratio > 0.2) & (interest_coverage_ratio <= 0.65):
        #Rating is C
        credit_spread = 0.1134
    if interest_coverage_ratio <= 0.2:
```

```
#Rating is D  
credit_spread = 0.1512  
  
cost_of_debt = RF + credit_spread  
print(cost_of_debt)  
return cost_of_debt
```

Salida

```
0.0076 # 0.76% es el costo estimado de la deuda para Microsoft
```

El costo estimado de la deuda de Microsoft es 0.76%. Pasemos ahora a calcular el costo del patrimonio.

8.5.2 Estimación del costo de patrimonio con Python

Tenga en cuenta que utilizamos el rendimiento anual del SP500 como indicador del rendimiento del mercado. El único elemento que no calculamos y tomamos directamente de la *API financialmodelingprep* es la beta de la empresa.

```
def costofequity(company):  
  
    #RF o tasa libre de riesgo  
    start = datetime.datetime(2019, 7, 10)  
    end= datetime.datetime.today().strftime('%Y-%m-%d')  
    #end = datetime.datetime(2020, 7, 10)  
  
    Treasury = web.DataReader(['TB1YR'], 'fred', start, end)  
    RF = float(Treasury.iloc[-1])  
    RF = RF/100  
  
    #Beta  
  
    beta = requests.get(f'https://financialmodelingprep.com/api/v3/company/profile/{company}?apikey={demo}')  
    beta = beta.json()  
    beta = float(beta['profile']['beta'])  
  
    #Rentabilidad del mercado  
    start = datetime.datetime(2019, 7, 10)
```

```
end= datetime.datetime.today().strftime('%Y-%m-%d')

SP500 = web.DataReader(['sp500'], 'fred', start, end)
# Quitar todos los valores no numéricos con el método drop.
SP500.dropna(inplace = True)

SP500yearlyreturn = (SP500['sp500'].iloc[-1]/ SP500['sp500'].iloc[-252])-1

cost_of_equity = RF+(beta*(SP500yearlyreturn - RF))
print(cost_of_equity)
return cost_of_equity
```

Salida

```
0.1265577165607343
```

Obtenemos que el costo de patrimonio para Microsoft es del 12,65%. Ahora obtengamos la tasa impositiva efectiva y la estructura de capital de Microsoft para finalmente calcular el WACC de Microsoft.

8.5.3 Costo de capital promedio ponderado (WACC) con Python

Finalmente, ahora podemos recuperar de la API la tasa impositiva efectiva, la deuda total y el capital total. Luego, tendremos todos los elementos necesarios para calcular el WACC para Microsoft (o cualquier otra empresa).

```
#Tasa de impuestos efectiva y estructura de capital
def wacc(company):
    FR = requests.get(f'https://financialmodelingprep.com/api/v3/ratios/{company}?apikey={demo}').json()

    ETR = FR[0]['effectiveTaxRate']

    #
    BS = requests.get(f'https://financialmodelingprep.com/api/v3/balance-sheet-statement/{company}?period=quarter&apikey={demo}').json()

    Debt_to = BS[0]['totalDebt'] / (BS[0]['totalDebt'] + BS[0]['totalStockholder-
```

```
sEquity'])
equity_to = BS[0]['totalStockholdersEquity'] / (BS[0]['totalDebt'] + BS[0]
['totalStockholdersEquity'])

WACC = (kd*(1-ETR)*Debt_to) + (ke*equity_to)
print(WACC, equity_to, Debt_to)
return WACC

company = 'MSFT'
demo = 'digite aquí su api key'

RF_and_IntCov = interest_coveraga_and_RF(company)
RF = RF_and_IntCov[0]
interest_coverage_ratio = RF_and_IntCov[1]
ke = costofequity(company)
kd = cost_of_debt(company,RF,interest_coverage_ratio)
wacc_company = wacc(company)
print('wacc of ' + company + ' is ' + str((wacc_company*100))+'%')
```

Salida

```
wacc of MSFT is 8.253167643608633%
```

¡Muy bien! Hemos construido un modelo para estimar el costo de capital promedio ponderado (WACC) con Python para cualquier empresa. Simplemente, pase el ticker de la empresa para la que le gustaría calcular el WACC. Tenga en cuenta que al ejecutar el código, si un ticker de la empresa no está disponible en la API financialmodelingprep, es posible que obtenga un error.

9

FLUJO DE CAJA LIBRE DESCONTADO: VALORACIÓN DE UNA EMPRESA CON PYTHON

En este capítulo utilizaremos el método del flujo de caja libre descontado para valorar una empresa. El modelo que construiremos será dinámico, es decir, podremos reutilizarlo para cualquier empresa sobre la cual queramos estimar el precio de sus acciones.

9.1 INTRODUCCIÓN AL MÉTODO DE FLUJO DE CAJA LIBRE DESCONTADO

El método del flujo de caja libre descontado se centra en las capacidades futuras de la empresa para generar efectivo. El efectivo es lo que la empresa necesita para poder pagar salarios, facturas, impuestos y también para retribuir a los proveedores de capital de la empresa.

El estado de flujo de efectivo clasifica las actividades de la empresa en tres componentes: actividades de operación, actividades de inversión y actividades de financiación. El método del flujo de caja libre descontado se enfoca principalmente en las actividades de operación, aquí se muestran las entradas y salidas de efectivo relacionadas con la producción y venta de bienes y servicios de la empresa.

El estado de flujo de efectivo es importante porque no podemos saber cuánto efectivo genera una empresa simplemente mirando el estado de resultados o el balance general. Por ejemplo, tome la utilidad neta. Al analizar la utilidad neta, no podemos saber cuánto efectivo ha generado una empresa. La utilidad neta está contaminada por elementos no monetarios como depreciación, ventas a crédito, entre otros.

Por lo tanto, dado que el método de flujo de caja libre descontado se basa en efectivo, comenzaremos nuestro modelo con la utilidad neta y ajustaremos ciertos elementos para obtener el flujo de efectivo operativo de la empresa:

- ▶ Ingresos y gastos no monetarios relacionados con operaciones
- ▶ Ajuste por otros elementos operativos que hayan generado una salida o entrada de efectivo, como el cobro de cuentas por cobrar o el pago de cuentas por pagar

Luego, una vez que tengamos el flujo de efectivo operativo, simplemente podemos deducir los gastos de capital para llegar al flujo de caja libre de la empresa (tabla 9.1). Esto es lo que usaremos para estimar el valor de una empresa.

Utilidad neta
(+) Depreciaciones y amortizaciones
(-) Variación cuentas por cobrar clientes
(-) Variación inventarios
(-) Variación otros activos
(+) Variación cuentas por pagar proveedores
(+) Variación otros pasivos
(=) Flujo de efectivo operaciones
(-) Variación activos fijos (CAPEX)
(=) FLUJO DE CAJA LIBRE

Tabla 9.1. Flujo de caja libre

9.2 VALOR DE LA EMPRESA CON EL MÉTODO DE FLUJO DE CAJA LIBRE DESCONTADO

A la utilidad operacional después de impuestos agregamos las depreciaciones y amortizaciones, esto considerando que no representan una salida real de efectivo. Posteriormente restamos el cambio en el capital de trabajo neto operativo (cuentas por cobrar clientes – inventario + cuentas por pagar proveedores) y finalmente restamos las inversiones en activos fijos. De esta forma obtenemos el flujo de caja libre.

El flujo de caja libre es lo que necesitamos en el método FCLD para estimar el valor de la empresa. Según el método de flujo de caja libre descontado, podemos obtener el valor de la empresa aplicando la siguiente fórmula:

$$EV = \sum_{i=1}^n \frac{FCFF_i}{(1+WACC)^i} + \frac{TV}{(1+WACC)^n}$$

$$TV = \frac{FCFF_{n+1}}{(WACC-g)} = \frac{FCFF_n \times (1+g)}{(WACC-g)}$$

Figura 9.1. Fórmula para determinar el valor de la empresa

Donde:

EV = Valor de la empresa

FCFF = Flujo de caja libre future

TV = Valor terminal

WACC = Costo promedio ponderado de capital

g = Tasa de crecimiento a perpetuidad del flujo de caja libre

n = Períodos relevantes de proyección

9.3 PRONÓSTICO DEL FLUJO DE CAJA LIBRE DE LA EMPRESA

El método del flujo de caja libre descontado requiere pronosticar el flujo de caja libre para los próximos años. En nuestro modelo, pronosticaremos los próximos 5 años.

Usaremos el método de porcentaje de ventas para pronosticar el flujo de caja de una empresa. Calcularemos las ventas futuras de la empresa y estimaremos el estado de resultados y las partidas del balance general requeridas en función del porcentaje de ventas.

Por ejemplo, la empresa A en el pasado tenía un costo de ventas que representaba el 30% de las ventas. Por lo tanto, usando el método de porcentaje de ventas, podríamos suponer que en el pronóstico del estado de resultados del próximo año, los costos de ventas permanecerán en el 30% de las ventas.

En resumen, seguiremos los siguientes pasos para valorar una empresa utilizando el método de flujo de caja descontado (FCLD) con Python:

- Estimar el crecimiento de los ingresos. Para simplificar, calcularemos el crecimiento de los ingresos del año pasado y asumiremos que el crecimiento de los ingresos será el mismo cada año para los años pronosticados. Un mejor enfoque sería encontrar empresas similares que nos ayuden a comprender cuánto podrían crecer las ventas con el tiempo

y qué tan madura es la industria. Con base en esto, podríamos estimar el crecimiento de los ingresos para los períodos proyectados.

- ▶ Proyectar los balances generales y los estados de resultados de la empresa utilizando el método del porcentaje de ventas
- ▶ Calcular el costo de capital WACC (costo promedio ponderado de capital)
- ▶ Utilizar el WACC como tasa de descuento para calcular el valor presente de los flujos de caja libre futuros para los 5 años pronosticados
- ▶ Calcular el valor terminal de la empresa. Ese es el valor de la empresa después de los 5 años previstos. Se requiere el valor terminal ya que solo hemos pronosticado los próximos cinco años de los flujos de caja de la empresa. ¿Qué pasa después de estos 5 años? Estimamos el valor terminal de la empresa
- ▶ Calcular el valor objetivo del patrimonio y la deuda total de la empresa. Finalmente, calcular el precio objetivo de las acciones

En la siguiente sección, nos trasladaremos a Python para comenzar a crear nuestro modelo.

9.4 FLUJO DE CAJA LIBRE DESCONTADO CON PYTHON

Es hora de empezar con la parte divertida. La idea es construir un modelo de flujo de caja libre descontado con Python. Pasaremos el ticker de la empresa para la que queremos estimar el precio de las acciones y Python hará el trabajo duro por nosotros. Entonces, el resultado del código será un precio estimado de las acciones considerando las suposiciones que hemos utilizado para llegar a ese precio objetivo.

Para utilizar el método de flujo de caja libre descontado, necesitamos obtener algunos datos financieros de la empresa. Tendremos que conseguir los estados financieros (el balance general y el estado de resultados) más recientes de la empresa. Los usaremos para construir nuestro modelo.

Para obtener los estados financieros, usaremos la API *financialmodelingprep*, una API financiera que ofrece hasta 250 solicitudes gratuitas por mes.

Como el código es un poco largo, se dividirá por secciones. El resultado del código será el precio estimado de las acciones de la empresa Alphabet (Google) utilizando el método de flujo de caja libre descontado.

9.4.1 Estimación del crecimiento futuro de los ingresos

Como se mencionó anteriormente, estimar cuánto crecerán los ingresos durante los próximos cinco años es la clave para la valoración por el método del flujo de caja libre descontado. Asumiremos que los ingresos de la compañía seguirán creciendo en el mismo porcentaje que en el último año.

NOTA

En el siguiente código, recuerde reemplazar el valor de la variable `demo` por su clave api de `financialmodelingprep`.

Extraemos la cuenta de resultados de Google de los últimos tres años. Luego, calculamos la variación de ingresos del último año y la almacenamos en una variable llamada `revenue_g`:

```
import requests
import numpy as np
import pandas as pd

company = 'GOOG'
demo = 'API key' #Aquí debe ingresar su clave
IS = requests.get(f'https://financialmodelingprep.com/api/v3/income-statement/{company}?apikey={demo}').json()
count = 0
# Obtener el crecimiento histórico de los ingresos para estimar los ingresos futuros
revenue_g = []
for item in IS:
    if count < 4:
        #print(item)
        revenue_g.append(item['revenue'])
        count = count + 1

revenue_g = (revenue_g[0] - revenue_g[1]) /revenue_g[1]
print(revenue_g)
```

Salida

```
0.12770532012826136
```

Como se muestra arriba, los ingresos de Google han crecido un 12,7% en el último año. Ese será nuestro supuesto de crecimiento anual de ingresos para los 5 años previstos.

9.4.2 Obtención del estado de resultados y el balance general

A continuación, obtendremos el estado de resultados y el balance general.

Primero, calculamos el estado de resultados completo como un porcentaje de los ingresos. Luego, pronosticamos el estado de resultados de los próximos 5 años multiplicando los ingresos del año anterior por el crecimiento en ventas (revenue_g) calculado en la sección anterior. Una vez hecho esto, podemos multiplicar cada uno de los elementos del estado de resultados por el % de ingresos para obtener un estado de resultados completo.

Hacemos lo mismo con el balance. Donde también lo mostramos como un porcentaje de los ingresos y pronosticamos los próximos 5 años:

```
#Obtener los ingresos netos
net_income = IS[0]['netIncome']

BS = requests.get(f'https://financialmodelingprep.com/api/v3/balance-sheet-state-
ment/{company}?apikey={demo}').json()

# Obtenga el estado de resultados como % de los ingresos para predicciones futu-
ras y pronostique 5 próximos años del estado de resultados

income_statement = pd.DataFrame.from_dict(IS[0],orient='index')

# Los [6:33] a continuación eliminan los elementos no necesarios que provienen
# de la API
income_statement = income_statement[6:33]
income_statement.columns = ['current_year']
income_statement['as_%_of_revenue'] = income_statement['current_year'] /
income_statement.iloc[0]

#Pronosticar los próximos 5 años del estado de resultados
income_statement['next_year'] = (income_statement['current_year']['revenue'] *
(1+revenue_g)) * income_statement['as_%_of_revenue']
income_statement['next_2_year'] = (income_statement['next_year']['revenue'] *
(1+revenue_g)) * income_statement['as_%_of_revenue']
income_statement['next_3_year'] = (income_statement['next_2_year']['revenue'] *
(1+revenue_g)) * income_statement['as_%_of_revenue']
```

```
income_statement['next_4_year'] = (income_statement['next_3_year']['revenue'] *  
(1+revenue_g)) * income_statement['as_%_of_revenue']  
income_statement['next_5_year'] = (income_statement['next_4_year']['revenue'] *  
(1+revenue_g)) * income_statement['as_%_of_revenue']  
  
# Obtener el balance general como porcentaje de los ingresos  
  
balance_sheet = pd.DataFrame.from_dict(BS[0],orient='index')  
balance_sheet = balance_sheet[6:-2]  
balance_sheet.columns = ['current_year']  
balance_sheet['as_%_of_revenue'] = balance_sheet['current_year'] /  
income_statement['current_year'].iloc[0]  
  
#Pronosticar los próximos 5 años del balance general  
  
balance_sheet['next_year'] = income_statement['next_year']['revenue'] * balance_sheet['as_%_of_revenue']  
balance_sheet['next_2_year'] = income_statement['next_2_year']['revenue'] * balance_sheet['as_%_of_revenue']  
balance_sheet['next_3_year'] = income_statement['next_3_year']['revenue'] * balance_sheet['as_%_of_revenue']  
balance_sheet['next_4_year'] = income_statement['next_4_year']['revenue'] * balance_sheet['as_%_of_revenue']  
balance_sheet['next_5_year'] = income_statement['next_5_year']['revenue'] * balance_sheet['as_%_of_revenue']
```

9.4.3 Proyección de los flujos de caja futuros de las operaciones

Finalmente, tenemos suficiente información para estimar los próximos cinco años de flujo de caja libre para Google.

No se asuste por la cantidad de líneas en el código a continuación. Simplemente se repite para pronosticar el flujo de caja para cada uno de los cinco años. En el siguiente párrafo, se explica lo que hace el código.

Creamos un diccionario vacío donde agregamos cada uno de los 5 años que vamos a pronosticar. Luego, comenzamos obteniendo la utilidad neta del año. A continuación, obtenemos el aumento de depreciación, cuentas por cobrar, cuentas por pagar e inventarios.

Teniendo estos elementos, podemos determinar el flujo de efectivo de las operaciones. Luego, obtenemos el CAPEX para el año calculando el cambio en el rubro de Propiedad y Equipo (PP&E) y agregando la depreciación. Finalmente,

sumamos el flujo de efectivo de operaciones y el CAPEX para estimar el flujo de caja libre para el primer año que requerimos. Luego, repetimos 4 veces más el mismo código para pronosticar los próximos 4 años.

```

CF_forecast = {}
CF_forecast['next_year'] = {}
CF_forecast['next_year']['netIncome'] = income_statement['next_year']['netIncome']
CF_forecast['next_year']['inc_depreciation'] = income_statement['next_year']['depreciationAndAmortization'] - income_statement['current_year']['depreciationAndAmortization']
CF_forecast['next_year']['inc_receivables'] = balance_sheet['next_year']['netReceivables'] - balance_sheet['current_year']['netReceivables']
CF_forecast['next_year']['inc_inventory'] = balance_sheet['next_year']['inventory'] - balance_sheet['current_year']['inventory']
CF_forecast['next_year']['inc_payables'] = balance_sheet['next_year']['accountPayables'] - balance_sheet['current_year']['accountPayables']
CF_forecast['next_year']['CF_operations'] = CF_forecast['next_year']['netIncome'] + CF_forecast['next_year']['inc_depreciation'] + (CF_forecast['next_year']['inc_receivables'] * -1) + (CF_forecast['next_year']['inc_inventory'] * -1) + CF_forecast['next_year']['inc_payables']
CF_forecast['next_year']['CAPEX'] = balance_sheet['next_year']['propertyPlantEquipmentNet'] - balance_sheet['current_year']['propertyPlantEquipmentNet'] + income_statement['next_year']['depreciationAndAmortization']

CF_forecast['next_year']['FCF'] = CF_forecast['next_year']['CAPEX'] + CF_forecast['next_year']['CF_operations']

CF_forecast['next_2_year'] = {}
CF_forecast['next_2_year']['netIncome'] = income_statement['next_2_year']['netIncome']
CF_forecast['next_2_year']['inc_depreciation'] = income_statement['next_2_year']['depreciationAndAmortization'] - income_statement['next_2_year']['depreciationAndAmortization']
CF_forecast['next_2_year']['inc_receivables'] = balance_sheet['next_2_year']['netReceivables'] - balance_sheet['next_2_year']['netReceivables']
CF_forecast['next_2_year']['inc_inventory'] = balance_sheet['next_2_year']['inventory'] - balance_sheet['next_2_year']['inventory']
CF_forecast['next_2_year']['inc_payables'] = balance_sheet['next_2_year']['accountPayables'] - balance_sheet['next_2_year']['accountPayables']
CF_forecast['next_2_year']['CF_operations'] = CF_forecast['next_2_year']['netIncome'] + CF_forecast['next_2_year']['inc_depreciation'] + (CF_forecast['next_2_year']['inc_receivables'] * -1) + (CF_forecast['next_2_year']['inc_inventory'] * -1) + CF_forecast['next_2_year']['inc_payables']

```

```
year'][‘inc_receivables’] * -1) + (CF_forecast[‘next_2_year’][‘inc_inventory’]
*-1) + CF_forecast[‘next_2_year’][‘inc_payables’]
CF_forecast[‘next_2_year’][‘CAPEX’] = balance_sheet[‘next_2_year’][‘property-
PlantEquipmentNet’] - balance_sheet[‘next_year’][‘propertyPlantEquipmentNet’] +
income_statement[‘next_2_year’][‘depreciationAndAmortization’]
CF_forecast[‘next_2_year’][‘FCF’] = CF_forecast[‘next_2_year’][‘CAPEX’] + CF_
forecast[‘next_2_year’][‘CF_operations’]

CF_forecast[‘next_3_year’] = {}
CF_forecast[‘next_3_year’][‘netIncome’] = income_statement[‘next_3_year’][‘ne-
tIncome’]

CF_forecast[‘next_3_year’][‘inc_depreciation’] = income_statement[‘next_3_year’]
[‘depreciationAndAmortization’] - income_statement[‘next_2_year’][‘depreciatio-
nAndAmortization’]
CF_forecast[‘next_3_year’][‘inc_receivables’] = balance_sheet[‘next_3_year’]
[‘netReceivables’] - balance_sheet[‘next_2_year’][‘netReceivables’]
CF_forecast[‘next_3_year’][‘inc_inventory’] = balance_sheet[‘next_3_year’][‘in-
ventory’] - balance_sheet[‘next_2_year’][‘inventory’]
CF_forecast[‘next_3_year’][‘inc_payables’] = balance_sheet[‘next_3_year’][‘ac-
countPayables’] - balance_sheet[‘next_2_year’][‘accountPayables’]
CF_forecast[‘next_3_year’][‘CF_operations’] = CF_forecast[‘next_3_year’][‘netIn-
come’] + CF_forecast[‘next_3_year’][‘inc_depreciation’] + (CF_forecast[‘next_3_
year’][‘inc_receivables’] * -1) + (CF_forecast[‘next_3_year’][‘inc_inventory’]
*-1) + CF_forecast[‘next_3_year’][‘inc_payables’]
CF_forecast[‘next_3_year’][‘CAPEX’] = balance_sheet[‘next_3_year’][‘property-
PlantEquipmentNet’] - balance_sheet[‘next_2_year’][‘propertyPlantEquipmentNet’]
+ income_statement[‘next_3_year’][‘depreciationAndAmortization’]
CF_forecast[‘next_3_year’][‘FCF’] = CF_forecast[‘next_3_year’][‘CAPEX’] + CF_
forecast[‘next_3_year’][‘CF_operations’]

CF_forecast[‘next_4_year’] = {}
CF_forecast[‘next_4_year’][‘netIncome’] = income_statement[‘next_4_year’][‘ne-
tIncome’]

CF_forecast[‘next_4_year’][‘inc_depreciation’] = income_statement[‘next_4_year’]
[‘depreciationAndAmortization’] - income_statement[‘next_3_year’][‘depreciatio-
nAndAmortization’]
CF_forecast[‘next_4_year’][‘inc_receivables’] = balance_sheet[‘next_4_year’]
[‘netReceivables’] - balance_sheet[‘next_3_year’][‘netReceivables’]
CF_forecast[‘next_4_year’][‘inc_inventory’] = balance_sheet[‘next_4_year’][‘in-
ventory’] - balance_sheet[‘next_3_year’][‘inventory’]
CF_forecast[‘next_4_year’][‘inc_payables’] = balance_sheet[‘next_4_year’][‘ac-
```

```

countPayables'] - balance_sheet['next_3_year']['accountPayables']
CF_forecast['next_4_year']['CF_operations'] = CF_forecast['next_4_year']['netIncome'] + CF_forecast['next_4_year']['inc_depreciation'] + (CF_forecast['next_4_year']['inc_receivables'] * -1) + (CF_forecast['next_4_year']['inc_inventory'] * -1) + CF_forecast['next_4_year']['inc_payables']
CF_forecast['next_4_year']['CAPEX'] = balance_sheet['next_4_year']['propertyPlantEquipmentNet'] - balance_sheet['next_3_year']['propertyPlantEquipmentNet'] + income_statement['next_4_year']['depreciationAndAmortization']
CF_forecast['next_4_year']['FCF'] = CF_forecast['next_4_year']['CAPEX'] + CF_forecast['next_4_year']['CF_operations']

CF_forecast['next_5_year'] = {}
CF_forecast['next_5_year']['netIncome'] = income_statement['next_5_year']['netIncome']

CF_forecast['next_5_year']['inc_depreciation'] = income_statement['next_5_year']['depreciationAndAmortization'] - income_statement['next_4_year']['depreciationAndAmortization']
CF_forecast['next_5_year']['inc_receivables'] = balance_sheet['next_5_year']['netReceivables'] - balance_sheet['next_4_year']['netReceivables']
CF_forecast['next_5_year']['inc_inventory'] = balance_sheet['next_5_year']['inventory'] - balance_sheet['next_4_year']['inventory']
CF_forecast['next_5_year']['inc_payables'] = balance_sheet['next_5_year']['accountPayables'] - balance_sheet['next_4_year']['accountPayables']
CF_forecast['next_5_year']['CF_operations'] = CF_forecast['next_5_year']['netIncome'] + CF_forecast['next_5_year']['inc_depreciation'] + (CF_forecast['next_5_year']['inc_receivables'] * -1) + (CF_forecast['next_5_year']['inc_inventory'] * -1) + CF_forecast['next_5_year']['inc_payables']
CF_forecast['next_5_year']['CAPEX'] = balance_sheet['next_5_year']['propertyPlantEquipmentNet'] - balance_sheet['next_4_year']['propertyPlantEquipmentNet'] + income_statement['next_5_year']['depreciationAndAmortization']
CF_forecast['next_5_year']['FCF'] = CF_forecast['next_5_year']['CAPEX'] + CF_forecast['next_5_year']['CF_operations']

```

9.4.4 Convertir los flujos de caja libres proyectados a pandas

Los flujos de caja libre proyectados los almacenamos en un DataFrame de Pandas.

```
#Llevar los flujos de caja proyectados a un DataFrame de pandas
CF_forec = pd.DataFrame.from_dict(CF_forecast,orient='columns')
```

```
#Formatear el DataFrame con separadores de miles
pd.options.display.float_format = '{:,.0f}'.format

print(CF_forec)
```

Salida

	next_year	next_2_year	next_3_year	next_4_year	\
netIncome	40,627,799,874	48,062,723,776	56,858,245,435	67,263,355,465	
inc_depreciation	2,155,933,591	2,550,471,376	3,017,209,931	3,569,362,061	
inc_receivables	5,031,060,715	5,951,749,349	7,040,924,831	8,329,420,404	
inc_inventory	182,817,898	216,273,738	255,852,026	302,673,177	
inc_payables	1,017,667,999	1,203,962,158	1,424,217,335	1,684,850,388	
CF_operations	38,587,522,851	45,649,074,223	54,002,895,845	63,885,474,333	
CAPEX	29,416,430,635	34,799,663,886	41,168,033,662	48,701,820,832	
FCF	68,063,953,486	80,448,738,109	95,170,929,507	112,587,295,165	
					next_5_year
netIncome		79,572,609,985			
inc_depreciation		4,222,558,527			
inc_receivables		9,853,711,826			
inc_inventory		358,862,641			
inc_payables		1,993,179,524			
CF_operations		75,576,573,568			
CAPEX		57,614,297,827			
FCF		133,190,871,396			

9.4.5 Estimación del costo de capital

Para descontar los flujos de caja libres (FCL) futuros, usaremos como tasa de descuento el WACC. Por lo tanto, necesitamos estimar el costo de capital promedio ponderado (WACC) de la empresa. A continuación, se presenta el respectivo código en Python:

```
import pandas_datareader.data as web
import datetime
import requests

demo = 'digite aquí su api key'
#Interest coverage ratio = EBIT / interest expenses

def interest_coverage_and_RF(company):
    IS= requests.get(f'https://financialmodelingprep.com/api/v3/income-statement/{company}?apikey={demo}').json()
    EBIT= IS[0]['ebitda'] - IS[0]['depreciationAndAmortization']
    interest_expense = IS[0]['interestExpense']
```

```
interest_coverage_ratio = EBIT / interest_expense

#RF o tasa libre de riesgo
start = datetime.datetime(2019, 7, 10)

end= datetime.datetime.today().strftime('%Y-%m-%d')
#end = datetime.datetime(2020, 7, 10)

Treasury = web.DataReader(['TB1YR'], 'fred', start, end)
RF = float(Treasury.iloc[-1])
RF = RF/100
return [RF,interest_coverage_ratio]

#Costo de deuda
def cost_of_debt(company, RF,interest_coverage_ratio):
    if interest_coverage_ratio > 8.5:
        #Rating is AAA
        credit_spread = 0.0063
    if (interest_coverage_ratio > 6.5) & (interest_coverage_ratio <= 8.5):
        #Rating is AA
        credit_spread = 0.0078
    if (interest_coverage_ratio > 5.5) & (interest_coverage_ratio <= 6.5):
        #Rating is A+
        credit_spread = 0.0098
    if (interest_coverage_ratio > 4.25) & (interest_coverage_ratio <= 5.49):
        #Rating is A
        credit_spread = 0.0108
    if (interest_coverage_ratio > 3) & (interest_coverage_ratio <= 4.25):
        #Rating is A-
        credit_spread = 0.0122
    if (interest_coverage_ratio > 2.5) & (interest_coverage_ratio <= 3):
        #Rating is BBB
        credit_spread = 0.0156
    if (interest_coverage_ratio > 2.25) & (interest_coverage_ratio <= 2.5):
        #Rating is BB+
        credit_spread = 0.02
    if (interest_coverage_ratio > 2) & (interest_coverage_ratio <= 2.25):
        #Rating is BB
        credit_spread = 0.0240
    if (interest_coverage_ratio > 1.75) & (interest_coverage_ratio <= 2):
        #Rating is B+
        credit_spread = 0.0351
    if (interest_coverage_ratio > 1.5) & (interest_coverage_ratio <= 1.75):
        #Rating is B
        credit_spread = 0.0421
```

```
if (interest_coverage_ratio > 1.25) & (interest_coverage_ratio <= 1.5):
    #Rating is B-
    credit_spread = 0.0515
if (interest_coverage_ratio > 0.8) & (interest_coverage_ratio <= 1.25):
    #Rating is CCC
    credit_spread = 0.0820
if (interest_coverage_ratio > 0.65) & (interest_coverage_ratio <= 0.8):
    #Rating is CC
    credit_spread = 0.0864
if (interest_coverage_ratio > 0.2) & (interest_coverage_ratio <= 0.65):
    #Rating is C
    credit_spread = 0.1134
if interest_coverage_ratio <= 0.2:
    #Rating is D
    credit_spread = 0.1512

cost_of_debt = RF + credit_spread
return cost_of_debt

def costofequity(company):

    #RF o tasa libre de riesgo
    start = datetime.datetime(2019, 7, 10)
    end= datetime.datetime.today().strftime('%Y-%m-%d')
    #end = datetime.datetime(2020, 7, 10)

    Treasury = web.DataReader(['TB1YR'], 'fred', start, end)
    RF = float(Treasury.iloc[-1])
    RF = RF/100

    #Beta

    beta = requests.get(f'https://financialmodelingprep.com/api/v3/company/profile/{company}?apikey={demo}')
    beta = beta.json()
    beta = float(beta['profile']['beta'])

    #Rentabilidad de mercado
    start = datetime.datetime(2019, 7, 10)
    end= datetime.datetime.today().strftime('%Y-%m-%d')
```

```

SP500 = web.DataReader(['sp500'], 'fred', start, end)
    #Eliminar los valores no numéricos usando el método drop
SP500.dropna(inplace = True)

SP500yearlyreturn = (SP500['sp500'].iloc[-1]/ SP500['sp500'].iloc[-252])-1

cost_of_equity = RF+(beta*(SP500yearlyreturn - RF))
return cost_of_equity

#Tasa efectivo de impuestos y estructura de capital
def wacc(company):
    FR = requests.get(f'https://financialmodelingprep.com/api/v3/ratios/{company}?apikey={demo}'.json())

    ETR = FR[0]['effectiveTaxRate']

    #
    BS = requests.get(f'https://financialmodelingprep.com/api/v3/balance-sheet-statement/{company}?apikey={demo}'.json()

    Debt_to = BS[0]['totalDebt'] / (BS[0]['totalDebt'] + BS[0]['totalStockholdersEquity'])
    equity_to = BS[0]['totalStockholdersEquity'] / (BS[0]['totalDebt'] + BS[0]['totalStockholdersEquity'])

    WACC = (kd*(1-ETR)*Debt_to) + (ke*equity_to)
    return WACC

RF_and_IntCov = interest_coveraga_and_RF(company)
RF = RF_and_IntCov[0]
interest_coverage_ratio = RF_and_IntCov[1]
ke = costofequity(company)
kd = cost_of_debt(company,RF,interest_coverage_ratio)
wacc_company = wacc(company)
print('wacc of ' + company + ' is ' + str((wacc_company*100))+'%')

```

Salida

wacc of GOOG is 35.44699077571782%

<https://dogramcode.com/programacion>

9.4.6 Obtención del valor presente de los flujos de caja libre futuros

Ahora que tenemos el WACC de Google y los flujos de caja libre proyectados para los próximos cinco años, podemos descontarlos para conocer el valor presente. Podemos usar el método numpy npv para hacerlo:

```
#Lista de los flujos de caja libre para cada año
FCF_List = CF_forec.iloc[-1].values.tolist()
npv = npf.npv(wacc_company, FCF_List)
```

9.4.7 Cálculo del valor terminal

El valor presente obtenido en la sección 9.4.6 cubre solo los próximos 5 años de los flujos de caja de la empresa. Pero, ¿qué pasará después de estos años proyectados?

Necesitamos calcular el valor terminal de la empresa. El valor terminal captura el valor de la empresa después del período de pronóstico. El valor terminal constituirá al menos el 50% del valor previsto. Por lo tanto, es muy importante prestar atención a las suposiciones que hacemos aquí. A continuación se muestra la fórmula que usaremos:

$$TV = \frac{FCFF_{n+1}}{(WACC - g)} = \frac{FCFF_n \times (1 + g)}{(WACC - g)}$$

FCFF_n será el último año de los flujos de caja libres proyectados. Entonces, necesitamos llegar a una tasa de crecimiento a perpetuidad g . La tasa de perpetuidad es la tasa constante a la que se espera que crezcan los flujos futuros para siempre.

Asumiremos una tasa de perpetuidad del 2%, ya que es una práctica común estar en línea con la tasa de inflación a largo plazo, que suele rondar el 2%. Luego, descontamos el valor usando la tasa WACC.

```
#Valor terminal
LTGrowth = 0.02

Terminal_value = (CF_forecast['next_5_year']['FCF'] * (1 + LTGrowth)) / (wacc_company - LTGrowth)
```

```
Terminal_value_Discounted = Terminal_value/(1+wacc_company)**4
Terminal_value_Discounted
```

Salida

```
104923793455.33406
```

9.4.8 Calcular el precio objetivo de Google

Finalmente, tenemos todo lo que necesitamos para valorar una empresa utilizando el método de flujo de caja libre descontado y Python. Obtenemos el valor patrimonial objetivo, la deuda de la empresa y el número de acciones.

Luego, para estimar el valor de la acción, simplemente se divide el valor objetivo por la cantidad de acciones:

```
target_equity_value = Terminal_value_Discounted + npv
debt = balance_sheet['current_year'][‘totalDebt’]
target_value = target_equity_value - debt
numero_of_shares = requests.get(f'https://financialmodelingprep.com/api/v3/enter-
prise-values/{company}?apikey={demo}').json()
numero_of_shares = numero_of_shares[0][‘numberOfShares’]

target_price_per_share = target_value/numero_of_shares
target_price_per_share

print(company + ‘ forecasted price per stock is ‘ + str(target_price_per_share)
)
print(‘the forecast is based on the following assumptions: ‘ + ‘revenue growth: ‘
+ str(revenue_g) + ‘ Cost of Capital: ‘ + str(wacc_company) )
print(‘perpetuity growth: ‘ + str(LTGrowth) )
```

Salida

```
GOOG forecasted price per stock is 495.47243748223946
the forecast is based on the following assumptions: revenue growth:
0.12770532012826136 Cost of Capital: 0.35446990775717824
perpetuity growth: 0.02
```

10

REPRESENTACIÓN GRÁFICA Y VISUAL DE DATOS FINANCIEROS EN PYTHON

Uno de los paquetes más populares utilizados para visualizaciones interactivas es Plotly. Plotly está construido sobre Python y permite producir gráficos profesionales y de gran apariencia con menos código. Se hizo popular debido a sus extensas categorías de gráficos que se pueden producir en poco tiempo. Las categorías incluyen gráficos básicos, gráficos estadísticos, gráficos científicos y gráficos financieros.

En este capítulo, se mostrará el proceso de creación de gráficos financieros interactivos y profesionales en Plotly con Python. También exploraremos la API de Yahoo Finance para extraer datos históricos de los precios de las acciones que usaremos para las visualizaciones.

10.1 IMPORTANDO LIBRERÍAS

Las principales librerías que utilizaremos son: Pandas para el procesamiento de datos, Yfinance para extraer los datos históricos de los precios de las acciones, Datetime para gestionar las fechas, finalmente, Plotly y sus dependencias para visualizaciones interactivas. Siga el código para importar las librerías a nuestro entorno de Python.

Implementación de Python:

```
# Importar librerías
```

```
import pandas as pd
```

```
import datetime as dt
import pandas_datareader.data as web
import plotly.express as px
import plotly.graph_objects as go
import yfinance as yf
from datetime import datetime
```

Nuestro próximo proceso será extraer los datos de los precios históricos de las acciones para visualizaciones utilizando las librerías Pandas y Yfinance.

10.2 EXTRAYENDO LOS PRECIOS HISTÓRICOS DE LAS ACCIONES

Para nuestras visualizaciones, vamos a extraer los datos históricos de seis empresas, a saber, Facebook, Amazon, Apple, Netflix, Google y Microsoft utilizando la API de Yahoo. ¡Extraigamos los datos en Python!

Implementación de Python:

```
# Obtener los datos

stocks = yf.download(['FB', 'AMZN', 'AAPL', 'NFLX', 'GOOGL', 'MSFT'], start='2010-01-01', end='2020-11-30', progress=False)
stocks_close = pd.DataFrame(yf.download(['FB', 'AMZN', 'AAPL', 'NFLX', 'GOOGL', 'MSFT'], start='2010-01-01', end='2020-11-30')['Close'])
```

En primer lugar, hemos definido la variable stocks que nos permite especificar los datos que requerimos(empresas y período). Posteriormente, almacenamos solo los datos de precios de cierre de las empresas en la variable “stocks_close”. Ahora, estamos listos para realizar visualizaciones con los precios históricos de las acciones.

10.3 GRÁFICO DE ÁREA

A menudo llamados gráficos de “montaña”, los gráficos de áreas son una interpretación más simplificada de los gráficos de líneas estándar. Trazan los precios de cierre durante un período determinado y el área debajo de la línea está sombreada. Siga el código para crear un gráfico de áreas con Plotly en Python.

Implementación de Python:

Gráfico de área

```
area_chart = px.area(stocks_close.FB, title = 'FACEBOOK SHARE PRICE (2013-2020)')

area_chart.update_xaxes(title_text = 'Date')
area_chart.update_yaxes(title_text = 'FB Close Price', tickprefix = '$')
area_chart.update_layout(showlegend = False)

area_chart.show()
```

Salida



El gráfico de áreas producido anteriormente es muy básico en comparación con las funciones eficientes de Plotly. Ahora, creemos un gráfico de áreas personalizado utilizando las poderosas funciones de Plotly. Siga el código para crear un gráfico de áreas personalizado con Plotly en Python:

Gráfico de área personalizado

```
c_area = px.area(stocks_close.FB, title = 'FACEBOOK SHARE PRICE (2013-2020)')

c_area.update_xaxes(
    title_text = 'Date',
```

```
rangeslider_visible = True,
rangeslider = dict(
    buttons = list([
        dict(count = 1, label = '1M', step = 'month', stepmode = 'backward'),
        dict(count = 6, label = '6M', step = 'month', stepmode = 'backward'),
        dict(count = 1, label = 'YTD', step = 'year', stepmode = 'todate'),
        dict(count = 1, label = '1Y', step = 'year', stepmode = 'backward'),
        dict(step = 'all')]))
c_area.update_yaxes(title_text = 'FB Close Price', tickprefix = '$')
c_area.update_layout(showlegend = False,
    title = {
        'text': 'FACEBOOK SHARE PRICE (2013-2020)',
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'}})
c_area.show()
```

Salida



10.4 GRÁFICO DE VELAS

Una vela es un tipo de gráfico de precios utilizado en el análisis técnico que muestra los precios máximos, mínimo, de apertura y de cierre de un valor durante un período específico. Los traders utilizan los gráficos de velas para determinar posibles movimientos de precios basados en patrones pasados. Las velas son útiles cuando se negocia, ya que muestran cuatro puntos de precio (apertura, cierre, máximo y mínimo) durante el período de tiempo especificado por el trader. Diversos algoritmos se basan en la misma información de precios que se muestra en los gráficos de velas. La negociación de acciones suele estar influenciada por las emociones, que se pueden leer en los gráficos de velas. Siga el código para producir un gráfico de velas con Plotly en Python.

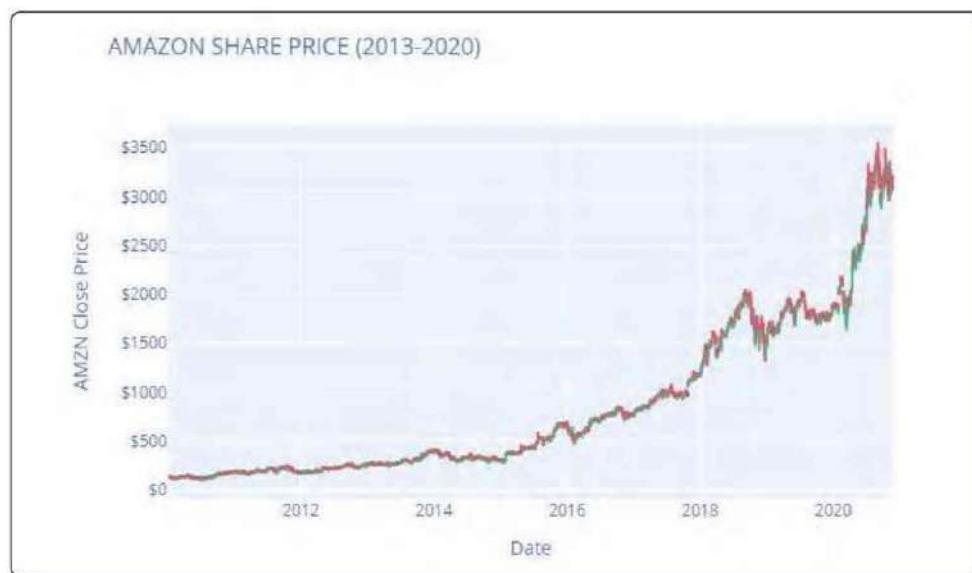
```
# Gráfico de velas

candlestick = go.Figure(data = [go.Candlestick(x = stocks.index,
                                                open = stocks[('Open',      'AMZN')]),
                                                high = stocks[('High',      'AMZN')]),
                                                low = stocks[('Low',       'AMZN')]),
                                                close = stocks[('Close',     'AMZN')])]

candlestick.update_layout(xaxis_rangeslider_visible = False, title = 'AMAZON
SHARE PRICE (2013-2020)')
candlestick.update_xaxes(title_text = 'Date')
candlestick.update_yaxes(title_text = 'AMZN Close Price', tickprefix = '$')

candlestick.show()
```

Salida



Al igual que el gráfico de áreas, los gráficos de velas también se pueden ajustar y personalizar para que se vean más profesionales. Siga el código para producir un gráfico de velas personalizado con Plotly en Python.

