پروژه پایانی درس الگوریتم معاملاتی

علی شیخ عطار

علی کمالی

دکتر انتظاری

دانشکده مهندسی کامپیوتر علم و صنعت

بررسی بخش ۱: جمع‌آوری و پردازش داده‌ها
هدف از این بخش جمع‌آوری داده‌های تاریخی ارزهای دیجیتال، پیش‌پردازش آن‌ها، تقسیم آن‌ها به داده‌های آموزشی و تست، و انجام تجزیه و تحلیل اکتشافی اولیه (EDA) است. این اطمینان حاصل می‌کند که داده‌ها پاک و آماده برای تحلیل‌های بیشتر در مراحل بعدی باشند.

توضیح مرحله به مرحله کد

۱. تنظیم بذر تصادفی برای تکرارپذیری
کد: `(np.random.seed(42`

- این اطمینان می‌دهد که هر عملیاتی که شامل تصادفیت است (اگر بعداً استفاده شود) نتایج یکسانی را در اجراهای مختلف تولید کند.
- مقدار بذر ۴۲ به طور معمول در عمل استفاده می‌شود و اطمینان می‌دهد که اعداد تصادفی تولید شده در مراحل آینده (در صورت وجود) قابل تکرار باشند.

---

۲. تابعی برای دریافت داده‌های ارز دیجیتال
تعریف تابع:

```
def fetch_crypto_data(symbols, start_date, end_date):
    data = {}
    for symbol in symbols:
        ticker = yf.Ticker(symbol)
        df = ticker.history(start=start_date, end=end_date)
        data[symbol] = df
    return data
```

توضیح:

- این تابع از yfinance (API Yahoo Finance) برای بازیابی داده‌های تاریخی قیمت ارزهای دیجیتال استفاده می‌کند.
- **ورودی‌ها:**
  - `symbols`: لیستی از نمادهای تیکر ارز دیجیتال (مثلاً BTC-USD, ETH-USD).
  - `start_date`: تاریخ شروع جمع‌آوری داده‌های تاریخی.
  - `end_date`: تاریخ پایان جمع‌آوری داده‌های تاریخی.
- **فرآیند:**
  - یک دیکشنری `data` برای ذخیره داده‌های بازیابی شده مقداردهی اولیه می‌شود.
  - یک حلقه از طریق هر نماد ارز دیجیتال تکرار می‌شود.
  - `(yf.Ticker(symbol` یک شیء برای بازیابی داده‌ها ایجاد می‌کند.

- `history()` داده‌های تاریخی روزانه را در بازه تاریخ مشخص شده بازیابی می‌کند.

- یک دیکشنری که هر کلید آن یک نماد است، و مقدار آن یک DataFrame شامل داده‌های OHLC (باز، بالا، پایین، بسته) می‌باشد.

---

۳. تعریف پارامترها

نمادها: ['BTC-USD', 'ETH-USD', 'BNB-USD', 'XRP-USD']
تاریخ شروع: '01-08-2023'
تاریخ پایان: '01-12-2024'
تاریخ تقسیم: '01-08-2024'

توضیح:

- این متغیرها دارایی‌های معاملاتی و بازه زمانی تحلیل را تعریف می‌کنند.
- `symbols`: لیستی از چهار ارز دیجیتال انتخاب‌شده.
- `start_date` و `end_date`: دوره جمع‌آوری داده‌ها را تعریف می‌کنند.
- `split_date`: بعداً برای تقسیم داده‌ها به مجموعه‌های آموزشی و تست استفاده می‌شود.

---

۴. دریافت داده‌ها

```
crypto_data = fetch_crypto_data(symbols, start_date, end_date)
```

- این تابع برای دانلود داده‌ها برای ارزهای دیجیتال مشخص‌شده در دوره تعیین‌شده فراخوانی می‌شود.
- دیکشنری `crypto_data` حاوی داده‌های تاریخی برای هر دارایی است.

---

۵. ایجاد یک DataFrame قیمت ترکیبی

تعریف تابع:

```
def create_price_df(data):
    df = pd.DataFrame()
    for symbol in symbols:
```

```
        df[symbol] = data[symbol]['Close']
    return df

price_df = create_price_df(crypto_data)
```

توضیح:

- این تابع قیمت‌های بسته شدن تنظیم‌شده برای تمام ارزهای دیجیتال را استخراج و در یک DataFrame تکی جمع‌آوری می‌کند.
- **فرآیند:**
  - یک DataFrame خالی `df` ایجاد می‌کند.
  - بر روی نمادها تکرار می‌کند و قیمت بسته شدن را از هر مجموعه داده‌ی ارز دیجیتال استخراج می‌کند.
  - DataFrame قیمت ترکیب‌شده را باز می‌گرداند.
- **نتیجه:**
  - `price_df` شامل قیمت‌های بسته شدن هر چهار ارز دیجیتال است، با تاریخ‌ها به عنوان شاخص.