```
# Gráfico de velas personalizado

c_candlestick = go.Figure(data = [go.Candlestick(x = stocks.index,
open = stocks[('Open',      'AMZN')], high = stocks[('High',     'AMZN')],
low = stocks[('Low',       'AMZN')], close = stocks[('Close',    'AMZN')]))]

c_candlestick.update_xaxes(
    title_text = 'Date',
    rangeslider_visible = True,
    rangeslider = dict(
        buttons = list([
            dict(count = 1, label = '1M', step = 'month', stepmode = 'backward'),
            dict(count = 6, label = '6M', step = 'month', stepmode = 'backward'),
            dict(count = 1, label = 'YTD', step = 'year', stepmode = 'todate'),
            dict(count = 1, label = '1Y', step = 'year', stepmode = 'backward'),
            dict(step = 'all')])))

c_candlestick.update_layout(
    title = {
        'text': 'AMAZON SHARE PRICE (2013-2020)',
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})

c_candlestick.update_yaxes(title_text = 'AMZN Close Price', tickprefix = '$')
c_candlestick.show()
```

Salida



10.5 GRÁFICO OHLC

Un gráfico OHLC es un tipo de gráfico de barras que muestra los precios de apertura, máximo, mínimo y de cierre para cada período. Los gráficos OHLC son útiles ya que muestran los cuatro puntos de datos principales durante un período, y muchos operadores consideran el precio de cierre como el más importante. El tipo de gráfico es útil porque puede mostrar un impulso creciente o decreciente. Cuando la apertura y el cierre están muy separados, muestra un fuerte impulso, y cuando se abren y cierran, muestran indecisión o un impulso débil. El máximo y el mínimo muestran el rango de precios completo del período, lo que resulta útil para evaluar la volatilidad. Siga el código para producir un gráfico OHLC con Plotly en Python.

Implementación en Python:

```
# Gráfico OHLC

ohlc = go.Figure(data = [go.Ohlc(x = stocks.index,
                                    open = stocks[['Open',      'AAPL']],
                                    high = stocks[['High',     'AAPL']],
                                    low = stocks[['Low',      'AAPL']],
                                    close = stocks[['Close',    'AAPL']]))

ohlc.update_layout(xaxis_rangeslider_visible = False, title = 'APPLE SHARE PRICE (2013-2020)')
ohlc.update_xaxes(title_text = 'Date')
ohlc.update_yaxes(title_text = 'AAPL Close Price', tickprefix = '$')

ohlc.show()
```

Salida



Ahora, personalicemos el gráfico OHLC predeterminado usando las funciones de Plotly para hacer que el gráfico sea más profesional y atractivo. Siga el código para crear un gráfico OHLC personalizado con Plotly en Python.

Implementación en Python:

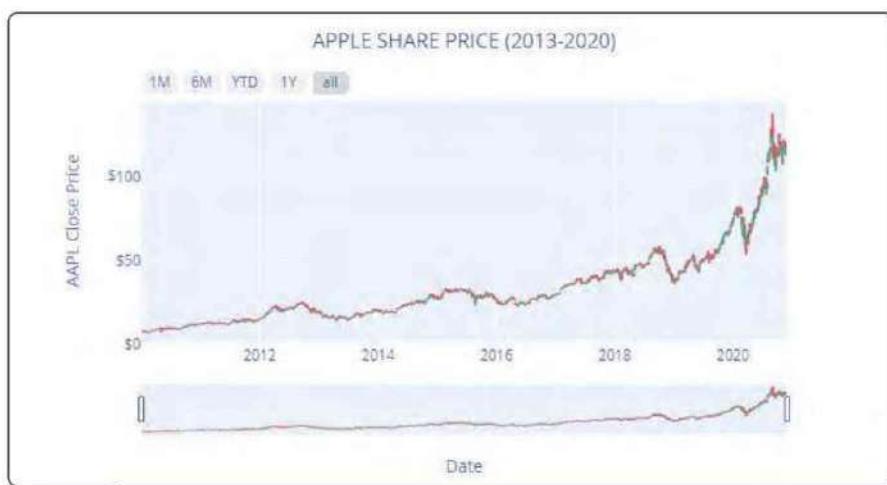
```
# Gráfico OHLC personalizado

c_ohlc = go.Figure(data = [go.Ohlc(x = stocks.index,
                                      open = stocks[('Open',      'AAPL')]),
                                      high = stocks[('High',     'AAPL')]),
                                      low = stocks[('Low',      'AAPL')]),
                                      close = stocks[('Close',    'AAPL')])])

c_ohlc.update_xaxes(
    title_text = 'Date',
    rangeslider_visible = True,
    rangeselector = dict(
        buttons = list([
            dict(count = 1, label = '1M', step = 'month', stepmode = 'backward'),
            dict(count = 6, label = '6M', step = 'month', stepmode = 'backward'),
            dict(count = 1, label = 'YTD', step = 'year', stepmode = 'todate'),
            dict(count = 1, label = '1Y', step = 'year', stepmode = 'backward'),
            dict(step = 'all')])))

c_ohlc.update_layout(
    title = {
        'text': 'APPLE SHARE PRICE (2013-2020)',
        'y': 0.9,
        'x': 0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
c_ohlc.update_yaxes(title_text = 'AAPL Close Price', tickprefix = '$')
c_ohlc.show()
```

Salida



10.6 GRÁFICO BULLET

Un gráfico Bullet es una variación de un gráfico de barras diseñado para comparar una única medida principal con una o más medidas para enriquecer su significado y lo muestra en el contexto de rangos cualitativos de desempeño. Los rangos cualitativos se muestran como bloques de un tono, pero con intensidad variable, haciéndolos discernibles para aquellos que son daltónicos y para restringir el uso de colores en el tablero al mínimo. Vamos a utilizar un gráfico bullet para representar el rango de días de una acción. Siga el código para crear un gráfico bullet con Plotly en Python. Implementación en Python:

```
# Gráfico Bullet personalizado

c_bullet = go.Figure()

c_bullet.add_trace(go.Indicator(
    mode = "number+gauge+delta",
    value = int(stocks_close['NFLX'].tail(1)),
    delta = {'reference': int(stocks_close['NFLX'].tail(2)[0])},
    domain = {'x': [0.25, 1],
              'y': [0.08, 0.25]},
    title = {'text': "<b>NETFLIX DAY<br>RANGE</b><br><span style='color: gray; font-size:0.8em'>U.S. $</span>",
              'font': {"size": 14}},
    gauge = {
        'shape': "bullet",
        'axis': {'range': [None, 550]},
        'threshold': {
            'line': {'color': "Red", 'width': 2},
            'thickness': 0.75,
            'value': 505},
        'steps': [
            {'range': [0, 350], 'color': "gray"},
            {'range': [350, 550], 'color': "lightgray"}],
        'bar': {'color': 'black'}}))

c_bullet.add_trace(go.Indicator(
    mode = "number+gauge+delta",
    value = int(stocks_close['GOOGL'].tail(1)),
    delta = {'reference': int(stocks_close['GOOGL'].tail(2)[0])},
    domain = {'x': [0.25, 1],
              'y': [0.4, 0.6]},
    title = {"text": "<b>GOOGLE DAY<br>RANGE</b><br><span style='color: gray; font-size:0.8em'>U.S. $</span>",
              'font': {"size": 14}},
    gauge = {
        'shape': "bullet",
        'axis': {'range': [None, 1800]},
```

```

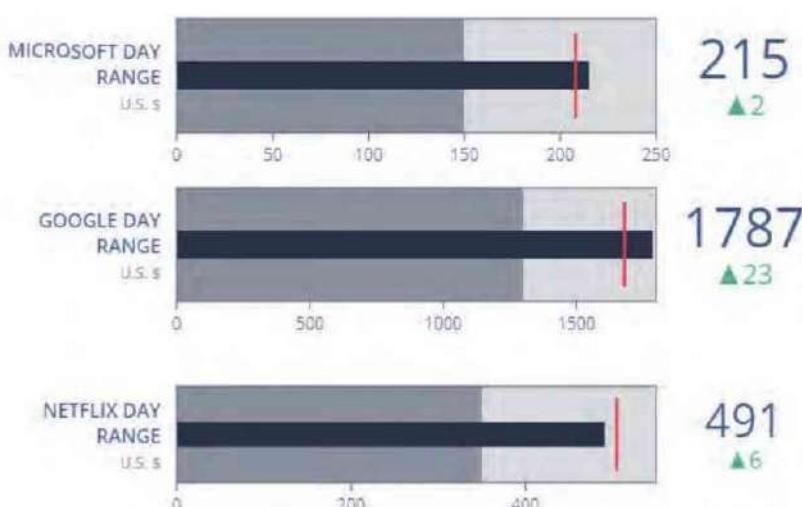
'threshold': {
    'line': {'color': "red", 'width': 2},
    'thickness': 0.75,
    'value': 1681},
'steps': [
    {'range': [0, 1300], 'color': "gray"},
    {'range': [1300, 1800], 'color': "lightgray"}],
'bar': {'color': 'black'}}))

c_bullet.add_trace(go.Indicator(
    mode = "number+gauge+delta",
    value = int(stocks_close['MSFT'].tail(1)),
    delta = {'reference': int(stocks_close['MSFT'].tail(2)[0])},
    domain = {'x': [0.25, 1],
               'y': [0.7, 0.9]},
    title = {'text': "<b>MICROSOFT DAY<br>RANGE</b><br><span style='color: gray; font-size:0.8em'>U.S. $</span>",
              'font': {"size": 14}},
    gauge = {
        'shape': "bullet",
        'axis': {'range': [None, 250]},
        'threshold': {
            'line': {'color': "red", 'width': 2},
            'thickness': 0.75,
            'value': 208},
        'steps': [
            {'range': [0, 150], 'color': "gray"},
            {'range': [150, 250], 'color': "lightgray"}],
        'bar': {'color': "black"}})

c_bullet.update_layout(height = 400 , margin = {'t':0, 'b':0, 'l':0})
c_bullet.show()

```

Salida



10.7 GRÁFICO DE CALIBRE RADIAL

Un gráfico de calibre radial tiene un arco circular, que muestra un valor único para estimar el progreso hacia una meta. La barra muestra el valor objetivo y el sombreado representa el progreso hacia ese objetivo. Gráficos de calibre, también conocidos como gráficos de velocímetro. Usando un gráfico de indicadores, vamos a representar el rango diario de una acción tal como usamos el gráfico bullet. Siga el código para producir un gráfico de indicadores con Plotly en Python.

```
# Gráfico de calibre

gauge = go.Indicator(
    domain = {'x': [0, 1],
              'y': [0, 1]},
    value = int(stocks_close['FB'].tail(1)),
    mode = "gauge+number+delta",
    title = {'text': "<b>FACEBOOK DAY RANGE</b><br><span style='color: gray; font-size:0.8em'>U.S. $</span>",
              'font': {"size": 20}},
    delta = {'reference': int(stocks_close['FB'].tail(2)[0])},
    gauge = {
        'axis': {'range': [None, 300]},
        'steps' : [
            {'range': [0, 200], 'color': "lightgray"},
            {'range': [200, 300], 'color': "gray"}],
        'threshold' : {'line': {"color": "red", 'width': 4},
                      'thickness': 0.75,
                      'value': 276}})

gauge.show()
```

Salida



11

ASIGNACIÓN DE ACTIVOS PARA UN PORTAFOLIO EFICIENTE EN PYTHON

La asignación de activos en un portafolio es la decisión más importante que cualquier inversionista debe enfrentar, y no existe una solución única que pueda funcionar para todos y cada uno de los inversionistas. Por asignación de activos, nos referimos a distribuir el monto total de la inversión del inversionista sobre ciertos activos (ya sean acciones, opciones, bonos o cualquier otro instrumento financiero). Al considerar la asignación, el inversionista desea equilibrar el riesgo y la rentabilidad potencial. Al mismo tiempo, la asignación depende de factores como los objetivos individuales (rendimiento esperado), la tolerancia al riesgo (cuánto riesgo está dispuesto a aceptar el inversionista) o el horizonte de inversión (inversión a corto o largo plazo).

El marco clave en la asignación de activos es la teoría moderna de portafolio (MPT por sus siglas en inglés: modern portfolio theory, también conocida como análisis de varianza media). Fue presentado por el ganador del Premio Nobel Harry Markowitz y describe cómo los inversionistas con aversión al riesgo pueden construir portafolios para maximizar sus expectativas de rentabilidad para un determinado nivel de riesgo. La idea principal de MPT es que los inversionistas no deben evaluar el desempeño de un activo solo (por métricas como el rendimiento esperado o la volatilidad), sino investigar cómo se afectaría el desempeño de su portafolio de activos.

MPT está estrechamente relacionado con el concepto de diversificación, lo que simplemente significa que poseer diferentes tipos de activos reduce el riesgo, ya que la pérdida o ganancia de un valor en particular tiene menos impacto en el rendimiento del portafolio. Otro concepto clave a tener en cuenta es que mientras el rendimiento del portafolio es el promedio ponderado de los rendimientos de los

activos individuales, esto no es cierto para el riesgo (volatilidad). También depende de las correlaciones entre los activos. Lo interesante es que gracias a la asignación de activos optimizada, es posible tener una portafolio con una volatilidad más baja que la volatilidad individual más baja de los activos en el portafolio. En principio, cuanto menor sea la correlación entre los activos que poseemos, mejor será para la diversificación. Con una correlación negativa perfecta, podríamos diversificar todo el riesgo.

Los principales supuestos de la teoría moderna de portafolios son:

- ▶ Los inversionistas son racionales y buscan maximizar sus rendimientos, evitando riesgos siempre que sea posible
- ▶ Los inversionistas comparten el objetivo de maximizar sus rendimientos esperados
- ▶ Todos los inversionistas tienen el mismo nivel de información sobre posibles inversiones
- ▶ Las comisiones, los impuestos y los costos de transacción no se tienen en cuenta
- ▶ Los inversionistas pueden pedir prestado y prestar dinero (sin límites) a una tasa libre de riesgo

En este capítulo, comenzamos con la estrategia de asignación de activos más básica y, sobre esta base, aprenderemos cómo evaluar el desempeño de los portafolios (también aplicable a los activos individuales). Más adelante, mostramos dos enfoques diferentes para obtener la Frontera Eficiente, al tiempo que relajamos algunos de los supuestos de MPT. Uno de los principales beneficios de aprender a abordar los problemas de optimización es que se pueden modificar fácilmente, por ejemplo, optimizando una función objetivo diferente. Esto requiere solo ligeras modificaciones al código, mientras que la mayoría del marco permanece igual.

En este capítulo se abordarán los siguientes temas:

- ▶ Evaluación del rendimiento de un portafolio básico de 1 / n
- ▶ Encontrar la frontera eficiente usando simulaciones de Monte Carlo
- ▶ Encontrar la frontera eficiente utilizando la optimización con `scipy` (librería o paquete científico de Python)

11.1 EVALUAR EL RENDIMIENTO DE UN 1 / N PORTAFOLIO BÁSICO

Comenzamos con la revisión de la estrategia de asignación de activos más básica: la de portafolio 1/n. La idea es asignar participaciones porcentuales iguales a todos los activos considerados, diversificando así el portafolio. Por simple que parezca, DeMiguel, Garlappi y Uppal (2007) muestran que puede ser difícil superar el rendimiento de la estrategia portafolio 1/n utilizando estrategias de asignación de activos más avanzadas.

El objetivo de la explicación es mostrar cómo crear un portafolio de 1/n, calcular su rendimiento y luego usar una librería de Python llamada pyfolio para obtener rápidamente todas las métricas de evaluación de portafolio relevantes en forma de una hoja de lágrimas (resumen de una página con información relevante de la empresa).

Cómo hacerlo

Ejecute los siguientes pasos para crear y evaluar la portafolio 1/n.

```
# 1. Importar librerías
import yfinance as yf
import numpy as np
import pandas as pd
import pyfolio as pf

# 2. Configurar los parámetros
RISKY_ASSETS = ['AAPL', 'IBM', 'MSFT', 'TWTR']
START_DATE = '2017-01-01'
END_DATE = '2018-12-31'
n_assets = len(RISKY_ASSETS)

# 3. Descargue los precios de las acciones de Yahoo Finance:
prices_df = yf.download(RISKY_ASSETS, start=START_DATE,
                        end=END_DATE, adjusted=True)

# 4. Calcular los rendimientos de los activos individuales:
returns = prices_df['Adj Close'].pct_change().dropna()

# 5. Definir los pesos o participaciones
portfolio_weights = n_assets * [1 / n_assets]

# 6. Calcule los rendimientos del portafolio:
portfolio_returns = pd.Series(np.dot(portfolio_weights, returns.T),
                               index=returns.index)

# 7. Crear la hoja de lágrimas
pf.create_simple_tear_sheet(portfolio_returns)
```

Salida

Start date	2017-01-04
End date	2018-12-28
Total months	23
Backtest	
Annual return	17.7%
Cumulative returns	38.1%
Annual volatility	21.8%
Sharpe ratio	0.86
Calmar ratio	0.70
Stability	0.87
Max drawdown	-25.3%
Omega ratio	1.17
Sortino ratio	1.21
Skew	-0.29
Kurtosis	3.93
Tail ratio	0.88

Describimos los indicadores presentados en la primera tabla de la hoja de lágrimas en la siguiente sección.

Cómo funciona

En los Pasos 1 a 4, seguimos el enfoque ya establecido: importamos las librerías, configuramos los parámetros, descargamos los precios de las acciones de cuatro compañías tecnológicas estadounidenses (Apple, IBM, Microsoft y Twitter) durante los años 2017-2018, y calculamos los rendimientos simples, utilizando los precios de cierre ajustados.

En el Paso 5, creamos una lista de ponderaciones, cada una igual a $1/n_{\text{activos}}$, donde n_{activos} es la cantidad de activos que queremos tener en nuestro portafolio. A continuación en el paso 6, calculamos los rendimientos del portafolio como una multiplicación matricial (también conocida como el producto punto —np.dot) de las participaciones porcentuales del portafolio y una matriz transpuesta de los rendimientos de los activos. Para transponer la matriz, utilizamos el método

T de un DataFrame de pandas. Luego, almacenamos los rendimientos del portafolio como un objeto de la serie pandas, porque esa es la entrada para el siguiente paso.

Por último en el paso 7, creamos una hoja de lágrimas usando pf.create_simple_tear_sheet. Decidimos usar una variante simple, que contiene las métricas más relevantes.

Los indicadores más importantes que vimos en la tabla anterior son:

Relación de Sharpe: una de las métricas de evaluación de desempeño más populares, mide el exceso de rendimiento (por encima de la tasa libre de riesgo) por unidad de desviación estándar. Cuando no se proporciona una tasa libre de riesgo, el supuesto predeterminado es que es igual al 0%. Cuanto mayor sea el índice de Sharpe, mejor será el rendimiento ajustado al riesgo del portafolio.

Max drawdown: una medida del riesgo a la baja de un portafolio mide la mayor pérdida de pico a valle (expresada como porcentaje) durante el curso de la inversión. Cuanto menor sea la reducción máxima, mejor.

Índice de Calmar: El índice se define como la tasa de rendimiento anual compuesta promedio dividida por la reducción máxima para ese mismo período de tiempo. Cuanto mayor sea la relación, mejor.

Relación Omega: la relación ponderada por probabilidad de ganancias sobre pérdidas para un umbral de retorno determinado (predeterminado establecido en 0). Su principal ventaja sobre el índice de Sharpe es que el índice de Omega, por construcción, considera todos los momentos de la distribución de rendimientos, mientras que el primero solo considera los dos primeros (media y varianza).

Relación de Sortino: una versión modificada de la relación de Sharpe, donde la desviación estándar en el denominador se reemplaza por una desviación a la baja. La desviación a la baja es similar a la desviación estándar; sin embargo, solo considera rendimientos negativos: descarta todos los cambios positivos de serie. También nos permite definir diferentes niveles de rendimientos mínimos aceptables (dependiendo del inversionista), y los rendimientos por debajo de ese umbral se utilizan para calcular la desviación a la baja.

Inclinación: la oblicuidad mide el grado de asimetría, es decir, cuánto es la distribución dada (aquí, de los rendimientos del portafolio) más sesgada que la distribución Normal. La asimetría negativa (distribuciones sesgadas a la izquierda) significa que grandes rendimientos negativos ocurren con mayor frecuencia que los grandes positivos.

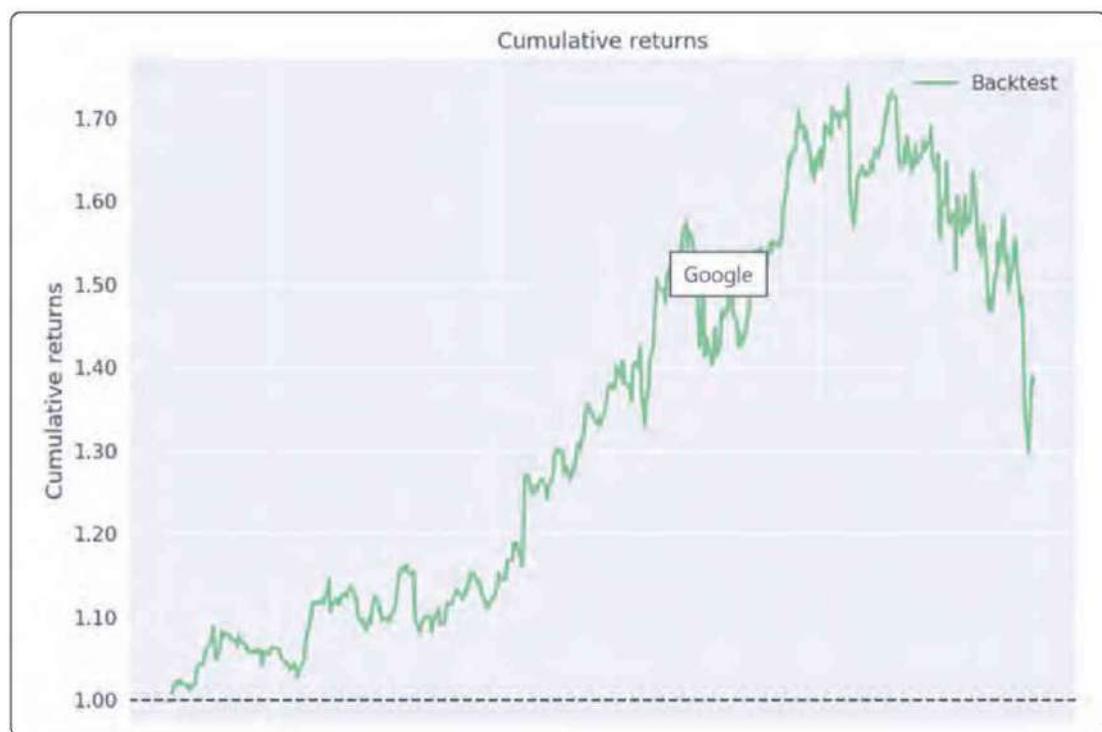
Curtosis: mide valores extremos en cualquiera de las colas. Las distribuciones con curtosis grande exhiben datos de cola que exceden las colas de la distribución gaussiana, lo que significa que los rendimientos grandes y pequeños ocurren con mayor frecuencia.

Relación de la cola: la relación (absoluta) entre el percentil 95 y 5 de los rendimientos diarios. Una relación de cola de ~ 0.8 significa que las pérdidas son ~ 1.25 veces más malas que las ganancias.

Valor diario en riesgo: Calculado como $\mu - 2\sigma$, donde μ es el rendimiento promedio del portafolio durante el período y σ la desviación estándar correspondiente.

La hoja de lágrimas también contiene los siguientes gráficos:

Gráfico de rentabilidad acumulada: presenta la evolución del valor de la portafolio a lo largo del tiempo:



Rolling Sharpe ratio: en lugar de informar un número a lo largo del tiempo, también es interesante ver qué tan estable era el ratio de Sharpe. Es por eso que el siguiente gráfico presenta esta métrica calculada de forma continua, utilizando datos de 6 meses:

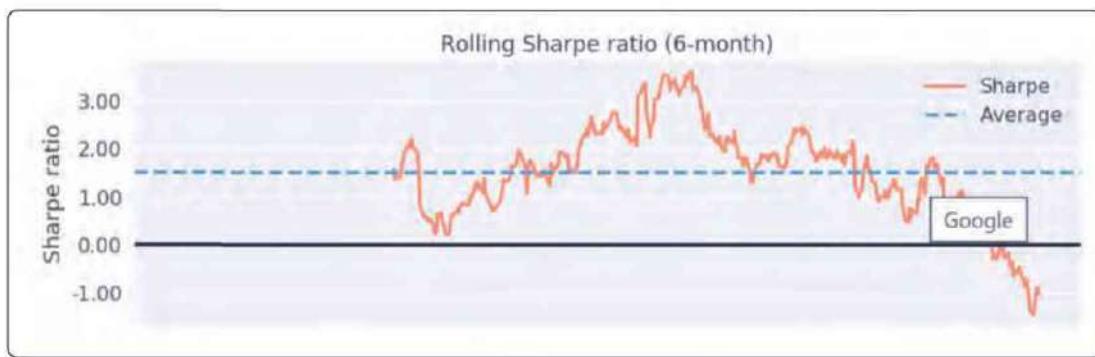
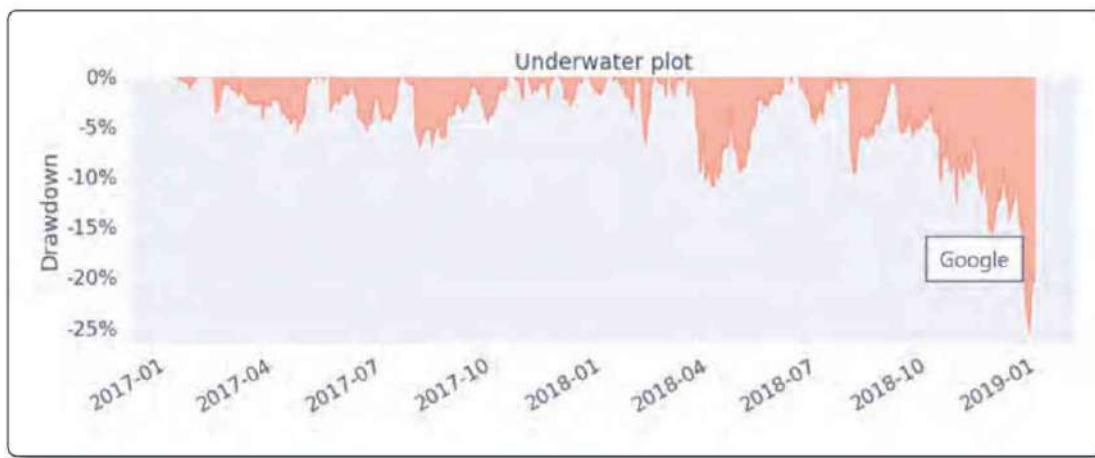


Gráfico subacuático (también conocido como curva de equidad subacuática): este gráfico presenta la inversión desde un punto de vista pesimista, ya que se centra en las pérdidas. Traza todos los períodos de reducción y cuánto duraron, hasta que el valor se recuperó a un nuevo máximo. Una de las ideas que podemos extraer de esto es cuánto duraron los períodos de pérdidas:



Para obtener todos los gráficos, utilizamos la función `pf.create_simple_tear_sheet`. Sin embargo, también podemos obtener solo las gráficas seleccionadas de las hojas de lágrimas. Por ejemplo, para crear el gráfico de relación de Sharpe continuo visto anteriormente, podemos usar la función `pf.plot_rolling_sharpe`. Al llamar a las funciones específicas directamente, podemos hacer que las gráficas se adapten más a nuestras necesidades. En este caso, uno de los parámetros de la función corresponde a la longitud de la ventana móvil.

Anteriormente, utilizamos la función `pf.create_simple_tear_sheet`, ya que contiene muchas métricas útiles. Para obtener aún más detalles, podemos usar la función `pf.create_returns_tear_sheet`.

Algunas de las nuevas características interesantes incluyen:

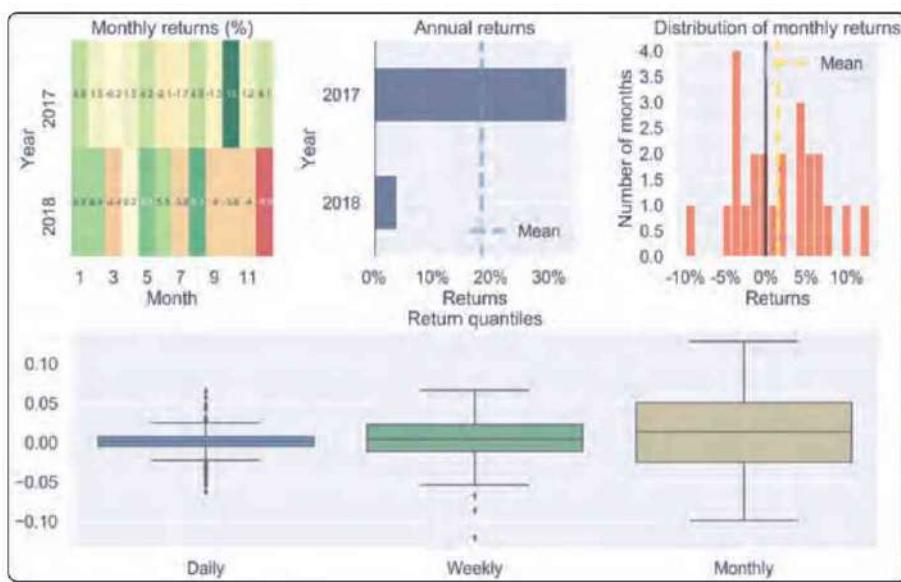
Una tabla con los cinco principales períodos de reducción: qué tan mala fue la reducción, las fechas pico / valle, la fecha de recuperación y la duración. Esta tabla complementa el análisis de la gráfica subacuática y se muestra en la siguiente imagen:

Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	25.32	2018-07-25	2018-12-24	NaT	NaN
1	10.93	2018-03-12	2018-04-02	2018-06-01	60
2	6.93	2017-07-20	2017-07-31	2017-10-12	61
3	6.71	2018-02-01	2018-02-05	2018-02-12	8
4	5.46	2017-02-08	2017-04-19	2017-05-01	59

Los cinco principales períodos de reducción también se visualizan en un diagrama separado:



Gráficos que describen la distribución de los rendimientos del portafolio: un resumen de cuáles fueron los rendimientos durante ciertos meses / años, cómo se distribuyeron los rendimientos mensuales y los cuantiles de los rendimientos, utilizando diferentes frecuencias. Esta información se presenta a continuación:



11.2 ENCONTRAR LA FRONTERA EFICIENTE USANDO SIMULACIÓN MONTECARLO

Según la teoría moderna de portafolio, la frontera eficiente es un conjunto de portafolios óptimos en el espectro de riesgo-rentabilidad. Esto significa que los portafolios en la frontera:

- ▶ Ofrecen el mayor rendimiento esperado para un determinado nivel de riesgo.
- ▶ Ofrecer el menor nivel de riesgo para un determinado nivel de rentabilidad esperada.

Todas los portafolios ubicados bajo la curva frontera eficiente se consideran subóptimos, por lo que siempre es mejor elegir las que están en la frontera.

En este apartado, se muestra cómo encontrar la frontera eficiente utilizando simulaciones de Monte Carlo. Creamos miles de portafolios, utilizando participaciones porcentuales asignadas al azar, y visualizamos los resultados. Para hacerlo, utilizamos los rendimientos de cuatro compañías tecnológicas de EE. UU. A partir de 2018.

Cómo hacerlo

Ejecute los siguientes pasos para encontrar la frontera eficiente utilizando simulaciones de Monte Carlo.

```
# 1. Importar librerías
import yfinance as yf
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

# 2. Configurar los parámetros
N_PORTFOLIOS = 10 ** 5
N_DAYS = 252
RISKY_ASSETS = ['FB', 'TSLA', 'TWTR', 'MSFT']
RISKY_ASSETS.sort()
START_DATE = '2018-01-01'
END_DATE = '2018-12-31'
n_assets = len(RISKY_ASSETS)

#3. Descargue los precios de las acciones de Yahoo Finance:

prices_df = yf.download(RISKY_ASSETS, start=START_DATE,
end=END_DATE, adjusted=True)
```

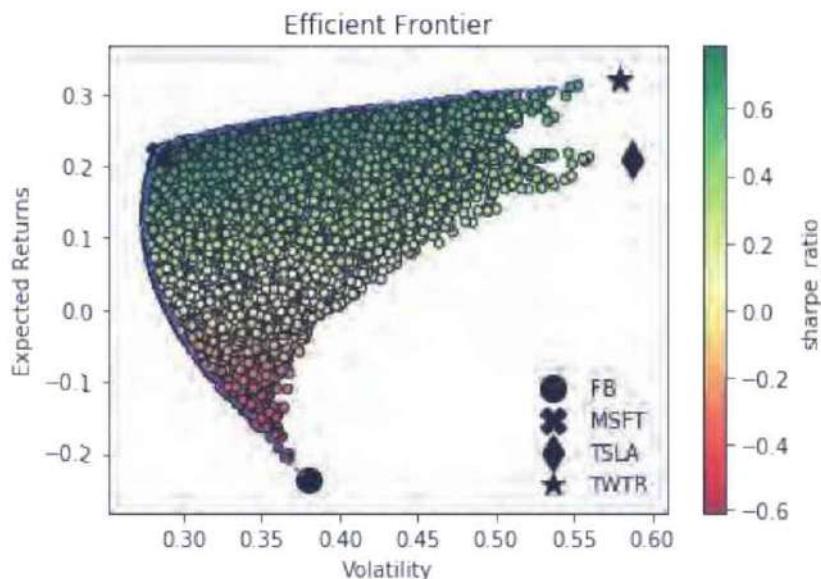
```
# 4. Calcule los rendimientos promedio anualizados y la desviación estándar co-
rrespondiente:
returns_df = prices_df['Adj Close'].pct_change().dropna()
avg_returns = returns_df.mean() * N_DAYS
cov_mat = returns_df.cov() * N_DAYS
# 5. Simule ponderaciones de portafolio aleatorias:
np.random.seed(42)
weights = np.random.random(size=(N_PORTFOLIOS, n_assets))
weights /= np.sum(weights, axis=1)[:, np.newaxis]
# 6. Calcule las métricas del portafolio:
portf_rtns = np.dot(weights, avg_returns)
portf_vol = []
for i in range(0, len(weights)):
    portf_vol.append(np.sqrt(np.dot(weights[i].T,
        np.dot(cov_mat, weights[i]))))
portf_vol = np.array(portf_vol)
portf_sharpe_ratio = portf_rtns / portf_vol

# 7. Cree un DataFrame que contenga todos los datos:
portf_results_df = pd.DataFrame({'returns': portf_rtns,
    'volatility': portf_vol,
    'sharpe_ratio':
    portf_sharpe_ratio})
# 8. Localice los puntos que crean la frontera eficiente:
N_POINTS = 100
portf_vol_ef = []
indices_to_skip = []
portf_rtns_ef = np.linspace(portf_results_df.returns.min(),
    portf_results_df.returns.max(),
    N_POINTS)
portf_rtns_ef = np.round(portf_rtns_ef, 2)
portf_rtns = np.round(portf_rtns, 2)
for point_index in range(N_POINTS):
    if portf_rtns_ef[point_index] not in portf_rtns:
        indices_to_skip.append(point_index)
        continue
    matched_ind = np.where(portf_rtns ==
                           portf_rtns_ef[point_index])
    portf_vol_ef.append(np.min(portf_vol[matched_ind]))
    portf_rtns_ef = np.delete(portf_rtns_ef, indices_to_skip)
# 9. Graficar la frontera eficiente
MARKS = ['o', 'X', 'd', '*']
fig, ax = plt.subplots()
portf_results_df.plot(kind='scatter', x='volatility',
    y='returns', c='sharpe_ratio',
```

```
cmap='RdYlGn', edgecolors='black',
ax=ax)
ax.set(xlabel='Volatility',
       ylabel='Expected Returns',
       title='Efficient Frontier')
ax.plot(portf_vol_ef, portf_rtms_ef, 'b--')
for asset_index in range(n_assets):
    ax.scatter(x=np.sqrt(cov_mat.iloc[asset_index, asset_index]),
               y=avg_returns[asset_index],
               marker=MARKS[asset_index],
               s=150,
               color='black',
               label=RISKY_ASSETS[asset_index])
ax.legend()
```

La ejecución del código anterior genera el gráfico con todos los portafolios creados al azar, cuatro puntos que indican los activos individuales y la frontera eficiente:

Salida



En la gráfica anterior, vemos la típica forma de bala de la frontera eficiente.

Cómo funciona

En el Paso 2, definimos los parámetros utilizados, como el marco temporal considerado, los activos de riesgo que queríamos usar para construir el portafolio y la cantidad de simulaciones. Una cosa importante a tener en cuenta aquí es que también ejecutamos RISKY_ASSETS.sort(), para ordenar la lista alfabéticamente. Esto es importante cuando se interpretan los resultados, ya que cuando se descargan datos de Yahoo Finance utilizando la librería yfinance, los precios obtenidos se ordenan alfabéticamente, no como se especifica en la lista provista. En los pasos 3 y 4 se descargaron los precios de las acciones, calculamos rendimientos simples usando el método pct_change de un DataFrame de pandas, y se descartó la primera fila que contiene NaNs.

Para evaluar los portafolios potenciales, necesitábamos el rendimiento anual promedio (esperado) y la matriz de covarianza correspondiente. Los obtuvimos utilizando los métodos mean() y cov() del DataFrame. También anualizamos ambas métricas multiplicándolas por 252 (el número promedio de días de negociación en un año).

En el Paso 5, calculamos las participaciones de los portafolios aleatorios. Siguiendo los supuestos del MPT, los porcentajes de participación deben ser positivos y sumar 1. Para lograr esto, primero generamos una matriz de números aleatorios (entre 0 y 1), usando np.random.random. La matriz era de tamaño N_SIMULATIONS x N_ASSETS. Para asegurarnos de que los porcentajes de participación sumaran 1, dividimos cada fila de la matriz por su suma.

En el Paso 6, calculamos las métricas del portafolio: rendimientos y desviación estándar. Para calcular los rendimientos anuales esperados del portafolio, tuvimos que multiplicar las participaciones por los promedios anuales calculados previamente. Para las desviaciones estándar, tuvimos que usar la siguiente fórmula: $W^T \Sigma W$, donde W es el vector de participaciones porcentuales y Σ es la matriz de covarianza histórica. Para calcular la desviación estándar, iteramos sobre todos los portafolios simuladas, usando un bucle for.

La implementación del bucle for es en realidad más rápida que el equivalente de matriz vectorizada: np.diag(np.sqrt(np.dot(weights, np.dot(cov_mat, Euros.T)))). La razón de esto es el rápido aumento del número de elementos fuera de la diagonal que se calcularán, lo que, al final, no importa para las métricas de interés. Este enfoque es más rápido que el bucle for solo para un número relativamente pequeño de simulaciones (~ 100).

Para este ejemplo, supusimos que la tasa libre de riesgo era del 0%, por lo que la relación de Sharpe del portafolio podría calcularse como rentabilidad /

volatilidad del portafolio. Otro enfoque posible sería calcular la tasa anual promedio libre de riesgo durante 2018 y utilizar el exceso de rentabilidad del portafolio para calcular la relación.

Los últimos tres pasos llevaron a visualizar los resultados. Primero, colocamos todas las métricas relevantes en un DataFrame de pandas. Segundo, creamos una matriz de rendimientos esperados de la muestra. Para hacerlo, utilizamos np.linspace, con los valores mínimo y máximo de los rendimientos calculados del portafolio. Redondeamos los números a dos decimales, para hacer los cálculos más suaves. Para cada rendimiento esperado, encontramos la volatilidad mínima observable. En los casos en que no hubo coincidencia, como puede suceder con puntos igualmente extendidos en el espacio lineal, omitimos ese punto.

En el último paso, trazamos los portafolios simulados, los activos individuales y la frontera eficiente aproximada. La forma de la frontera era un poco irregular, lo que se puede esperar cuando se usan solo valores simulados que no son tan frecuentes en algunas zonas extremas. Además, coloreamos los puntos que representan los portafolios simulados por el valor de la relación de Sharpe.

Hay más

Habiendo simulado 100,000 portafolios aleatorios, también podemos investigar cuál tiene la relación de Sharpe más alta (rendimiento máximo esperado por unidad de riesgo, también conocida como la portafolio de tangencia) o volatilidad mínima. Para ubicar estos portafolios entre los simulados, usamos np.argmin y np.argmax, que devuelven el índice de un valor mínimo / máximo en la matriz.

El código es el siguiente:

```
max_sharpe_ind = np.argmax(portf_results_df.sharpe_ratio)
max_sharpe_portf = portf_results_df.loc[max_sharpe_ind]
min_vol_ind = np.argmin(portf_results_df.volatility)
min_vol_portf = portf_results_df.loc[min_vol_ind]
#También podemos investigar los componentes de estas portafolios:
print('Maximum Sharpe ratio portfolio ----')
print('Performance')
for index, value in max_sharpe_portf.items():
    print(f'{index}: {100 * value:.2f}%', end='', flush=True)
print('\nWeights')
for x, y in zip(RISKY_ASSETS,
                weights[np.argmax(portf_results_df.sharpe_ratio)]):
    print(f'{x}: {100*y:.2f}%', end='', flush=True)
```

Salida

```
Maximum Sharpe ratio portfolio ----
Performance
returns: 23.42% volatility: 29.77% sharpe_ratio: 78.68%
Weights
FB: 0.01% MSFT: 75.18% TSLA: 5.80% TWTR: 19.00%
```

El portafolio de relación máxima de Sharpe asigna la mayoría de los recursos (~ 75%) a Microsoft y prácticamente nada a Facebook. Esto se debe a que el rendimiento promedio anualizado de Facebook para 2018 fue negativo.

Para calcular el portafolio de volatilidad mínima ejecutamos el siguiente código:

```
print('Minimum Volatility portfolio ----')
print('Performance')
for index, value in min_vol_portf.items():
    print(f'{index}: {100 * value:.2f}%', end='', flush=True)
print('\nWeights')
for x, y in zip(RISKY_ASSETS, weights[np.argmin(portf_results_df.volatility)]):
    print(f'{x}: {100*y:.2f}%', end='', flush=True)
```

Salida

```
Minimum Volatility portfolio ----
Performance
returns: 13.45% volatility: 27.56% sharpe_ratio: 48.79%
Weights
FB: 17.80% MSFT: 78.75% TSLA: 2.94% TWTR: 0.50%
```

El portafolio de volatilidad mínima asigna ~ 79% del peso a Microsoft, ya que es la acción con la volatilidad más baja.

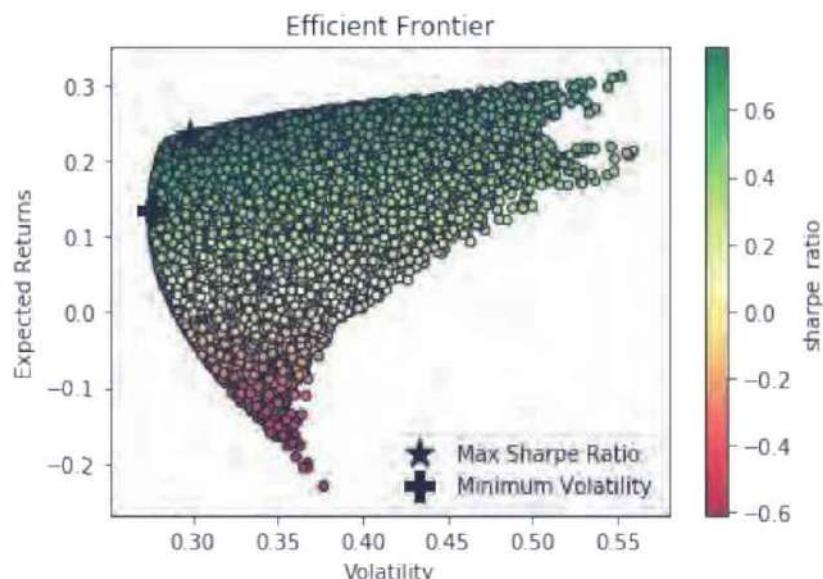
Por último, marcamos estos dos portafolios en el diagrama Frontera Eficiente. Para hacerlo, agregamos dos diagramas de dispersión adicionales, cada uno con un punto correspondiente al portafolio seleccionado. Luego definimos la forma del marcador con el argumento marcador, y el tamaño del marcador con el argumento de s. Aumentamos el tamaño de los marcadores para que los portafolios sean más visibles entre todos los otros.

El código es el siguiente:

```
fig, ax = plt.subplots()
portf_results_df.plot(kind='scatter', x='volatility',
                      y='returns', c='sharpe_ratio',
                      cmap='RdYlGn', edgecolors='black',
                      ax=ax)
ax.scatter(x=max_sharpe_portf.volatility,
            y=max_sharpe_portf.returns,
            c='black', marker='*',
            s=200, label='Max Sharpe Ratio')
ax.scatter(x=min_vol_portf.volatility,
            y=min_vol_portf.returns,
            c='black', marker='P',
            s=200, label='Minimum Volatility')
ax.set(xlabel='Volatility', ylabel='Expected Returns',
       title='Efficient Frontier')
ax.legend()
```

La ejecución del código genera el siguiente diagrama:

Salida



No trazamos los activos individuales y la línea de la frontera eficiente, para evitar que la trama se vuelva demasiado abarrotada.

11.3 ENCONTRAR LA FRONTERA EFICIENTE USANDO OPTIMIZACIÓN CON SCIPY

En la explicación anterior, encontramos la frontera eficiente utilizando simulaciones de Monte Carlo, utilizamos un enfoque de fuerza bruta basado en simulaciones de Monte Carlo para visualizar la frontera eficiente. A continuación, usamos un método más refinado para determinar la frontera.

Desde su definición, la frontera eficiente está formada por un conjunto de portafolios que ofrecen el mayor rendimiento esperado para una determinada volatilidad, o que ofrecen el menor riesgo (volatilidad) para un cierto nivel de rendimiento esperado. Podemos aprovechar este hecho y usarlo en forma de optimización numérica. El objetivo de la optimización es encontrar el mejor valor (óptimo) de la función objetivo ajustando las variables objetivo y teniendo en cuenta algunos límites y restricciones (que tienen un impacto en las variables objetivo). En este caso, la función objetivo es una función que devuelve la volatilidad del portafolio, y las variables objetivo son las ponderaciones de la portafolio.

Matemáticamente, el problema se puede expresar como:

$$\begin{aligned} \min \quad & W^T \Sigma W \\ \text{s.t.} \quad & W^T = 1 \\ & w \geq 0 \\ & W^T \mu = \mu_p \end{aligned}$$

Aquí, W es un vector de participaciones porcentuales, Σ es la matriz de covarianza, μ es un vector de rendimientos y μ_p es el rendimiento esperado del portafolio.

Repetimos la rutina de optimización utilizada para encontrar los porcentajes de participación óptimos del portafolio en un rango de rendimientos esperados del portafolio, y esto da como resultado la frontera eficiente.

En esta explicación, trabajamos con el mismo conjunto de datos que en el anterior, para mostrar que los resultados obtenidos por ambos enfoques son similares. Por dicha razón necesitamos usar nuevamente las simulaciones de Montecarlo.

Ejecute los siguientes pasos para encontrar la frontera eficiente utilizando la optimización con scipy.

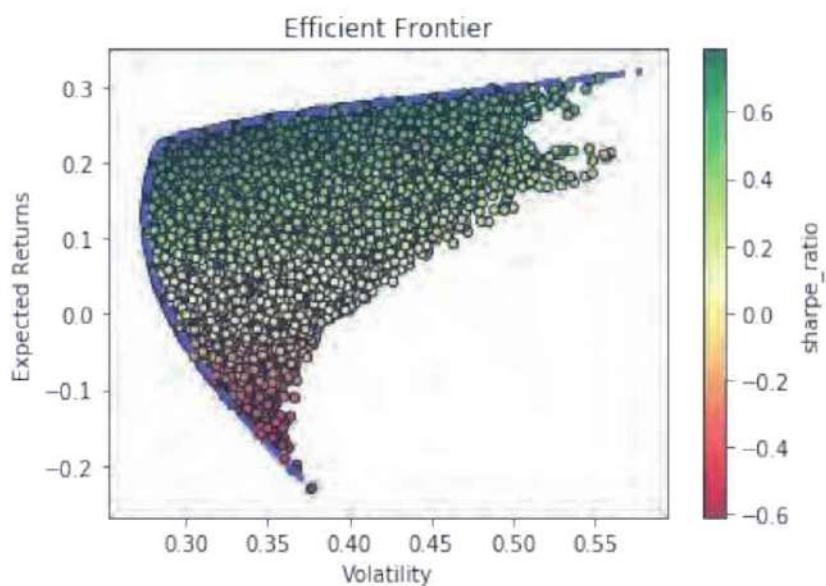
```
# 1. Importar librerías
import numpy as np
import scipy.optimize as sco
# 2. Definir funciones para calcular los rendimientos de la portafolio y la vola-
tilidad:
```

```
def get_portf_rtn(w, avg_rtns):
    return np.sum(avg_rtns * w)
def get_portf_vol(w, avg_rtns, cov_mat):
    return np.sqrt(np.dot(w.T, np.dot(cov_mat, w)))
# 3. Defina la función que calcula la frontera eficiente:
def get_efficient_frontier(avg_rtns, cov_mat, rtns_range):
    efficient_portfolios = []
    n_assets = len(avg_returns)
    args = (avg_returns, cov_mat)
    bounds = tuple((0,1) for asset in range(n_assets))
    initial_guess = n_assets * [1. / n_assets, ]
    for ret in rtns_range:
        constraints = ({'type': 'eq',
                        'fun': lambda x: get_portf_rtn(x, avg_rtns)
                        - ret},
                       {'type': 'eq',
                        'fun': lambda x: np.sum(x) - 1})
        efficient_portfolio = sco.minimize(get_portf_vol,
                                            initial_guess,
                                            args=args,
                                            method='SLSQP',
                                            constraints=constraints,
                                            bounds=bounds)
        efficient_portfolios.append(efficient_portfolio)
    return efficient_portfolios
# 4. Defina el rango considerado de rendimientos:
rtns_range = np.linspace(-0.22, 0.32, 200)
# 5.Calcule la frontera eficiente:
efficient_portfolios = get_efficient_frontier(avg_returns,
                                                cov_mat,
                                                rtns_range)
# 6.Extraiga las volatilidades de las portafolios eficientes:
vols_range = [x['fun'] for x in efficient_portfolios]
# 7.Trace la frontera eficiente calculada, junto con las portafolios simuladas:
fig, ax = plt.subplots()
portf_results_df.plot(kind='scatter', x='volatility',
                      y='returns', c='sharpe_ratio',
                      cmap='RdYlGn', edgecolors='black',
                      ax=ax)
ax.plot(vols_range, rtns_range, 'b--', linewidth=3)
ax.set(xlabel='Volatility',
       ylabel='Expected Returns',
       title='Efficient Frontier')
```

Salida

```
[Text(0, 0.5, 'Expected Returns'),
 Text(0.5, 0, 'Volatility'),
 Text(0.5, 1.0, 'Efficient Frontier')]
```

La siguiente imagen presenta un gráfico de la frontera eficiente, calculado mediante la optimización numérica:



Vemos que la frontera eficiente tiene una forma muy similar a la obtenida usando simulaciones de Monte Carlo. La única diferencia es que la línea es más suave.

```
# 8. Identifique el portafolio de volatilidad mínima:
min_vol_ind = np.argmin(vols_range)
min_vol_portf_rtn = rtns_range[min_vol_ind]
min_vol_portf_vol = efficient_portfolios[min_vol_ind]['fun']
min_vol_portf = {'Return': min_vol_portf_rtn,
                 'Volatility': min_vol_portf_vol,
                 'Sharpe Ratio': (min_vol_portf_rtn /
                                   min_vol_portf_vol)}
# 9.Imprima el resumen de rendimiento:
print('Minimum volatility portfolio ----')
print('Performance')
```

```
for index, value in min_vol_portf.items():
    print(f'{index}: {100 * value:.2f}%', end='', flush=True)
print('\nWeights')
for x, y in zip(RISKY_ASSETS,
                 efficient_portfolios[min_vol_ind]['x']):
    print(f'{x}: {100*y:.2f}%', end='', flush=True)
```

La ejecución del código da como resultado el siguiente resumen:

```
Minimum volatility portfolio ----
Performance
Return: 13.01% Volatility: 27.54% Sharpe Ratio: 47.22%
Weights
FB: 18.65% MSFT: 77.34% TSLA: 4.01% TWTR: 0.00%
```

El portafolio de volatilidad mínima se logra invirtiendo principalmente en Microsoft y Facebook, sin invertir en Twitter en absoluto.

Cómo funciona

Como se mencionó anteriormente, continuamos con el ejemplo de la explicación anterior. Es por eso que tuvimos que ejecutar del Paso 1 al Paso 4 de la primera explicación. Encontrar la frontera eficiente utilizando simulaciones de Monte Carlo (no se muestra aquí por brevedad), para tener todos los datos requeridos. Como requisito previo adicional, tuvimos que importar el módulo de optimización de `scipy`.

En el Paso 2, definimos dos funciones, que devolvieron el rendimiento esperado del portafolio y la volatilidad, dados los datos históricos y las participaciones porcentuales del portafolio. Tuvimos que definir estas funciones en lugar de calcular estas métricas directamente, como las usamos en el procedimiento de optimización. El algoritmo intenta iterativamente diferentes participaciones porcentuales y necesita poder usar los valores actuales de las variables objetivo (participaciones) para llegar a la métrica que intenta optimizar.

En el Paso 3, definimos una función llamada `get_efficient_frontier`. Su objetivo es devolver una lista que contenga los portafolios eficientes, dadas las métricas históricas y considerando el rango de rendimientos. Este es el paso más importante del modelo y contiene diversos matices.

Describimos la lógica de la función secuencialmente:

- ▶ El esquema de la función es que ejecuta el procedimiento de optimización para cada retorno esperado en el rango considerado, y almacena el portafolio óptimo resultante en una lista.
- ▶ Fuera del ciclo for, definimos un par de objetos que pasamos al optimizador:
 - Los argumentos que se pasan a la función objetivo. En este caso, estos son los rendimientos promedio históricos y la matriz de covarianza. La función que optimizamos debe aceptar los argumentos como entradas. Es por eso que pasamos los rendimientos a la función `get_portfolio_volatility`, aunque no sean necesarios para los cálculos.
 - Límites (una tupla anidada): para cada variable objetivo (peso), una tupla que contiene los límites: mínimo y máximo de los valores permitidos. En este caso, los valores abarcan el rango de 0 a 1 (no porcentajes de participación negativos, según el MPT).
 - `initial_guess`, que es la suposición inicial de las variables objetivo. El objetivo de usar la suposición inicial es hacer que la optimización se ejecute más rápido y más eficiente. En este caso, la suposición es el 1/n asignación.
- ▶ Dentro del bucle for, definimos el último elemento utilizado para la optimización: las restricciones. Definimos dos restricciones:
 - El rendimiento esperado del portafolio debe ser igual al valor proporcionado.
 - La suma de las participaciones porcentuales debe ser igual a 1.
- ▶ La primera restricción es la razón por la cual la tupla de la restricción se define dentro del ciclo: a medida que el ciclo pasa sobre el rango considerado de los rendimientos esperados del portafolio, y para cada valor, encontramos el nivel de riesgo óptimo.
- ▶ Ejecutamos el optimizador con el algoritmo de programación de mínimos cuadrados secuenciales (SLSQP), que se usa con frecuencia para problemas de minimización genéricos. Para minimizar la función, pasamos la función `get_portfolio_volatility`.

El optimizador establece la restricción de igualdad (eq) en 0. Es por eso que la restricción prevista, `np.sum(participación) == 1`, se expresa como `np.sum(participación) - 1 == 0`

En los Pasos 4 y 5, definimos el rango de rendimiento esperado del portafolio (basado en el rango que observamos empíricamente en la explicación inicial) y ejecutamos la función de optimización.

En el Paso 6, iteramos sobre la lista de portafolios eficientes y extrajimos las volatilidades óptimas. Extrajimos la volatilidad del objeto `scipy.optimize.OptimizeResult` accediendo al elemento *fun*. Esto representa la función objetivo optimizada, en este caso, la volatilidad del portafolio.

En el Paso 7, agregamos la Frontera Eficiente calculada en la parte superior de la gráfica de la explicación anterior, usando simulaciones de Monte Carlo. Todos los portafolios simulados se encuentran en o debajo de la frontera eficiente, que es lo que esperábamos que sucediera.

En los pasos 8 y 9, identificamos el portafolio de volatilidad mínima, imprimimos las métricas de rendimiento y mostramos las participaciones porcentuales del portafolio (extraídos de la frontera eficiente).

Ahora podemos comparar los dos portafolios de volatilidad mínima: el obtenido mediante simulaciones de Monte Carlo y la que recibimos de la optimización. El patrón predominante en la asignación es el mismo: asignar la mayoría de los recursos disponibles a Facebook y Microsoft. También podemos ver que la volatilidad de la estrategia optimizada es ligeramente menor. Esto significa que entre las 100,000 portafolios, no hemos simulado el portafolio de volatilidad mínima real.

Hay más

También podemos utilizar el enfoque de optimización para encontrar las participaciones porcentuales que generan un portafolio con el índice de Sharpe más alto esperado: el portafolio de tangencia. Para hacerlo, primero necesitamos definir la función objetivo, que es el negativo de la relación de Sharpe. La razón por la que usamos el negativo es que los algoritmos de optimización ejecutan problemas de minimización. Podemos abordar fácilmente los problemas de maximización cambiando el signo de la métrica objetiva:

```
# 1. Defina la función objetivo (relación de Sharpe negativa):
def neg_sharpe_ratio(w, avg_rtns, cov_mat, rf_rate):
    portf_returns = np.sum(avg_rtns * w)
    portf_volatility = np.sqrt(np.dot(w.T, np.dot(cov_mat, w)))
    portf_sharpe_ratio = (portf_returns - rf_rate) / portf_volatility
    return -portf_sharpe_ratio
```

El segundo paso es muy similar a lo que ya hemos hecho con Frontera Eficiente, esta vez sin el bucle for, ya que solo estamos buscando un conjunto de participaciones porcentuales. Incluimos la tasa libre de riesgo en los argumentos (aunque suponemos que es 0%, por simplicidad) y solo usamos una restricción: la suma de las variables objetivo debe ser igual a 1.

```
# 2. Encontrar el portafolio óptimo
n_assets = len(avg_returns)
RF_RATE = 0

args = (avg_returns, cov_mat, RF_RATE)
constraints = ({'type': 'eq',
                 'fun': lambda x: np.sum(x) - 1})
bounds = tuple((0,1) for asset in range(n_assets))
initial_guess = n_assets * [1. / n_assets]

max_sharpe_portf = sco.minimize(neg_sharpe_ratio,
                                 x0=initial_guess,
                                 args=args,
                                 method='SLSQP',
                                 bounds=bounds,
                                 constraints=constraints)

# 3. Extraiga información sobre el portafolio con la máxima relación de Sharpe:
max_sharpe_portf_w = max_sharpe_portf['x']
max_sharpe_portf = {'Return': get_portf_rtn(max_sharpe_portf_w,
                                             avg_returns),
                     'Volatility': get_portf_vol(max_sharpe_portf_w,
                                                 avg_returns,
                                                 cov_mat),
                     'Sharpe Ratio': -max_sharpe_portf['fun']}

# 4. Imprima el resumen de rendimiento:
print('Maximum Sharpe Ratio portfolio ----')
print('Performance')
for index, value in max_sharpe_portf.items():
    print(index, value)
```

```
print(f'{index}: {100 * value:.2f}%', end='', flush=True)
print('\nWeights')
for x, y in zip(RISKY_ASSETS, max_sharpe_portf_w):
    print(f'{x}: {100*y:.2f}%', end='', flush=True)
```

Salida

```
Maximum Sharpe Ratio portfolio ----
Performance
Return: 23.10% Volatility: 29.25% Sharpe Ratio: 78.97%
Weights
FB: 0.00% MSFT: 81.50% TSLA: 2.79% TWTR: 15.72%
```

La información de salida nos muestra un resumen del portafolio que maximiza la relación de Sharpe.

Para lograr la relación máxima de Sharpe, el inversionista debería invertir principalmente en Microsoft y Twitter, con una asignación del 0% a Facebook, ya que el rendimiento promedio de Facebook durante 2018 fue negativo.

www.dogramcode.com

<https://dogramcode.com/programacion>

12

SIMULACIÓN MONTE CARLO EN FINANZAS

Las simulaciones de Monte Carlo son una clase de algoritmos computacionales que utilizan muestreo aleatorio repetido para resolver cualquier problema que tenga una interpretación probabilística. En finanzas, una de las razones por las que ganaron popularidad es que pueden usarse para estimar con precisión las integrales.

La idea principal de las simulaciones de Monte Carlo es producir una multitud de rutas de muestra: posibles escenarios / resultados, a menudo durante un período de tiempo determinado. El horizonte se divide en un número específico de pasos de tiempo y el proceso de hacerlo se llama discretización. Su objetivo es aproximar el tiempo continuo, ya que el precio de los instrumentos financieros ocurre en tiempo continuo.

Los resultados de todas estas rutas de muestra simuladas se pueden utilizar para calcular métricas como el porcentaje de veces que ocurrió un evento, el valor promedio de un instrumento en el último paso, etc. Históricamente, el principal problema con el enfoque de Monte Carlo era que requería una gran potencia computacional para calcular todos los escenarios posibles. Hoy en día, se está volviendo menos problemático ya que podemos ejecutar simulaciones bastante avanzadas en un computador de escritorio o en un computador portátil.

Al final de este capítulo, habremos visto cómo podemos usar los métodos de Monte Carlo en varios escenarios y tareas. En algunos de ellos, crearemos las simulaciones desde cero, mientras que, en otros, utilizaremos las librerías modernas de Python para facilitar aún más el proceso. Debido a la flexibilidad del método, Monte Carlo es una de las técnicas más importantes en finanzas computacionales. Se puede adaptar a diversos problemas, como la fijación de precios de derivados sin una solución de forma cerrada (opciones americanas / exóticas), la valoración de

bonos (por ejemplo, un bono de cupón cero), la estimación de la incertidumbre de un portafolio (por ejemplo, mediante el cálculo del Valor en riesgo y déficit esperado), o realizar pruebas de stress en la gestión de riesgos. Mostramos cómo resolver algunos de estos problemas en este capítulo.

En este capítulo, cubrimos los siguientes temas:

- ▶ Simulación de la dinámica del precio de las acciones mediante el movimiento browniano geométrico
- ▶ Precios de las opciones europeas mediante simulaciones
- ▶ Estimación del valor en riesgo utilizando Monte Carlo

12.1 SIMULANDO LA DINÁMICA DEL PRECIO DE LAS ACCIONES UTILIZANDO MOVIMIENTO BROWNIANO GEOMÉTRICO

Gracias a la imprevisibilidad de los mercados financieros, la simulación de los precios de las acciones juega un papel importante en la valoración de muchos derivados, como las opciones. Debido a la aleatoriedad antes mencionada en el movimiento de precios, estas simulaciones se basan en ecuaciones diferenciales estocásticas (SDE).

Se dice que un proceso estocástico sigue al Movimiento Browniano Geométrico (GBM) cuando satisface la siguiente SDE:

$$dS = \mu S dt + \sigma S dW_t$$

Donde:

- ▶ S: precio de las acciones
- ▶ μ : el coeficiente de deriva, es decir, el rendimiento promedio durante un período determinado o el rendimiento esperado instantáneo
- ▶ σ : el coeficiente de difusión, es decir, cuánta volatilidad hay en la deriva
- ▶ W_t : El movimiento browniano

No investigaremos las propiedades del Movimiento Browniano con demasiada profundidad, ya que está fuera del alcance de este libro. Baste decir que los incrementos brownianos se calculan como un producto de una variable aleatoria normal estándar ($rv \sim N(0,1)$) y la raíz cuadrada del incremento de tiempo. Otra forma de decir esto es que el incremento browniano proviene de $rv \sim N(0, t)$, donde t es el incremento de tiempo. Obtenemos el camino browniano tomando la suma acumulativa de los incrementos brownianos.

El SDE tiene una solución de forma cerrada (solo unos pocos SDE lo tienen):

$$S(T) = S_0 e^{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W_t}$$

Aquí $S_0 = S(0)$ está el valor inicial del proceso, que en este caso es el precio inicial de una acción. La ecuación anterior presenta la relación en comparación con el precio inicial de las acciones.

Para simulaciones, podemos usar la siguiente fórmula recursiva:

$$S(t_{i+1}) = S(t_i) \exp \left(\mu - \frac{1}{2}\sigma^2 \right) (t_{i+1} - t_i) + \sigma \sqrt{(t_{i+1} - t_i)} Z_{i+1}$$

Aquí, Z_i es una variable aleatoria normal estándar y t_i es el índice de tiempo. Esta especificación es posible porque los incrementos de W son independientes y se distribuyen normalmente.

A continuación, utilizamos los métodos de Monte Carlo y el Movimiento Geométrico Browniano para simular los precios de las acciones de Microsoft un mes antes.

Cómo hacerlo

Ejecute los siguientes pasos para simular los precios de las acciones de Microsoft con un mes de anticipación.

```
# 1. Importar librerías
import numpy as np
import pandas as pd
import yfinance as yf

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import matplotlib.pyplot as plt
import warnings

plt.style.use('seaborn')
# plt.style.use('seaborn-colorblind') # Alternativa
plt.rcParams['figure.figsize'] = [8, 4.5]
plt.rcParams['figure.dpi'] = 300
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
# 2. Definir parámetros para descargar datos:
```

```
RISKY_ASSET = 'MSFT'  
START_DATE = '2019-01-01'  
END_DATE = '2019-07-31'
```

```
# 3. Descargar los datos de Yahoo Finance:
```

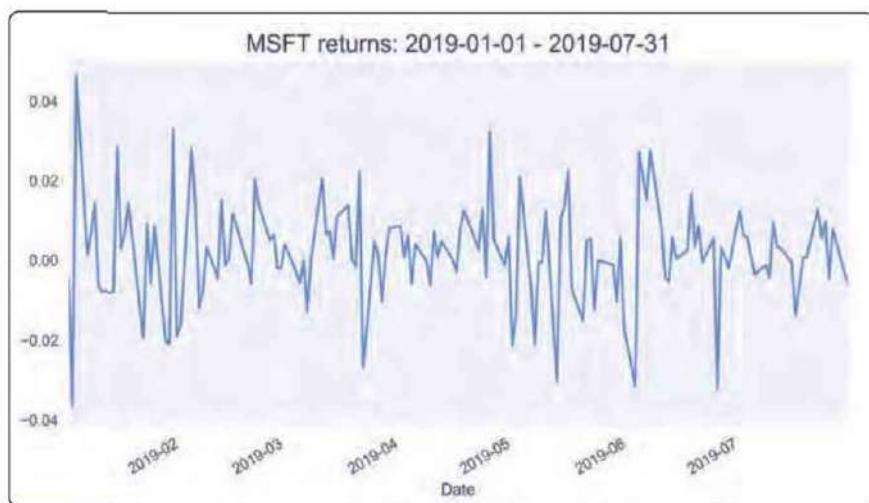
```
df = yf.download(RISKY_ASSET, start=START_DATE,  
                 end=END_DATE, adjusted=True)  
print(f'Downloaded {df.shape[0]} rows of data.')
```

```
# 4.Calcular los rendimientos diarios
```

```
adj_close = df['Adj Close']  
returns = adj_close.pct_change().dropna()  
  
ax = returns.plot()  
ax.set_title(f'{RISKY_ASSET} returns: {START_DATE} - {END_DATE}',  
            fontsize=16)  
  
plt.tight_layout()  
#plt.savefig('images/ch6_im1.png')  
plt.show()  
  
print(f'Average return: {100 * returns.mean():.2f}%')
```

Salida

El código produce el siguiente gráfico:



Y la siguiente línea:

```
Average return: 0.24%  
  
# 5. Divida los datos en conjuntos de entrenamiento y prueba:  
train = returns['2019-01-01':'2019-06-30']  
test = returns['2019-07-01':'2019-07-31']  
  
# 6. Especifique los parámetros de la simulación:  
T = len(test)  
N = len(test)  
S_0 = adj_close[train.index[-1].date()]  
N_SIM = 100  
mu = train.mean()  
sigma = train.std()  
  
# 7. Defina la función para simulaciones:  
def simulate_gbm(s_0, mu, sigma, n_sims, T, N,  
                  random_seed=42):  
    ...  
    Función utilizada para simular rendimientos de acciones usando Movimiento Browniano Geométrico.  
  
Parámetros  
-----  
s_0 : float  
      Precio inicial de La acción  
mu : float  
      Coeficiente de deriva  
sigma : float  
      Coeficiente de difusión  
n_sims : int  
      Número de trayectorias de simulación  
dt : float  
      Incremento de tiempo, usualmente un día  
T : float  
      Longitud del horizonte de pronóstico  
N : int  
      Número de incrementos de tiempo en el horizonte de pronóstico  
random_seed : int  
      Semilla aleatoria para La reproducibilidad  
Returns  
-----  
S_t : np.ndarray  
      Matrix (size: n_sims x (T+1)) contiene Los resultados de La simulación.  
      Las filas representan trayectorias, mientras que Las columnas son puntos de tiempo.
```

```
    ...
    np.random.seed(random_seed)

    dt = T/N
    dW = np.random.normal(scale = np.sqrt(dt), size=(n_sims, N))
    W = np.cumsum(dW, axis=1)

    time_step = np.linspace(dt, T, N)
    time_steps = np.broadcast_to(time_step, (n_sims, N))

    S_t = s_0 * np.exp((mu - 0.5 * sigma**2) * time_steps
                       + sigma * W)
    S_t = np.insert(S_t, 0, s_0, axis=1)

    return S_t

# 8. Ejecute las simulaciones:
gbm_simulations = simulate_gbm(S_0, mu, sigma, N_SIM, T, N)

# 9. Graficar resultados de simulación:
#Preparar objetos para graficar

last_train_date = train.index[-1].date()
first_test_date = test.index[0].date()
last_test_date = test.index[-1].date()
plot_title = (f'{RISKY_ASSET} Simulation '
              f'({first_test_date}:{last_test_date})')

selected_indices = adj_close[last_train_date:last_test_date].index
index = [date.date() for date in selected_indices]

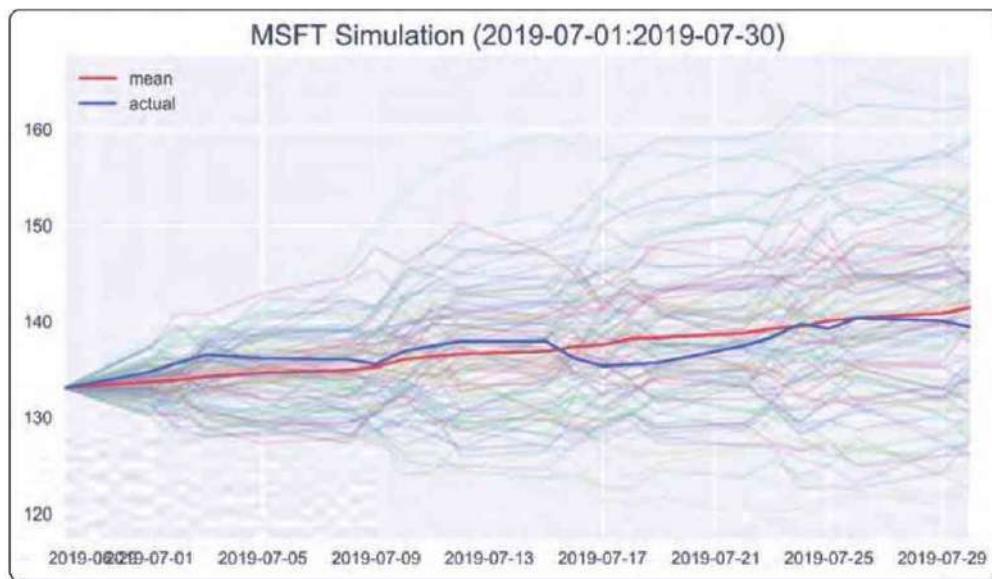
gbm_simulations_df = pd.DataFrame(np.transpose(gbm_simulations),
                                    index=index)

# Graficar

ax = gbm_simulations_df.plot(alpha=0.2, legend=False)
line_1, = ax.plot(index, gbm_simulations_df.mean(axis=1),
                  color='red')
line_2, = ax.plot(index, adj_close[last_train_date:last_test_date],
                  color='blue')
ax.set_title(plot_title, fontsize=16)
ax.legend((line_1, line_2), ('mean', 'actual'))

plt.tight_layout()
#plt.savefig('images/ch6_im2.png')
plt.show()
```

Salida



En la gráfica anterior, observamos que el valor promedio de las simulaciones exhibe una tendencia positiva debido al término de deriva positiva:

Cómo funciona

En los pasos 2 a 4, descargamos los precios de las acciones de Microsoft y calculamos los rendimientos simples. En el siguiente paso, dividimos los datos en los conjuntos de entrenamiento y prueba. Calculamos el promedio y la desviación estándar de los rendimientos del conjunto de entrenamiento para obtener los coeficientes de deriva (μ) y difusión (σ), que luego usamos para las simulaciones. Además, en el Paso 6, definimos los siguientes parámetros:

- ▀ T: horizonte de pronóstico; en este caso, el número de días en el conjunto de prueba
- ▀ N: Número de incrementos de tiempo en el horizonte de pronóstico
- ▀ S_0: precio inicial. Para esta simulación, tomamos la última observación del conjunto de entrenamiento
- ▀ N_SIM: número de rutas simuladas

Las simulaciones de Monte Carlo utilizan un proceso llamado discretización. La idea es aproximar el precio continuo de los activos financieros dividiendo el

horizonte temporal considerado en una gran cantidad de intervalos discretos. Es por eso que, a excepción de considerar el horizonte de pronóstico, también necesitamos indicar el número de incrementos de tiempo para encajar en el horizonte.

El paso 7 es donde definimos la función para ejecutar las simulaciones. Es una buena práctica definir una función / clase para tal problema, ya que también será útil en las siguientes aplicaciones. Comenzamos definiendo el incremento de tiempo (dt) y los incrementos brownianos (dW). En la matriz de incrementos (tamaño: $n_sims \times N$), cada fila describe una ruta de muestra. A partir de ahí, calculamos las rutas brownianas (W) ejecutando una suma acumulativa (`np.cumsum`) sobre las filas. Luego, creamos una matriz que contiene los pasos de tiempo (`time_steps`). Para hacerlo creamos una matriz de valores espaciados uniformemente dentro de un intervalo (el horizonte de la simulación). Para eso, utilizamos `np.linspace`. Luego, transmitimos la matriz a la forma deseada usando `np.broadcast_to`. Utilizamos la fórmula de forma cerrada para calcular el precio de las acciones en cada momento. Finalmente, insertamos el valor inicial en la primera posición de cada fila.

En los pasos anteriores, podemos reconocer la deriva como $(mu - 0.5 * sigma ** 2) * time_steps$ y la difusión como $sigma * W$.

Al definir esta función, seguimos el enfoque vectorizado. Al hacerlo, evitamos escribir bucles `for`, lo que sería ineficiente en el caso de simulaciones grandes.

En el Paso 9, visualizamos las rutas de muestra simuladas. Para hacerlo, transpusimos los datos y los convertimos en un DataFrame de pandas. Hicimos la transposición para tener una ruta por columna, lo que simplifica el uso del método de trazado de pandas DataFrame. Esto también se puede hacer usando matplotlib puro.

Aparte de la trama principal, agregamos dos líneas adicionales. La primera representa el valor promedio de todas las rutas de muestra en un punto dado en el tiempo. El segundo es el precio real de las acciones de Microsoft en el conjunto de prueba. Para visualizar los precios de las acciones simuladas, elegimos `alpha = 0.2` para hacer que las líneas sean transparentes. Al hacer esto, es más fácil ver las dos líneas adicionales.

12.2 PRECIOS DE OPCIONES EUROPEAS MEDIANTE SIMULACIONES

Las opciones son un tipo de instrumento derivado porque su precio está vinculado al precio del valor subyacente, como las acciones. La compra de un contrato de opciones otorga el derecho, pero no la obligación, de comprar o vender un activo subyacente a un precio establecido (conocido como strike o precio de ejercicio) en / antes de una cierta fecha. La razón principal de la popularidad de

las opciones es porque evitan la exposición al precio de un activo que se mueve de forma indeseable.

Una opción de compra / venta europea nos da el derecho (pero de nuevo, sin obligación) de comprar / vender un determinado activo en una determinada fecha de vencimiento (comúnmente denominado T). Algunos métodos populares de valoración de opciones:

- ▶ Usando fórmulas analíticas
- ▶ Enfoque del árbol binomial
- ▶ Diferencias finitas
- ▶ Simulaciones de Monte Carlo

Las opciones europeas son una excepción en el sentido de que existe una fórmula analítica para su valoración, que no es el caso para derivados más avanzados, como las opciones Americanas u opciones Exóticas.

Para fijar el precio de las opciones utilizando simulaciones de Monte Carlo, utilizamos una valoración neutral al riesgo, según la cual el valor razonable de un derivado es el valor esperado de sus pagos futuros. En otras palabras, suponemos que la prima de la opción crece a la misma tasa que la tasa libre de riesgo, que usamos para descontar el valor presente. Para cada una de las rutas simuladas, calculamos el pago de la opción al vencimiento, tomamos el promedio de todas las rutas y lo descontamos al valor presente.

En el siguiente ejemplo, se muestra cómo codificar la solución de forma cerrada para el modelo Black-Scholes y luego usar el enfoque de simulación. Para simplificar, usamos datos de entrada ficticios, pero los datos de vida real podrían usarse de forma análoga.

Cómo hacerlo

Ejecute los siguientes pasos para fijar el precio de las opciones europeas utilizando el modelo Black-Scholes y simulaciones de Monte Carlo.

```
# 1. Importar las librerías
import numpy as np
from scipy.stats import norm

# 2. Defina los parámetros para la valoración:
S_0 = 100
K = 100
r = 0.05
```

```

sigma = 0.50
T = 1 # 1 año
N = 252 # 252 días en el año
dt = T / N # pasos
N_SIMS = 1000000 # número de simulaciones
discount_factor = np.exp(-r * T)

# 3.Defina la función utilizando la solución analítica:
def black_scholes_analytical(S_0, K, T, r, sigma, type='call'):
    """
        Función utilizada para calcular el precio de las opciones europeas utilizando
        la forma analítica del modelo Black-Scholes.

    Parametros """
    -----
    S_0 : float
        Precio inicial de la acción
    K : float
        Precio de ejercicio
    T : float
        Tiempo hasta el vencimiento en años
    r : float
        Tasa libre de riesgo anualizada
    sigma : float
        Desviación estándar del rendimiento de las acciones
    type : str
        Tipo de opción: ['call', 'put']

    Returns
    option_premium : float
        La prima de la opción calculada utilizando el modelo Black-Scholes.
    """

    d1 = (np.log(S_0 / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = (np.log(S_0 / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))

    if type == 'call':
        val = (S_0 * norm.cdf(d1, 0, 1) - K * np.exp(-r * T) * norm.cdf(d2, 0,
1))
    elif type == 'put':
        val = (K * np.exp(-r * T) * norm.cdf(-d2, 0, 1) - S_0 * norm.cdf(-d1, 0,
1))
    else:
        raise ValueError('Wrong input for type!')

    return val

```

```
# 4. Valore la opción de compra utilizando los parámetros especificados:  
  
black_scholes_analytical(S_0=S_0, K=K, T=T, r=r, sigma=sigma, type='call')  
  
Salida  
21.79260421286685  
  
# 5. Simule la ruta del precio de la acción con GBM (movimiento browniano  
geométrico:  
  
>>gbm_sims = simulate_gbm(s_0=S_0, mu=r, sigma=sigma,  
n_sims=N_SIMS, T=T, N=N)  
  
# 6. Calcule la prima de la opción:  
  
premium = discount_factor * np.mean(np.maximum(0, gbm_sims[:, -1] - K))  
premium  
  
Salida  
21.756178586245806
```

Aquí, podemos ver que la prima de la opción que calculamos usando simulaciones de Monte Carlo es cercana a la de una solución de forma cerrada del modelo Black-Scholes. Para aumentar la precisión de la simulación, podríamos aumentar el número de rutas simuladas (utilizando el parámetro n_sims).

Cómo funciona

En el Paso 2, definimos los parámetros que usamos para este ejemplo:

- ▀ S_0: precio de la acción inicial
- ▀ K: Precio de ejercicio, es decir, al que podemos comprar / vender al vencimiento
- ▀ r: tasa anual libre de riesgo
- ▀ sigma: volatilidad subyacente de las acciones (anualizada)
- ▀ T: Tiempo hasta el vencimiento en años.
- ▀ N: número de incrementos de tiempo para simulaciones
- ▀ n_sims: número de rutas de muestra simuladas
- ▀ discount_factor: factor de descuento, que se utiliza para calcular el valor presente de la recompensa futura

En el Paso 3, definimos una función para calcular la prima de la opción utilizando la solución de forma cerrada para el modelo Black-Scholes (para acciones que no pagan dividendos). Lo usamos en el Paso 4 para calcular el punto de referencia para las simulaciones de Monte Carlo. La solución analítica a las opciones de compra y venta es:

$$C(S_t, t) = N(d_1) S_t - N(d_2) K e^{-r(T-t)} \quad (1)$$

$$P(S_t, t) = N(-d_2) K e^{-r(T-t)} - N(-d_1) S_t \quad (2)$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}} [\ln(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})(T-t)]$$

$$d_2 = d_1 - \sigma \sqrt{T-t}$$

Aquí, $N()$ representa la función de distribución acumulativa (CDF) de la distribución Normal estándar y $T - t$ es el tiempo de vencimiento expresado en años. La ecuación 1 representa la fórmula del precio de una opción de compra europea, mientras que la ecuación 2 representa el precio de la opción de venta europea. Informalmente, los dos términos en la ecuación 1 pueden considerarse como:

- ▶ El precio actual de la acción, ponderado por la probabilidad de ejercer la opción de comprar la acción ($N(d_1)$), en otras palabras, lo que podríamos recibir
- ▶ El precio con descuento de ejercer la opción (strike), ponderado por la probabilidad de ejercer la opción ($N(d_2)$), en otras palabras, lo que vamos a pagar

En el Paso 5, utilizamos la función de simulación GBM del ejemplo anterior para obtener 1,000,000 de posibles rutas del activo subyacente. Para calcular la prima de la opción, solo miramos los valores terminales y, para cada ruta, calculamos la recompensa de la siguiente manera:

- ▶ $\max(S_T - K, 0)$ para la opción call
- ▶ $\max(K - S_T, 0)$ para la opción put

En el Paso 6, tomamos el promedio de los pagos y lo descontamos a valor presente utilizando el factor de descuento.

En los pasos anteriores, mostramos cómo reutilizar la simulación GBM para calcular la prima de la opción de compra europea. Sin embargo, podemos hacer los cálculos más rápido, ya que en el caso de las opciones europeas solo estamos interesados en el precio final de las acciones. Los pasos intermedios no importan. Es por eso que solo necesitamos simular el precio en el momento T y usar estos valores para calcular la rentabilidad esperada. Mostramos cómo hacer esto usando un ejemplo de opción de venta europea con los mismos parámetros que usamos antes.

Comenzamos calculando la prima de la opción utilizando la fórmula analítica:

```
black_scholes_analytical(S_0=S_0, K=K, T=T, r=r, sigma=sigma, type='put')
```

```
16.915546662938254
```

Luego, definimos la función de simulación modificada, que solo analiza los valores terminales de las rutas de simulación:

```
def european_option_simulation(S_0, K, T, r, sigma, n_sims, type='call', random_seed=42):
    """


```

Función utilizada para calcular el precio de Las opciones europeas mediante simulaciones de Monte Carlo.

Parámetros

```
-----
```

```
S_0 : float
```

Precio inicial de La acción

```
K : float
```

Precio de ejercicio

```
T : float
```

Tiempo hasta el vencimiento en años

```
r : float
```

Tasa Libre de riesgo anualizada

```
sigma : float
```

Desviación estándar del rendimiento de Las acciones

```
n_sims : int
```

Número de trayectorias a simular

```
type : str
```

Tipo de opción: ['call', 'put']

```
random_seed : int
```

Semilla aleatoria para la reproducibilidad

```
Returns
-----
option_premium : float
    La prima de la opción calculada mediante simulaciones de Monte Carlo.
    ,
np.random.seed(random_seed)
rv = np.random.normal(0, 1, size=n_sims)
S_T = S_0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * rv)

if type == 'call':
    payoff = np.maximum(0, S_T - K)
elif type == 'put':
    payoff = np.maximum(0, K - S_T)
else:
    raise ValueError('Wrong input for type!')

premium = np.mean(payoff) * np.exp(-r * T)
return premium
```

Luego, ejecutamos las simulaciones:

```
european_option_simulation(S_0, K, T, r, sigma, N_SIMS, type='put')
```

```
16.948225203893127
```

12.3 ESTIMACIÓN DEL VALOR EN RIESGO UTILIZANDO MONTE CARLO

El valor en riesgo es una métrica financiera muy importante que mide el riesgo asociado con una posición o portafolio. Se abrevia comúnmente como VaR. VaR informa la peor pérdida esperada - a un nivel dado de confianza - en un cierto horizonte en condiciones normales de mercado. La forma más fácil de entenderlo es mirando un ejemplo. Digamos que el VaR de 95% de 1 día de nuestro portafolio es de € 100. Esto significa que el 95% del tiempo (en condiciones normales de mercado), no perderemos más de € 100 al mantener nuestro portafolio durante un día.

Es común presentar la pérdida dada por VaR como un valor positivo (absoluto). Es por eso que en este ejemplo, un VaR de € 100 significa perder no más de € 100.

Hay varias formas de calcular el VaR, algunas de las cuales son:

- ▀ Enfoque paramétrico (varianza-covarianza)
- ▀ Enfoque de simulación histórica
- ▀ Simulaciones de Monte Carlo

En el siguiente ejemplo, solo consideramos el último método. Suponemos que tenemos un portafolio que consta de dos activos (Facebook y Google) y que queremos calcular un valor a riesgo de 1 día.

Cómo hacerlo

Ejecute los siguientes pasos para estimar el valor en riesgo utilizando Monte Carlo.

```
# 1. Importar librerías
import numpy as np
import pandas as pd
import yfinance as yf
import seaborn as sns

# 2. Establecer semilla aleatoria para la reproducibilidad
np.random.seed(42)

# 3. Defina los parámetros que se utilizarán para este ejercicio:
RISKY_ASSETS = ['GOOG', 'FB']
SHARES = [5, 5]
START_DATE = '2018-01-01'
END_DATE = '2018-12-31'
T = 1
N_SIMS = 10 ** 5

# 4. Descargar datos de Yahoo Finance:
df = yf.download(RISKY_ASSETS, start=START_DATE,
                  end=END_DATE, adjusted=True)
print(f'Downloaded {df.shape[0]} rows of data.')