---

۶. تقسیم داده‌ها به مجموعه‌های آموزشی و تست

```
train_df = price_df[price_df.index < split_date]
test_df = price_df[price_df.index >= split_date]
```

توضیح:

- `price_df` DataFrame بر اساس `split_date` تقسیم می‌شود.
  - **مجموعه آموزشی:** شامل داده‌ها قبل از 01-08-2024.
  - **مجموعه تست:** شامل داده‌ها از 01-08-2024 به بعد.
- این تفکیک به ما اجازه می‌دهد تا مدل‌ها را روی داده‌های تاریخی آموزش دهیم و عملکرد آن‌ها را روی داده‌های آینده و دیده نشده ارزیابی کنیم.

---

۷. محاسبه بازده‌های روزانه

```
returns_df = price_df.pct_change().dropna()
```

توضیح:

- بازده‌های درصدی روزانه با استفاده از تابع `pct_change()` محاسبه می‌شود.
- ردیف اول معمولاً شامل NaN است (زیرا قیمت قبلی برای مقایسه وجود ندارد)، بنابراین `dropna()` مقادیر گمشده را حذف می‌کند.
- `returns_df` تغییرات درصدی روزانه را برای هر ارز دیجیتال ارائه می‌دهد.

---

۸. تجسم داده‌های ابتدایی

```
plt.figure(figsize=(15, 8))
for symbol in symbols:
    plt.plot(price_df.index, price_df[symbol]/price_df[symbol].iloc[0],
label=symbol)
plt.title('Normalized Price Evolution')
plt.xlabel('Date')
plt.ylabel('Normalized Price')
plt.legend()
plt.grid(True)
plt.show()
```

توضیح:

- تجسم روند قیمت‌ها در طول زمان ایجاد می‌کند.
- قیمت‌ها نرمال‌سازی می‌شوند (تقسیم بر اولین مقدار)، که به مقایسه دارایی‌ها با مقیاس قیمت متفاوت کمک می‌کند.
- عناصر کلیدی در نمودار:
  - `figsize=(15,8)`: اندازه شکل را تعیین می‌کند.
  - یک حلقه هر ارز دیجیتالی را با تکامل قیمت نرمال‌سازی شده رسم می‌کند.
  - `title`, `xlabel`, و `ylabel` برچسب‌های توصیفی اضافه می‌کنند.
  - شبکه و legend خوانایی را بهبود می‌بخشند.
- **هدف:**
  - بینش‌هایی در مورد روندهای عملکرد نسبی را فراهم می‌کند.

---

۹. نمایش آمار ابتدایی

```
print("\nBasic Statistics of Daily Returns:")
print(returns_df.describe())
```

توضیح:

- `describe()` آمار خلاصه کلیدی برای بازده‌های روزانه را ارائه می‌دهد:
  - **Mean**: بازده متوسط.
  - **Standard Deviation**: نوسانات را اندازه‌گیری می‌کند.
  - **Min/Max**: حداکثر و حداقل در بازده‌های روزانه.
  - **Percentiles 25th/50th/75th**: توزیع بازده‌ها.

---

۱۰. بررسی برای مقادیر گمشده

```
missing_data = price_df.isnull().sum()
print()
if (not missing_data.any()):
    print("there is no missing data")
else:
    print("\nMissing Values:")
    print(missing_data)
```

توضیح:

- برای مقادیر گمشده در داده‌های قیمت بررسی می‌کند.
- `isnull().sum()` تعداد ورودی‌های گمشده در هر دارایی را محاسبه می‌کند.
- بررسی شرطی:
  - اگر مقادیر گمشده‌ای یافت نشود، تأییدیه چاپ می‌شود.
  - در غیر این صورت، جزئیات داده‌های گمشده نمایش داده می‌شود.

خلاصه نکات کلیدی بخش ۱:

۱. **جمع‌آوری داده‌ها:**

- استفاده از Yahoo Finance برای بازیابی داده‌های ارزهای دیجیتال.

۲. **آماده‌سازی داده‌ها:**

- استخراج قیمت‌های بسته شدن تنظیم‌شده و تقسیم مجموعه داده‌ها.