[*****100%*****] 2 of 2 downloaded
Downloaded 250 rows of data.
df.head()
```

Salida

Date	Adj Close		Close		High		Low		Open		Volume	
	FB	GOOG	FB	GOOG								
2018-01-02	181.419998	1065.000000	181.419998	1065.000000	181.580002	1066.939941	177.550003	1045.229980	177.679993	1048.339966	18151900	1237600
2018-01-03	184.689998	1082.479980	184.689998	1082.479980	184.779989	1086.290039	181.300002	1083.209981	181.880005	1064.310059	16888600	1430200
2018-01-04	184.330002	1086.400024	184.330002	1086.400024	186.210007	1093.509946	184.100006	1084.001953	184.869994	1068.000000	13880900	1004600
2018-01-05	186.850006	1102.229980	186.850006	1102.229980	186.899994	1104.250000	184.929993	1092.000000	185.589996	1094.000000	13574500	1279100
2018-01-08	188.279999	1106.939941	188.279999	1106.939941	188.899994	1111.270020	186.330002	1101.519995	187.199997	1102.229980	17994700	1047600

5. Calcular los rendimientos diarios

```

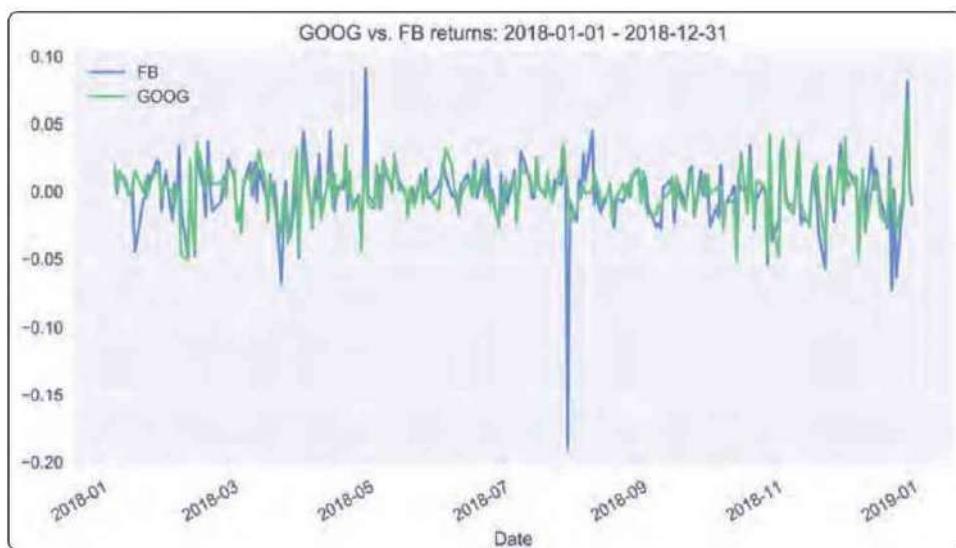
adj_close = df['Adj Close']
returns = adj_close.pct_change().dropna()
plot_title = f'{` vs. `.join(RISKY_ASSETS)} returns: {START_DATE} - {END_DATE}'
returns.plot(title=plot_title)

plt.tight_layout()
#plt.savefig('images/ch6_im3.png')
plt.show()

print(f'Correlation between returns: {returns.corr().values[0,1]:.2f}')

```

También graficamos los rendimientos y calculamos la correlación de Pearson de las dos series:



La correlación entre las dos series es 0.62

```
.....  
# 6. Calcule la matriz de covarianza  
  
cov_mat = returns.cov()  
  
# 7. Realice la descomposición de Cholesky de la matriz de covarianza:  
  
chol_mat = np.linalg.cholesky(cov_mat)  
chol_mat  
  
array([[0.02397822, 0.          ],  
       [0.01105642, 0.01389753]])  
  
# 8. Dibuje números aleatorios correlacionados de la distribución Normal estándar  
  
rv = np.random.normal(size=(N_SIMS, len(RISKY_ASSETS)))  
correlated_rv = np.transpose(np.matmul(chol_mat, np.transpose(rv)))  
  
# 9. Defina las métricas que se usarán para las simulaciones  
r = np.mean(returns, axis=0).values  
sigma = np.std(returns, axis=0).values  
S_0 = adj_close.values[-1, :]  
P_0 = np.sum(SHARES * S_0)  
  
# 10. Calcule el precio terminal de las acciones consideradas  
  
S_T = S_0 * np.exp((r - 0.5 * sigma ** 2) * T +  
                    sigma * np.sqrt(T) * correlated_rv)  
  
#11. Calcule el valor final y los rendimientos del portafolio  
  
P_T = np.sum(SHARES * S_T, axis=1)  
P_diff = P_T - P_0  
  
# 12. Calcule el VaR para los niveles de confianza seleccionados
```

```
P_diff_sorted = np.sort(P_diff)
percentiles = [0.01, 0.1, 1.]
var = np.percentile(P_diff_sorted, percentiles)

for x, y in zip(percentiles, var):
    print(f'1-day VaR with {100-x}% confidence: {-y:.2f}€')
```

La ejecución del código anterior da como resultado la siguiente salida:

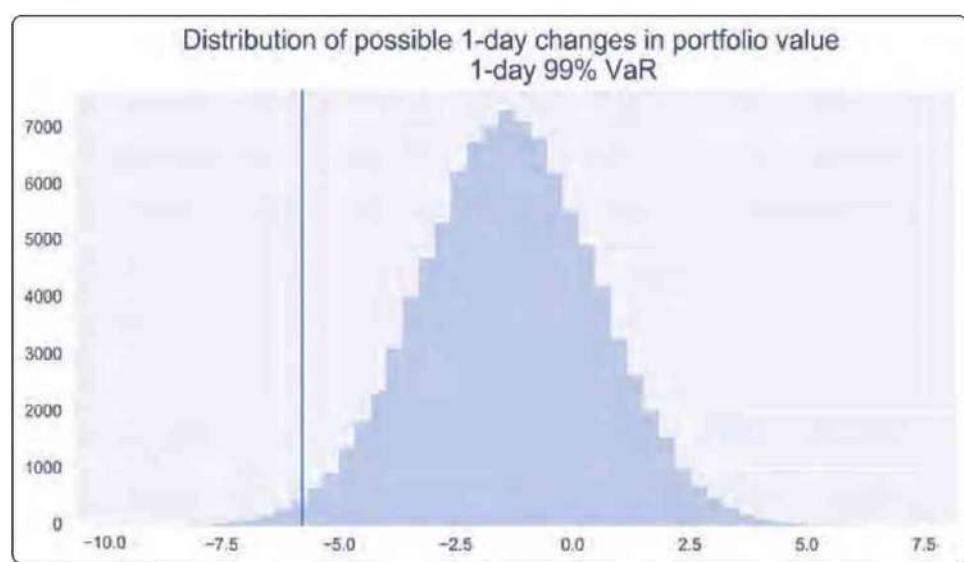
```
1-day VaR with 99.99% confidence: 8.24€
1-day VaR with 99.9% confidence: 7.14€
1-day VaR with 99.0% confidence: 5.78€
```

13. Presente los resultados en un gráfico

```
ax = sns.distplot(P_diff, kde=False)
ax.set_title('Distribution of possible 1-day changes in portfolio value
              1-day 99% VaR', fontsize=16)
ax.axvline(var[2], 0, 10000)

plt.tight_layout()
plt.savefig('images/ch6_im4.png')
plt.show()
```

La ejecución del código da como resultado el siguiente diagrama:



El gráfico anterior muestra la distribución de los posibles valores de portafolio con 1 día de anticipación. Presentamos el valor en riesgo con la línea vertical.

Cómo funciona

En los pasos 3 a 5, descargamos los precios diarios de las acciones de Google y Facebook, trajimos los precios de cierre ajustados y los convertimos en rendimientos simples. También definimos algunos parámetros, como el número de simulaciones y el número de acciones que tenemos en nuestro portafolio. Hay dos formas de abordar los cálculos de VaR:

- ▶ Calcular el VaR a partir de los precios: utilizando el número de acciones y los precios de los activos, podemos calcular el valor del portafolio ahora y su posible valor X días antes.
- ▶ Calcular el VaR a partir de los rendimientos: utilizando las participaciones porcentuales de cada activo en el portafolio y los rendimientos esperados de los activos, podemos calcular el rendimiento esperado de la portafolio X días antes. Luego, podemos expresar el VaR como el monto en euros basado en ese rendimiento y el valor actual del portafolio.

El enfoque de Monte Carlo para determinar el precio de un activo emplea variables aleatorias extraídas de la distribución normal estándar. Para el caso del cálculo del VaR del portafolio, debemos tener en cuenta el hecho de que los activos de nuestro portafolio pueden estar correlacionados. Para hacerlo, en los Pasos 6 a 8, calculamos la matriz de covarianza histórica, usamos la descomposición de Cholesky y multiplicamos la matriz resultante por la matriz de variables aleatorias. De esta manera, agregamos correlación a las variables aleatorias generadas.

Otra posible opción para hacer que las variables aleatorias estén correlacionadas es usar la Descomposición de Valores Singulares (SVD) en lugar de la descomposición Cholesky. La función que podemos usar para esto es `np.linalg.svd`.

En el Paso 9, calculamos métricas como los promedios históricos del rendimiento del activo, las desviaciones estándar que lo acompañan, los últimos precios de acciones conocidos y el valor inicial de la portafolio. En el Paso 10, aplicamos la solución analítica a la ecuación diferencial estocástica del movimiento browniano geométrico y calculamos los posibles precios de las acciones con 1 día de anticipación para ambos activos.

Para calcular el VaR del portafolio, calculamos los posibles valores de la portafolio con 1 día de anticipación y las diferencias que lo acompañan () y los

clasificamos en orden ascendente. El X% VaR es simplemente el percentil (1-X) de las diferencias de portafolio ordenadas.

Los bancos con frecuencia calculan el VaR de 1 y 10 días. Para llegar a este último, pueden simular el valor de sus activos en un intervalo de 10 días utilizando pasos de 1 día (discretización). Sin embargo, también pueden calcular el VaR de 1 día y multiplicarlo por la raíz cuadrada de 10. Esto podría ser beneficioso para el banco si conduce a menores requisitos de capital.

13

MODELADO DE SERIES DE TIEMPO FINANCIERAS

En la modelación financiera de cualquier empresa del sector real o financiero que se ve afectada por variables de mercado es de gran utilidad pronosticar y simular estas variables para gestionar y proyectar los resultados financieros de la empresa. El análisis de series de tiempo financieras es uno de los campos más desarrollados en las finanzas en los últimos años, y es el método principal para incorporar la incertidumbre de variables de mercado en el análisis financiero de las empresas.

En este capítulo, presentaremos los conceptos básicos del modelado de series de tiempo. Comenzamos explicando los componentes básicos de las series de tiempo y cómo separarlos usando métodos de descomposición. Más adelante, presentaremos el concepto de estacionariedad: por qué es importante, cómo probarlo y, en última instancia, cómo lograrlo en caso de que la serie original no sea estacionaria.

También veremos dos de los enfoques más utilizados para el modelado de series de tiempo: los métodos de suavizado exponencial y los modelos de clase ARIMA. En ambos casos, se mostrará cómo ajustar los modelos, evaluar la bondad del ajuste y pronosticar los valores futuros de la serie temporal. Además, se presentará un enfoque novedoso para modelar una serie temporal utilizando el modelo aditivo de la librería Prophet de Facebook.

Cubrimos los siguientes temas en este capítulo:

- ▶ Descomposición de series de tiempo
- ▶ Descomposición de series de tiempo usando Prophet de Facebook
- ▶ Prueba de estacionariedad en series de tiempo
- ▶ Corrección de la estacionariedad en series de tiempo

- ▶ Modelado de series de tiempo con métodos de suavizado exponencial
- ▶ Modelado de series de tiempo con modelos de clase ARIMA

13.1 DESCOMPOSICIÓN DE SERIES DE TIEMPO

El objetivo de la descomposición de series de tiempo es aumentar nuestra comprensión de los datos dividiendo la serie en varios componentes. Proporciona información en términos de complejidad de modelado y qué enfoques seguir para capturar con precisión cada uno de los componentes.

Estos componentes se pueden dividir en dos tipos: sistemáticos y no sistemáticos. Los sistemáticos se caracterizan por la coherencia y el hecho de que pueden describirse y modelarse. Por el contrario, los no sistemáticos no pueden modelarse directamente.

Los siguientes son los componentes sistemáticos:

- ▶ Nivel: el valor medio de la serie
- ▶ Tendencia: una estimación de la tendencia, es decir, el cambio de valor entre puntos de tiempo sucesivos en un momento dado. Puede asociarse con la pendiente (creciente / decreciente) de la serie
- ▶ Estacionalidad: Desviaciones de la media provocadas por la repetición de ciclos de corta duración

El siguiente es el componente no sistemático:

- ▶ Ruido: la variación aleatoria en la serie

Hay dos tipos de modelos que se utilizan para descomponer series de tiempo: aditivo y multiplicativo.

Las siguientes son las características del modelo aditivo:

- ▶ Forma del modelo: $y(t) = \text{nivel} + \text{tendencia} + \text{estacionalidad} + \text{ruido}$
- ▶ Modelo lineal: los cambios en el tiempo son consistentes en tamaño
- ▶ La tendencia es lineal (línea recta)
- ▶ Estacionalidad lineal con la misma frecuencia (ancho) y amplitud (alto) de ciclos a lo largo del tiempo

Las siguientes son las características del modelo multiplicativo:

- ▶ Forma del modelo: $y(t) = \text{nivel} * \text{tendencia} * \text{estacionalidad} * \text{ruido}$
- ▶ Modelo no lineal: los cambios en el tiempo no son consistentes en tamaño, por ejemplo, exponencial
- ▶ Una tendencia curva, no lineal
- ▶ Estacionalidad no lineal con frecuencia creciente / decreciente y amplitud de ciclos a lo largo del tiempo

Puede darse el caso de que no queramos trabajar con el modelo multiplicativo. Una posible solución es aplicar ciertas transformaciones para hacer lineal la tendencia / estacionalidad. Un ejemplo de transformación podría ser tomar el logaritmo de una serie en la que observamos un crecimiento exponencial.

En el siguiente ejemplo, presentamos cómo llevar a cabo la descomposición de series de tiempo de los precios mensuales del oro descargados de Quandl.

Ejecute los siguientes pasos para llevar a cabo la descomposición de la serie temporal.

```
# 1. Importar las librerías

import pandas as pd
import quandl
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
import warnings

# 2. Descargue los precios del oro para 2000-2011 y vuelva a muestrear a valores
mensuales:
QUANDL_KEY = '{key}' # reemplace {key} con su propia clave de la API
quandl.ApiConfig.api_key = QUANDL_KEY

df = quandl.get(dataset='WGC/GOLD_MONAVG_USD',
                 start_date='2000-01-01',
                 end_date='2011-12-31')

df.rename(columns={'Value': 'price'}, inplace=True)
df = df.resample('M').last()
```

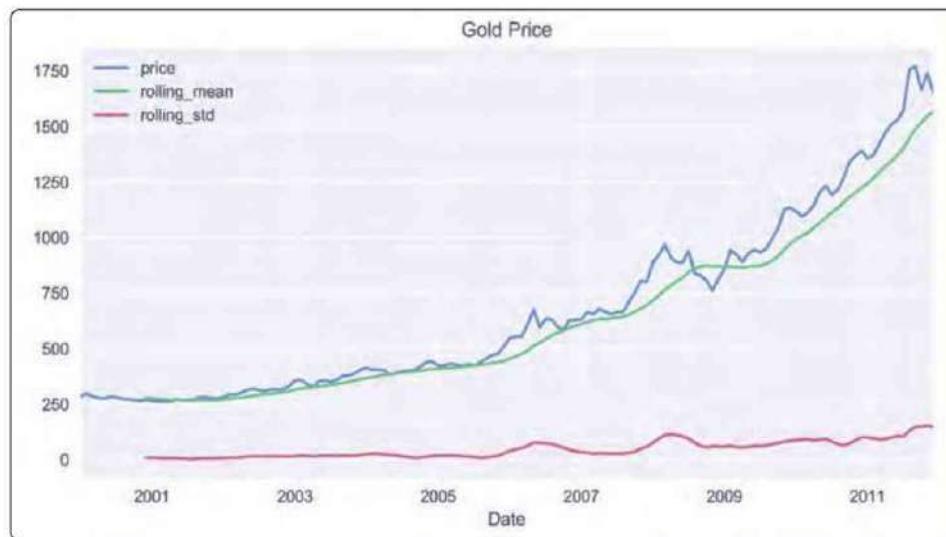
```
print(f'Shape of DataFrame: {df.shape}')
df.head()
```

Hay algunos valores duplicados en la serie. Por ejemplo, hay una entrada para 2000-04-28 y 2000-04-30, ambos con el mismo valor. Para solucionar este problema, volvemos a muestrear los datos mensuales tomando solo el último valor disponible (esto no cambia ninguno de los valores reales; solo elimina los posibles duplicados en cada mes).

```
# 3. Agregue la media móvil y la desviación estándar:
WINDOW_SIZE = 12
df['rolling_mean'] = df.price.rolling(window=WINDOW_SIZE).mean()
df['rolling_std'] = df.price.rolling(window=WINDOW_SIZE).std()
df.plot(title='Gold Price')

plt.tight_layout()
#plt.savefig('images/ch3_im1.png')
plt.show()
```

Salida

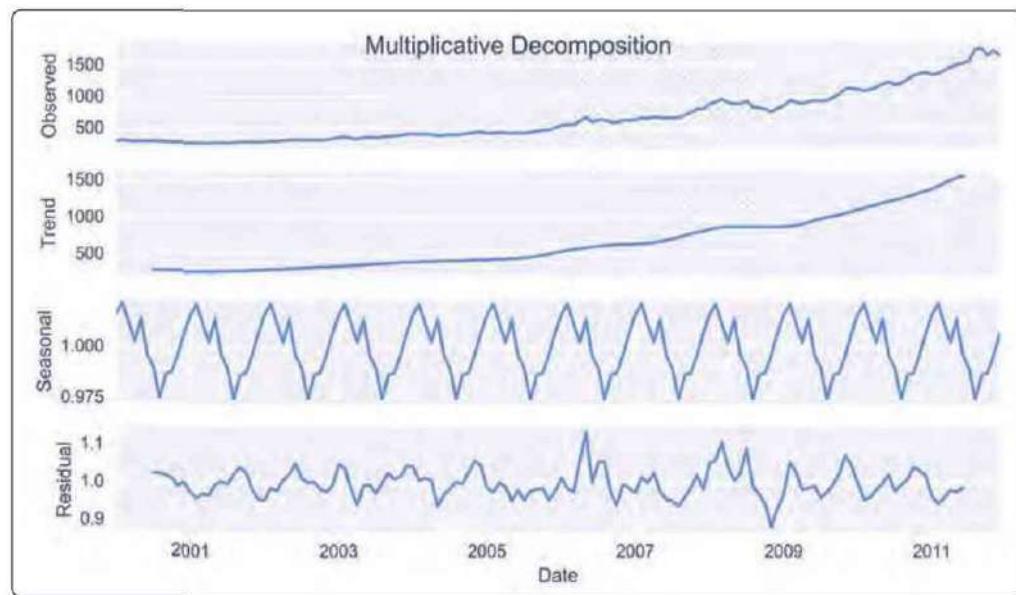


En el gráfico anterior, podemos ver que hay un patrón de crecimiento no lineal en el promedio móvil de 12 meses y que la desviación estándar móvil aumenta con el tiempo. Por eso decidimos utilizar el modelo multiplicativo.

```
# 4. Realice la descomposición estacional mediante el modelo multiplicativo:  
decomposition_results = seasonal_decompose(df.price,  
                                              model='multiplicative')  
decomposition_results.plot() \  
    .suptitle('Multiplicative Decomposition',  
              fontsize=14)  
  
plt.tight_layout()  
# plt.savefig('images/ch3_im2.png')  
plt.show()
```

Salida

Se genera la siguiente gráfica de descomposición:



En la gráfica de descomposición, podemos ver la serie de componentes extraídos: tendencia, estacional y aleatoria (residual). Para evaluar si la descomposición tiene sentido, podemos mirar el componente aleatorio. Si no hay un patrón discernible (en otras palabras, el componente aleatorio es de hecho aleatorio), entonces el ajuste tiene sentido. Por ejemplo, si hubiéramos aplicado el modelo aditivo, habría un patrón creciente en los residuos a lo largo del tiempo. En este caso, parece que la varianza de los residuos es ligeramente mayor en la segunda mitad del conjunto de datos.

Cómo funciona

Después de descargar los datos en el Paso 2, nos aseguramos de que la serie solo contuviera un punto de datos por mes (hicimos cumplir esto volviendo a muestrear los datos a la frecuencia mensual). Para calcular las estadísticas, utilizamos el método rolling de un DataFrame de pandas y se especificó el tamaño de ventana deseada (12 meses).

Usamos la función season_decompose de la librería statsmodels para realizar la descomposición clásica. Al hacerlo, indicamos qué tipo de modelo nos gustaría usar; los valores posibles son aditivos y multiplicativos.

13.2 DESCOMPOSICIÓN DE SERIES DE TIEMPO USANDO PROPHET DE FACEBOOK

Un enfoque alternativo para la descomposición de series de tiempo es utilizar un modelo aditivo, en el que una serie de tiempo se representa como una combinación de patrones en diferentes escalas de tiempo (diaria, semanal, mensual, anual, etc.) junto con la tendencia general. Prophet de Facebook hace exactamente eso, junto con funcionalidades más avanzadas como la contabilidad de puntos de cambio (cambios rápidos en el comportamiento), festivos y mucho más. Un beneficio práctico del uso de esta librería es que podemos pronosticar valores futuros de la serie de tiempo, junto con un intervalo de confianza que indica el nivel de incertidumbre.

En el siguiente ejemplo, intentaremos ajustar el modelo aditivo de Prophet a los precios diarios del oro de 2000 a 2004 y predecir los precios durante 2005.

Cómo hacerlo

```
# 1. Importe las librerías y autentíquese con Quandl:  
import pandas as pd  
import seaborn as sns  
import quandl  
from fbprophet import Prophet  
  
QUANDL_KEY = '{key}' # reemplace {key} con su propia clave  
quandl.ApiConfig.api_key = QUANDL_KEY  
  
# 2. Descargue los precios diarios del oro y cambie el nombre de las columnas:  
df = quandl.get(dataset='WGC/GOLD_DAILY_USD',  
                 start_date='2000-01-01',  
                 end_date='2005-12-31')
```

```
df.reset_index(drop=False, inplace=True)
df.rename(columns={'Date': 'ds', 'Value': 'y'}, inplace=True)

# 3. Divida la serie en conjuntos de entrenamiento y prueba:
train_indices = df.ds.apply(lambda x: x.year).values < 2005
df_train = df.loc[train_indices].dropna()
df_test = df.loc[~train_indices].reset_index(drop=True)

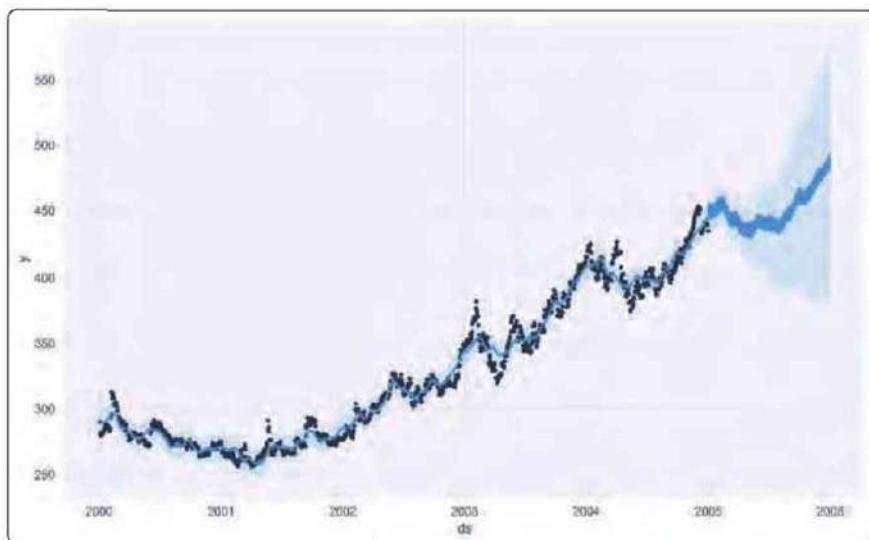
# 4. Cree la instancia del modelo y ajústela a los datos:
model_prophet = Prophet(seasonality_mode='additive')
model_prophet.add_seasonality(name='monthly', period=30.5, fourier_order=5)
model_prophet.fit(df_train)

<fbprophet.forecaster.Prophet at 0x11be16b00>

# 5. Pronostique los precios del oro con 1 año de anticipación y trace los resultados:
df_future = model_prophet.make_future_dataframe(periods=365)
df_pred = model_prophet.predict(df_future)
model_prophet.plot(df_pred)

plt.tight_layout()
# plt.savefig('images/ch3_im3.png')
plt.show()
```

La gráfica resultante es la siguiente:

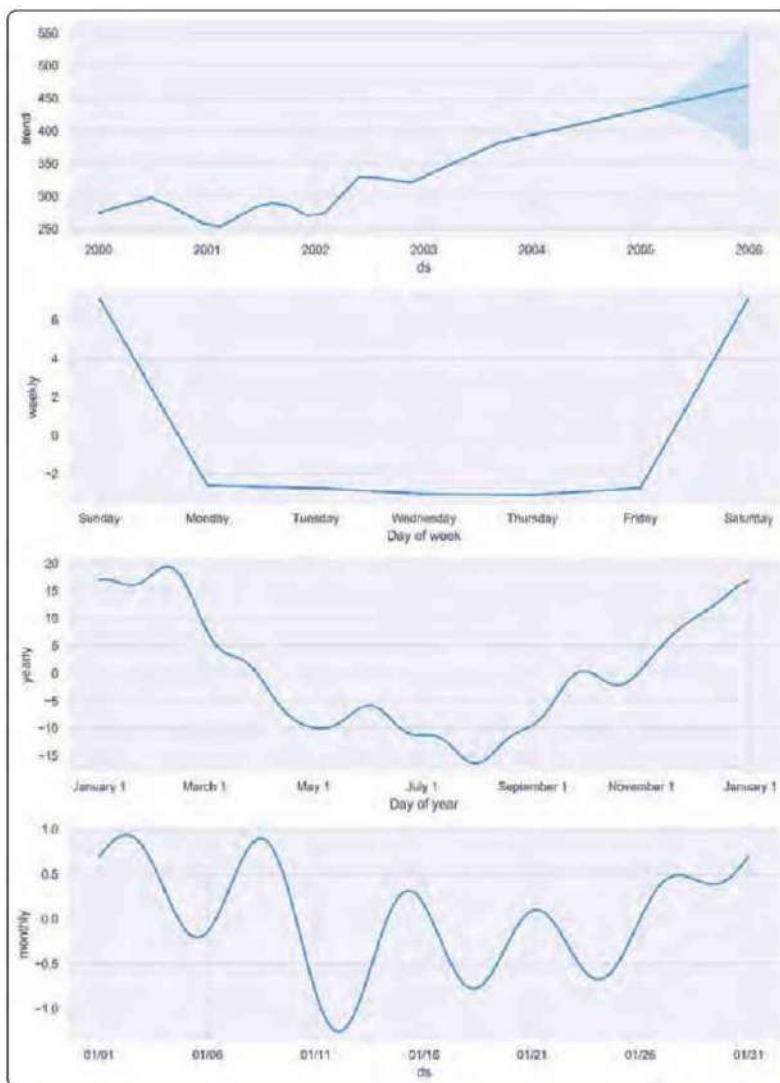


Los puntos negros son las observaciones reales del precio del oro. La línea azul que representa el ajuste no coincide exactamente con las observaciones, ya que el modelo suaviza el ruido en los datos (también reduce la posibilidad de sobreajuste). Una característica importante es que Prophet cuantifica la incertidumbre, que está representada por los intervalos azules alrededor de la línea ajustada.

```
# 6. Inspeccione la descomposición de la serie temporal:  
model_prophet.plot_components(df_pred)
```

```
plt.tight_layout()  
#plt.savefig('images/ch3_im4.png')  
plt.show()
```

La descomposición se presenta en la siguiente gráfica:



Tras una inspección más cercana, podemos ver que la tendencia general está aumentando y que el precio del oro parece ser más alto durante el comienzo y el final del año, con una caída en el verano. A nivel mensual, hay algún movimiento, pero la escala es mucho menor que en el caso del patrón anual. No hay mucho movimiento en el gráfico semanal (no miramos los fines de semana ya que no hay precios para los fines de semana), lo cual tiene sentido porque, con una disminución en la escala de tiempo, el ruido comienza a borrar la señal. Por esta razón, podríamos desactivar el nivel semanal por completo.

Cómo funciona

Prophet fue diseñado para analizar series de tiempo con observaciones diarias (lo que no significa que no haya formas de usar datos semanales o mensuales) que exhiben patrones en diferentes escalas de tiempo (semanal, mensual, anual, etc.).

En el Paso 2, descargamos los precios diarios del oro de Quandl y creamos un DataFrame de pandas con dos columnas: ‘ds’, que indica la marca de tiempo, e ‘y’, que es la variable objetivo. Esta estructura (nombres de columna) es necesaria para trabajar con Prophet. Luego, dividimos el DataFrame en conjuntos de entrenamiento (años 2000-2004) y de prueba (año 2005) dividiéndolos en el tiempo.

En el Paso 4, instanciamos el modelo con estacionalidad aditiva. Además, agregamos la estacionalidad mensual mediante el método add_seasonality con valores sugeridos por la documentación de Prophet. Para ajustar el modelo, utilizamos el método de ajuste, que se conoce de la popular librería scikit-learn.

En el Paso 5, usamos el modelo ajustado para las predicciones. Para crear pronósticos con Prophet, tuvimos que crear un future_DataFrame usando el método make_future_DataFrame e indicando cuántos períodos queríamos obtener (de forma predeterminada, esto se mide en días). Creamos las predicciones utilizando el método de predicción del modelo ajustado.

En el paso 6, Inspeccionamos los componentes del modelo (la descomposición). Para hacerlo, usamos el método plot_components con la predicción DataFrame como argumento.

También estamos interesados en alguna evaluación básica del desempeño del modelo ajustado. Ejecute los siguientes pasos para inspeccionar visualmente los precios del oro previstos frente a los reales en 2005.

```
# 1. Fusionar el conjunto de prueba con las previsiones:  
selected_columns = ['ds', 'yhat_lower', 'yhat_upper', 'yhat']  
  
df_pred = df_pred.loc[:, selected_columns].reset_index(drop=True)  
df_test = df_test.merge(df_pred, on=['ds'], how='left')
```

```
df_test.ds = pd.to_datetime(df_test.ds)
df_test.set_index('ds', inplace=True)
```

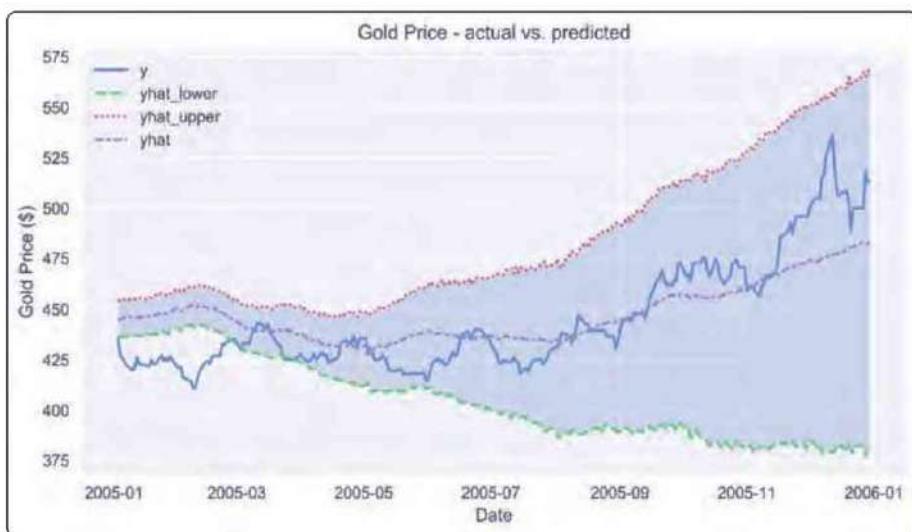
Fusionamos el conjunto de prueba con el DataFrame de predicción. Usamos una combinación izquierda, que devuelve todas las filas de la tabla izquierda (conjunto de prueba) y las filas coincidentes de la tabla derecha (marco de datos de predicción), dejando vacías las filas no coincidentes. De esta manera, también conservamos solo las fechas que estaban en el conjunto de prueba (Prophet creó predicciones para los próximos 365 días, incluidos los fines de semana y posibles días festivos).

2. Trace los valores de prueba frente a las predicciones:

```
fig, ax = plt.subplots(1, 1)

ax = sns.lineplot(data=df_test[['y', 'yhat_lower',
                                'yhat_upper', 'yhat']])
ax.fill_between(df_test.index,
                df_test.yhat_lower,
                df_test.yhat_upper,
                alpha=0.3)
ax.set(title='Gold Price - actual vs. predicted',
       xlabel='Date',
       ylabel='Gold Price ($)')
plt.tight_layout()
# plt.savefig('images/ch3_im5.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:



De la gráfica anterior, podemos ver que Prophet predijo con precisión (al menos visualmente) el precio del oro durante 2005. Fue sólo durante los dos primeros meses que los precios observados estuvieron fuera del intervalo de confianza.

13.3 PRUEBA DE ESTACIONARIEDAD EN SERIES DE TIEMPO

Una serie de tiempo estacionaria es una serie en la que las propiedades estadísticas como la media, la varianza y la covarianza son constantes en el tiempo. La estacionariedad es una característica deseada de las series de tiempo, ya que hace que el modelado y la extrapolación (pronóstico) al futuro sean más factibles. Algunos inconvenientes de los datos no estacionarios son:

- ▀ La varianza puede estar mal especificada por el modelo
- ▀ Peor ajuste del modelo
- ▀ No se pueden aprovechar los patrones valiosos que dependen del tiempo en los datos

En el siguiente ejemplo, se muestra cómo probar la estacionariedad de la serie temporal. Para ello empleamos los siguientes métodos:

- ▀ La prueba Augmented Dickey-Fuller (ADF)
- ▀ La prueba de Kwiatkowski-Phillips-Schmidt-Shin (KPSS)
- ▀ Gráficos de la función de autocorrelación (parcial) (PACF / ACF)

Investigamos la estacionariedad de los precios mensuales del oro desde los años 2000-2011.

Usaremos los mismos datos que usamos en el ejemplo de descomposición de series de tiempo. En el gráfico que presenta la media móvil y la desviación estándar de los precios del oro, ya hemos visto que las estadísticas parecen aumentar con el tiempo, lo que sugiere no estacionariedad.

Ejecute los siguientes pasos para probar la estacionariedad de la serie temporal dada.

1. Importar las librerías.

```
import pandas as pd
import quandl
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, kpss
```

2. Descargar los datos

```
QUANDL_KEY = '{key}' # reemplace {key} con su clave
quandl.ApiConfig.api_key = QUANDL_KEY

df = quandl.get(dataset='WGC/GOLD_MONAVG_USD',
                 start_date='2000-01-01',
                 end_date='2011-12-31')

df.rename(columns={'Value': 'price'}, inplace=True)
df = df.resample('M').last()
```

3. Defina una función para ejecutar la prueba ADF:

```
def adf_test(x):
    """
    Función para realizar la prueba de estacionariedad de Dickey-Fuller aumentada

    Hipótesis nula: La serie de tiempo no es estacionaria
    Hipótesis alternativa: La serie de tiempo es estacionaria

    Parámetros
    -----
    x : pd.Series / np.array
        La serie de tiempo que se comprobará para determinar la estacionariedad.

    Returns
    -----
    results: pd.DataFrame
        Un DataFrame con los resultados de la prueba ADF
    """

    indices = ['Test Statistic', 'p-value',
               '# of Lags Used', '# of Observations Used']

    adf_test = adfuller(x, autolag='AIC')
    results = pd.Series(adf_test[0:4], index=indices)

    for key, value in adf_test[4].items():
        results[f'Critical Value ({key})'] = value

    return results
adf_test(df.price)
```

Ahora, podemos ejecutar la prueba:

```
Test Statistic      3.510499
p-value           1.000000
# of Lags Used   14.000000
# of Observations Used 129.000000
Critical Value (1%) -3.482088
Critical Value (5%) -2.884219
Critical Value (10%) -2.578864
dtype: float64
```

La hipótesis nula de la prueba ADF establece que la serie temporal no es estacionaria. Con un valor p de 1 (o equivalentemente, el estadístico de prueba mayor que el valor crítico para el nivel de confianza seleccionado), no tenemos ninguna razón para rechazar la hipótesis nula, lo que significa que podemos concluir que la serie no es estacionaria.

#4. Definimos una función para ejecutar la prueba KPSS:

```
def kpss_test(x, h0_type='c'):
    """
        Función para realizar La prueba de estacionariedad Kwiatkowski-Phillips-
        Schmidt-Shin
    """

    Hipótesis nula: La serie de tiempo es estacionaria
    Hipótesis alternativa: La serie de tiempo no es estacionaria

    Parámetros
    -----
    x: pd.Series / np.array
        La serie de tiempo que se comprobará para determinar La estacionariedad.
    h0_type: str{'c', 'ct'}
        Indica la hipótesis nula de La prueba KPSS:
        * 'c': Los datos son estacionarios alrededor de una constante
        (predeterminado)
        * 'ct': Los datos son estacionarios en torno a una tendencia.

    Returns
    -----
    results: pd.DataFrame
        Un DataFrame con Los resultados de La prueba KPSS
    """

    indices = ['Test Statistic', 'p-value', '# of Lags']
```

```

kpss_test = kpss(x, regression=h0_type)
results = pd.Series(kpss_test[0:3], index=indices)

for key, value in kpss_test[3].items():
    results[f'Critical Value ({key})'] = value

return results

```

Ahora, podemos ejecutar la prueba:

```
kpss_test(df.price)
```

El código genera el siguiente resumen:

```

Test Statistic      0.985671
p-value            0.010000
# of Lags          14.000000
Critical Value (10%) 0.347000
Critical Value (5%)   0.463000
Critical Value (2.5%) 0.574000
Critical Value (1%)   0.739000
dtype: float64

```

La hipótesis nula de la prueba KPSS es que la serie temporal es estacionaria. Con un valor de p de 0.01 (o estadístico de prueba mayor que el valor crítico seleccionado), tenemos razones para rechazar la hipótesis nula a favor de la alternativa, lo que significa que la serie no es estacionaria.

```

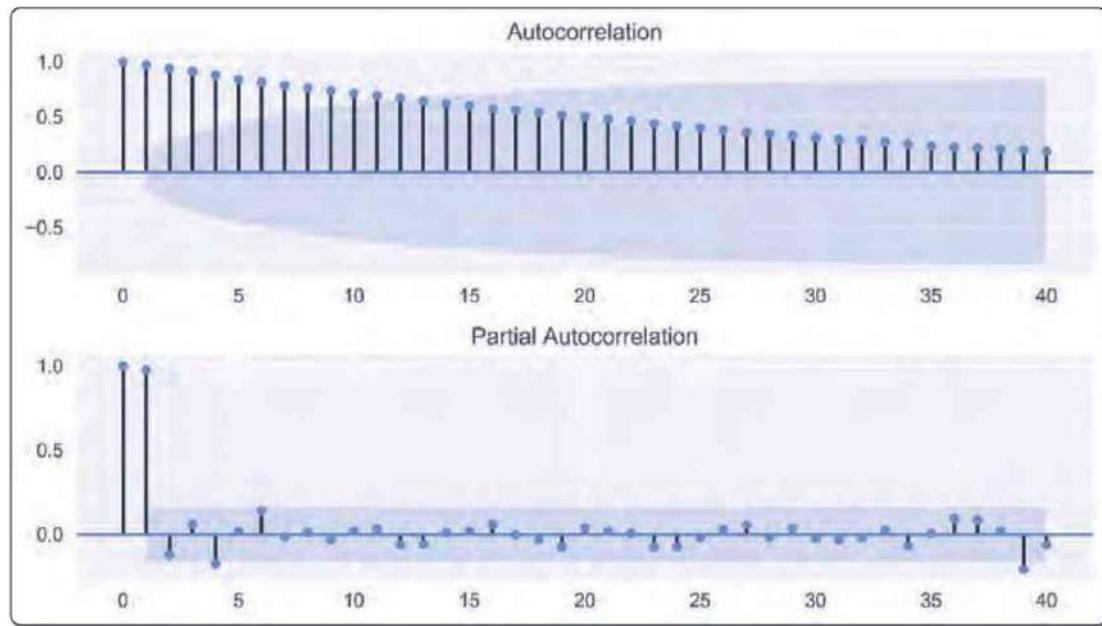
# 5. Genere las gráficas ACF / PACF:
N_LAGS = 40
SIGNIFICANCE_LEVEL = 0.05

fig, ax = plt.subplots(2, 1)
plot_acf(df.price, ax=ax[0], lags=N_LAGS,
          alpha=SIGNIFICANCE_LEVEL)
plot_pacf(df.price, ax=ax[1], lags=N_LAGS,
          alpha=SIGNIFICANCE_LEVEL)

plt.tight_layout()
#plt.savefig('images/ch3_im8.png')
plt.show()

```

El resultado es el siguiente:



En el gráfico ACF, podemos ver que hay autocorrelaciones significativas (por encima del intervalo de confianza del 95%, correspondiente al nivel de significancia del 5% seleccionado). También hay algunas autocorrelaciones significativas en los rezagos 1 y 4 en el gráfico PACF.

Cómo funciona

En el Paso 3, definimos una función utilizada para ejecutar la prueba ADF e imprimir los resultados. Especificamos autolag = 'AIC' en la función adfuller, por lo que el número de retrasos considerados se selecciona automáticamente en función del criterio de información de Akaike (AIC). Alternativamente, podríamos seleccionar este valor manualmente.

Para la función kpss (Paso 4), especificamos el argumento de regresión. Un valor de 'c' corresponde a la hipótesis nula que establece que la serie es estacionaria en el nivel, mientras que 'ct' corresponde a estacionaria en la tendencia (eliminar la tendencia de la serie la haría estacionaria en el nivel).

Para todas las pruebas y las gráficas de autocorrelación, seleccionamos el nivel de significancia del 5%, que es la probabilidad de rechazar la hipótesis nula (H_0) cuando, de hecho, es verdadera.

13.4 CORRECCIÓN DE LA ESTACIONARIEDAD EN SERIES DE TIEMPO

En este ejemplo, analizaremos cómo hacer estacionaria una serie de tiempo no estacionaria mediante el uso de las siguientes transformaciones:

Deflación: contabilización de la inflación en series monetarias utilizando el índice de precios al consumidor (IPC).

Logaritmo natural: acercar la tendencia exponencial a lineal.

Diferenciación: tomando la diferencia entre la observación actual y un valor rezagado (observación x puntos de tiempo antes).

Usamos los mismos datos que usamos en el ejemplo Prueba de estacionariedad en series de tiempo. La conclusión de ese ejemplo fue que la serie de tiempo de los precios mensuales del oro de 2000 a 2011 no era estacionaria.

Cómo hacerlo

Ejecute los siguientes pasos para transformar la serie de no estacionaria a estacionaria.

1. Importar librerías y actualizar datos de inflación:

```
import pandas as pd
import quandl
import cpi
import numpy as np
from datetime import date
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, kpss
from chapter_3_utils import test_autocorrelation

# update the CPI data (if needed)
# cpi.update()
```

2. Descargar los datos:

```
QUANDL_KEY = '{key}' # reemplace {key} con su clave
quandl.ApiConfig.api_key = QUANDL_KEY

df = quandl.get(dataset='WGC/GOLD_MONAVG_USD',
                start_date='2000-01-01',
                end_date='2011-12-31')
```

```
df.rename(columns={'Value': 'price'}, inplace=True)
df = df.resample('M').last()

# 3. Deflacte los precios del oro (a los valores de 2011-12-31 USD) y grafique los resultados:

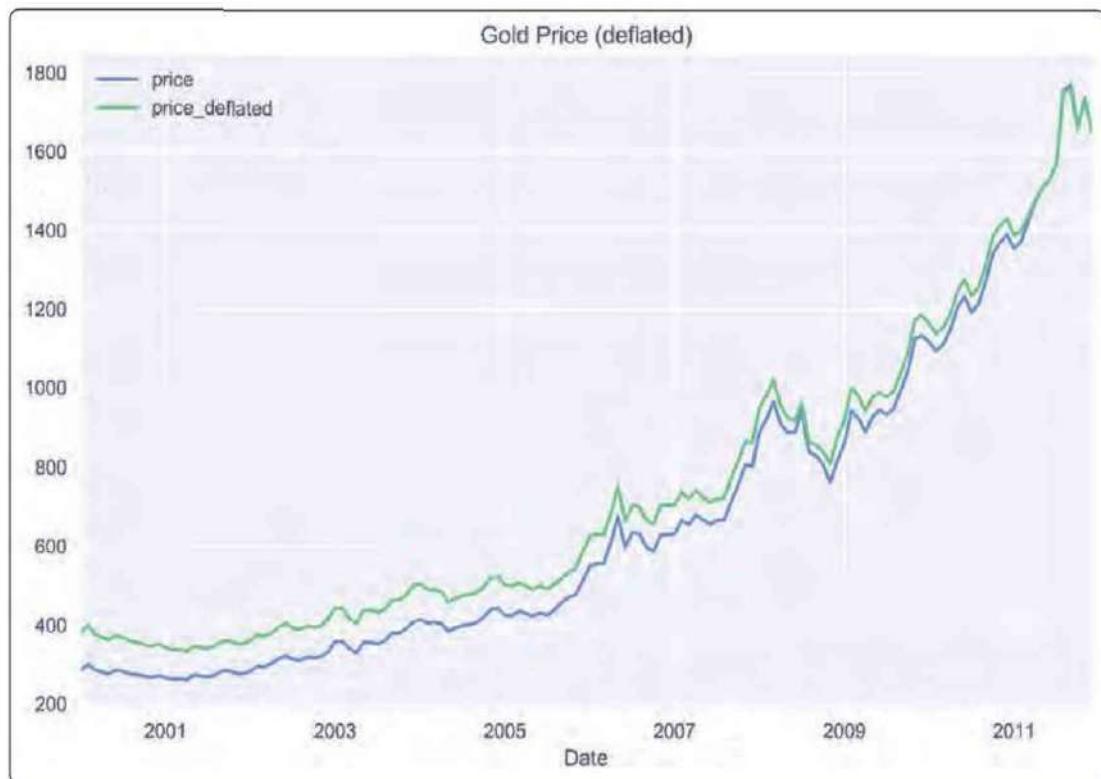
DEFL_DATE = date(2011, 12, 31)

df['dt_index'] = df.index.map(lambda x: x.to_pydatetime().date())
df['price_deflated'] = df.apply(lambda x: cpi.inflate(x.price,
                                                       x.dt_index,
                                                       DEFL_DATE),
                                 axis=1)
df[['price', 'price_deflated']].plot(title='Gold Price (deflated)')

plt.tight_layout()
# plt.savefig('images/ch3_im9.png')
plt.show()
```

Salida

Podemos observar los precios ajustados por inflación en la siguiente gráfica:



También podríamos ajustar los precios del oro a otro momento, siempre que sea el mismo punto para toda la serie.

```
# 4. Deflactar la serie usando un logaritmo natural y graficando con las métricas continuas

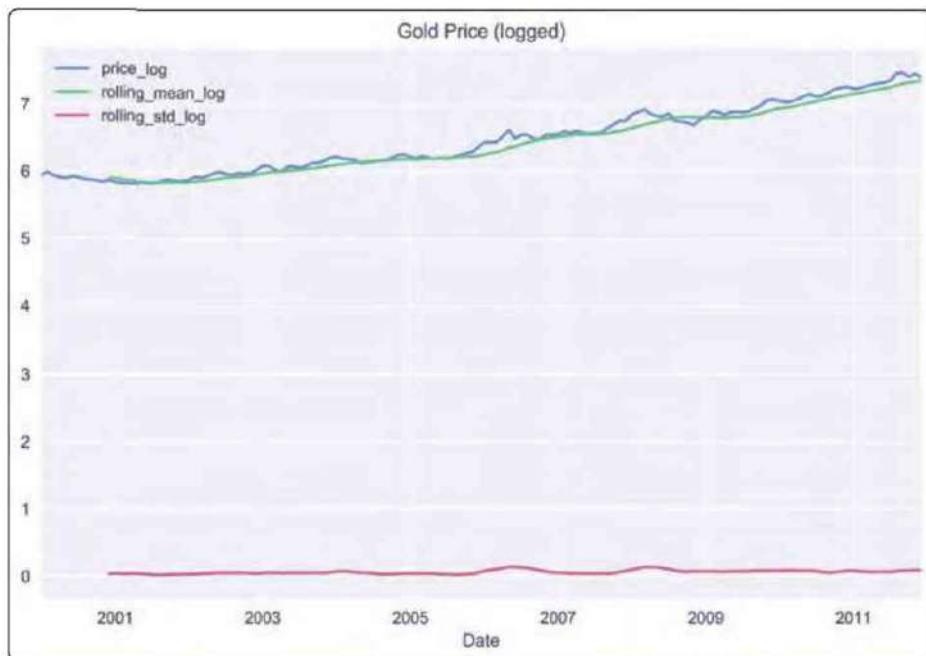
WINDOW = 12
selected_columns = ['price_log', 'rolling_mean_log',
                     'rolling_std_log']

df['price_log'] = np.log(df.price_deflated)
df['rolling_mean_log'] = df.price_log.rolling(WINDOW) \
                           .mean()
df['rolling_std_log'] = df.price_log.rolling(WINDOW) \
                           .std()

df[selected_columns].plot(title='Gold Price (logged)')

plt.tight_layout()
# plt.savefig('images/ch3_im10.png')
plt.show()
```

La ejecución del código da como resultado la siguiente salida:



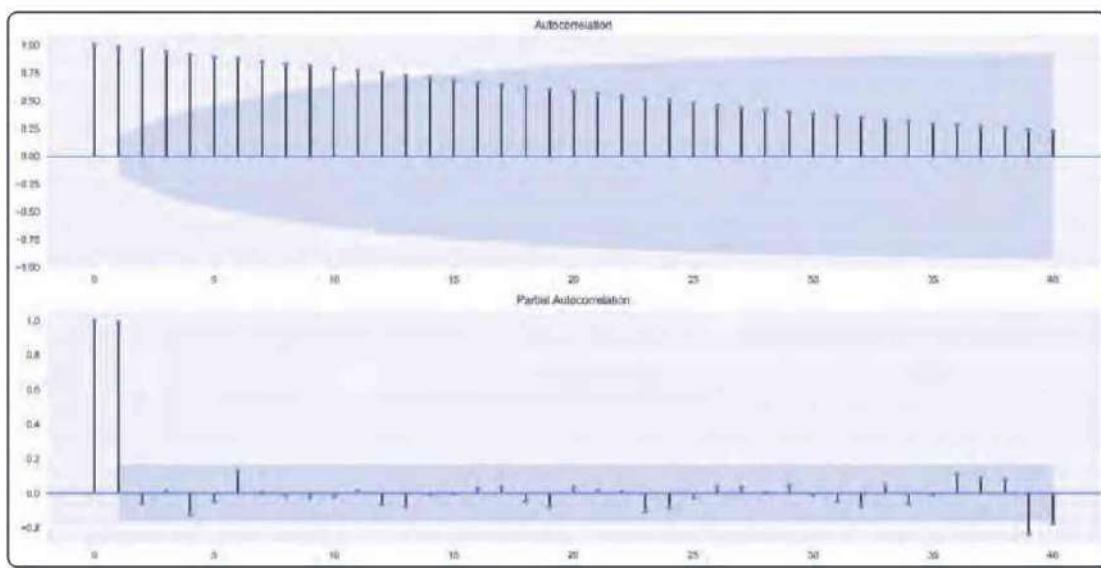
En el gráfico anterior, podemos ver que la transformación logarítmica hizo su trabajo, es decir, hizo lineal la tendencia exponencial.

```
#5. Utilice test_autocorrelation para investigar si la serie se volvió estacionaria:
```

```
fig = test_autocorrelation(df.price_log)

plt.tight_layout()
# plt.savefig('images/ch3_im11.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:



También recibimos los resultados de las pruebas estadísticas:

ADF test statistic: 0.89 (p-val: 0.99)

KPSS test statistic: 1.04 (p-val: 0.01)

Después de inspeccionar los resultados de las pruebas estadísticas y las gráficas de ACF / PACF, podemos concluir que la deflación y un algoritmo natural no fueron suficientes para hacer los precios del oro estacionarios.

A continuación, aplicamos la diferenciación a la serie y graficamos los resultados:

6. Diferenciación de la serie

```

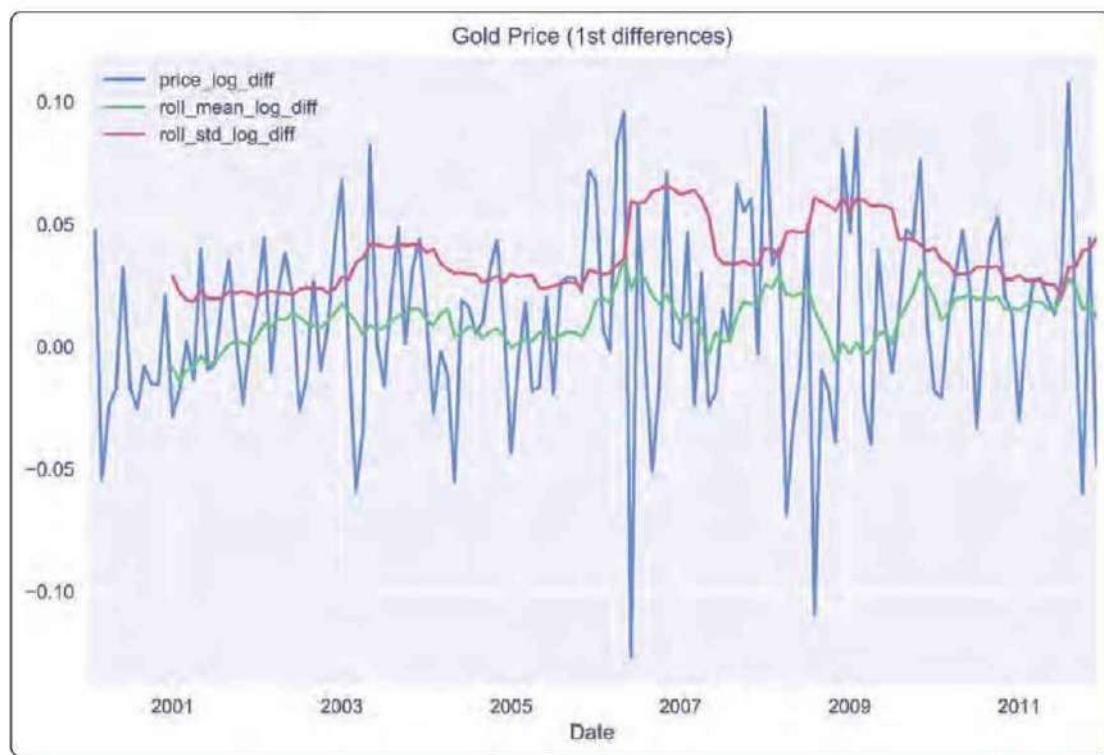
selected_columns = ['price_log_diff', 'roll_mean_log_diff',
                    'roll_std_log_diff']

df['price_log_diff'] = df.price_log.diff(1)
df['roll_mean_log_diff'] = df.price_log_diff.rolling(WINDOW) \
                           .mean()
df['roll_std_log_diff'] = df.price_log_diff.rolling(WINDOW) \
                           .std()
df[selected_columns].plot(title='Gold Price (1st differences)')

plt.tight_layout()
# plt.savefig('images/ch3_im12.png')
plt.show()

```

La ejecución del código da como resultado la siguiente salida:



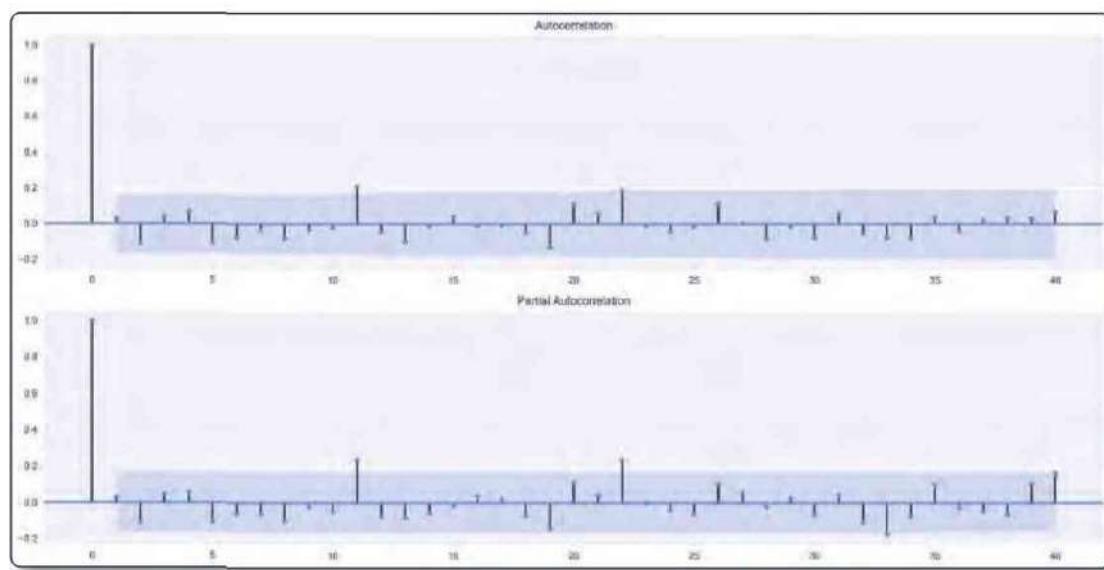
Los precios del oro transformados dan la impresión de estar estacionarios: la serie oscila alrededor de 0 con una varianza más o menos constante. Al menos no hay una tendencia visible.

Probaremos si la serie se volvió estacionaria:

```
# 7. Utilice test_autocorrelation para investigar si la serie se volvió estacionaria f
ig = test_autocorrelation(df.price_log_diff.dropna())

plt.tight_layout()
#plt.savefig('images/ch3_im13.png')
plt.show()
```

La ejecución del código anterior da como resultado la siguiente gráfica:



También recibimos los resultados de las pruebas estadísticas:

```
ADF test statistic: -9.13 (p-val: 0.00)
KPSS test statistic: 0.37 (p-val: 0.09)
```

Después de aplicar las primeras diferencias, la serie se volvió estacionaria al nivel de significancia del 5% (según ambas pruebas). En los gráficos de ACF / PACF, podemos ver que hubo un valor significativo de la función en el retardo 11 y 22. Esto podría indicar algún tipo de estacionalidad o simplemente ser una señal falsa. El uso de un nivel de significancia del 5% significa que el 5% de los valores pueden estar fuera del intervalo de confianza del 95%, incluso cuando el proceso subyacente no muestra ninguna autocorrelación o autocorrelación parcial.

Cómo funciona

Abordamos cada transformación por separado:

Deflación: En el Paso 3, usamos la librería cpi para contabilizar la inflación en dólares estadounidenses. La librería se basa en el índice CPI-U recomendado por la Oficina de Estadísticas Laborales. Para que funcione, creamos una columna de índice artificial que contiene fechas como objetos de la clase `datetime.date`. La función `inflate` toma los siguientes argumentos:

- ▀ valor: El valor en dólares que queremos ajustar
- ▀ year_or_month: la fecha de la que proviene el valor en dólares
- ▀ to: Opcionalmente, la fecha a la que queremos ajustarnos. Si no proporcionamos este argumento, la función se ajustará al año más reciente.

Transformación logarítmica: en el paso 4, aplicamos el logaritmo natural (`np.log`) a todos los valores para hacer lineal la tendencia exponencial. Esta operación se aplicó a precios que ya habían sido corregidos por inflación.

Tomando la primera diferencia: En el Paso 6, usamos el método `diff` para calcular la diferencia entre el valor en el tiempo t y el tiempo $t-1$ (la configuración predeterminada corresponde a la primera diferencia). Podemos especificar un número diferente cambiando el argumento del período.

Los precios del oro considerados no contienen una estacionalidad obvia. Sin embargo, si el conjunto de datos muestra patrones estacionales, existen algunas posibles soluciones:

- ▀ Ajuste por diferenciación: en lugar de usar la diferenciación de primer orden, use una de orden superior, por ejemplo, si hay estacionalidad anual en los datos mensuales, use `diff(12)`
- ▀ Ajuste por modelado: podemos modelar directamente la estacionalidad y luego eliminarla de la serie. Una posibilidad es extraer el componente estacional de `estacional_decomposición` u otro algoritmo de descomposición automática más avanzado. En este caso, debemos restar el componente estacional cuando se usa el modelo aditivo o dividir por él si el modelo es multiplicativo. Otra solución es usar `np.polyfit()` para ajustar el mejor polinomio de un orden elegido a la serie de tiempo seleccionada y luego restarlo de la serie original

La transformación de Box-Cox es otro tipo de ajuste que podemos utilizar en los datos de series de tiempo. Combina diferentes funciones de transformación exponencial para hacer que la distribución sea más similar a la distribución Normal (Gaussiana). Podemos usar boxcox de scipy, que nos permite encontrar automáticamente el valor del parámetro lambda para el mejor ajuste. Una condición a tener en cuenta es que todos los valores de la serie deben ser positivos, la transformación no debe usarse después de las primeras diferencias o cualquier transformación que introduzca valores negativos en la serie.

Una librería llamada pmdarima contiene dos funciones que emplean pruebas estadísticas para determinar cuántas veces debemos diferenciar la serie para lograr la estacionalidad.

Podemos emplear las siguientes pruebas para investigar la estacionariedad: ADF, KPSS y Phillips - Perron (PP):

```
from pmdarima.arima import ndiffs, nsdiffs

print(f"Suggested # of differences (ADF): {ndiffs(df.price, test='adf')}")  
print(f"Suggested # of differences (KPSS): {nsdiffs(df.price, test='kpss')}")  
print(f"Suggested # of differences (PP): {nsdiffs(df.price, test='pp')}")
```

La salida del código anterior es la siguiente:

```
Suggested # of differences (ADF): 1  
Suggested # of differences (KPSS): 2  
Suggested # of differences (PP): 1  
  
print(f"Suggested # of differences (OSCB): {nsdiffs(df.price, m=12,  
test='ocsb')}")  
print(f"Suggested # of differences (CH): {nsdiffs(df.price, m=12, test='ch')}")  
  
Suggested # of differences (OSCB): 0  
Suggested # of differences (CH): 0
```

Para la prueba KPSS, también podemos especificar qué tipo de hipótesis nula queremos contrastar. El valor predeterminado es la estacionariedad de nivel (nulo = ‘nivel’). Los resultados de las pruebas sugieren que la serie (sin ninguna diferenciación) no es estacionaria.

La librería también contiene dos pruebas de diferencias estacionales:

- ▶ Osborn, Chui, Smith, and Birchenhall (OCSB)
- ▶ Canova-Hansen (CH)

Para ejecutarlos, también necesitamos especificar la frecuencia de nuestros datos (12, en nuestro caso) ya que estamos trabajando con datos mensuales:

```
print(f"Suggested # of differences (OSCB): {nsdiffs(df.price, m=12,  
test='ocsb')}")  
print(f"Suggested # of differences (CH): {nsdiffs(df.price, m=12, test='ch')}")
```

El resultado es el siguiente:

```
Suggested # of differences (OSCB): 0  
Suggested # of differences (CH): 0
```

Los resultados sugieren que no hay estacionalidad en los precios del oro.

13.5 MODELADO DE SERIES DE TIEMPO CON MÉTODOS DE SUAVIZADO EXPONENCIAL

Los métodos de suavizado exponencial son adecuados para datos no estacionarios (es decir, datos con tendencia y / o estacionalidad) y funcionan de manera similar a las medias móviles exponenciales. Los pronósticos son promedios ponderados de observaciones pasadas. Estos modelos ponen más énfasis en las observaciones recientes a medida que los pesos porcentuales se vuelven exponencialmente más pequeños con el tiempo. Los métodos de suavizado son populares porque son rápidos (no se requieren muchos cálculos) y relativamente confiables cuando se trata de pronósticos:

Suavizado exponencial simple: el modelo más básico se llama Suavizado exponencial simple (SES). Esta clase de modelos es más adecuada para los casos en que la serie de tiempo considerada no muestra ninguna tendencia o estacionalidad. También funcionan bien con series que tienen pocos puntos de datos.

El modelo está parametrizado por un parámetro de suavizado α con valores entre 0 y 1. Cuanto mayor sea el valor, más peso se dará a las observaciones recientes. Cuando $\alpha = 0$, los pronósticos para el futuro son iguales al promedio de los datos históricos (al que se ajustó el modelo). Cuando $\alpha = 1$, todos los pronósticos tienen el mismo valor que la última observación en los datos de entrenamiento.

La función de pronóstico del Suavizado Exponencial Simple es plana, es decir, todos los pronósticos, independientemente del horizonte temporal, son iguales al mismo valor: el último componente de nivel. Por eso, este método solo es adecuado para series sin tendencia ni estacionalidad.

Método de tendencia lineal de Holt: el modelo de Holt es una extensión de SES que da cuenta de una tendencia en la serie al agregar el componente de tendencia en la especificación del modelo. Este modelo debe usarse cuando hay una tendencia en los datos, pero no estacionalidad.

Un problema con el modelo de Holt es que la tendencia es constante en el futuro, lo que significa que aumenta / disminuye indefinidamente. Es por eso que una extensión del modelo amortigua la tendencia agregando el parámetro de amortiguación, ϕ . Hace que la tendencia converja a un valor constante en el futuro, aplanándolo efectivamente. Hyndman y Athanasopoulos (2018) afirman que ϕ rara vez es menor que 0,8, ya que la amortiguación tiene un efecto muy fuerte para valores menores de ϕ .

La mejor práctica es restringir los valores de ϕ para que se encuentren entre 0,8 y 0,98, porque para $\phi = 1$ el modelo amortiguado es equivalente al modelo sin amortiguación.

A continuación, se muestra cómo aplicar métodos de suavizado a los precios de las acciones mensuales de Google (datos no estacionarios con una tendencia y sin estacionalidad visible). Ajustamos el modelo a los precios de 2010-2017 y hacemos previsiones para 2018.

En las siguientes explicaciones, trazaremos varias líneas en las mismas gráficas, cada una de las cuales representará una especificación de modelo diferente. Por eso queremos asegurarnos de que estas líneas son claramente distinguibles, especialmente en blanco y negro. Por esa razón, desde ahora hasta la última explicación de este capítulo, usaremos una paleta de colores diferente para las tramas, es decir, cubehelix:

En el siguiente código, definimos una lista de cuatro colores. Los usaremos en lugar de usar los códigos de color estándar (rojo / verde / azul / gris).

1. Definir lista de colores

```
import seaborn as sns

plt.set_cmap('cubehelix')
sns.set_palette('cubehelix')

COLORS = [plt.cm.cubehelix(x) for x in [0.1, 0.3, 0.5, 0.7]]
```

A continuación, se ejecutan los siguientes pasos para utilizar los métodos de suavizado exponencial con el fin de crear pronósticos de los precios de las acciones de Google.

2. Importar las librerías

```
import pandas as pd
import numpy as np
import yfinance as yf
from datetime import date
from statsmodels.tsa.holtwinters import (ExponentialSmoothing,
                                           SimpleExpSmoothing,
                                           Holt)
```

3. Descargar los precios ajustados de las acciones de Google:

```
df = yf.download('GOOG',
                 start='2010-01-01',
                 end='2018-12-31',
                 adjusted=True,
                 progress=False)

print(f'Downloaded {df.shape[0]} rows of data.')
```

Salida

```
Downloaded 2264 rows of data.
```

4. Agregado a la frecuencia mensual:

```
goog = df.resample('M') \
    .last() \
    .rename(columns={'Adj Close': 'adj_close'}) \
    .adj_close
```

5. Se crea la división de entrenamiento / prueba:

```
train_indices = goog.index.year < 2018
goog_train = goog[train_indices]
goog_test = goog[~train_indices]

test_length = len(goog_test)
```

```
# 6. Se grafican los precios:
```

```
goog.plot(title="Google's Stock Price")  
  
plt.tight_layout()  
#plt.savefig('images/ch3_im14.png')  
plt.show()
```

El código anterior genera el siguiente gráfico:



```
# 7. Se ajustan tres modelos SES y se crean pronósticos para ellos:
```

```
ses_1 = SimpleExpSmoothing(goog_train).fit(smoothing_level=0.2)  
ses_forecast_1 = ses_1.forecast(test_length)  
  
ses_2 = SimpleExpSmoothing(goog_train).fit(smoothing_level=0.5)  
ses_forecast_2 = ses_2.forecast(test_length)  
  
ses_3 = SimpleExpSmoothing(goog_train).fit()  
alpha = ses_3.model.params['smoothing_level']  
ses_forecast_3 = ses_3.forecast(test_length)
```

```
# 8. Se grafican los precios originales y los resultados de los modelos:
```

```
goog.plot(color=COLORS[0],  
          title='Simple Exponential Smoothing',
```

```
label='Actual',
legend=True)

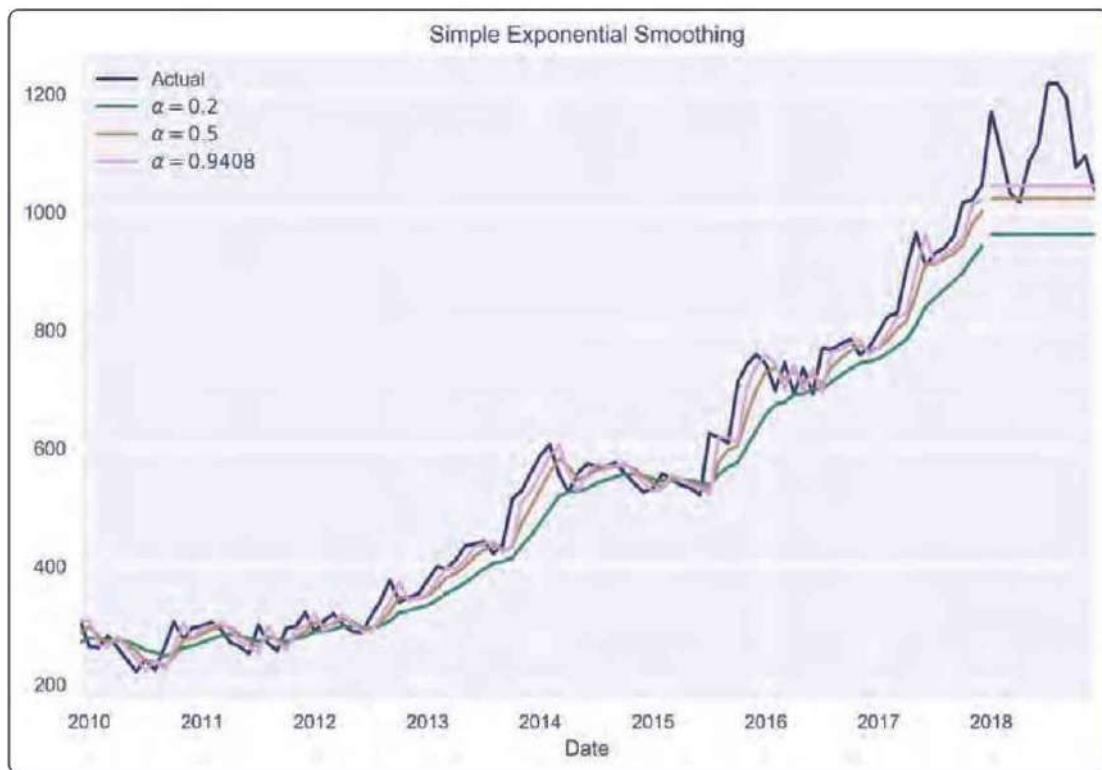
ses_forecast_1.plot(color=COLORS[1], legend=True,
                     label=r'$\alpha=0.2$')
ses_1.fittedvalues.plot(color=COLORS[1])

ses_forecast_2.plot(color=COLORS[2], legend=True,
                     label=r'$\alpha=0.5$')
ses_2.fittedvalues.plot(color=COLORS[2])

ses_forecast_3.plot(color=COLORS[3], legend=True,
                     label=r'$\alpha={0:.4f}$'.format(alpha))
ses_3.fittedvalues.plot(color=COLORS[3])

plt.tight_layout()
# plt.savefig('images/ch3_im15.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:



En el gráfico anterior, podemos ver la característica del SES que describimos en la parte inicial de esta sección: el pronóstico es una línea plana. También podemos ver que el valor óptimo que fue seleccionado por la rutina de optimización de statsmodels está cerca de 1. Además, la línea ajustada del tercer modelo es efectivamente la línea de los precios observados desplazados hacia la derecha.

```
# 9. Se ajustan tres variantes del modelo de suavizado de Holt y se crean pronósticos:
```

```
# Modelo de Holt con tendencia Lineal
hs_1 = Holt(goog_train).fit()
hs_forecast_1 = hs_1.forecast(test_length)

# Modelo de Holt con tendencia exponencial
hs_2 = Holt(goog_train, exponential=True).fit()
# equivalent to ExponentialSmoothing(goog_train, trend='mul').fit()
hs_forecast_2 = hs_2.forecast(test_length)

# Modelo de Holt con tendencia exponencial y atenuación
hs_3 = Holt(goog_train, exponential=False,
            damped=True).fit(damping_slope=0.99)
hs_forecast_3 = hs_3.forecast(test_length)
```

```
# 10. Se trazan los precios originales y los resultados de los modelos:
```

```
goog.plot(color=COLORS[0],
           title="Holt's Smoothing models",
           label='Actual',
           legend=True)

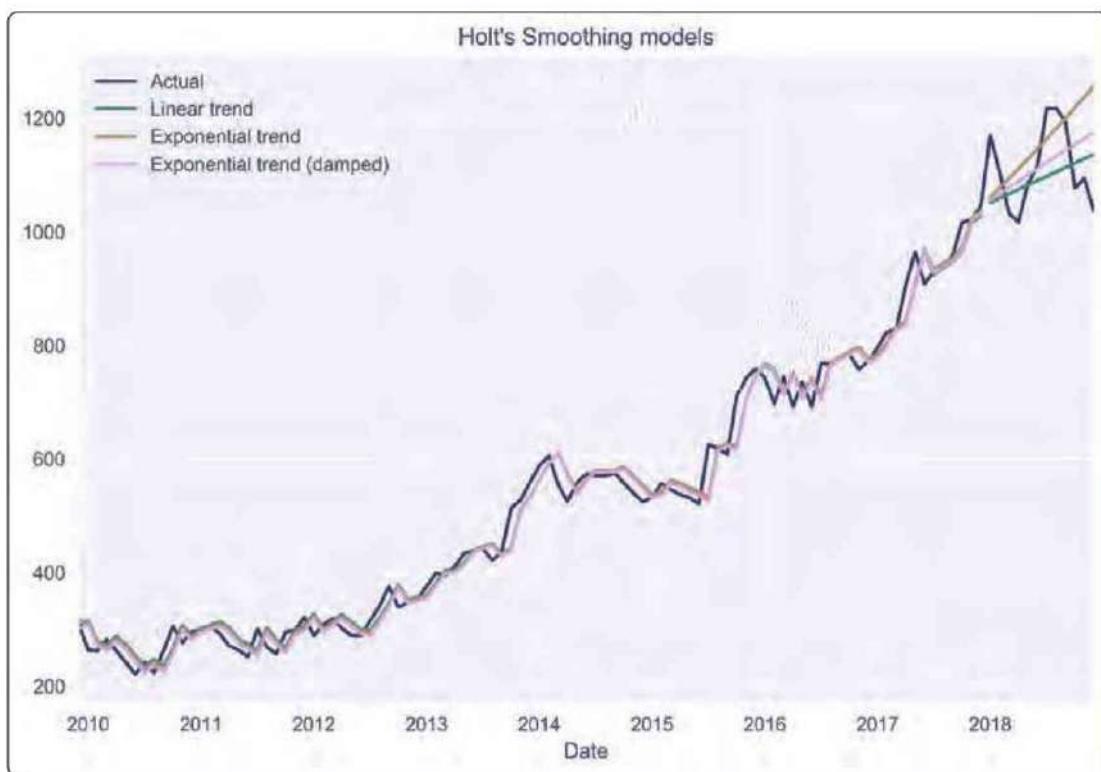
hs_1.fittedvalues.plot(color=COLORS[1])
hs_forecast_1.plot(color=COLORS[1], legend=True,
                   label='Linear trend')

hs_2.fittedvalues.plot(color=COLORS[2])
hs_forecast_2.plot(color=COLORS[2], legend=True,
                   label='Exponential trend')

hs_3.fittedvalues.plot(color=COLORS[3])
hs_forecast_3.plot(color=COLORS[3], legend=True,
                   label='Exponential trend (damped)')

plt.tight_layout()
#plt.savefig('images/ch3_im16.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:



Podemos observar una mejora, las líneas ya no son planas, en comparación con SES.

Cómo funciona

En los pasos 3 a 6, descargamos los precios de las acciones de Google de 2010-2018, volvimos a muestrear los valores a una frecuencia mensual, dividimos los datos en conjuntos de entrenamiento (2010-2017) y de prueba (2018), y graficamos la serie.

En el paso 7, ajustamos tres modelos de SES diferentes usando la clase `SimpleExpSmoothing` y su método de ajuste. Para el ajuste, solo usamos los datos de entrenamiento. Podríamos haber seleccionado manualmente el valor del parámetro de suavizado (`smoothing_level`), sin embargo, la mejor práctica es dejar que `statsmodels` lo optimice para lograr el mejor ajuste. Esta optimización se realiza minimizando la suma de los residuos al cuadrado (errores). Creamos los pronósticos utilizando el método de `forecast`, que requiere la cantidad de períodos para los que queremos pronosticar (que es igual a la longitud del equipo de prueba). En el paso 8, visualizamos los resultados y los comparamos con los precios de las acciones

reales. Los valores ajustados del modelo se extrajeron mediante el método de valores ajustados del modelo ajustado.

En el paso 9, usamos la clase Holt (que es una envoltura de la clase ExponentialSmoothing más general) para ajustar el modelo de tendencia lineal de Holt. De forma predeterminada, la tendencia en el modelo es lineal, pero podemos hacerla exponencial especificando `exponential = True` y agregar amortiguación con `damped = True`. Como en el caso de SES, el uso del método de ajuste sin argumentos da como resultado la ejecución de la rutina de optimización para determinar el valor óptimo del parámetro. Podemos acceder a él ejecutando `fitted_model.params`. En nuestro ejemplo, especificamos manualmente que el valor del parámetro de amortiguación fuera 0,99, ya que el optimizador seleccionó 1 para ser el valor óptimo, y esto sería indistinguible en el gráfico. En el paso 10, visualizamos los resultados.

Existe una extensión del método de Holt llamada Suavizado estacional de Holt-Winter. Da cuenta de la estacionalidad en la serie temporal. No hay una clase separada para este modelo, pero podemos ajustar la clase ExponentialSmoothing agregando `seasonal` y argumentos de `seasonal_periods`.

Sin entrar en demasiados detalles, este método es más adecuado para datos con tendencia y estacionalidad. Hay dos variantes de este modelo y tienen estacionalidades aditivas o multiplicativas. En el primero, las variaciones estacionales son más o menos constantes a lo largo de la serie temporal. En este último, las variaciones cambian en proporción al paso del tiempo.

Empezamos ajustando los modelos:

```
SEASONAL_PERIODS = 12

# Modelo de Holt-Winter con tendencia exponencial
hw_1 = ExponentialSmoothing(goog_train,
                            trend='mul',
                            seasonal='add',
                            seasonal_periods=SEASONAL_PERIODS).fit()

hw_forecast_1 = hw_1.forecast(test_length)

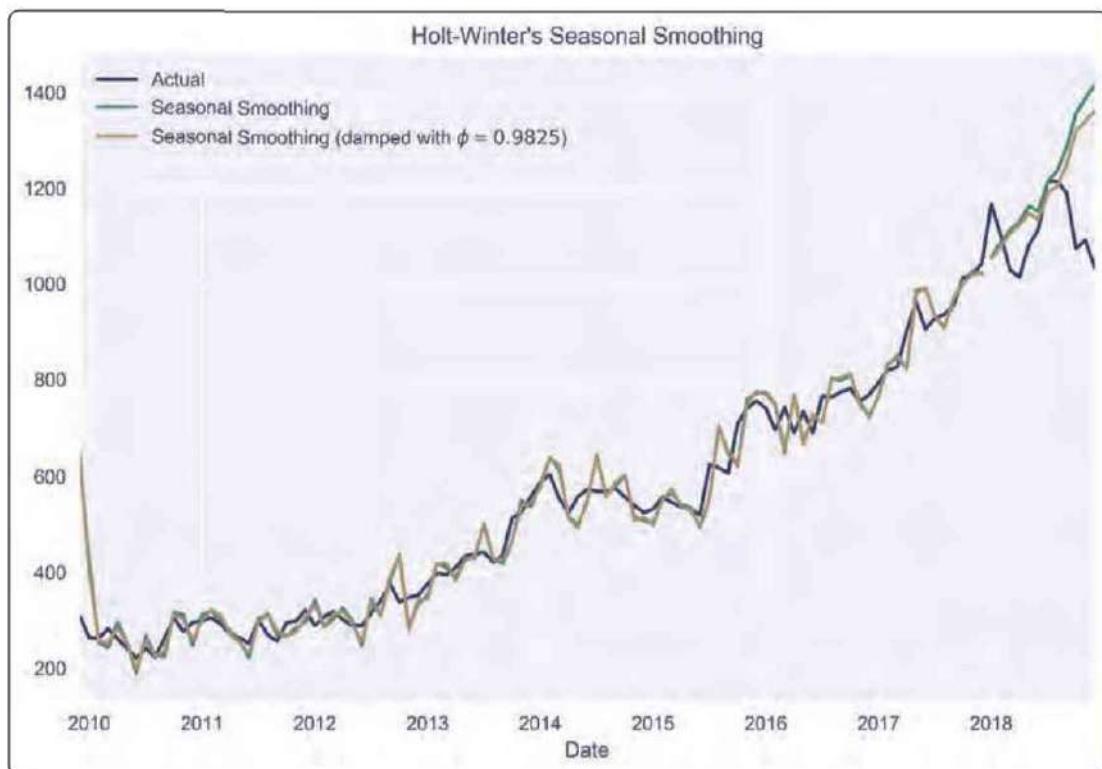
# Modelo de Holt-Winter con tendencia exponencial y atenuación
hw_2 = ExponentialSmoothing(goog_train,
                            trend='mul',
                            seasonal='add',
                            seasonal_periods=SEASONAL_PERIODS,
                            damped=True).fit()

hw_forecast_2 = hw_2.forecast(test_length)
```

Luego, graficamos los resultados:

```
goog.plot(color=COLORS[0],  
          title="Holt-Winter's Seasonal Smoothing",  
          label='Actual',  
          legend=True)  
  
hw_1.fittedvalues.plot(color=COLORS[1])  
hw_forecast_1.plot(color=COLORS[1], legend=True,  
                    label='Seasonal Smoothing')  
  
phi = hw_2.model.params['damping_trend']  
plot_label = f'Seasonal Smoothing (damped with $\phi={phi:.4f}$)'  
  
hw_2.fittedvalues.plot(color=COLORS[2])  
hw_forecast_2.plot(color=COLORS[2], legend=True,  
                    label=plot_label)  
  
plt.tight_layout()  
#plt.savefig('images/ch3_im17.png')  
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:



De los pronósticos graficados, podemos ver que el modelo es más flexible en comparación con los modelos de tendencia lineal de SES y Holt. Los valores extremos ajustados al comienzo de la serie son el resultado de no tener suficientes observaciones para mirar hacia atrás (seleccionamos períodos_estacionales = 12 ya que estamos tratando con datos mensuales).

13.6 MODELADO DE SERIES DE TIEMPO CON MODELOS DE CLASE ARIMA

Los modelos ARIMA son una clase de modelos estadísticos que se utilizan para analizar y pronosticar datos de series de tiempo. Su objetivo es hacerlo describiendo las autocorrelaciones en los datos. ARIMA significa Autoregressive Integrated Moving Average y es una extensión de un modelo ARMA más simple. El objetivo del componente de integración adicional es garantizar la estacionariedad de la serie porque, a diferencia de los modelos de suavizado exponencial, la clase ARIMA requiere que la serie de tiempo sea estacionaria. En los siguientes párrafos, repasaremos brevemente los componentes básicos de los modelos ARIMA.

Modelo AR (autorregresivo):

- ▀ Este tipo de modelo utiliza la relación entre una observación y sus valores rezagados
- ▀ En el contexto financiero, el modelo autorregresivo intenta tener en cuenta el impulso y los efectos de reversión a la media

I (integración)

- ▀ La integración, en este caso, se refiere a diferenciar la serie de tiempo original (restando el valor del período anterior del valor del período actual) para hacerla estacionaria
- ▀ El parámetro responsable de la integración es d (llamado grado / orden de diferenciación) e indica el número de veces que necesitamos aplicar la diferenciación

Modelo MA (media móvil)

- ▀ Este tipo de modelo utiliza la relación entre una observación y los términos de ruido blanco (choques que ocurrieron en las últimas q observaciones).
- ▀ En el contexto financiero, los modelos de promedio móvil intentan dar cuenta de los choques impredecibles (observados en los residuales) que influyen en la serie temporal observada. Algunos ejemplos de estos choques podrían ser desastres naturales, noticias de última hora relacionadas con una determinada empresa, etc.

Todos estos componentes encajan y se especifican directamente en la notación comúnmente utilizada conocida como ARIMA (p, d, q).

Configurando los parámetros del modelo ARIMA, podemos obtener algunos casos especiales:

- ▶ ARIMA (0,0,0): ruido blanco
- ▶ ARIMA (0,1,0) sin constante: Paseo aleatorio
- ▶ ARIMA ($p, 0, q$): ARMA (p, q)
- ▶ ARIMA ($p, 0, 0$): modelo AR (p)
- ▶ ARIMA ($0, 0, q$): modelo MA (q)
- ▶ ARIMA (0,1,2): Modelo de Holt amortiguado
- ▶ ARIMA (0,1,1) sin constante: modelo SES
- ▶ ARIMA (0,2,2): método lineal de Holt con errores aditivos

Una de las debilidades conocidas de los modelos de clase ARIMA en el contexto financiero es su incapacidad para capturar la agrupación de volatilidad que se observa en la mayoría de los activos financieros.

A continuación, pasaremos por todos los pasos necesarios para estimar correctamente un modelo ARIMA y aprenderemos a verificar que se ajusta correctamente a los datos. Para este ejemplo, utilizamos los precios de las acciones semanales de Google de 2015 a 2018.

Cómo hacerlo

Ejecute los siguientes pasos para ajustar y evaluar un modelo ARIMA utilizando el precio de las acciones de Google.