۳. **تحلیل اکتشافی داده‌ها (EDA):**

- تجسم روند قیمت نرمال‌سازی شده و تحلیل آمار بازده‌های ابتدایی.

۴. **بررسی کیفیت داده‌ها:**

- اطمینان از کامل بودن داده‌ها از طریق بررسی مقادیر گمشده.

بهبودها و ملاحظات پیشنهادی:

- افزودن مدیریت خطا هنگام بازیابی داده‌ها برای مقابله با مشکلات احتمالی API.
- معرفی بازده‌های لگاریتمی به جای بازده‌های ساده برای مدل‌سازی مالی.
- انجام پیش‌پردازش‌های اضافی مانند شناسایی مقادیر پرت.
- ذخیره داده‌های جمع‌آوری‌شده در دیسک برای استفاده مجدد بدون نیاز به فراخوانی مکرر API.

بررسی بخش ۲: پیش‌بینی نوسان
هدف این بخش تخمین نوسانات قیمت ارزهای دیجیتال با استفاده از روش‌های مختلف است. هدف تحلیل و مقایسه مدل‌های مختلف نوسان برای ارزیابی کارایی آن‌ها در پیش‌بینی تغییرات قیمت می‌باشد.

اجزای اصلی پیاده‌سازی:
۱. **روش‌های تخمین نوسان:**

- پروکسی‌های نوسان: Yang-Zhang، Garman-Klass، Parkinson، Historical.
- مدل‌های خانواده GARCH: GARCH، EGARCH، FIGARCH.

۲. **برآورد با پنجره متحرک:**

- محاسبه نوسان با استفاده از پنجره‌های ۷ روزه و ۳۰ روزه.

۳. **تجسم و تحلیل همبستگی:**

- مقایسه مقادیر نوسان مختلف از طریق نمودارها و ماتریس‌های همبستگی.

توضیح مرحله به مرحله کد:

۱. وارد کردن کتابخانه‌ها و مدیریت هشدارها

```
import arch
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

- `arch`: ابزارهایی برای تخمین مدل‌های ARCH، GARCH، EGARCH، و FIGARCH ارائه می‌دهد.
- `scipy.stats`: برای محاسبات آماری استفاده می‌شود.
- سرکوب هشدارها مانع از شلوغ شدن خروجی با هشدارهای جزئی مانند هشدارهای همگرایی می‌شود.

---

۲. تعریف کلاس VolatilityEstimator

این کلاس روش‌های مختلفی برای محاسبه نوسان با استفاده از مدل‌های مختلف را ارائه می‌دهد.

```
class VolatilityEstimator:
    def __init__(self, prices_df):
        self.prices = prices_df
        self.returns = prices_df.pct_change().dropna()
```

- **ورودی‌ها:**
  - `prices_df`: DataFrame شامل قیمت‌های بسته شدن ارزهای دیجیتال.
- **ویژگی‌ها:**
  - `self.prices`: ذخیره داده‌های خام قیمت.
  - `self.returns`: محاسبه بازده‌های روزانه برای پردازش‌های بعدی.

---

۳. محاسبات پروکسی نوسان

الف. نوسان تاریخی

```
def calculate_historical_volatility(self, window):
    return self.returns.rolling(window=window).std() * np.sqrt(252)
```

- انحراف معیار بازده‌ها در پنجره متحرک تعیین‌شده را محاسبه می‌کند.
- نوسان را با ضرب در ۲۵۲√ سالانه‌سازی می‌کند (با فرض ۲۵۲ روز معاملاتی در سال).

ب. نوسان پارکینسون

```
def calculate_parkinson_volatility(self, window):
    high = pd.DataFrame()
    low = pd.DataFrame()
    for symbol in self.prices.columns:
        high[symbol] = self.prices[symbol].rolling(window).max()
        low[symbol] = self.prices[symbol].rolling(window).min()
    k = 1 / (4 * np.log(2))
    return np.sqrt(k * (np.log(high/low)**2).rolling(window).mean() * 252)
```

- از محدوده قیمت بالا-پایین برای تخمین نوسان استفاده می‌کند که ممکن است دقیق‌تر از نوسان تاریخی ساده باشد.
- ثابت k برای تنظیم ویژگی‌های توزیعی محدوده بالا-پایین استفاده می‌شود.

ج. نوسان گارمن-کلاس

```
def calculate_garman_klass_volatility(self, window):
    log_hl = (self.prices.rolling(window).max() /
              self.prices.rolling(window).min()).apply(np.log)
    return np.sqrt(0.5 * log_hl**2 * 252)
```

- پارامترهای باز و بسته شدن قیمت‌ها را به اندازه پارکینسون اضافه می‌کند و دقت را بهبود می‌بخشد.

د. نوسان یانگ-ژانگ