1. Importar librerías

```
import yfinance as yf
import pandas as pd
import numpy as np
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.stats.diagnostic import acorr_ljungbox
import scipy.stats as scs
```

```
# 2. Descargue los precios de las acciones de Google y vuelva a muestrear con frecuencia semanal:
```

```
df = yf.download('GOOG',
                 start='2015-01-01',
                 end='2018-12-31',
                 adjusted=True,
                 progress=False)

goog = df.resample('W') \
    .last() \
    .rename(columns={'Adj Close': 'adj_close'}) \
    .adj_close
```

```
# 3. Aplique las primeras diferencias a la serie de precios y grafíquelas juntas:
```

```
goog_diff = goog.diff().dropna()

fig, ax = plt.subplots(2, sharex=True)
goog.plot(title = "Google's stock price", ax=ax[0])
goog_diff.plot(ax=ax[1], title='First Differences')

plt.tight_layout()
# plt.savefig('images/ch3_im18.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:

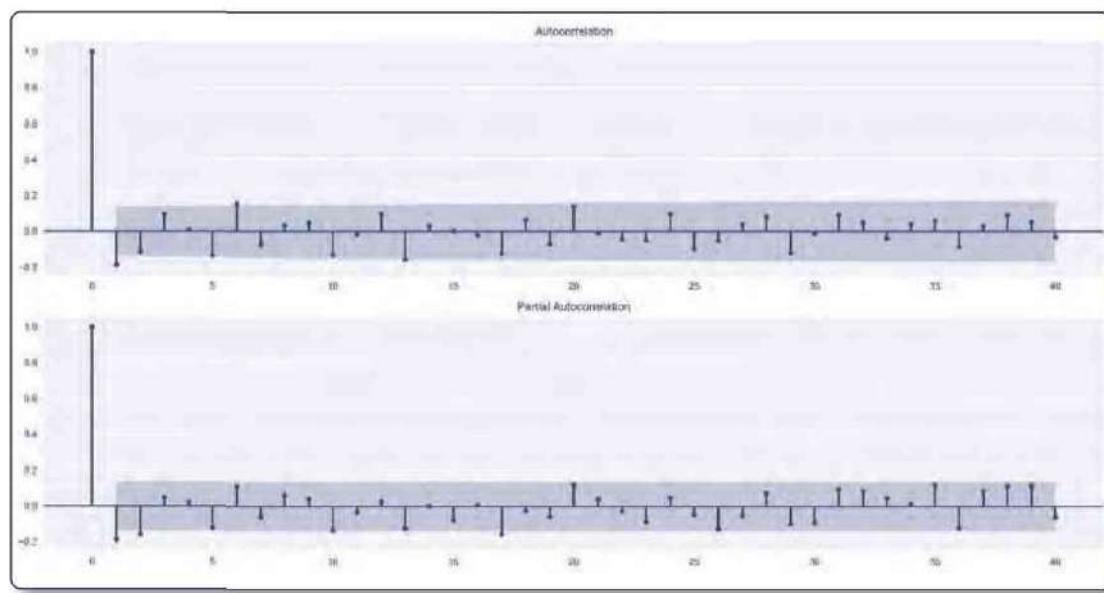


```
# 4. Pruebe la serie diferenciada para determinar la estacionariedad
```

```
fig = test_acf_pacf(goog_diff)

plt.tight_layout()
# plt.savefig('images/ch3_im19.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:



También recibimos los resultados de las pruebas estadísticas:

```
ADF test statistic: -12.79 (p-val: 0.00)
KPSS test statistic: 0.11 (p-val: 0.10)
```

Los resultados indican que los precios diferenciados son estacionarios.

```
# 5. Con base en los resultados de las pruebas, se especifica el modelo ARIMA y
se ajusta a los datos:
```

```
arima = ARIMA(goog, order=(2, 1, 1)).fit(disp=0)
arima.summary()
```

Obtenemos el siguiente resultado:

Out[40]:	Dep. Variable:	D.adj_close	No. Observations:	208		
Model:	ARIMA(2, 1, 1)		Log Likelihood	-987.233		
Method:	css-mle		S.D. of innovations	27.859		
Date:	Fri, 24 Jan 2020		AIC	1984.466		
Time:	23:47:31		BIC	2001.154		
Sample:	01-11-2015		HQIC	1991.214		
	- 12-30-2018					
	coef	std err	z	P> z	[0.025	0.975]
const	2.4700	1.441	1.714	0.088	-0.354	5.294
ar.L1.D.adj_close	-0.3908	0.280	-1.398	0.164	-0.939	0.157
ar.L2.D.adj_close	-0.1910	0.082	-2.322	0.021	-0.352	-0.030
ma.L1.D.adj_close	0.1781	0.280	0.637	0.525	-0.370	0.726
	Real	Imaginary	Modulus	Frequency		
AR.1	-1.0233	-2.0467j	2.2883	-0.3238		
AR.2	-1.0233	+2.0467j	2.2883	0.3238		

6. Preparamos una función para diagnosticar el ajuste del modelo en función de sus residuos:

'''

Función para diagnosticar el ajuste de un modelo ARIMA investigando los residuos.

`def arima_diagnostics(resids, n_Lags=40):`

Parámetros

`resids : np.array`

Una matriz que contiene los residuos de un modelo ajustado

`n_lags : int`

Número de retrasos para la gráfica de autocorrelación

Returns

`fig : matplotlib.figure.Figure`

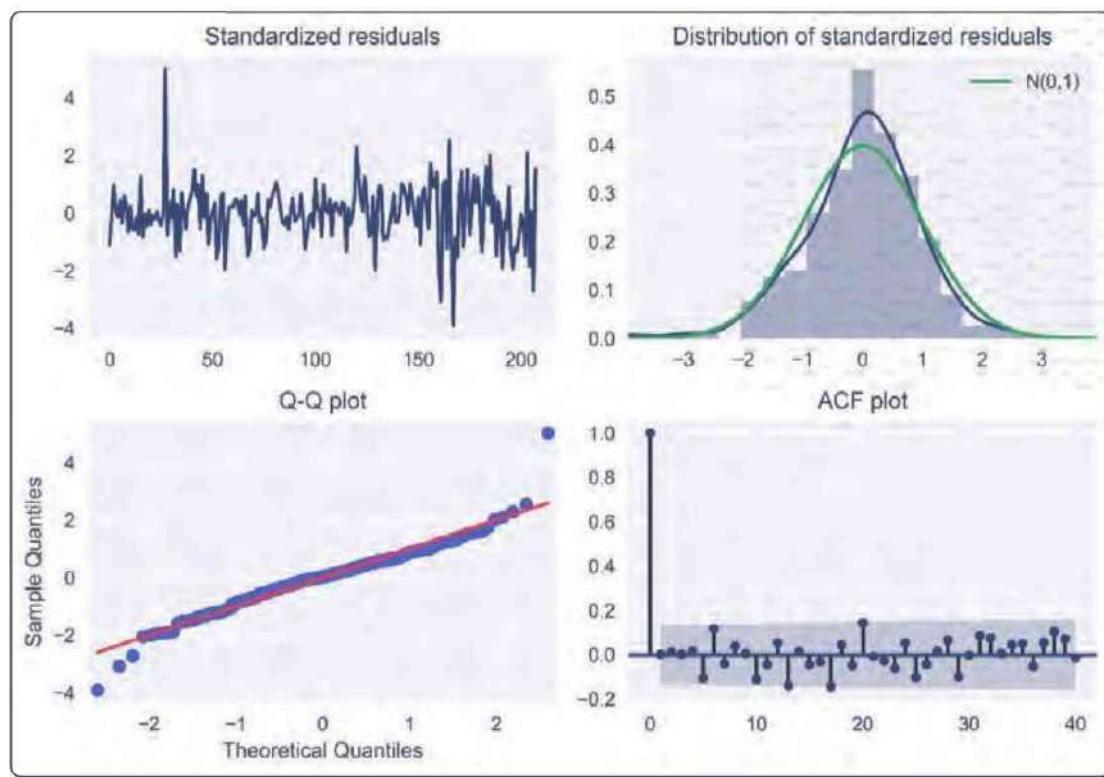
Figura creada

```
'''  
  
# Crear subtramas  
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)  
  
r = resids  
resids = (r - np.nanmean(r)) / np.nanstd(r)  
resids_nonmissing = resids[~(np.isnan(resids))]  
  
  
# Residuales sobre el tiempo  
sns.lineplot(x=np.arange(len(resids)), y=resids, ax=ax1)  
ax1.set_title('Standardized residuals')  
  
# Distribución de residuales  
x_lim = (-1.96 * 2, 1.96 * 2)  
r_range = np.linspace(x_lim[0], x_lim[1])  
norm_pdf = scs.norm.pdf(r_range)  
  
sns.distplot(resids_nonmissing, hist=True, kde=True,  
             norm_hist=True, ax=ax2)  
ax2.plot(r_range, norm_pdf, 'g', lw=2, label='N(0,1)')  
ax2.set_title('Distribution of standardized residuals')  
ax2.set_xlim(x_lim)  
ax2.legend()  
  
# Gráfico Q-Q  
qq = sm.qqplot(resids_nonmissing, line='s', ax=ax3)  
ax3.set_title('Q-Q plot')  
  
# Gráfico ACF  
plot_acf(resids, ax=ax4, lags=n_lags, alpha=0.05)  
ax4.set_title('ACF plot')  
  
return fig
```

7. Se prueban los residuos del modelo ARIMA ajustado:

```
arima_diagnostics(arima.resid, 40)  
  
plt.tight_layout()  
#plt.savefig('images/ch3_im21.png')  
plt.show()
```

La distribución de los residuos estandarizados se parece a la distribución normal:



El promedio de los residuales es cercano a 0 (-0,05), y al inspeccionar el gráfico ACF se llega a la conclusión de que los residuales no están correlacionados. Estas dos características hablan a favor de un buen ajuste. Sin embargo, las colas de la distribución son un poco más pesadas que en normalidad, lo que podemos observar en el gráfico Q-Q.

8. Aplicamos la prueba de Ljung-Box para no autocorrelación en los residuos y graficamos los resultados:

```
ljung_box_results = acorr_ljungbox(arima.resid)

fig, ax = plt.subplots(1, figsize=[16, 5])
sns.scatterplot(x=range(len(ljung_box_results[1])),  

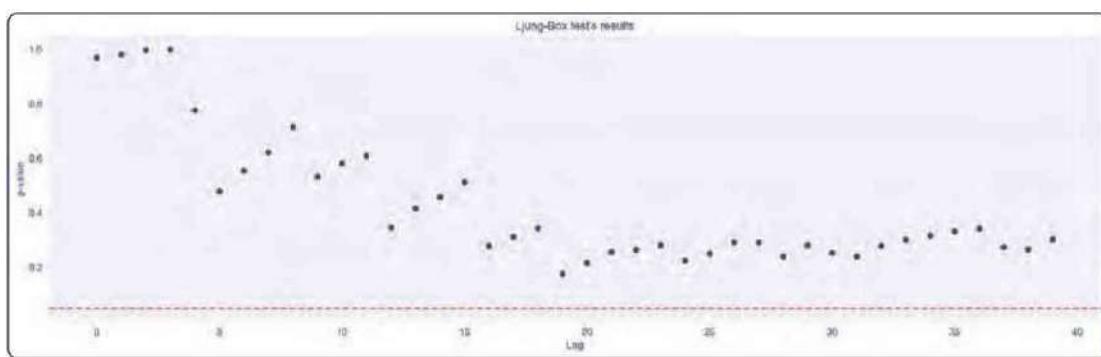
                 y=ljung_box_results[1],  

                 ax=ax)
ax.axhline(0.05, ls='--', c='r')
ax.set(title="Ljung-Box test's results",
       xlabel='Lag',
```

```
ylabel='p-value')

plt.tight_layout()
#plt.savefig('images/ch3_im22.png')
plt.show()
```

La ejecución del código da como resultado la siguiente gráfica:



Los resultados de la prueba de Ljung-Box no nos dan ninguna razón para rechazar la hipótesis nula de no autocorrelación significativa para ninguno de los rezagos seleccionados. Esto indica un buen ajuste del modelo.

Cómo funciona

En el paso 2 comenzamos descargando los precios de las acciones de Google de los años dados y volviéndolos a muestrear a la frecuencia semanal tomando el último precio de cierre (ajustado) de cada semana. En el Paso 3, aplicamos la primera diferencia para hacer que la serie sea estacionaria.

Si queremos diferenciar una serie dada más de una vez, deberíamos usar la función `np.diff` ya que implementa la diferenciación recursiva. El uso del método `diff` de un `DataFrame / Series` con `periodos > 1` da como resultado la diferencia entre las observaciones actuales y la de tantos períodos anteriores.

El paso 5 es muy importante porque determinamos el orden del modelo ARIMA con base a estos resultados. Probamos la estacionariedad usando una función personalizada llamada `test_autocorrelation`. Primero, la serie resultó ser estacionaria, por lo que sabíamos que el orden de integración era $d = 1$. Determinamos el orden de retraso sugerido (p) observando el último retraso después de que la función PACF cruzó el intervalo de confianza. En este caso, fue $p = 2$. De manera análoga, para el orden de promedio móvil, miramos el gráfico ACF para determinar $q = 1$. De esta

manera, especificamos el modelo como ARIMA (2,1,1) en el paso 5. Imprimimos un resumen utilizando el método de resumen del modelo ajustado.

Hyndman y Athanasopoulos (2018) advirtieron que si tanto p como q son positivos, las gráficas ACF / PACF podrían no ser útiles para determinar la especificación del modelo ARIMA.

En el Paso 6 y el Paso 7, investigamos la bondad del ajuste observando los residuos del modelo. Si el ajuste es bueno, los residuos deben ser similares al ruido blanco. Es por eso que usamos cuatro tipos diferentes de gráficos para investigar qué tan cerca se parecen los residuos al ruido blanco.

Finalmente en el paso 8, empleamos la prueba de Ljung-Box (la función `acorr_ljungbox` de `statsmodels`) para no tener una correlación significativa y graficamos los resultados.

Diferentes fuentes sugieren un número diferente de rezagos a considerar en la prueba de Ljung-Box. El valor predeterminado en `statsmodels` es `min((nobs // 2 - 2), 40)`, mientras que otras variantes de uso común incluyen `min(20, nobs - 1)` y `ln(nobs)`.

AUTO-ARIMA: como la selección manual de los parámetros ARIMA podría no conducir a descubrir la especificación óptima del modelo, existe una librería llamada `pmdarima` (que transfiere las funcionalidades del famoso paquete R llamado pronóstico a Python). La clase clave de la librería se llama `auto_arima` y se ajusta automáticamente al mejor modelo para nuestra serie de tiempo.

Para hacerlo, necesitamos introducir una métrica que la función optimizará. Una opción popular es Akaike Information Criterion (AIC), que ofrece una compensación entre la bondad de ajuste del modelo y su simplicidad: el AIC se ocupa de los riesgos de sobreajuste y desajuste. Cuando comparamos varios modelos, cuanto menor es el valor de AIC, mejor es el modelo.

`auto_arima` itera sobre el rango especificado de parámetros posibles y selecciona el modelo con el AIC más bajo. También facilita la estimación de modelos SARIMA.

Nos gustaría verificar si el modelo que seleccionamos con base en los gráficos de ACF / PACF es el el mejor que pudimos haber seleccionado.

Comenzamos importando la librería.

```
import pmdarima as pm
```

Ejecutamos `auto_arima` con la mayoría de las configuraciones establecidas en los valores predeterminados. Solo excluimos la estacionalidad potencial.

```
auto_arima = pm.auto_arima(goog,
                           error_action='ignore',
                           suppress_warnings=True,
                           seasonal=False)
auto_arima.summary()
```

La ejecución del código genera el siguiente resumen:

Out[47]:	Dep. Variable:	D.y	No. Observations:	208
	Model:	ARIMA(0, 1, 1)	Log Likelihood	-988.749
	Method:	css-mle	S.D. of innovations	28.065
	Date:	Fri, 24 Jan 2020	AIC	1983.497
	Time:	23:59:16	BIC	1993.510
	Sample:	1	HQIC	1987.546

	coef	std err	z	P> z	[0.025	0.975]
const	2.4561	1.486	1.653	0.100	-0.456	5.368
ma.L1.D.y	-0.2376	0.071	-3.339	0.001	-0.377	-0.098

	Real	Imaginary	Modulus	Frequency
MA.1	4.2096	+0.0000j	4.2096	0.0000

Parece que un modelo más simple proporciona un mejor ajuste. ARIMA (0,1,1) corresponde en realidad a uno de los casos especiales: SES.

En el siguiente paso intentamos sintonizar la búsqueda de los parámetros óptimos:

```
auto_arima = pm.auto_arima(goog,
                           error_action='ignore',
                           suppress_warnings=True,
                           seasonal=False,
                           stepwise=False,
```

```
approximation=False,
n_jobs=-1)

auto_arima.summary()
```

Recibimos el siguiente resumen:

Out[48]:	Dep. Variable:	D.y	No. Observations:	208
	Model:	ARIMA(3, 1, 2)	Log Likelihood	-982.698
	Method:	css-mle	S.D. of innovations	27.235
	Date:	Fri, 24 Jan 2020	AIC	1979.397
	Time:	23:59:29	BIC	2002.759
	Sample:	1	HQIC	1988.843

	coef	std err	z	P> z	[0.025	0.975]
const	2.4675	1.478	1.670	0.096	-0.429	5.364
ar.L1.D.y	-1.5445	0.124	-12.445	0.000	-1.788	-1.301
ar.L2.D.y	-1.1844	0.132	-8.982	0.000	-1.443	-0.926
ar.L3.D.y	-0.3043	0.072	-4.230	0.000	-0.445	-0.163
ma.L1.D.y	1.3587	0.116	11.731	0.000	1.132	1.586
ma.L2.D.y	0.7929	0.119	6.655	0.000	0.559	1.026

	Real	Imaginary	Modulus	Frequency
AR.1	-0.7904	-0.8927j	1.1924	-0.3653

Esta vez, el modelo sugerido es ARIMA (3,1,2), que tiene un AIC más bajo. Entonces, ¿por qué la ejecución inicial no descubrió este modelo? La razón es que, de forma predeterminada, auto_arima usa un algoritmo paso a paso para recorrer el espacio de parámetros. Además, existen algunas aproximaciones para acelerar la búsqueda. Generalmente, se ajustará un conjunto mucho mayor de modelos si desactivamos tanto el paso a paso como la aproximación. Establecemos n_jobs = -1 para usar todos los núcleos para la búsqueda.

Podemos utilizar el método plot_diagnostics de un pm.auto_arima ajustado para obtener una evaluación de los residuales similar a la que se obtuvo a través de la función arima_diagnostics personalizada para los modelos estimados usando statsmodels.

También hay muchas configuraciones diferentes con las que podemos experimentar, como las siguientes:

- Seleccionar el valor inicial para la búsqueda
- Limitando los valores máximos de parámetros en la búsqueda
- Seleccionar diferentes pruebas estadísticas para determinar el número de diferencias (también estacionales)
- Seleccionar un período de evaluación fuera de la muestra (`out_of_sample_size`). Esto hará que el algoritmo ajuste los modelos a los datos hasta cierto punto en el tiempo (la última observación - `out_of_sample_size`) y evalúe en el conjunto retenido

Enumeramos algunas de las extensiones notables del marco ARIMA:

- ARIMAX: Agrega variable (s) exógena (s) al modelo
- SARIMA (ARIMA estacional): amplía ARIMA para tener en cuenta la estacionalidad en la serie temporal. La especificación completa es SARIMA (p, d, q) (P, D, Q) m, donde los parámetros en mayúscula son análogos a los originales, pero se refieren al componente estacional de la serie temporal. m se refiere al período de estacionalidad

En statsmodels, los modelos ARIMA se pueden instalar usando dos clases: ARIMA y SARIMAX. Este último es más flexible y nos permite incluir variables exógenas, así como contabilizar el componente estacional.

Sin embargo, los modelos de estas dos clases utilizan una formulación diferente de ARIMA, donde este último utiliza la formulación de espacio de estados. Es por eso que ajustar el mismo ARIMA (1,1,1) usando estas dos clases producirá resultados ligeramente diferentes.

14

INTEGRACIÓN DE PYTHON CON EXCEL

Excel es una aplicación de hoja de cálculo que fue desarrollada por Microsoft en el año 1987. Es oficialmente compatible con casi todos los sistemas operativos como Windows, Macintosh, Android, etc. Viene preinstalado con el sistema operativo Windows y se puede integrar fácilmente con otras plataformas de SO. Microsoft Excel es la mejor y más accesible herramienta cuando se trata de trabajar con datos estructurados.

Excel permite organizar, analizar y almacenar los datos en forma de fila-columna tabular. Puede realizar cálculos y crear tablas dinámicas, gráficos y mucho más. Desde su lanzamiento, este software ganó popularidad y se usa ampliamente en muchos campos y dominios diferentes en todo el mundo. El campo de las finanzas no es la excepción y podríamos afirmar que Excel es la principal herramienta de trabajo de los profesionales de las finanzas.

Desde el día en que se creó Internet, su uso ha crecido exponencialmente, al igual que la cantidad de datos. El crecimiento de los datos ha impulsado la necesidad de que las personas comprendan cómo analizarlos. Las empresas y los gobiernos recopilan inmensas cantidades de datos.

Cuando trabaje con datos, deberá ocuparse de hojas de cálculo en algún momento; sin embargo, trabajar directamente con hojas de cálculo puede resultar en un dolor de cabeza, especialmente cuando se tienen grandes cantidades de datos y se requieren cálculos con alguna complejidad. Para deshacerse de este problema, los desarrolladores de Python idearon formas de leer, escribir y analizar todo tipo de formatos de archivo, incluidas las hojas de cálculo.

Este capítulo tratará principalmente sobre cómo usar el lenguaje de programación Python y trabajar con Excel. Le proporcionará una experiencia práctica con la librería pandas que puede utilizar para cargar, leer, escribir y analizar hojas de cálculo.

14.1 PANDAS PARA LEER UN ARCHIVO DE EXCEL

Pandas es la librería estándar para la manipulación de datos dentro del lenguaje de programación Python. Si trabaja con datos en cualquier forma usando Python, requiere pandas. Para obtener pandas, simplemente realice la instalación con el administrador de paquetes PIP:

```
pip install pandas --upgrade
```

La librería pandas proporciona varios métodos convenientes para leer desde diferentes fuentes de datos, incluidos archivos Excel y CSV. Exploraremos dos métodos aquí: `pd.read_excel()` y `pd.read_csv()`.

14.1.1 Método y argumentos `pd.read_excel()`

El método `read_excel()` contiene aproximadamente dos docenas de argumentos, la mayoría de los cuales son opcionales. Para empezar, solo veremos algunos argumentos aquí.

```
read_excel(io, sheet_name=0, header=0, names=None, usecols=None)
```

- **io** suele ser una de dos cosas: una cadena que representa una ruta de archivo o un objeto `ExcelFile`
- **sheet_name** puede ser una cadena o un entero, esta es la hoja que se desea que pandas lea
- **header** suele ser un número entero para decirle a pandas qué fila de la hoja usar como encabezado del `DataFrame`
- **names** normalmente una lista de nombres que puede utilizar como encabezado de columna
- **usecols** puede ser un número entero, una cadena o una lista para indicar a pandas que solo extraiga ciertas columnas del archivo de Excel

Veamos algunos ejemplos usando este archivo de muestra (`'users.xlsx'`).

Por convención, “`pd`” es la abreviatura de “`pandas`” y “`df`” es la abreviatura de “`DataFrame`”.

```
import pandas as pd  
df= pd.read_excel('users.xlsx')
```

io y sheet_name

- ▶ pd.read_excel ('users.xlsx') es la forma más simple, que (por defecto) nos dará la primera hoja del archivo de entrada de Excel, que es la hoja “User_info”.
- ▶ pf.read_excel ('users.xlsx', sheet_name = 'purchase') significa que obtendremos la segunda hoja, que se llama “purchase”.
- ▶ pf.read_excel ('users.xlsx', sheet_name = [0,2]) devolverá la primera y la tercera hoja del archivo de Excel. El valor devuelto es un diccionario de DataFrame.

Encabezado

Si por alguna razón, los datos en su hoja de Excel no comienzan en la fila 1, puede usar el encabezado para decirle a pandas “oye, el encabezado de estos datos está en la fila X”. La cuarta hoja del archivo de ejemplo de Excel comienza en la fila 5. Al leer esa hoja sin instrucciones especiales, pandas interpreta que nuestros datos no tienen nombres de columna.

```
df = pd.read_excel('users.xlsx', sheet_name = 3)
```

Salida

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3
0	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	ID	Customer	purchase	Date
4	101	Forrest Gump	Dragon Ball	2020-08-12 00:00:00
5	102	Mary Jane	Evangelion	2020-01-01 00:00:00
6	103	Harry Porter	Kill la Kill	2020-08-01 00:00:00
7	104	Jean Grey	Dragon Ball	1999-01-01 00:00:00
8	105	Mary Jane	Evangelion	2019-12-31 00:00:00
9	106	Harry Porter	Ghost in the Shell	2020-01-01 00:00:00
10	107	Jean Grey	Evangelion	2018-04-01 00:00:00

La anterior salida no fue la adecuada y el DataFrame requiere una limpieza. Por el contrario, podemos cambiar un poco el código especificando el argumento del encabezado. Recuerde que Python usa un índice basado en 0, por lo que la quinta fila tiene un índice de 4.

```
df = pd.read_excel('users.xlsx', sheet_name = 3, header = 4)
df
```

Salida

	ID	Customer	purchase	Date
0	101	Forrest Gump	Dragon Ball	2020-08-12
1	102	Mary Jane	Evangelion	2020-01-01
2	103	Harry Porter	Kill la Kill	2020-08-01
3	104	Jean Grey	Dragon Ball	1999-01-01
4	105	Mary Jane	Evangelion	2019-12-31
5	106	Harry Porter	Ghost in the Shell	2020-01-01
6	107	Jean Grey	Evangelion	2018-04-01

Esta vez estuvo mucho mejor.

nombres

Si no le gustan los nombres de los encabezados en el archivo de origen de Excel, no dude en crear los suyos propios utilizando el argumento de nombres.

```
df = pd.read_excel('users.xlsx', sheet_name = 3, header = 4, names = ['Customer_ID','Customer_Name','Customer_Purchase','Purchase_Date'])
```

Salida

	Customer_ID	Customer_Name	Customer_Purchase	Purchase_Date
0	101	Forrest Gump	Dragon Ball	2020-08-12
1	102	Mary Jane	Evangelion	2020-01-01
2	103	Harry Porter	Kill la Kill	2020-08-01
3	104	Jean Grey	Dragon Ball	1999-01-01
4	105	Mary Jane	Evangelion	2019-12-31
5	106	Harry Porter	Ghost in the Shell	2020-01-01
6	107	Jean Grey	Evangelion	2018-04-01

usecols

Al especificar usecols, estamos limitando las columnas de Excel para que se carguen en Python, lo cual es una buena práctica si tiene un conjunto de datos grande y no necesita todas las columnas. El siguiente ejemplo leerá solo el nombre del cliente y las columnas de compra en Python.

```
df = pd.read_excel('users.xlsx', sheet_name = 3, header = 4, usecols = "B:C")
```

Salida

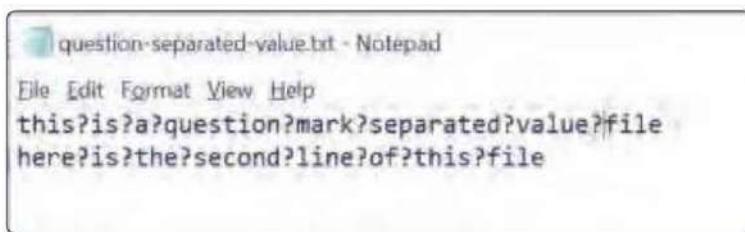
Out[26]:		
	Customer	purchase
0	Forrest Gump	Dragon Ball
1	Mary Jane	Evangelion
2	Harry Porter	Kill la Kill
3	Jean Grey	Dragon Ball
4	Mary Jane	Evangelion
5	Harry Porter	Ghost in the Shell
6	Jean Grey	Evangelion

14.1.2 Método y argumentos pd.read_csv ()

Como sugiere el nombre, este método lee un archivo csv.

CSV significa “valor separado por comas”, por lo que un archivo .csv es básicamente un archivo de texto con valores separados por comas. Significa que también puede usar este método para leer cualquier archivo .txt en Python.

El argumento para read_csv () es similar a read_excel (), por lo que no los repetiremos aquí. Sin embargo, vale la pena señalar un argumento: sep o delimitador. Se usa para decirle a pandas qué delimitador usar para separar los datos. Utilizamos este archivo de texto de muestra (question-separated-value.txt) para ver que básicamente puede usar cualquier carácter como delimitador.



question-separated-value.txt - Notepad
File Edit Format View Help
this?is?a?question?mark?separated?value?file
here?is?the?second?line?of?this?file

```
df = pd.read_csv('question-separated-value.txt',sep='?')
```

Salida

Out[28]:
this is a question mark separated value file
0 here is the second line of this file

14.2 LEER VARIAS HOJAS DE EXCEL CON PANDAS

En la sección anterior, mencionamos cómo leer un archivo de Excel en Python. Aquí intentaremos leer varias hojas de Excel (del mismo archivo) con pandas. Podemos hacer esto de dos maneras: use el método `pd.read_excel()`, con el argumento opcional `sheet_name`; la alternativa es crear un objeto `pd.ExcelFile` y luego analizar los datos de ese objeto.

14.2.1 método `pd.read_excel()`

Utilicemos el siguiente ejemplo:

- ▶ Seleccionar hojas para leer por índice: `sheet_name = [0,1,2]` significa las primeras tres hojas
- ▶ Seleccione hojas para leer por nombre: `sheet_name = ['User_info', 'compuesto']`. Este método requiere que conozca los nombres de las hojas de antemano
- ▶ Seleccione todas las hojas: `sheet_name = None`

```
import pandas as pd
df = pd.read_excel('users.xlsx', sheet_name = [0,1,2])
df = pd.read_excel('users.xlsx', sheet_name = ['User_info','compound'])
df = pd.read_excel('users.xlsx', sheet_name = None) # Lee todas las hojas
```

Leeremos todas las hojas del archivo de ejemplo de Excel y luego usaremos ese DataFrame para los ejemplos más adelante.

El df devuelve un diccionario de DataFrame. Las claves del diccionario contienen los nombres de hojas y los valores del diccionario el contenido de las hojas.

```
df.keys()
```

Salida

```
dict_keys(['User_info', 'purchase', 'compound', 'header_row5'])  
df.values()
```

Salida

```
dict_values([

|   | User         | Name   | Country  | City | Gender | Age |
|---|--------------|--------|----------|------|--------|-----|
| 0 | Forrest Gump | USA    | New York | M    | 50     |     |
| 1 | Mary Jane    | CANADA | Toronto  | F    | 30     |     |
| 2 | Harry Porter | UK     | London   | M    | 20     |     |
| 3 | Jean Grey    | CHINA  | Shanghai | F    | 30,    |     |


| ID   | Customer         | purchase           | Date        |
|------|------------------|--------------------|-------------|
| 0    | 101 Forrest Gump | Dragon Ball        | 2020-08-12  |
| 1    | 102 Mary Jane    | Evangelion         | 2020-01-01  |
| 2    | 103 Harry Porter | Kill la Kill       | 2020-08-01  |
| 3    | 104 Jean Grey    | Dragon Ball        | 1999-01-01  |
| 4    | 105 Mary Jane    | Evangelion         | 2019-12-31  |
| 5    | 106 Harry Porter | Ghost in the Shell | 2020-01-01  |
| 6    | 107 Jean Grey    | Evangelion         | 2018-04-01, |
| .... | ]                |                    |             |


```

Para obtener datos de una hoja específica, simplemente haga referencia a la clave en el diccionario. Por ejemplo, df ['header_row5'] devuelve la hoja en la que los datos comienzan en la fila 5.

```
df['header_row5']
```

Salida

```
Unnamed: 0    Unnamed: 1        Unnamed: 2        Unnamed: 3  
0      NaN        NaN        NaN        NaN  
1      NaN        NaN        NaN        NaN  
2      NaN        NaN        NaN        NaN  
3      ID      Customer        purchase        Date  
4      101 Forrest Gump      Dragon Ball 2020-08-12 00:00:00
```

5	102	Mary Jane	Evangelion	2020-01-01 00:00:00
6	103	Harry Porter	Kill la Kill	2020-08-01 00:00:00
7	104	Jean Grey	Dragon Ball	1999-01-01 00:00:00
8	105	Mary Jane	Evangelion	2019-12-31 00:00:00
9	106	Harry Porter	Ghost in the Shell	2020-01-01 00:00:00
10	107	Jean Grey	Evangelion	2018-04-01 00:00:00

14.2.2 Método pd.ExcelFile ()

Con este enfoque, creamos un objeto pd.ExcelFile para representar el archivo de Excel. No es necesario que especifiquemos qué hojas leer al utilizar este método. Tenga en cuenta que el método read_excel () anterior devuelve un DataFrame o un diccionario de DataFrame; mientras que pd.ExcelFile () devuelve un objeto de referencia al archivo de Excel.

```
f = pd.ExcelFile('users.xlsx')
f
```

Salida

```
<pandas.io.excel._base.ExcelFile object at 0x00000138DAE66670>
```

Para obtener los nombres de las hojas, podemos utilizar todos los atributos sheet_names del objeto ExcelFile, que devuelve una lista de los nombres de las hojas (cadena).

```
f.sheet_names
```

Salida

```
['User_info', 'purchase', 'compound', 'header_row5']
```

Para obtener datos de una hoja, podemos usar el método parse() y proporcionar el nombre de la hoja.

```
f.parse(sheet_name = 'User_info')
```

Salida

	User Name	Country	City	Gender	Age
0	Forrest Gump	USA	New York	M	50
1	Mary Jane	CANADA	Toronto	F	30
2	Harry Porter	UK	London	M	20
3	Jean Grey	CHINA	Shanghai	F	30

Una cosa a tener en cuenta es que el método `pd.ExcelFile.parse()` es equivalente al método `pd.read_excel()`, por lo que eso significa que puede pasar los mismos argumentos usados en `read_excel()`.

14.3 LEER MÚLTIPLES ARCHIVOS DE EXCEL CON PYTHON

Anteriormente discutimos cómo leer datos de un solo archivo de Excel. A continuación, aprenderemos cómo leer varios archivos de Excel en Python usando la librería pandas. Podemos hacerlo de dos formas:

14.3.1 Método 1: Obtener archivos de la carpeta - estilo PowerQuery

Excel PowerQuery tiene una función “Obtener datos de la carpeta” que nos permite cargar todos los archivos de una carpeta específica. Podemos hacer esto fácilmente en Python. El flujo de trabajo es el siguiente:

- ▶ Dada una carpeta, busque todos los archivos dentro de ella
- ▶ Reduzca la selección de archivos, ¿qué archivos necesito cargar?
- ▶ Cargue los datos de los archivos seleccionados, uno por uno

Para lograr el flujo de trabajo anterior, necesitaremos librerías de `os` y `pandas`. La librería `os` proporciona formas de interactuar con el sistema operativo de su computador, como averiguar qué archivos existen en una carpeta. `os.listdir()` devuelve una lista de todos los nombres de archivo (cadena) dentro de una carpeta específica. Una vez que tenemos la lista de nombres de archivos, podemos recorrerlos y cargar datos en Python.

```
import os
import pandas as pd
folder = r'C:\Users\JZ\Desktop\PythonInOffice\Python_excel_series_read_multiple_excel_files'
files = os.listdir(folder)
for file in files:
```

```

if file.endswith('.xlsx'):
    df = pd.read_excel(os.path.join(folder,file))
for file in files:

```

Salida

```

['data.pdf', 'File_1.xlsx', 'File_2.xlsx', 'File_3.xlsx', 'header-background.
PNG', 'multiple files.py']

```

Nuestra carpeta de trabajo contiene varios tipos de archivos (archivos PDF, Excel, Image y Python). Pero el archivo.endswith ('. Xlsx') asegura que solo leamos los archivos de Excel en Python.

os.path.join () proporciona una forma eficaz de crear una ruta de archivo. Esto debe usarse siempre que sea posible, en lugar de carpeta + “\” + archivo.

14.3.2 Método 2: usar un archivo de entrada de Excel

El segundo método requiere que tengamos un archivo de Excel separado que actúa como un “archivo de entrada”. Contiene enlaces a archivos individuales que pretendemos leer en Python. Para replicar el ejemplo que acabamos de analizar, necesitamos crear un archivo de Excel que se parezca al siguiente, esencialmente una columna con enlaces a otros archivos.

A	B	C	D	E	F	G	H	I	J
1 File path									
2 C:\Users\JZ\Desktop\PythonInOffice\python_excel_series_read_multiple_excel_files\File_1.xlsx									
3 C:\Users\JZ\Desktop\PythonInOffice\python_excel_series_read_multiple_excel_files\File_2.xlsx									
4 C:\Users\JZ\Desktop\PythonInOffice\python_excel_series_read_multiple_excel_files\File_3.xlsx									
5									
6									
7									
8									

Este método tiene las siguientes ventajas:

- ▶ Es posible organizar y almacenar información (nombres de archivos, enlaces, etc.) en un entorno (hoja de cálculo) conocido.
- ▶ Si se requiere actualizar o agregar nuevos archivos para leer, solo es necesario actualizar el archivo de entrada. No se requiere ningún cambio de codificación.

El flujo de trabajo es similar al método anterior. Primero, debemos dejar que Python conozca las rutas de los archivos, que se pueden obtener del archivo de entrada.

```
df_files = pd.read_excel('Excel_input.xlsx')
df_files
```

Salida

```
File path
0 C:\Users\JZ\Desktop\PythonInOffice\Python_exce...
1 C:\Users\JZ\Desktop\PythonInOffice\Python_exce...
2 C:\Users\JZ\Desktop\PythonInOffice\Python_exce...
```

Este es básicamente un DataFrame simple con una sola columna, que contiene los enlaces de archivos. Ahora podemos recorrer la lista y leer archivos de Excel.

```
for file in df_files['File path']:
    df = pd.read_excel(file)
```

¿Cuándo usar el método 1 (Obtener archivos de la carpeta) frente al método 2 (archivo de entrada de Excel)?

Se pueden plantear dos preguntas sencillas para determinar qué método utilizar.

La primera pregunta es: ¿La carpeta de origen contiene archivos adicionales que no necesito?

- ▶ Por ejemplo, si una carpeta contiene 20 archivos csv y solo necesito 10 de ellos. Probablemente sea más fácil utilizar el método de archivo de entrada de Excel. Editar un archivo de entrada de Excel es mucho más fácil y rápido que escribir código para manejar diferentes escenarios en Python.
- ▶ Sin embargo, si la carpeta contiene 50 archivos, de los cuales 20 son csv, los necesito todos. Luego, usaré el método Obtener archivo de carpeta, porque podemos seleccionar fácilmente todos los archivos .csv de la lista de archivos.

La segunda pregunta es: ¿Todos los archivos se encuentran dentro de la misma carpeta?

- Si los archivos están en carpetas diferentes, tiene más sentido usar un archivo de entrada de Excel para almacenar las rutas de los archivos

Ya hemos aprendido a leer datos de archivos de Excel. Otra cosa importante que se debe saber es cómo guardar datos en un archivo de Excel usando Python.

14.4 GUARDANDO LOS DATOS EN UN ARCHIVO DE EXCEL USANDO PYTHON

Guardar datos en un archivo de Excel también es fácil con pandas. La forma más sencilla es la siguiente: `df.to_excel()`, que guarda el DataFrame en un archivo de Excel. Similar a `df.read_excel()`, este método `to_excel()` también tiene muchos argumentos opcionales. Veremos solo algunos de los argumentos aquí, si desea conocer la lista completa de argumentos, le sugiero que lea la documentación oficial de pandas. Veamos un ejemplo: primero necesitamos tener un DataFrame listo para guardar. Trabajaremos con el mismo archivo usado para el ejemplo `read_excel()`.

```
import pandas as pd  
df = pd.read_excel('users.xlsx')  
df
```

Salida

```
User Name Country      City Gender  Age  
0  Forrest Gump     USA  New York    M   50  
1    Mary Jane  CANADA  Toronto    F   30  
2  Harry Porter     UK   London    M   20  
3   Jean Grey    CHINA  Shanghai   F   30
```

```
df.to_excel('saved_file.xlsx')
```

Name	Date modified	Type	Size
save_to_file.py	8/16/2020 11:02 PM	Python File	1 KB
saved_file.xlsx	8/16/2020 11:03 PM	XLSX Worksheet	5 KB
users.xlsx	8/12/2020 1:49 AM	XLSX Worksheet	14 KB

Después de ejecutar el código anterior, tendremos un nuevo archivo llamado “Saved_file.xlsx”, que fue creado por Python. Abramos el archivo y veamos si contiene los mismos datos.

A	B	C	D	E	F
1	User Name	Country	City	Gender	Age
2	0 Forrest Gump	USA	New York	M	50
3	1 Mary Jane	CANADA	Tornoto	F	30
4	2 Harry Porter	UK	London	M	20
5	3 Jean Grey	CHINA	Shanghai	F	30

Inmediatamente notamos algo extraño, la columna A contiene algo que parece una lista que comienza desde 0. Si está de acuerdo con dejarlo allí, está bien. Pero es posible eliminarla.

Eliminar el índice inicial al guardar un archivo de Excel usando pandas

El método `.to_excel()` proporciona un índice de argumento opcional, que es para controlar esa lista que acabamos de ver. Podemos eliminar esa lista de nuestro archivo de salida de Excel de la siguiente manera:

```
df.to_excel('saved_file.xlsx', index = False)
```

Otros argumentos opcionales útiles:

- ▶ `sheet_name`: puedes nombrar la hoja si no te gusta “Sheet1” de forma predeterminada.
- ▶ `na_rep`: valor para reemplazar los valores “Null” en el DataFrame, por defecto esta es una cadena vacía “”. Sin embargo, si su marco de datos contiene números, es posible que desee establecer esto en `np_rep = 0` en su lugar.
- ▶ `columnas`: elija las columnas que desea generar. Normalmente no uso esto, ya que coloco las columnas en el DataFrame antes de guardar en el archivo.

14.5 GUARDAR EN ARCHIVO CSV

Podemos guardar el mismo DataFrame en un archivo csv usando df.to_csv(). Los argumentos son similares a to_excel(), así que no los repetiré aquí. Solo quiero señalar una pequeña diferencia, pero esta es realmente una diferencia entre Excel y el archivo CSV.

El archivo CSV es básicamente un archivo de texto, contiene solo 1 hoja, por lo que no podemos cambiar el nombre de la hoja.

14.6 GUARDE VARIAS HOJAS EN UN ARCHIVO DE EXCEL CON PYTHON

Recientemente aprendimos cómo guardar una hoja en un archivo de Excel. Ahora guardaremos varias hojas en un (el mismo) archivo de Excel usando pandas.

Solo un recordatorio: df significa marco de datos y pd es la abreviatura de pandas.

Seguiremos usando el método df.to_excel(), pero necesitaremos ayuda de otra clase pd.ExcelWriter(). Como sugiere su nombre, esta clase escribe en archivos de Excel. Si lee atentamente la documentación de pd.to_excel(), ExcelWriter es en realidad el primer argumento.

Modelos de DataFrame

Creemos dos modelos de DataFrame, para que tengamos algo con que trabajar. El primero está formado por números aleatorios de 20 filas por 10 columnas; y el segundo DataFrame es de 10 filas por 1 columna.

```
import pandas as pd
import numpy as np
df_1 = pd.DataFrame(np.random.rand(20,10))
df_2 = pd.DataFrame(np.random.rand(10,1))
```

Veremos dos métodos para guardar archivos de Excel de varias hojas. La idea es prácticamente la misma entre los 2 métodos: creamos un ExcelWriter, luego lo pasamos a df.to_excel() para guardar el DataFrame en un archivo de Excel. Los dos métodos son ligeramente diferentes en sintaxis pero funcionan de la misma manera.

14.6.1 Método 1

Este es el método demostrado en la documentación oficial de pandas.

```
.....  
with pd.ExcelWriter('mult_sheets_1.xlsx') as writer1:  
    df_1.to_excel(writer1, sheet_name = 'df_1', index = False)  
    df_2.to_excel(writer1, sheet_name = 'df_2', index = False)  
.....
```

14.6.2 Método 2

Este es mi método preferido. Permitidme mostrar cómo se ve y luego decirle por qué prefiero este método al método 1.

```
.....  
writer2 = pd.ExcelWriter('mult_sheets_2.xlsx')  
df_1.to_excel(writer2, sheet_name = 'df_1', index = False)  
df_2.to_excel(writer2, sheet_name = 'df_2', index = False)  
writer2.save()  
.....
```

¡A estas alturas debes pensar que estos dos métodos son iguales! Bueno, sí y no.

Es cierto que ambos métodos hacen exactamente lo mismo: guardar los dos DataFrames en un solo archivo de Excel. Sin embargo, el mecanismo es bastante diferente.

La diferencia

En primer lugar, debido al bloque `with` en el método 1, todos sus DataFrames deben estar en el mismo ámbito. Esto significa que si tiene un DataFrame fuera del ámbito actual, deberá traerlo primero.

Mientras que para el método 2, los DataFrame pueden estar en diferentes ámbitos y seguirá funcionando. Esto es especialmente útil cuando su código es complicado.

14.7 OBTENGA VALORES, FILAS Y COLUMNAS EN DATAFRAME DE PANDAS

Hemos analizado la parte leer y guardar datos en archivos de Excel con Python. Pasemos a algo más interesante. En Excel podemos ver las filas, columnas y celdas. Podemos hacer referencia a los valores usando un signo “=” o dentro de una fórmula. En Python, los datos se almacenan en la memoria del computador (es decir, no son visibles directamente para los usuarios); afortunadamente, la librería de pandas proporciona formas fáciles de obtener valores, filas y columnas.

Primero preparamos un DataFrame. Usaremos el archivo del ejemplo anterior.

```
import pandas as pd  
df = pd.read_excel('users.xlsx')  
df
```

Salida

```
User Name Country      City Gender  Age  
0  Forrest Gump      USA  New York    M   50  
1    Mary Jane CANADA  Toronto    F   30  
2  Harry Porter      UK   London    M   20  
3   Jean Grey   CHINA Shanghai   F   30
```

	A	B	C	D	E
1	User Name	Country	City	Gender	Age
2	Forrest Gump	USA	New York	M	50
3	Mary Jane	CANADA	Tornoto	F	30
4	Harry Porter	UK	London	M	20
5	Jean Grey	CHINA	Shanghai	F	30
6					

Algunas observaciones sobre esta pequeña tabla / DataFrame:

- ▶ Hay cinco columnas con nombres: “User Name”, “Country”, “City”, “Gender”, “Age”.
- ▶ Hay 4 filas (excluyendo la fila de encabezado).

df.index devuelve la lista del índice, en nuestro caso, son solo números enteros 0, 1, 2, 3.

df.columns da la lista de los nombres de las columnas (encabezados).

df.shape muestra la dimensión del DataFrame, en este caso son 4 filas por 5 columnas.

```
df.index
```

Salida

```
RangeIndex(start=0, stop=4, step=1)
df.columns
```

Salida

```
Index(['User Name', 'Country', 'City', 'Gender', 'Age'], dtype='object')
df.shape
```

Salida

```
(4, 5)
```

14.7.1 Obteniendo columnas con pandas

Hay varias formas de obtener columnas en pandas. Cada método tiene sus pros y sus contras, por lo que los usaría de manera diferente según la situación.

La notación de puntos

Podemos escribir df.Country para obtener la columna “Country”. Esta es una forma rápida y sencilla de obtener columnas. Sin embargo, si el nombre de la columna contiene espacio, como “User Name”. Este método no funcionará.

```
df.Country
```

Salida

```
0      USA
1    CANADA
2      UK
3    CHINA
Name: Country, dtype: object
df.Age
```

Salida

```
0    50
1    30
2    20
3    30
Name: Age, dtype: int64
df.User Name
```

Salida

```
SyntaxError: invalid syntax
```

La notación de corchetes

Este es mi favorito. Requiere un nombre de DataFrame y un nombre de columna, que es así: DataFrame [nombre de columna]. El nombre de la columna dentro de los corchetes es una cadena, así que tenemos que usar comillas alrededor. Aunque requiere más escritura que la notación de puntos, este método siempre funcionará en cualquier caso. Debido a que envolvemos la cadena (nombre de la columna) con una cita, aquí también se permiten nombres con espacios.

```
df['User Name']
```

Salida

```
0    Forrest Gump
1    Mary Jane
2    Harry Porter
3    Jean Grey
```

```
Name: User Name, dtype: object
```

```
df['City']
```

Salida

```
0    New York
1    Toronto
2    London
3   Shanghai
Name: City, dtype: object
```

14.7.2 Obteniendo varias columnas

La notación de corchetes facilita la obtención de varias columnas. La sintaxis es similar, pero en su lugar, pasamos una lista de cadenas entre corchetes. Preste atención a los corchetes dobles:

DataFrame [[nombre de columna 1, nombre de columna 2, nombre de columna 3, etc.]].

```
df[['User Name', 'Age', 'Gender']]
```

Salida

```
      User Name  Age Gender
0  Forrest Gump   50      M
1    Mary Jane   30      F
2  Harry Porter   20      M
3   Jean Grey   30      F
```

14.7.3 Obteniendo filas con Pandas

Podemos usar .loc [] para obtener filas. Tenga en cuenta los corchetes aquí en lugar del paréntesis (). La sintaxis es la siguiente: df.loc [fila, columna]. La columna es opcional, y si se deja en blanco, podemos obtener la fila completa. Debido a que Python usa un índice de base cero, df.loc [0] devuelve la primera fila del marco de datos.

```
df.loc[0]
```

Salida

```
User Name    Forrest Gump
Country          USA
City        New York
Gender            M
Age             50
Name: 0, dtype: object
df.loc[2]
```

Salida

```
User Name    Harry Porter
Country          UK
City        London
Gender            M
Age             20
Name: 2, dtype: object
```

14.7.4 Obteniendo varias filas

Tendremos que usar indexación / segmentación para obtener varias filas. En pandas, esto se hace de manera similar a cómo indexar / dividir una lista de Python.

Para obtener las tres primeras filas, podemos hacer lo siguiente:

```
df.loc[0:2]
```

Salida

	User Name	Country	City	Gender	Age
0	Forrest Gump	USA	New York	M	50
1	Mary Jane	CANADA	Toronto	F	30
2	Harry Porter	UK	London	M	20

14.7.5 Obteniendo valores de celda

Para obtener valores de celda individuales, necesitamos usar la intersección de filas y columnas. Piense en cómo hacemos referencia a las celdas dentro de Excel, como una celda “C10” o un rango “C10: E20”. Los siguientes dos enfoques siguen esta idea de fila y columna.

Notación de corchetes

Usando la notación de corchetes, la sintaxis es así: DataFrame [nombre de columna] [índice de fila]. Esto a veces se denomina indexación encadenada. Una forma más fácil de recordar esta notación es: el DataFrame [nombre de la columna] da una columna, luego agregar otro [índice de fila] dará el elemento específico de esa columna.

Supongamos que queremos obtener la Ciudad para Mary Jane (en la fila 2).

```
.....  
df['City'][1]  
.....
```

Salida

```
.....  
'Toronto'  
.....
```

Para obtener la 2^a y 4^a fila, y solo las columnas User Name, Gender y Age, podemos pasar las filas y columnas como dos listas como la siguiente.

```
.....  
df[['User Name', 'Age', 'Gender']].loc[[1,3]]  
.....
```

Salida

```
.....  
User Name  Age  Gender  
1  Mary Jane    30      F  
3  Jean Grey    30      F  
.....
```

Recuerde, df [['User Name', 'Age', 'Gender']] devuelve un nuevo DataFrame con solo tres columnas. Entonces .loc [[1,3]] devuelve la 1^a y 4^a filas de ese DataFrame.

.loc [] método

Como se mencionó anteriormente, la sintaxis de .loc es df.loc [fila, columna]. ¿Necesita un recordatorio sobre cuáles son los posibles valores para filas (índice) y columnas?

```
df.index
```

Salida

```
RangeIndex(start=0, stop=4, step=1)  
df.columns
```

Salida

```
Index(['User Name', 'Country', 'City', 'Gender', 'Age'], dtype='object')
```

Intentemos obtener el nombre del país de Harry Potter, que está en la fila 3.

```
df.loc[2, 'Country']
```

Salida

```
'UK'
```

Para obtener la 2^a y 4^a fila, y solo las columnas User Name, Gender y Age, podemos pasar las filas y columnas como dos listas a los argumentos posicionales de “fila” y “columna”.

Salida

```
df.loc[[1,3],['User Name', 'Age', 'Gender']]  
User Name  Age  Gender  
1  Mary Jane    30      F  
3  Jean Grey    30      F
```

14.8 OBTENIENDO DATOS DE LA TABLA DE UNA PÁGINA WEB USANDO PYTHON Y LA LIBRERÍA PANDAS

Hemos venido hablando sobre la integración de Python con Excel, ¿por qué obtener datos de la web?" No se sorprenda cuando vea el título de esta sección. Hoy en día, las personas están conectadas a Internet en cualquier momento y lugar. Internet es probablemente la base de datos pública más grande que existe, por lo que aprender a obtener datos de Internet es fundamental. Es por eso que quiero hablar sobre cómo obtener datos de tablas desde una página web usando Python y la librería de pandas. Además, si ya está utilizando Excel PowerQuery, esto es equivalente a "Obtener datos de la Web", pero 100 veces más potente.

14.8.1 Obtener datos de un sitio web (web scraping)

HTML es el lenguaje detrás de cada sitio web. Cuando visitamos un sitio web, lo que sucede atrás es lo siguiente: 1. Escribimos una dirección (URL) en la barra de direcciones del navegador, el navegador envía una solicitud al servidor del sitio web de destino. 2. El servidor recibe la solicitud y devuelve el código HTML que compone la página web. 3. Nuestro navegador recibe el código HTML, lo ejecuta sobre la marcha y crea una página web para que la veamos.

Así que ahora, los humanos vemos las hermosas páginas web, pero las máquinas solo ven código. Y eso es bueno porque el código es más fácil de digerir mediante programación. El web scraping básicamente significa que, en lugar de usar un navegador, podemos usar Python para enviar una solicitud al servidor de un sitio web, recibir el código HTML y luego extraer los datos que queremos.

No cubriremos demasiado HTML ya que este no es un tutorial de diseño web, pero quiero presentar los aspectos básicos para que tengamos una comprensión básica de cómo funcionan los sitios web y el web scraping. Los elementos HTML o "etiquetas HTML" son determinadas palabras clave envueltas en <>. Tenga en cuenta que la mayoría de los elementos HTML requieren una etiqueta de apertura (e, g, <title>) y una etiqueta de cierre correspondiente (por ejemplo, </title>).

De manera similar, el siguiente código dibujará una tabla en su navegador, puede intentar copiarlo y pegarlo en un bloc de notas, luego guardarlo como un archivo "table-example.html". Debería poder abrirlo dentro de un navegador. Algunas notas rápidas:

- ▀ <table> ... </table> dibuja la tabla
- ▀ <tr> ... </tr> dibuja una fila dentro de la tabla
- ▀ <th> ... </th> indica el encabezado de la tabla
- ▀ <td> ... </td> indica datos de la table

```
<html>
  <table>
    <tr>
      <th>User Name</th>
      <th>Country</th>
      <th>City</th>
      <th>Gender</th>
      <th>Age</th>
    </tr>
    <tr>
      <td>Forrest Gump</td>
      <td>USA</td>
      <td>New York</td>
      <td>M</td>
      <td>50</td>
    </tr>
    <tr>
      <td>Mary Jane</td>
      <td>CANADA</td>
      <td>Toronto</td>
      <td>F</td>
      <td>30</td>
    </tr>
  </table>
</html>
```

14.8.2 Requisitos para usar pandas para web scraping

Ahora que comprende los componentes básicos de un sitio web y cómo interpretar HTML (bueno, al menos la parte de la tabla). La razón por la que solo cubrí la tabla HTML es porque, la mayoría de las veces, cuando intentamos obtener datos de un sitio web, está en formato de tabla. Y pandas es la herramienta perfecta para obtener datos de formato de tabla de un sitio web.

Entonces, el único requisito para usar pandas para obtener datos de un sitio web es que los datos deben almacenarse dentro de una tabla o, en términos HTML, dentro de las etiquetas `<table> ... </table>`. Pandas podrá extraer la tabla, los encabezados y las filas de datos utilizando esas etiquetas HTML que mencionamos hace un momento.

Si intenta utilizar pandas para “extraer datos” de una página web que no contiene ninguna tabla (etiquetas `<table> ... </table>`), no podrá obtener ningún dato. Para aquellos datos que no están almacenados en una tabla, necesitamos otras formas de rastrear el sitio web.

14.8.3 Web Scraping en acción

Nuestros ejemplos anteriores eran en su mayoría tablas pequeñas con algunos datos, usemos algo un poco más grande para ensayar.

Vamos a obtener los nombres de las empresas del S&P 500 y los símbolos de acciones de wikipedia: https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

```
import pandas as pd

df=pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')

type(df)
```

Salida

```
<class 'list'>
len(df)
```

Salida

```
2
```

Nuestro df anterior en realidad es una lista, eso es interesante ... y parece que hay 2 elementos en esa lista. Veamos qué datos obtuvo pandas:

```
df[0]
```

Salida

	Symbol	Security	...	CIK	Founded
0	MMM	3M Company	...	66740	1902
1	ABT	Abbott Laboratories	...	1800	1888
2	ABBV	AbbVie Inc.	...	1551152	2013 (1888)
3	ABMD	ABIOMED Inc	...	815094	1981
4	ACN	Accenture plc	...	1467373	1989
...

```

500    YUM        Yum! Brands Inc ... 1041061 1997
501   ZBRA       Zebra Technologies ... 877212 1969
502    ZBH      Zimmer Biomet Holdings ... 1136869 1927
503   ZION       Zions Bancorp ... 109380 1873
504    ZTS        Zoetis ... 1555280 1952
[505 rows x 9 columns]

```

```
df[1]
```

Salida

	Date	Reason
	Date	Reason
0	June 22, 2020	Market capitalization change.[6]
1	June 22, 2020	Market capitalization change.[6]
2	June 22, 2020	Market capitalization change.[6]
3	May 22, 2020	Market capitalization change.[7]
4	May 12, 2020	Market capitalization change.[8]
..
244	December 5, 2000	Market Cap changes.
245	December 5, 2000	Market Cap changes.
246	December 5, 2000	Market Cap changes.
247	July 27, 2000	Market Cap change.[202]
248	December 7, 1999	Market Cap change.[203]

```
[249 rows x 6 columns]
```

Parece que el primer DataFrame df [0] contiene la lista S&P 500, y el segundo DataFrame df [1] es otra tabla en esa página. También observe que al final del primer DataFrame, dice [505 filas x 9 columnas]. Así que siempre pensé que solo hay 500 empresas en el S&P 500, ¡pero no realmente!

Siempre verifique lo que devuelve pd.read_html (), una página web puede contener varias tablas, por lo que obtendrá una lista de DataFrames en lugar de un solo DataFrame.

14.9 REPLICAR LAS FUNCIONES BUSCAR DE EXCEL EN PYTHON

Las funciones de Excel LOOKUP (BUSCAR) son probablemente unas de las funciones más utilizadas. Así que replicaremos la función de xlookup en Python.

De hecho, podemos usar la misma técnica para replicar cualquiera de VLOOKUP (CONSULTAV), HLOOKUP (BUSCARH), XLOOKUP (BUSCARX) o INDEX (INDICE)/ MATCH (COINCIDIR) en Python.

El ejemplo

Tenemos dos tablas de Excel, una contiene información básica del cliente y la otra contiene información sobre pedidos de clientes. Nuestra tarea es llevar algunos datos de una tabla a otra. ¿Le suena familiar esta situación?

Solución Excel

Para abordar este problema, podemos usar: búsqueda o función INDICE / COINCIDIR. VLOOKUP (CONSULTAV) es probablemente el más utilizado, pero está restringido por el formato de la tabla: la clave de búsqueda debe estar en la columna más a la izquierda del conjunto de datos que estamos buscando. En otras palabras, si los valores que estamos intentando traer están en el lado izquierdo de la clave de búsqueda, VLOOKUP no funcionará.

La solución de Microsoft es la fórmula XLOOKUP (BUSCARX), pero solo está disponible en Office 2016 u Office 365. Por lo tanto, usaremos la fórmula xlookup para resolver este problema. En la captura de pantalla siguiente, la columna F “compra” es lo que queríamos traer de la segunda tabla (hacia abajo), y la columna G muestra la fórmula utilizada para la columna F. Aunque la tabla 2 contiene varias entradas para los mismos clientes, con fines de demostración, solo usaremos el valor de la primera entrada. P.ej. para Harry, queremos traer la compra “Kill la Kill”.

A	B	C	D	E	F	G
1 User Name	Country	City	Gender	Age	purchase	
2 Forrest Gump	USA	New York	M	50	Dragon Ball	=XLOOKUP(A2,\$D\$9:\$D\$15,\$B\$9:\$B\$15)
3 Mary Jane	CANADA	Toronto	F	30	Evangelion	=XLOOKUP(A3,\$D\$9:\$D\$15,\$B\$9:\$B\$15)
4 Harry Porter	UK	London	M	20	Kill la Kill	=XLOOKUP(A4,\$D\$9:\$D\$15,\$B\$9:\$B\$15)
5 Jean Grey	CHINA	Shanghai	F	30	Dragon Ball	=XLOOKUP(A5,\$D\$9:\$D\$15,\$B\$9:\$B\$15)
6						
7						
8 ID	purchase	Date	Customer			
9 101	Dragon Ball	8/12/2020	Forrest Gump			
10 102	Evangelion	1/1/2020	Mary Jane			
11 103	Kill la Kill	8/1/2020	Harry Porter			
12 104	Dragon Ball	1/1/1999	Jean Grey			
13 105	Evangelion	12/31/2019	Mary Jane			
14 106	Ghost in the	1/1/2020	Harry Porter			
15 107	Evangelion	4/1/2018	Jean Grey			

Replicar Buscarx en Python

Usaremos la librería pandas, que es casi equivalente a una aplicación de hoja de cálculo para Python, para replicar la fórmula de Excel. Pandas ofrece una amplia selección de herramientas para que podamos replicar la función Buscarx de varias formas. Recorremos un camino aquí, que es una combinación de filtros y apply () .

Primero, carguemos las tablas en nuestra “aplicación de hoja de cálculo Python”.

```
import pandas as pd
df1 = pd.read_excel('users.xlsx', sheet_name = 'User_info')
df2 = pd.read_excel('users.xlsx', sheet_name = 'purchase')
df1
```

Salida

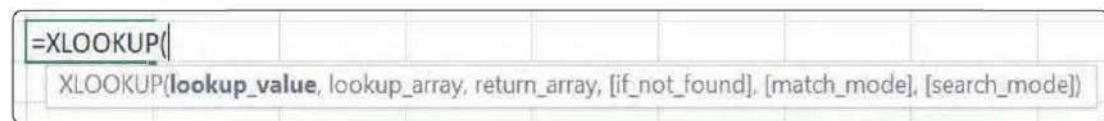
	User	Name	Country	City	Gender	Age
0	Forrest	Gump	USA	New York	M	50
1	Mary	Jane	CANADA	Toronto	F	30
2	Harry	Porter	UK	London	M	20
3	Jean	Grey	CHINA	Shanghai	F	30

df2

Salida

	ID	purchase	Date	Customer
0	101	Dragon Ball	2020-08-12	Forrest Gump
1	102	Evangelion	2020-01-01	Mary Jane
2	103	Kill la Kill	2020-08-01	Harry Porter
3	104	Dragon Ball	1999-01-01	Jean Grey
4	105	Evangelion	2019-12-31	Mary Jane
5	106	Ghost in the Shell	2020-01-01	Harry Porter
6	107	Evangelion	2018-04-01	Jean Grey

La idea detrás de la función BUSCARX es similar a INDICE / COINCIDIR pero más sencilla. Dado un lookup_value, encontramos su posición en el lookup_array, luego devolvemos el valor en la misma posición del return_array. A continuación se muestran los argumentos disponibles de la fórmula de Excel Buscarx. Escribiremos la función Python usando los mismos nombres para los argumentos para que sea más fácil compararla con la fórmula BUSCARX (XLOOKUP) de Excel.



Replicación en Python

Podemos usar un filtro de pandas para lograr esto. Además de los tres argumentos obligatorios, también implementaremos los dos argumentos opcionales `if_not_found` y `search_mode` (que se actualizarán más adelante). Aquí está el código de Python:

```
def xlookup(lookup_value, lookup_array, return_array, if_not_found:str = ''):  
    match_value = return_array.loc[lookup_array == lookup_value]  
    if match_value.empty:  
        return f'{lookup_value} not found!' if if_not_found == '' else if_not_  
        found  
  
    else:  
        return match_value.tolist()[0]
```

De acuerdo, están sucediendo muchas cosas en las pocas líneas de código anteriores. Por eso me encanta Python: es simple pero puede expresar una lógica compleja. Analicemos el código anterior.

En la primera línea, estamos definiendo una función llamada `xlookup` con algunos argumentos.

- ▶ `lookup_value`: el valor que nos interesa será un valor de cadena
- ▶ `lookup_array`: esta es una columna dentro del marco de datos de pandas de origen, estamos buscando el “`lookup_value`” dentro de esta matriz / columna
- ▶ `return_array`: esta es una columna dentro del DataFrame de pandas de origen, queremos devolver valores de esta columna
- ▶ `if_not_found`: se devolverá si no se encuentra “`lookup_value`”

En las siguientes líneas:

- `lookup_array == lookup_value` devuelve un índice booleano, que pandas utiliza para filtrar los resultados
- `return_array.loc []` devuelve una serie pandas con los valores basados en el índice booleano anterior, solo se devuelven valores verdaderos
- Una cosa buena de la Serie pandas es su atributo `.empty`, que nos dice si la Serie contiene valor o está vacía, si el valor de coincidencia resulta estar vacío, entonces sabemos que no se encontró ninguna coincidencia. Entonces podemos informar al usuario que no se encuentra `lookup_value` en los datos
- Por el contrario, si `match_value` no está vacío, sabemos que se ha encontrado algún valor. Podemos convertir `match_value` (que es una serie pandas) en una lista por `.tolist ()`
- Finalmente, debido a que queremos mantener solo el primer valor (si hay varias entradas), elegimos el primer elemento especificando `[0]` de la lista devuelta

Probemos la función, parece que funciona bien. Tenga en cuenta que `df1` es la tabla a la que queremos traer valores, y `df2` es la tabla de origen desde la que estamos buscando valores, y estamos pasando dos columnas del DataFrame a la función para `lookup_array` y `return_array`.

```
xlookup('Mary Jane', df2['Customer'], df2['purchase'])
```

Salida

```
'Evangelion'  
xlookup('Forrest Gump', df2['Customer'], df2['purchase'])
```

Salida

```
'Dragon Ball'  
xlookup('Forrest Gump', df2['Customer'], df2['purchase'])
```

Salida

```
“Forrest Gump” not found!
```

Fórmula completa, ahora “arrastrar hacia abajo”.

Bueno, ya que estamos haciendo todo en código, no podemos simplemente hacer doble clic en algo para “arrastrar hacia abajo” la fórmula. Pero esencialmente “arrastrar hacia abajo” es la parte del bucle; solo necesitamos aplicar la función xlookup a cada fila de la tabla df1. Y recuerde, nunca deberíamos recorrer un DataFrame usando el bucle for.

Método Apply () en lugar de for loop

Resulta que pandas proporciona un método para hacer exactamente esto, y su nombre es .apply (). Veamos su sintaxis. A continuación se muestra una lista simplificada de argumentos, si prefiere ver la lista completa de argumentos, consulte la documentación oficial de pandas en Apply.

DataFrame.apply (func, axis = 0, args = ())

- ▶ func: la función que estamos aplicando
- ▶ axis: podemos aplicar la función tanto en filas como en columnas. De forma predeterminada, es = 0, que son filas. eje = 1 significa columnas
- ▶ args = (): esta es una tupla que contiene los argumentos posicionales que queremos pasar a la función

Así es como podemos aplicar la función xlookup en toda la columna de un marco de datos.

```
df1[‘purchase’] = df1[‘User Name’].apply(xlookup, args = (df2[‘Customer’],  
df2[‘purchase’]))
```

Una cosa a la que hay que prestar atención es cómo apply () pasa argumentos a la función original, que es xlookup en nuestro caso. Por diseño, apply pasará automáticamente todos los datos del DataFrame de la persona que llama (serie). En nuestro ejemplo, apply () pasará df1 [‘User Name’] como el primer argumento en la función xlookup. Sin embargo, nuestro xlookup toma tres argumentos en total. Ahí

es donde el argumento args = () se vuelve útil. Tenga en cuenta que debemos pasar estos argumentos en el orden correcto.

```
df1['purchase'] = df1['User Name'].apply(xlookup, args = (df2['Customer'], df2['purchase']))
```

```
xlookup('Mary Jane', df2['Customer'], df2['purchase'])
```

```
df1
```

Salida

	User Name	Country	City	Gender	Age	purchase
0	Forrest Gump	USA	New York	M	50	Dragon Ball
1	Mary Jane	CANADA	Toronto	F	30	Evangelion
2	Harry Porter	UK	London	M	20	Kill la Kill

15

COMO CONSTRUIR UNA HERRAMIENTA DE ANÁLISIS DE SENTIMIENTOS PARA EL TRADING DE ACCIONES

En este capítulo, crearemos una herramienta de análisis de sentimientos para los titulares de las operaciones bursátiles. Este proyecto le permitirá perfeccionar sus habilidades en Web Scraping, análisis y manipulación de datos, y visualización para construir una herramienta completa de análisis de sentimientos.

La hoja de ruta que seguiremos es la siguiente:

- ▀ Usaremos BeautifulSoup en Python para extraer los titulares de los artículos de FinViz
- ▀ Luego, usaremos Pandas (Librería de análisis de datos de Python) para analizar y ejecutar análisis de sentimientos en los titulares de los artículos.
- ▀ Finalmente, usaremos Matplotlib para visualizar nuestros resultados

15.1 ¿QUÉ ES EL ANÁLISIS DE SENTIMIENTOS?

El análisis de sentimientos es una subcategoría del procesamiento del lenguaje natural (PNL), cuyo objetivo es detectar la polaridad (es decir, opiniones positivas y negativas) dentro de un texto proporcionado. En esencia, el Análisis de sentimiento mide la actitud, el sentimiento y las emociones presentadas dentro de una muestra de texto, devolviendo valores continuos correspondientes a puntuaciones positivas, negativas o neutrales.

15.2 RECOPILACIÓN Y ANÁLISIS DE DATOS DE FINVIZ

FinViz es un sitio web gratuito que hace que los datos bursátiles sean fácilmente accesibles para traders e inversionistas. Recopilaremos los datos bursátiles de FinViz para una cotización bursátil específica. Por ejemplo, así es como se ve la página web del ticker de AMZN en FinViz:



Si se desplaza hacia abajo, verá los artículos de la acción que estamos tratando de analizar. Ver fuente muestra el código HTML exacto que contiene el nombre del artículo la acción y la fecha en que se publicó:

Jul-21-20 04:14PM Amazon Prime Day Officially Postponed Due To Covid-19	<td width="110" align="right" style="white-space: nowrap">Jul-21-20 04:14PM &nbsp&nbsp&nbsp</td><td align="left"><div class="news-link-container"><div class="news-link-left">Amazon Prime Day Officially Postponed Due To Coronavirus Impact</div><div class="news-link-right"> Investor's Business Daily</div></div></td>
03:38PM S&P 500 Turns Positive for 2020, but Most Stocks	
02:59PM Microsoft Makes First Climate Fund Investment	
02:38PM Only a Few Stocks Are Fueling the Markets' Rise	
02:20PM France's Macron: we want an EU-wide digital tax	
01:44PM Amazon Prime Day Officially Postponed Due To Covid-19	
01:25PM US STOCKS-S&P 500, Dow Rise After Positiveearnings	
01:14PM We need to back more female founders, 'not just' Amazon	
01:05PM Amazon Sets New Lobbying Record as Tech Anti-trust	
12:29PM Prime day is not 'a requirement for Amazon base'	
12:16PM Amazon Small Business Sales Surge 60% Amid Pandemic	

Sigamos adelante y escribamos el siguiente código para almacenar estos artículos en un dataset o tabla de datos.

#1. Importar librerías

```
from urllib.request import urlopen, Request
from bs4 import BeautifulSoup
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import pandas as pd
import matplotlib.pyplot as plt
```

#2. Crear la URL para cada ticker

```
finviz_url = 'https://finviz.com/quote.ashx?t='
tickers = ['AMZN', 'GOOG', 'FB']
```

#3. Almacenar los artículos de noticias

```
news_tables = {}
for ticker in tickers:
    url = finviz_url + ticker

    req = Request(url=url, headers={'user-agent': 'my-app'})
    response = urlopen(req)

    html = BeautifulSoup(response, features='html.parser')
    news_table = html.find(id='news-table')
    news_tables[ticker] = news_table

parsed_data = []
```

Aquí, hemos creado una serie de tickers, y para cada uno creamos la URL completa de FinViz para analizar los datos. Usando el módulo Request en Python, obtenemos la respuesta html del sitio web y la lanzamos a BeautifulSoup para que podamos analizarla fácilmente. El elemento HTML con id ‘news-table’ contiene todos nuestros artículos de noticias, por lo que estamos guardando ese elemento BeautifulSoup en un diccionario. Analicemos ese diccionario ahora mismo:

#4. Agregamos las noticias a la matriz y luego la transformamos en un DataFrame

```
for ticker, news_table in news_tables.items():

    for row in news_table.findAll('tr'):
```

```
title = row.a.text
date_data = row.td.text.split(' ')
if len(date_data) == 1:
    time = date_data[0]
else:
    date = date_data[0]
    time = date_data[1]

parsed_data.append([ticker, date, time, title])

df = pd.DataFrame(parsed_data, columns=['ticker', 'date',
                                         'time', 'title'])
```

Nuestro código de análisis simplemente manipula la tabla de noticias que guardamos mientras recopilamos los resultados y analiza los valores específicos que necesitamos. Buscamos todas las filas de la tabla en la tabla de artículos de noticias y recopilamos el título, la fecha y la hora de cada artículo publicado. Una vez que tengamos esos valores, podemos guardar cada dato como un objeto de matriz en nuestra matriz parsed_data.

En las últimas 2 líneas, convertimos nuestra matriz parsed_data en un DataFrame de pandas y configuramos la columna date para que tenga el formato Python Datetime. Esto nos permitirá aplicar fácilmente el análisis de sentimientos y visualizar los datos con Matplotlib.

15.3 APlicar el análisis de sentimientos

Aplicar el análisis de sentimiento en los títulos es en realidad la parte más fácil de todo el proyecto. Con NLTK (Natural Language Toolkit) viene un hermoso submódulo llamado Vader que nos permite pasar una cadena a su función y obtener un resultado de aspecto miedoso como este:

```
Very bad movie.
compound: -0.5849, neg: 0.655, neu: 0.345, pos: 0.0,
```

Podemos ver que la cadena “Very bad movie”. devuelve la respuesta de 4 variables, compuesta, negativa, neutra y positiva. El resultado compuesto es un rango entre -1 a 1, siendo -1 abrumadoramente negativo y +1 siendo respectivamente

positivo. Este será el resultado del cual deduciremos si un artículo sobre una acción es positivo o negativo.

Sigamos adelante y apliquemos el análisis de sentimientos en nuestro DataFrame:

```
#5. Aplicamos el análisis de sentimientos

vader = SentimentIntensityAnalyzer()

f = lambda title: vader.polarity_scores(title)[‘compound’]
df[‘compound’] = df[‘title’].apply(f)
df[‘date’] = pd.to_datetime(df.date).dt.date
```

Inicializamos SentimentIntensityAnalyzer, y luego creamos una función lambda que toma una cadena de título, aplica la función vader.polarity_scores () en ella para obtener los resultados según la imagen de arriba y luego solo devuelve la puntuación compuesta. Usando la función de *apply* en Pandas, podemos crear una nueva columna ‘compuesta’ en el DataFrame con todas las puntuaciones compuestas de cada título.

15.4 VISUALIZACIÓN DE LOS RESULTADOS EN MATPLOTLIB

Por último, pero no menos importante, debemos visualizar este DataFrame en MatPlotLib para ver cómo le fue a nuestras acciones todos los días según la percepción pública en los artículos de noticias.

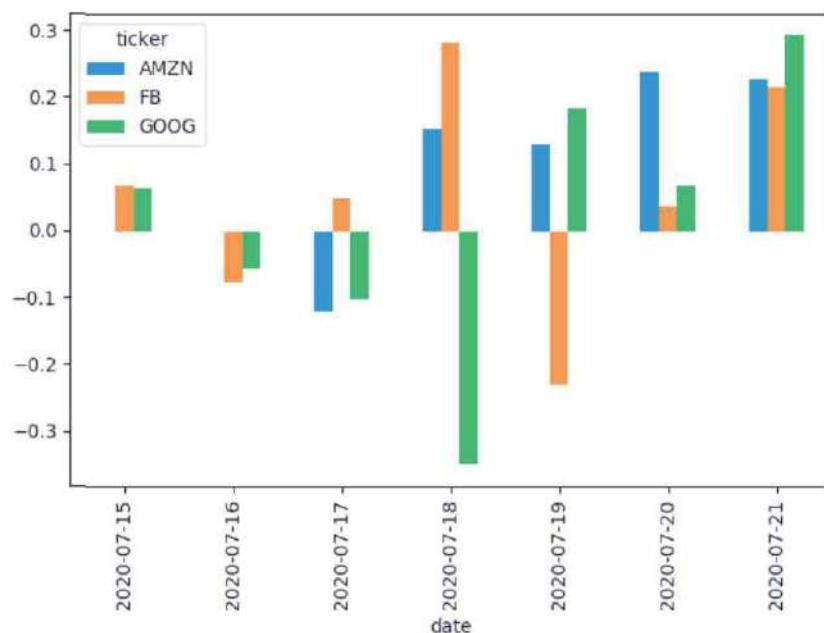
Visualicemos los resultados en un gráfico de barras, agrupando los datos en función de los tickers y las fechas:

```
#6. Visualización de los resultados

plt.figure(figsize=(10,8))
mean_df = df.groupby([‘ticker’, ‘date’]).mean().unstack()
mean_df = mean_df.xs(‘compound’, axis=“columns”)
mean_df.plot(kind=‘bar’)
plt.show()
```

El código de visualización anterior agrupa nuestro conjunto de datos en función del ticker y las fechas de cada fila, y luego visualiza la puntuación compuesta

promedio de cada día. Tomamos la sección transversal de las filas ‘compuestas’, volteamos el marco de datos para que tengamos las fechas como el eje x, y luego lo trazamos como un gráfico de barras.



Acabamos de crear una herramienta de análisis de sentimientos para la negociación de acciones.

16

WEB SCRAPING PARA ESTADOS FINANCIEROS CON PYTHON

No hay muchos casos de uso de ciencia de datos para la contabilidad. Sin embargo, los trabajos de contabilidad son una de las categorías tediosas que deben automatizarse. Una de las tareas que solía hacer era obtener estados financieros de empresas individuales para poder analizar su desempeño frente a la industria. Básicamente, buscaría sus nombres en Yahoo Finance y reuniría los indicadores clave en Excel.

Manipular datos web puede ser complicado a veces, especialmente cuando el sitio web se actualiza, pero dominar los siguientes pocos pasos le ahorrará una enorme cantidad de tiempo en el futuro.

16.1 ¿QUÉ ES EL WEB SCRAPING?

Web scraping o raspado web, es una técnica utilizada mediante programas de software para extraer información de sitios web. El web scraping se enfoca más en la transformación de datos sin estructura en la web (como el formato HTML) en datos estructurados que pueden ser almacenados y analizados en una base de datos central, en una hoja de cálculo o en alguna otra fuente de almacenamiento.

A continuación, se muestra un ejemplo de cómo obtener estados financieros de Yahoo Finance usando Python.

16.2 IMPORTANDO LAS LIBRERÍAS URLLIB Y BEAUTIFUL SOUP

Urllib.request es una librería de código abierto que analiza el contenido de una página web. Cuando la usas, básicamente le pides al sitio web que te entregue los datos de ese sitio web.

Existe otra librería denominada Beautiful Soup que facilita mucho la lectura de datos almacenados en formato XML. XML es un formato similar al HTML que tiene valores de almacenamiento entre etiquetas. Se ve un poco desordenado si lo abres. Al igual que cuando obtienes el código fuente de una página web.

1. Importar librerías

```
import pandas as pd
from bs4 import BeautifulSoup
import urllib.request as ur
```

16.3 PROCESANDO LOS DATOS

Aquí hay un truco simple que permite ajustar de manera flexible el símbolo (ticker symbol) de las acciones y conectarlo al enlace URL. Será útil más adelante si desea extraer cientos de estados financieros de la empresa.

2. Definir la acción y el link a la URLs

```
index= 'MSFT'

url_is = 'https://finance.yahoo.com/quote/' + index + '/financials?p=' + index
url_bs = 'https://finance.yahoo.com/quote/' + index + '/balance-sheet?p=' + index
url_cf = 'https://finance.yahoo.com/quote/' + index + '/cash-flow?p=' + index
```

Ahora tenemos el enlace URL guardado. Si lo abre manualmente en un navegador web, se verá así.

Show: [Income Statement](#) [Balance Sheet](#) [Cash Flow](#) Annual

Income Statement All numbers in thousands

Get access to 15+ years of historical data with Yahoo Finance Premium. [Learn more](#)

Breakdown	TTM	6/30/2019	6/30/2018	6/30/2017	6/30/2016
Total Revenue	129,814,000	125,843,000	110,360,000	89,950,000	85,320,000
Cost of Revenue	43,411,000	42,910,000	38,353,000	34,261,000	32,780,000
Gross Profit	86,403,000	82,933,000	72,007,000	55,689,000	52,540,000
▼ Operating Expenses					
Research Development	17,464,000	16,876,000	14,726,000	13,037,000	11,988,000
Selling General and Administrative	23,249,000	23,098,000	22,223,000	20,020,000	19,260,000
Total Operating Expenses	40,713,000	39,974,000	36,949,000	33,057,000	31,248,000
Operating Income or Loss	45,690,000	42,959,000	35,058,000	22,632,000	21,292,000
Interest Expense	2,649,000	2,686,000	2,733,000	2,222,000	1,243,000
Total Other Income/Expenses Net	3,112,000	3,415,000	4,149,000	2,739,000	-298,000
Income Before Tax	46,153,000	43,688,000	36,474,000	23,149,000	19,751,000
Income Tax Expense	5,059,000	4,448,000	19,903,000	1,945,000	2,953,000
Income from Continuing Operations	41,094,000	39,240,000	16,571,000	21,204,000	16,798,000
Net Income	41,094,000	39,240,000	16,571,000	21,204,000	16,798,000
Net Income available to common s...	41,094,000	39,240,000	16,571,000	21,204,000	16,798,000

16.4 LEYENDO LA URL

A continuación, solo necesitamos abrir el enlace y leerlo en un formato adecuado llamado lxml.

3. Leyendo la URL

```
read_data = ur.urlopen(url_is).read()
soup_is= BeautifulSoup(read_data,'lxml')
```

Si abre soup_is, se verá como un desastre porque los elementos estaban originalmente en formato HTML. Todos los elementos se organizan sistemáticamente en clases.

```
In [15]: soup_is
Out[15]: <!DOCTYPE html>
<html class="NoJs featurephone" id="atomic" lang="en-US"><head prefix="og: http://ogp.me/ns#><script>window.performance
    & window.performance.mark & window.performance.mark('PageStart');</script><meta charset="utf-8"/><title>Microso
    ft Corporation (MSFT) Income Statement</title><meta content="income statement,gross profit,revenue,operating expense
    s,operating income,net income,earning,earning per share" name="keywords"/><meta content="on" http-equiv="x-dns-prefet
    ch-control"/><meta content="on" property="twitter:dnt"/><meta content="90376669494" property="fb:app_id"/><meta conte
    nt="#400090" name="theme-color"/><meta content="width=device-width, initial-scale=1" name="viewport"/><meta content
    ="Get the detailed quarterly/annual income statement for Microsoft Corporation (MSFT). Find out the revenue, expenses
    and profit or loss over the last fiscal year." lang="en-US" name="description"/><meta content="guce.yahoo.com" name
    ="oath:guce:consent-host"/><meta content="A9862C0E621BE95BCE0BF3D0298FD58B" name="mavalance.01"/><link href="/manife
    st.json" rel="manifest"/><link href="//l.yimg.com" rel="dns-prefetch"/><link href="//s.yimg.com" rel="dns-prefetch"/>
    <link href="//csc.beap.bc.yahoo.com" rel="dns-prefetch"/><link href="//geo.query.yahoo.com" rel="dns-prefetch"/><link
    href="//y.analytics.yahoo.com" rel="dns-prefetch"/><link href="//b.scorecardresearch.com" rel="dns-prefetch"/><link h
    ref="//jquery.finance.yahoo.com" rel="dns-prefetch"/><link href="//fc.yahoo.com" rel="dns-prefetch"/><link href="//vi
    deo-api.yql.yahoo.com" rel="dns-prefetch"/><link href="//ytas.btrll.com" rel="dns-prefetch"/><link href="//shim.btrll
    .com" rel="dns-prefetch"/><link href="//consent.cmp.oath.com" rel="dns-prefetch"/><link href="//geo.yahoo.com" rel
    ="dns-prefetch"/><link crossorigin="anonymous" href="//l.yimg.com" rel="preconnect"/><link crossorigin="anonymous" hr
    ef="//s.yimg.com" rel="preconnect"/><link href="//csc.beap.bc.yahoo.com" rel="preconnect"/><link href="//geo.query.ya
    hoo.com" rel="preconnect"/><link href="//ytas.btrll.com" rel="preconnect"/><link href="//shim.btrll.com" rel="preconne
    ct"/>
```

16.5 MANIPULANDO LOS DATOS

Pero ¿cómo saber en qué clases se almacenan los datos relevantes?

Después de algunas búsquedas, sabemos que están almacenados en “div”, podemos crear una lista vacía y usar un bucle for para encontrar todos los elementos y agregarlos a la lista.

```
# 4. Crear una lista y encontrar estructuras de datos 'div'

ls= [] # Crear una lista vacía
for l in soup_is.find_all('div'):

    #Encontrar las estructuras de datos que son 'div'
    ls.append(l.string) # adiciona cada elemento uno por uno a la lista

ls = [e for e in ls if e not in ('Operating Expenses','Non-recurring Events')] # Excluir esas columnas
```

Encontrará que hay muchos elementos “none” en ls porque no todos los “div” tienen un elemento. Solo tenemos que filtrarlos.

```
new_ls = list(filter(None,ls))
new_ls
```

```
In [25]: M new_ls
Out[25]: ['Yahoo Finance',
  "(function () { var inputEl = document.getElementById('yfin-usr-qry'); var perf = window.performance; if (inputEl && inputEl.addEventListener && perf && perf.mark && perf.getEntriesByName) { var listener = function (e) { inputEl.removeEventListener('keyup', listener); if (perf.getEntriesByName('Fin.Search first keyup').length === 0) { perf.mark('Fin.Search first keyup'); } }; inputEl.addEventListener('keyup', listener); } }()); if (window && window.FinSearch && window.FinSearch.init) { window.FinSearch.init({}); }",
  "(function () { var inputEl = document.getElementById('yfin-usr-qry'); var perf = window.performance; if (inputEl && inputEl.addEventListener && perf && perf.mark && perf.getEntriesByName) { var listener = function (e) { inputEl.removeEventListener('keyup', listener); if (perf.getEntriesByName('Fin.Search first keyup').length === 0) { perf.mark('Fin.Search first keyup'); } }; inputEl.addEventListener('keyup', listener); } }()); if (window && window.FinSearch && window.FinSearch.init) { window.FinSearch.init({}); }",
  'react-empty: 2',
  'react-empty: 2',
  'Microsoft Corporation (MSFT)',
  'NasdaqGS - NasdaqGS Real Time Price. Currency in USD',
  'As of 11:28AM EST. Market open.',
  'Income Statement',
  'Balance Sheet',
  'Cash Flow',
  'Annual']
```

Si damos un paso más y comenzamos a leer la lista comenzando en la posición 12.

```
new_ls = new_ls[12:]
```

Bueno, ahora tenemos una lista. Pero ¿cómo lo convertimos en un DataFrame? Primero, necesitamos iterar 6 elementos a la vez y almacenarlos en tuplas. Sin embargo, queremos una lista para que la librería de pandas pueda leerla en un DataFrame.

```
is_data = list(zip(*[iter(new_ls)]*6))
is_data
```

Salida

```
In [27]: M is_data
Out[27]: [('Yahoo Finance',
  "(function () { var inputEl = document.getElementById('yfin-usr-qry'); var perf = window.performance; if (inputEl && inputEl.addEventListener && perf && perf.mark && perf.getEntriesByName) { var listener = function (e) { inputEl.removeEventListener('keyup', listener); if (perf.getEntriesByName('Fin.Search first keyup').length === 0) { perf.mark('Fin.Search first keyup'); } }; inputEl.addEventListener('keyup', listener); } }()); if (window && window.FinSearch && window.FinSearch.init) { window.FinSearch.init({}); }",
  "(function () { var inputEl = document.getElementById('yfin-usr-qry'); var perf = window.performance; if (inputEl && inputEl.addEventListener && perf && perf.mark && perf.getEntriesByName) { var listener = function (e) { inputEl.removeEventListener('keyup', listener); if (perf.getEntriesByName('Fin.Search first keyup').length === 0) { perf.mark('Fin.Search first keyup'); } }; inputEl.addEventListener('keyup', listener); } }()); if (window && window.FinSearch && window.FinSearch.init) { window.FinSearch.init({}); }",
  'react-empty: 2',
  'react-empty: 2',
  'Microsoft Corporation (MSFT)',
  'NasdaqGS - NasdaqGS Real Time Price. Currency in USD',
  'As of 11:28AM EST. Market open.',
  'Income Statement',
  'Balance Sheet',
  'Cash Flow',
  'Annual']
```

Perfecto, eso es exactamente lo que queremos. Ahora, solo tenemos que leerlo en un DataFrame.

```
Income_st = pd.DataFrame(is_data[0:])
Income_st
```

Salida

	In [29]:	Out[29]:	0	1	2	3	4	5
0			Yahoo Finance	(function () { var inputEl = document.getEleme...	(function () { var inputEl = document.getEleme...	readEmpty: 2	readEmpty: 2	Microsoft Corporation (MSFT)
1	NasdaqGS - NasdaqGS Real Time Price, Currency ...	As of 11:28AM EST. Market open.			Income Statement	Balance Sheet	Cash Flow	Annual
2	Quarterly	Expand All			Item	6/30/2020	6/30/2019	6/30/2018
3	6/30/2017	147,114,000			143,015,000	125,843,000	110,386,000	89,950,000
4	Cost of Revenue	46,674,000			46,078,000	42,910,000	38,353,000	34,281,000
5	Gross Profit	100,440,000			96,937,000	82,033,000	72,007,000	55,689,000
6	44,291,000	43,978,000			39,974,000	36,049,000	33,057,000	Operating Income
7	56,149,000	52,959,000			42,859,000	35,054,000	22,032,000	-2,543,000
8	-2,591,000	-2,686,000			-2,733,000	-2,222,000	2,868,000	2,668,000
9	3,415,000	4,149,000			2,730,000	Pretax Income	56,474,000	53,036,000
10	43,688,000	36,474,000			23,149,000	Tax Provision	8,978,000	8,755,000
11	4,448,000	19,903,000			1,045,000	47,486,000	44,281,000	-39,240,000
12	16,571,000	21,204,000		Diluted NI Available to Com Stockholders	47,496,000	44,281,000		39,240,000
13	16,571,000	21,204,000		Basic EPS	-	0.0058		0.0051
14	0.0022	0.0027		Diluted EPS	-	0.0058		0.0051
15	0.0021	0.0027		Basic Average Shares	-	7,610,000		7,673,000
16	7,706,000	7,746,000		Diluted Average Shares	-	7,683,000		7,753,000
17	7,794,000	7,832,000	Total Operating Income as Reported		56,149,000	52,959,000		42,959,000

16.6 LIMPIANDO LOS DATOS

Ya casi está hecho. Solo necesitamos leer la primera fila como columna y la primera columna como índice de fila. Aquí hay algunas cosas para limpiar.

5. Limpiando los datos

```
Income_st.columns = Income_st.iloc[0] # Nombre de las columnas a la primera fila del dataframe
```

```
Income_st = Income_st.iloc[1:,] # Empezar a leer la primera fila
```

```
Income_st = Income_st.T # Transponer el DataFrame
```

```

Income_st.columns = Income_st.iloc[0] # Nombre de las columnas a la primera fila
del dataframe

Income_st.drop(Income_st.index[0], inplace=True) # Omitir la primera fila del índice

Income_st.index.name = "" # Quitar el nombre del índice

Income_st.rename(index={'ttm': '12/31/2019'}, inplace=True) # Renombrar ttm en la
columna índice por el fin de año.

Income_st = Income_st[Income_st.columns[:-5]] # Remover las 5 columnas irrelevantes

```

Después de utilizar las mismas técnicas para el estado de resultados, el balance y el flujo de caja, sus DataFrames deberían tener el siguiente aspecto.

Annual	Total Revenue	Cost of Revenue	Gross Profit	Research Development	Selling General and Administrative	Total Operating Expenses	Operating Income or Loss	Interest Expense	Total Other Income/Expenses Net	Income Before Tax	Income Tax Expense	Ir Cont Oper
12/31/2019	129,814,000	43,411,000	86,403,000	17,464,000	23,249,000	40,713,000	45,690,000	2,649,000	3,112,000	46,153,000	5,059,000	41,0
6/29/2019	125,843,000	42,910,000	82,933,000	16,876,000	23,098,000	39,974,000	42,959,000	2,686,000	3,415,000	43,688,000	4,448,000	39,2
6/29/2018	110,360,000	38,353,000	72,007,000	14,726,000	22,223,000	36,949,000	35,058,000	2,733,000	4,149,000	36,474,000	19,903,000	16,5
6/29/2017	89,950,000	34,261,000	55,689,000	13,037,000	20,020,000	33,057,000	22,632,000	2,222,000	2,739,000	23,149,000	1,945,000	21,2
6/29/2016	85,320,000	32,780,000	52,540,000	11,988,000	19,260,000	31,248,000	21,292,000	1,243,000	-298,000	19,751,000	2,953,000	16,7

Annual	Net Income	Depreciation & amortization	Deferred income taxes	Stock based compensation	Change in working capital	Accounts receivable	Inventory	Accounts Payable	Other working capital	Other non-cash items	Common stock repurchased	Dividends Paid
2019-12-31	41,094,000	11,816,000	-5,708,000	4,807,000	-3,132,000	+1,916,000	992,000	80,000	38,638,000	4,011,000	-20,711,000	-14,101,000
2019-06-29	39,240,000	11,682,000	-3,534,000	4,652,000	-3,525,000	-2,812,000	597,000	232,000	38,260,000	4,462,000	-19,543,000	-13,811,000
2018-06-29	16,571,000	10,261,000	13,040,000	3,940,000	-3,638,000	-3,862,000	-465,000	1,148,000	32,252,000	5,922,000	-10,721,000	-12,699,000
2017-06-29	21,204,000	8,778,000	-3,296,000	3,266,000	1,652,000	-925,000	50,000	81,000	31,378,000	67,711,000	-11,788,000	-11,845,000
2016-06-29	16,798,000	6,622,000	332,000	2,668,000	-2,076,000	-530,000	600,000	88,000	24,982,000	57,072,000	-15,969,000	-11,006,000

Cash And Cash Equivalents	Short Term Investments	Total Cash	Net Receivables	Inventory	Other Current Assets	Total Current Assets	Gross property, plant and equipment	Accumulated Depreciation	Net property, plant and equipment	Deferred taxes liabilities	Deferred Revenue	
6/29/2019	11,356,000	122,463,000	133,819,000	29,524,000	2,063,000	10,146,000	175,552,000	79,186,000	-35,330,000	43,856,000	233,000	4,530,00
6/29/2018	11,946,000	121,822,000	133,768,000	26,481,000	2,662,000	6,751,000	169,662,000	65,369,000	-29,223,000	36,146,000	541,000	3,615,00
6/29/2017	7,663,000	125,318,000	132,981,000	19,792,000	2,181,000	4,897,000	159,851,000	47,913,000	-24,179,000	23,734,000	531,000	10,377,00
6/29/2016	6,510,000	106,730,000	113,240,000	18,277,000	2,251,000	5,892,000	139,660,000	38,156,000	-19,800,000	16,356,000	1,476,000	6,441,00

17

TRADING ALGORÍTMICO CON PYTHON

El trading algorítmico, o trading basado en reglas y procesos, es una modalidad de operación en mercados financieros (trading) que se caracteriza por el uso de algoritmos, reglas y procedimientos automatizados en diferentes grados, para ejecutar operaciones de compra o venta de instrumentos financieros.

El trading algorítmico viene creciendo considerablemente con la evolución de las tecnologías y la facilidad de acceso a internet, Python se ha constituido en una fuerte herramienta para diseñar, implementar y validar estrategias propias del trading algorítmico. En este capítulo se ilustrarán dos estrategias y su implementación en Python.

17.1 TABLERO DE TRADING CON YFINANCE & PYTHON

El objetivo de esta sección es proporcionar al inversionista minorista una forma rápida y sencilla de extraer datos en directo, utilizar esos datos para resaltar los indicadores clave y crear una tabla agradable y legible antes de invertir en una determinada empresa (s). Este proceso le ayudará a sacar la emoción de la ecuación y le dará suficiente información para tomar decisiones informadas.

17.1.1 Extrayendo datos con la api Yfinance

Sustituya cualquier ticker de acciones que desee en la parte inferior del bloque de código. En este ejemplo “net” corresponde al ticker de la empresa Cloudflare Inc.

```
# 1. Importar librerías
import numpy as np
```

```

import pandas as pd
import hvplot.pandas
from pathlib import Path
import yfinance as yf

# 2. Elegir la empresa
net = yf.Ticker("net") # empresa Cloudflare Inc.

# 3. Establezca el período de tiempo que le interesa analizar.
net_historical = net.history(start="2018-01-01", end="2020-12-31", interval="1d")

# 4. Cree un nuevo DataFrame llamado signals, conservando solamente las columnas
# 'Date' y 'Close'
signals_df = net_historical.drop(columns=['Open', 'High', 'Low',
                                           'Volume', 'Dividends', 'Stock Splits'])

```

17.1.2 Establecer las ventanas cortas y largas (medias móviles)

A continuación, queremos crear columnas para las ventanas cortas y largas, también conocidas como medias móviles simples. En este caso, utilizaremos los promedios de 50 y 100 días.

En el siguiente código, necesitaremos establecer las señales de trading como 0 o 1. Esto le dirá a Python en qué puntos debemos comprar o vender una posición.

Tenga en cuenta que cuando el SMA50 cruza por encima del SMA100 o el nivel de resistencia, esta es una señal de ruptura alcista.

```

# 5. Establecer los marcos temporales corto y largo
short_window = 50
long_window = 100

# 6. Genere las medias móviles cortas y largas (50 y 100 días, respectivamente)
signals_df['SMA50'] = signals_df['Close'].rolling(window=short_window).mean()
signals_df['SMA100'] = signals_df['Close'].rolling(window=long_window).mean()
signals_df['Signal'] = 0.0

# 7. Genere las señales de trading 0 o 1, donde 0 es cuando el SMA50 está debajo
# de SMA100, y 1 donde 1 es cuando la SMA50 está por encima (o cruza) la SMA100

signals_df['Signal'][short_window:] = np.where(
    signals_df['SMA50'][short_window:] > signals_df['SMA100'][short_window:],
```

```

1.0, 0.0
)

# 8. Calcule los puntos en el tiempo en los que se debe tomar una posición 1 o
-1 signals_df['Entry/Exit'] = signals_df['Signal'].diff()

# 9. Imprima el DataFrame
signals_df.tail(10)

```

Salida

	Out[3]:	Close	SMA50	SMA100	Signal	Entry/Exit
	Date					
2020-11-27	74.76	54.3126	46.2071	1.0	0.0	
2020-11-30	75.08	55.0722	46.5606	1.0	0.0	
2020-12-01	72.29	55.7388	46.8827	1.0	0.0	
2020-12-02	71.90	56.3794	47.2342	1.0	0.0	
2020-12-03	74.13	57.0624	47.6080	1.0	0.0	
2020-12-04	77.35	57.8390	48.0203	1.0	0.0	
2020-12-07	81.96	58.6774	48.4889	1.0	0.0	
2020-12-08	83.55	59.5308	48.9542	1.0	0.0	
2020-12-09	77.31	60.2636	49.3436	1.0	0.0	
2020-12-10	84.05	61.1234	49.8023	1.0	0.0	

17.1.3 Generando señales de trading

El tercer paso hacia la construcción de nuestro tablero es crear un gráfico con marcadores de señales verdes y rojos para los indicadores de Entrada / Salida.

Trazar las medias móviles con HvPlot:

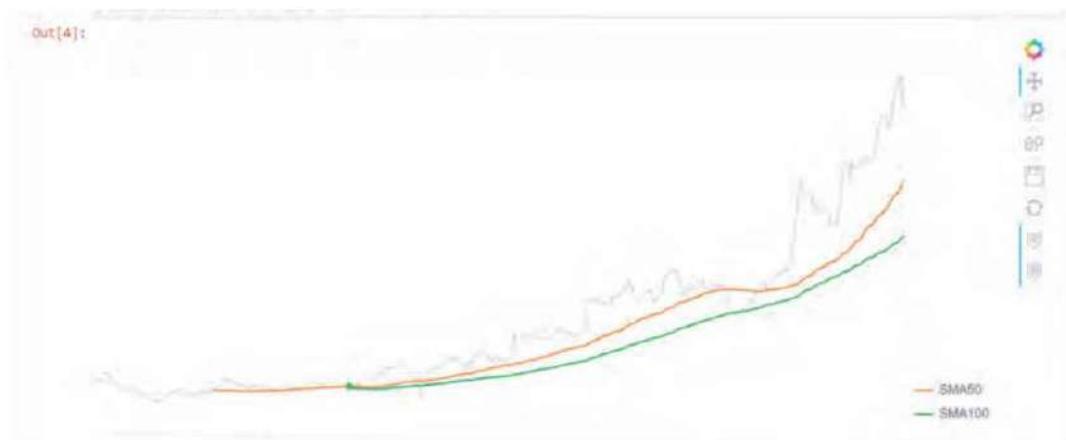
```

# 10. Visualice la posición de salida en relación con el precio de cierre.
exit = signals_df[signals_df['Entry/Exit'] == -1.0]['Close'].hvplot.scatter(
    color='red',
    legend=False,
    ylabel='Price in $',
    width=1000,
    height=400
)
# 11. Visualice la posición de entrada en relación con el precio de cierre
entry = signals_df[signals_df['Entry/Exit'] == 1.0]['Close'].hvplot.scatter(
    color='green',
    legend=False,

```

```
        ylabel='Price in $',
        width=1000,
        height=400
    )
# 12. Visualice el precio de cierre de la inversión
security_close = signals_df[['Close']].hvplot(
    line_color='lightgray',
    ylabel='Price in $',
    width=1000,
    height=400
)
# 13. Visualice los promedio móviles
moving_avgs = signals_df[['SMA50', 'SMA100']].hvplot(
    ylabel='Price in $',
    width=1000,
    height=400
)
# 14. Gráficos superpuestos
entry_exit_plot = security_close * moving_avgs * entry * exit
entry_exit_plot.opts(xaxis=None)
```

Salida



17.1.4 Trazando puntos de entrada y salida

A continuación, estableceremos una participación de inversión inicial de capital y estableceremos el número de acciones. Para este ejemplo, digamos que queremos comprar 500 acciones de Cloudflare.

```
.....  
# 15. Establecer el capital inicial  
initial_capital = float(100000)  
  
# 16. Establecer la cantidad de acciones  
share_size = 500  
  
#17. Tome una posición de 500 acciones donde el cruce del promedio móvil dual es  
1 (SMA50 es mayor que SMA100)  
signals_df['Position'] = share_size * signals_df['Signal']  
  
# 18. Encuentre los puntos en el tiempo en los que se compra o vende una posi-  
ción de 500 acciones  
signals_df['Entry/Exit Position'] = signals_df['Position'].diff()  
  
# 19. Multiplique el precio de la acción por las posiciones de entrada / salida  
y obtenga la suma acumulativa  
signals_df['Portfolio Holdings'] = signals_df['Close'] * signals_df['Entry/Exit  
Position'].cumsum()  
  
# 20. Reste del capital inicial las compras realizadas para obtener la cantidad  
de efectivo líquido  
signals_df['Portfolio Cash'] = initial_capital - (signals_df['Close'] * signals_  
df['Entry/Exit Position']).cumsum()  
  
#21. Obtenga el valor total del portafolio sumando el monto en efectivo y las  
inversiones  
signals_df['Portfolio Total'] = signals_df['Portfolio Cash'] + signals_  
df['Portfolio Holdings']  
  
# 22. Calcule la rentabilidad diaria del portafolio  
signals_df['Portfolio Daily Returns'] = signals_df['Portfolio Total'].pct_change()  
  
# 23. Calcule los rendimientos acumulados  
signals_df['Portfolio Cumulative Returns'] = (1 + signals_df['Portfolio Daily  
Returns']).cumprod() - 1  
  
# 24. Imprima el DataFrame  
signals_df.tail(10)
```

.....

Salida

Date	Close	SMA50	SMA100	Signal	Entry/Exit	Position	Entry/Exit Position	Portfolio Holdings	Portfolio Cash	Portfolio Total	Portfolio Daily Returns	Portfolio Cumulative Returns
2020-11-27	74.76	54.3126	46.2071	1.0	0.0	500.0	0.0	37380.0	91000.0	128380.0	0.018929	0.28389
2020-11-30	75.08	55.0722	46.5606	1.0	0.0	500.0	0.0	37540.0	91000.0	128540.0	0.001246	0.28540
2020-12-01	72.29	55.7386	46.8527	1.0	0.0	500.0	0.0	36145.0	91000.0	127145.0	-0.010883	0.27145
2020-12-02	71.90	56.3794	47.2342	1.0	0.0	500.0	0.0	35650.0	91000.0	128950.0	-0.001534	0.26950
2020-12-03	74.13	57.0624	47.6060	1.0	0.0	500.0	0.0	37065.0	91000.0	128065.0	0.008783	0.28065
2020-12-04	77.35	57.8380	48.6203	1.0	0.0	500.0	0.0	38675.0	91000.0	129675.0	-0.012572	0.29675
2020-12-07	81.95	58.5774	48.4889	1.0	0.0	500.0	0.0	40680.0	91000.0	131980.0	-0.017775	0.31980
2020-12-08	83.55	59.5308	48.9542	1.0	0.0	500.0	0.0	41775.0	91000.0	132775.0	0.006024	0.32775
2020-12-09	77.31	60.2636	49.3438	1.0	0.0	500.0	0.0	36655.0	91000.0	129655.0	-0.023498	0.29555
2020-12-10	84.05	61.1234	49.8023	1.0	0.0	500.0	0.0	42025.0	91000.0	133025.0	0.025992	0.33025

Visualice las posiciones de salida relativas a nuestro portafolio:

```
# 25. Visualice la posición de salida en relación con el valor total del portafolio
exit = signals_df[signals_df['Entry/Exit'] == -1.0]['Portfolio Total'].hvplot.
scatter(
    color='red',
    legend=False,
    ylabel='Total Portfolio Value',
    width=1000,
    height=400
)
# 26. Visualice la posición de entrada en relación con el valor total del portafolio
entry = signals_df[signals_df['Entry/Exit'] == 1.0]['Portfolio Total'].hvplot.
scatter(
    color='green',
    legend=False,
    ylabel='Total Portfolio Value',
    width=1000,
    height=400
)
# 27. Visualice el valor total del portafolio para la inversión
total_portfolio_value = signals_df[['Portfolio Total']].hvplot(
    line_color='lightgray',
    ylabel='Total Portfolio Value',
    width=1000,
    height=400
)
```

```
# 28. Gráficos superpuestos
portfolio_entry_exit_plot = total_portfolio_value * entry * exit
portfolio_entry_exit_plot.opts(xaxis=None)
```

Salida



```
# 29. Preparar el DataFrame para los indicadores
```

```
metrics = [
    'Annual Return',
    'Cumulative Returns',
    'Annual Volatility',
    'Sharpe Ratio',
    'Sortino Ratio']
columns = ['Backtest']
```

```
# 30. Inicialice el DataFrame con el índice establecido en los indicadores de evaluación y la columna como `Backtest`
```

```
portfolio_evaluation_df = pd.DataFrame(index=metrics, columns=columns)
```

Salida

Backtest	
Annual Return	NaN
Cumulative Returns	NaN
Annual Volatility	NaN
Sharpe Ratio	NaN
Sortino Ratio	NaN

17.1.5 Realizar Backtest

En esta sección se destacan los siguientes indicadores:

- ▶ Rendimiento acumulado: rendimiento total de la inversión.
- ▶ Rentabilidad anual - rentabilidad de la inversión recibida ese año.
- ▶ Volatilidad anual: la volatilidad diaria multiplicada por la raíz cuadrada de 252 días de negociación.
- ▶ Índice de Sharpe: mide el rendimiento de una inversión en comparación con un activo libre de riesgo, después de ajustar su riesgo.
- ▶ Relación de Sortino: diferencia la volatilidad negativa (desviación estándar de los rendimientos negativos) de la volatilidad general total utilizando la desviación estándar del activo de los rendimientos negativos de la cartera, la desviación a la baja, en lugar de la desviación estándar total de los rendimientos de la cartera.

```
# 31. Calcular el rendimiento acumulado
portfolio_evaluation_df.loc['Cumulative Returns'] = signals_df['Portfolio Cumulative Returns'][-1]

# 32. Calcular la rentabilidad anual
portfolio_evaluation_df.loc['Annual Return'] = (
    signals_df['Portfolio Daily Returns'].mean() * 252
)

# 33. Calcular la volatilidad anual
portfolio_evaluation_df.loc['Annual Volatility'] = (
    signals_df['Portfolio Daily Returns'].std() * np.sqrt(252)
)

# 34. Calcular el indicador Sharpe Ratio
portfolio_evaluation_df.loc['Sharpe Ratio'] = (
    signals_df['Portfolio Daily Returns'].mean() * 252) / (
    signals_df['Portfolio Daily Returns'].std() * np.sqrt(252)
)

# 35. Calcular el rendimiento a la baja
sortino_ratio_df = signals_df[['Portfolio Daily Returns']].copy()
sortino_ratio_df.loc[:, 'Downside Returns'] = 0
target = 0
mask = sortino_ratio_df['Portfolio Daily Returns'] < target
sortino_ratio_df.loc[mask, 'Downside Returns'] = sortino_ratio_df['Portfolio Daily Returns']**2
portfolio_evaluation_df
```

```
# 36. Calcular el indicador Sortino Ratio
down_stdev = np.sqrt(sortino_ratio_df['Downside Returns'].mean()) * np.sqrt(252)
expected_return = sortino_ratio_df['Portfolio Daily Returns'].mean() * 252
sortino_ratio = expected_return/down_stdev
portfolio_evaluation_df.loc['Sortino Ratio'] = sortino_ratio
portfolio_evaluation_df.head()
```

Salida

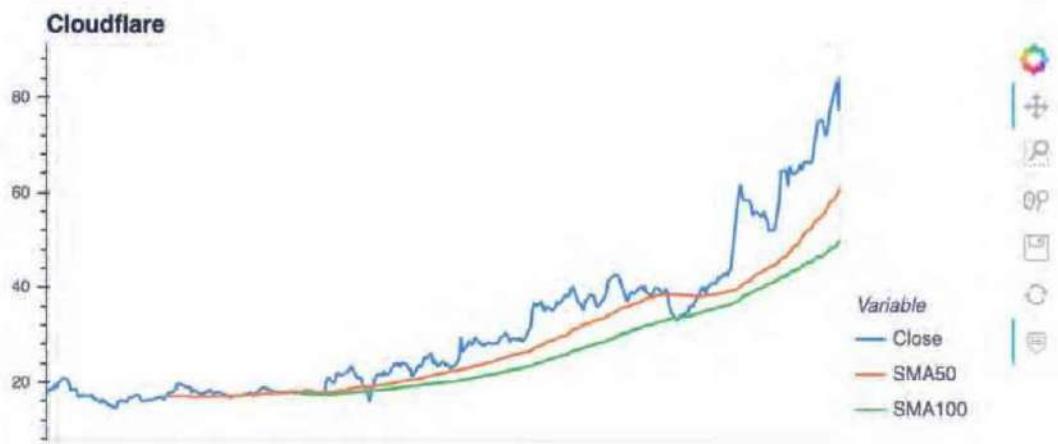
Out[8]:	
Backtest	
Annual Return	0.234779
Cumulative Returns	0.33025
Annual Volatility	0.0997506
Sharpe Ratio	2.35366
Sortino Ratio	4.40517

```
# 37. Iniciar el DataFrame con las columnas para la evaluación de las transacciones
trade_evaluation_df = pd.DataFrame(
    columns=[
        'Stock',
        'Entry Date',
        'Exit Date',
        'Shares',
        'Entry Share Price',
        'Exit Share Price',
        'Entry Portfolio Holding',
        'Exit Portfolio Holding',
        'Profit/Loss']
)
```

- Recorra DataFrame, si la operación “Entrada / Salida” es 1, configure las métricas de la operación de Entrada
- Si “Entrada / Salida” es -1, establezca métricas de operaciones de salida y calcule las ganancias
- Adjunte el registro al DataFrame de evaluación de trading

```
.....  
# 38. Inicializar las variables a iterar  
entry_date = ''  
exit_date = ''  
entry_portfolio_holding = 0  
exit_portfolio_holding = 0  
share_size = 0  
entry_share_price = 0  
exit_share_price = 0  
  
for index, row in signals_df.iterrows():  
    if row['Entry/Exit'] == 1:  
        entry_date = index  
        entry_portfolio_holding = abs(row['Portfolio Holdings'])  
        share_size = row['Entry/Exit Position']  
        entry_share_price = row['Close']  
    elif row['Entry/Exit'] == -1:  
        exit_date = index  
        exit_portfolio_holding = abs(row['Close'] * row['Entry/Exit Position'])  
        exit_share_price = row['Close']  
        profit_loss = entry_portfolio_holding - exit_portfolio_holding  
        trade_evaluation_df = trade_evaluation_df.append(  
            {  
                'Stock': 'NET',  
                'Entry Date': entry_date,  
                'Exit Date': exit_date,  
                'Shares': share_size,  
                'Entry Share Price': entry_share_price,  
                'Exit Share Price': exit_share_price,  
                'Entry Portfolio Holding': entry_portfolio_holding,  
                'Exit Portfolio Holding': exit_portfolio_holding,  
                'Profit/Loss': profit_loss  
            },  
            ignore_index=True)  
  
# 39. Graficar los resultados  
  
price_df = signals_df[['Close', 'SMA50', 'SMA100']]  
price_chart = price_df.hvplot.line()  
price_chart.opts(title='Cloudflare', xaxis=None)
```

Salida



17.1.6 Generar Dashboard o Tablero de Control

```
# 40. Generar tablero de control
```

```
portfolio_evaluation_df.reset_index(inplace=True)
portfolio_evaluation_table = portfolio_evaluation_df.hvplot.table()
portfolio_evaluation_table
```

Salida

index	Backtest
Annual Return	0.2347785334442904
Cumulative Returns	0.3302500000000026
Annual Volatility	0.09975058037938116
Sharpe Ratio	2.353655813844468
Sortino Ratio	4.405167733907506

17.2 ESTRATEGIA DE NEGOCIACIÓN ALGORÍTMICA CON MACD

En esta sección, aprenderá una estrategia de trading simple que se utiliza para determinar cuándo comprar y vender acciones utilizando el lenguaje de programación Python. Más específicamente, aprenderá a realizar operaciones algorítmicas. Es extremadamente difícil intentar predecir la dirección del impulso del mercado de valores, pero vamos a intentarlo. Incluso las personas con un buen conocimiento de las estadísticas y las probabilidades tienen dificultades para hacer esto.

El trading algorítmico es un proceso para ejecutar órdenes utilizando instrucciones de trading automatizadas y preprogramadas para tener en cuenta variables como precio, tiempo y volumen.

Python es uno de los lenguajes de programación más populares para las finanzas junto con otros como C # y R. La estrategia de trading que se utilizará en este artículo se llama crossover MACD.

17.2.1 ¿Qué es MACD Crossover?

El cruce de la divergencia/convergencia de la media móvil (MACD) es un indicador técnico que utiliza la diferencia entre las medias móviles exponenciales (EMA) para determinar el impulso y la dirección del mercado. El cruce de MACD ocurre cuando la línea MACD y la línea de señal se interceptan, lo que a menudo indica un cambio en el impulso / tendencia del mercado.

17.2.2 Componentes del MACD

El indicador / línea MACD: la línea MACD es la diferencia entre las dos medias móviles exponenciales (generalmente los últimos 12 y 26 días o semanas) y generalmente se la conoce como la línea más rápida.

Línea de señal: la línea de señal suele ser un promedio de 9 períodos suavizado exponencialmente de la línea MACD y se denominará línea más lenta.

Línea cero: las líneas MACD fluctúan por encima y por debajo de una línea cero, lo que le da al MACD las cualidades de un oscilador.

Histograma: el histograma consta de líneas verticales que muestran la extensión entre las dos líneas MACD.

Aquí nos vamos a centrar en la línea del indicador MACD y la línea de señal para determinar cuándo comprar y vender acciones; sin embargo, el uso de la línea de señal cero y el componente de histograma pueden ayudar y brindar más información sobre la compra o venta de un activo.

17.2.3 ¿Cómo se calcula el MACD?

El indicador o línea MACD se puede calcular simplemente restando la media móvil exponencial a largo plazo (por ejemplo, EMA de 26 períodos) de la media móvil exponencial a corto plazo (por ejemplo, la EMA de 12 períodos).

$$\text{MACD} = \text{EMA de 12 períodos} - \text{EMA de 26 períodos}.$$

Un MACD positivo indica que la EMA de 12 períodos está por encima de la EMA de 26 períodos. Los valores positivos aumentan a medida que la EMA más corta se aleja más de la EMA más larga. Esto significa que el impulso alcista está aumentando. Los valores negativos de MACD indican que la EMA de 12 períodos está por debajo de la EMA de 26 períodos. Los valores negativos aumentan a medida que la EMA más corta diverge más por debajo de la EMA más larga. Esto significa que el impulso a la baja está aumentando.

Un MACD positivo está por encima de la línea cero y un MACD negativo está por debajo de la línea cero.

¿Cuándo comprar y vender acciones con MACD?

Cuando el indicador MACD cruza la línea de señal, esto indica un cambio de impulso en el precio de las acciones. Por ejemplo, si el indicador MACD es mayor que la línea de señal, esto se considera un cruce alcista e indica un buen momento para comprar, y cuando el indicador MACD es menor que la línea de señal, esto se considera un cruce bajista e indica un buen momento para vender.

17.2.4 Implementación del MACD en Python

El siguiente algoritmo en Python muestra paso a paso la construcción de la estrategia, veamos:

Se importan las librerías requeridas:

1. Importar librerías

```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

Se carga un archivo con extensión CSV llamado ‘AAPL.csv’ que contiene los siguientes datos de la acción de Apple Corporation entre el 20 de abril y el 23 de agosto de 2017: fecha, precio de apertura, precio alto, precio de cierre, precio de cierre ajustado y volumen.

A continuación, se almacenan los datos en una variable.

```
# 2. Almacene los datos en la variable df
```

```
df = pd.read_csv('AAPL.csv')
```

Ahora se muestran los datos:

```
# 3. Establecer la fecha como índice para los datos
df = df.set_index(pd.DatetimeIndex(df['Date'].values))
```

```
# 4. Mostrar el DataFrame
df
```

Salida

Out[3]:

	Date	Open	High	Close	Adj Close	Volume
2017-04-20	4/20/2017	35.305000	35.730000	35.610001	33.930340	93278400
2017-04-21	4/21/2017	35.610001	35.669998	35.567501	33.889851	69283600
2017-04-24	4/24/2017	35.875000	35.987499	35.910000	34.216194	68537200
2017-04-25	4/25/2017	35.977501	36.224998	36.132500	34.428196	75486000
2017-04-26	4/26/2017	36.117500	36.150002	35.919998	34.225719	80164800
...						
2017-08-17	8/17/2017	40.130001	40.177502	39.465000	37.906998	111762400
2017-08-18	8/18/2017	39.465000	39.875000	39.375000	37.820560	109712400
2017-08-21	8/21/2017	39.375000	39.472500	39.302502	37.750916	105474000
2017-08-22	8/22/2017	39.557499	40.000000	39.945000	38.368046	86418400
2017-08-23	8/23/2017	39.767502	40.117500	39.994999	38.416069	77596400

88 rows × 6 columns

El precio de la acción se representa gráficamente:

```
# 5. Mostrar visualmente el precio de las acciones, crear la figura

title = 'Close Price History'      # Crear el título

my_stocks = df # Conseguir las acciones

plt.figure(figsize=(12.2,4.5)) #width = 12.2in, height = 4.5
plt.plot( my_stocks['Close'], label='Close')#plt.plot( X-Axis , Y-Axis, line_
width, alpha_for_blending, label)
plt.xticks(rotation=45)
plt.title(title)
plt.xlabel('Date',fontsize=18)
plt.ylabel('Price USD ($)',fontsize=18)
plt.show()
```

Salida



Se calculan los indicadores MACD y Línea de señal.

```
# 6. Calcular la media móvil exponencial a corto plazo
```

```
ShortEMA = df.Close.ewm(span=12, adjust=False).mean() # Promedio móvil rápido
```

7. Calcular la media móvil exponencial a largo plazo

```
LongEMA = df.Close.ewm(span=26, adjust=False).mean() # Promedio móvil lento
```

8. Calcular la media móvil Convergence/Divergence (MACD)

```
MACD = ShortEMA - LongEMA
```

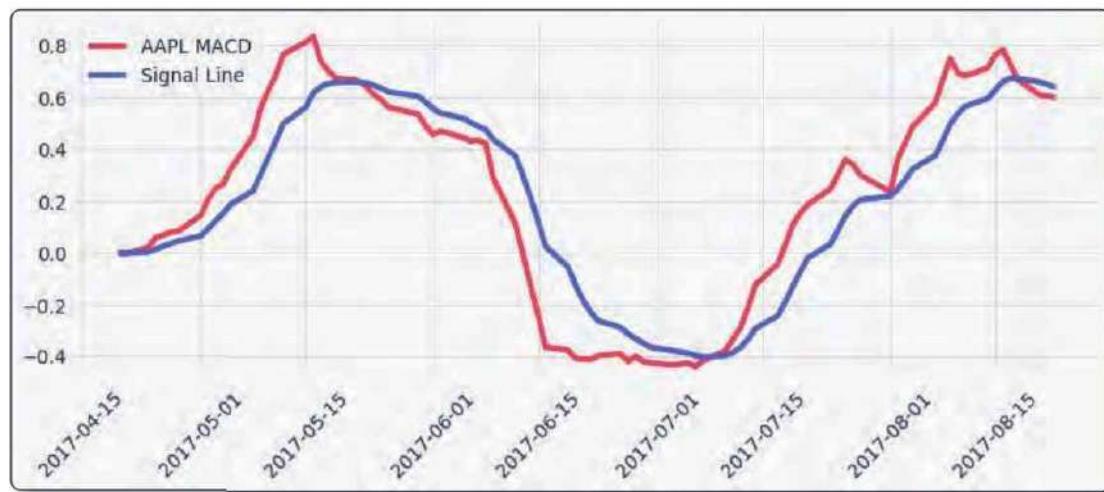
9. Calcular la línea de señal

```
signal = MACD.ewm(span=9, adjust=False).mean()
```

Graficar el MACD y la línea de señal. Nos interesa ver cuándo se cruzan las dos líneas, ya que es una indicación para comprar o vender el activo.

10. Graficar la figura del MACD

```
plt.figure(figsize=(12.2,4.5)) #width = 12.2in, height = 4.5
plt.plot(df.index, MACD, label='AAPL MACD', color = 'red')
plt.plot(df.index, signal, label='Signal Line', color='blue')
plt.xticks(rotation=45)
plt.legend(loc='upper left')
plt.show()
```

Salida

Se representa el conjunto de datos en columnas y se agregan nuevas columnas para el MACD y la línea de señal.

```
# 11. Crea nueva columnas para el DataFrame
```

```
df['MACD'] = MACD
df['Signal Line'] = signal
```

```
# 12. Mostrar el nuevo DataFrame
df
```

Salida

Out[7]:

	Date	Open	High	Close	Adj Close	Volume	MACD	Signal Line
2017-04-20	4/20/2017	35.305000	35.730000	35.610001	33.930340	93278400	0.000000	0.000000
2017-04-21	4/21/2017	35.610001	35.669998	35.567501	33.889851	69283600	-0.003390	-0.000678
2017-04-24	4/24/2017	35.875000	35.987499	35.910000	34.216194	68537200	0.021314	0.003720
2017-04-25	4/25/2017	35.977501	36.224998	36.132500	34.428196	75486000	0.058176	0.014611
2017-04-26	4/26/2017	36.117500	36.150002	35.919998	34.225719	80184800	0.069441	0.025577
<hr/>								
2017-08-17	8/17/2017	40.130001	40.177502	39.465000	37.906998	111762400	0.725700	0.672078
2017-08-18	8/18/2017	39.465000	39.875000	39.375000	37.820560	109712400	0.666326	0.670928
2017-08-21	8/21/2017	39.375000	39.472500	39.302502	37.750916	105474000	0.606431	0.658028
2017-08-22	8/22/2017	39.557499	40.000000	39.945000	38.368046	86418400	0.603848	0.647192
2017-08-23	8/23/2017	39.767502	40.117500	39.994999	38.416069	77596400	0.598930	0.637540

88 rows × 8 columns

Se crea una función para señalar cuándo comprar y vender acciones.

```
# 13. Cree una función para señalar cuándo comprar y vender un activo
def buy_sell(signal):
    sigPriceBuy = []
    sigPriceSell = []
    flag = -1
    for i in range(0,len(signal)):

        # si MACD > línea de señal entonces compra sino vende
        if signal['MACD'][i] > signal['Signal Line'][i]:
            if flag != 1:
                sigPriceBuy.append(signal['Close'][i])
                sigPriceSell.append(np.nan)
                flag = 1
            else:
```

```

        sigPriceBuy.append(np.nan)
        sigPriceSell.append(np.nan)
    elif signal['MACD'][i] < signal['Signal Line'][i]:
        if flag != 0:
            sigPriceSell.append(signal['Close'][i])
            sigPriceBuy.append(np.nan)
            flag = 0
        else:
            sigPriceBuy.append(np.nan)
            sigPriceSell.append(np.nan)
    else: #Handling nan values
        sigPriceBuy.append(np.nan)
        sigPriceSell.append(np.nan)

return (sigPriceBuy, sigPriceSell)

```

Se crean y muestran las columnas con información de compra y venta.

```

# 14. Crear columnas de compra y venta
x = buy_sell(df)
df['Buy_Signal_Price'] = x[0]
df['Sell_Signal_Price'] = x[1]
#Show the data frame
df

```

Salida

	Date	Open	High	Close	Adj Close	Volume	MACD	Signal Line	Buy_Signal_Price	Sell_Signal_Price
2017-04-20	4/20/2017	35.305000	35.730000	35.610001	33.930340	93278400	0.000000	0.000000	NaN	NaN
2017-04-21	4/21/2017	35.810001	35.869998	35.567501	33.886985	69283600	-0.003380	-0.000678	NaN	35.567501
2017-04-24	4/24/2017	35.875000	35.987499	35.910000	34.216194	68537200	0.021314	0.003720	35.91	NaN
2017-04-25	4/25/2017	35.877501	36.224998	36.132800	34.428198	75486000	0.058176	0.014611	NaN	NaN
2017-04-26	4/26/2017	36.117500	38.150002	35.919998	34.225719	80164800	0.069441	0.025577	NaN	NaN
									—	—
2017-08-17	8/17/2017	40.130001	40.177502	39.465000	37.906998	111762400	0.725700	0.672078	NaN	NaN
2017-08-18	8/18/2017	39.465000	39.875000	39.375000	37.820560	109712400	0.666326	0.670928	NaN	39.375000
2017-08-21	8/21/2017	39.375000	39.472500	39.302502	37.750916	105474000	0.606431	0.658028	NaN	NaN
2017-08-22	8/22/2017	39.557499	40.000000	39.945000	38.368048	86418400	0.603848	0.647192	NaN	NaN
2017-08-23	8/23/2017	39.767502	40.117500	39.994999	38.418068	775986400	0.598930	0.637540	NaN	NaN

88 rows × 10 columns

Se representan visualmente las señales de compra y venta de acciones.

```
# 15. Mostrar visualmente las señales de compra y venta de acciones, crear figura
```

```
title = 'Close Price History Buy / Sell Signals' # Crear el título  
  
my_stocks = df #Conseguir las acciones  
  
plt.figure(figsize=(12.2,4.5)) #width = 12.2in, height = 4.5  
plt.scatter(my_stocks.index, my_stocks['Buy_Signal_Price'], color = 'green',  
label='Buy Signal', marker = '^', alpha = 1)  
plt.scatter(my_stocks.index, my_stocks['Sell_Signal_Price'], color = 'red',  
label='Sell Signal', marker = 'v', alpha = 1)  
plt.plot( my_stocks['Close'], label='Close Price', alpha = 0.35) #plt.plot( X-  
Axis , Y-Axis, line_width, alpha_for_blending, label)  
plt.xticks(rotation=45)  
plt.title(title)  
plt.xlabel('Date', fontsize=18)  
plt.ylabel('Close Price USD ($)', fontsize=18)  
plt.legend( loc='upper left')  
plt.show()
```

Salida



<https://dogramcode.com/programacion>

MATERIAL ADICIONAL

El material adicional de este libro puede descargarlo en nuestro portal web:
<http://www.ra-ma.es>.

Debe dirigirse a la ficha correspondiente a esta obra, dentro de la ficha encontrará el enlace para poder realizar la descarga.

Cuando descomprima el fichero obtendrá los archivos que complementan al libro para que pueda continuar con su aprendizaje.

INFORMACIÓN ADICIONAL Y GARANTÍA

- ▀ RA-MA EDITORIAL garantiza que estos contenidos han sido sometidos a un riguroso control de calidad.
- ▀ Los archivos están libres de virus, para comprobarlo se han utilizado las últimas versiones de los antivirus líderes en el mercado.
- ▀ RA-MA EDITORIAL no se hace responsable de cualquier pérdida, daño o costes provocados por el uso incorrecto del contenido descargable.
- ▀ Este material es gratuito y se distribuye como contenido complementario al libro que ha adquirido, por lo que queda terminantemente prohibida su venta o distribución.