```
def calculate_yang_zhang_volatility(self, window):
    returns = self.returns
    open_close = returns.rolling(window).std() * np.sqrt(252)
```

```
    high_low = self.calculate_parkinson_volatility(window)
    k = 0.34 / (1.34 + (window + 1) / (window - 1))
    return np.sqrt(open_close**2 + k * high_low**2)
```

- نوسان باز-بسته و بالا-پایین را ترکیب می‌کند و تخمینی قوی‌تر ارائه می‌دهد.

---

۴. تخمین‌های خانواده GARCH

الف. GARCH(1,1)

```
def calculate_garch_volatility(self, window):
    volatility = pd.DataFrame()
    for symbol in self.returns.columns:
        returns_series = self.returns[symbol].dropna()
        model = arch.arch_model(returns_series, vol='Garch', p=1, q=1)
```

- نوسان را با مدل GARCH(1,1) پرکاربرد تخمین می‌زند.

ب. EGARCH(1,1)

```
def calculate_egarch_volatility(self, window):
    model = arch.arch_model(returns_series, vol='EGARCH', p=1, q=1)
```

- مدل GARCH نمایی (EGARCH) اثرات نامتقارن در خوشه‌بندی نوسان را مدل می‌کند.

ج. FIGARCH(2,2)

```
def calculate_figarch_volatility(self, window):
    model = arch.arch_model(returns_series, vol='Garch', p=2, q=2)
```

- مدل FIGARCH اثرات حافظه بلندمدت در نوسان را ثبت می‌کند.

---

۵. اجرای تمام محاسبات نوسان

```
def calculate_all_volatilities(prices_df, windows=[7, 30]):
    vol_estimator = VolatilityEstimator(prices_df)
    volatility_results = {}
    for window in windows:
        volatility_results[f'window_{window}'] = {
            'historical':
vol_estimator.calculate_historical_volatility(window),
            'parkinson':
vol_estimator.calculate_parkinson_volatility(window),
            'garman_klass':
vol_estimator.calculate_garman_klass_volatility(window),
            'yang_zhang':
vol_estimator.calculate_yang_zhang_volatility(window),
            'garch': vol_estimator.calculate_garch_volatility(window),
            'egarch': vol_estimator.calculate_egarch_volatility(window),
            'figarch': vol_estimator.calculate_figarch_volatility(window)
        }
    return volatility_results
```

- تمام مقادیر نوسان را برای هر پنجره (۷ و ۳۰ روزه) محاسبه می‌کند و در یک دیکشنری تو در تو ذخیره می‌کند.

---

۶. تجسم نتایج نوسان

```
def plot_volatility_comparison(volatility_results, symbol, window):
    plt.figure(figsize=(15, 8))
    methods = ['historical', 'parkinson', 'garman_klass', 'yang_zhang',
               'garch', 'egarch', 'figarch']
    for method in methods:
        vol = volatility_results[f'window_{window}'][method][symbol]
        plt.plot(vol.index, vol, label=method.capitalize(), alpha=0.7)
    plt.title(f'{symbol} Volatility Estimates ({window}-day window)')
    plt.xlabel('Date')
    plt.ylabel('Annualized Volatility')
    plt.legend()
    plt.grid(True)
    plt.show()
```

- این تابع نمودار تخمین‌های نوسان را برای هر ارز دیجیتال ترسیم می‌کند.

---

۷. تحلیل همبستگی

```
def analyze_volatility_correlations(volatility_results, symbol, window):
    vol_data = pd.DataFrame()
    for method in methods:
        vol_data[method.capitalize()] =
volatility_results[f'window_{window}'][method][symbol]
    sns.heatmap(vol_data.corr(), annot=True, cmap='coolwarm', center=0)
```

- همبستگی تخمین‌های نوسان مختلف را برای هر ارز دیجیتال مقایسه می‌کند.

نکات کلیدی و بهبودها:

۱. **مزایا:**

- تکنیک‌های جامع تخمین نوسان.
- تجسم قوی و تحلیل همبستگی.

۲. **بهبودهای پیشنهادی:**

- بهینه‌سازی انتخاب مدل GARCH برای کارایی بیشتر.
- افزودن اعتبارسنجی متقابل برای دقت بیشتر مدل.
- ذخیره نتایج برای جلوگیری از محاسبات تکراری.

بررسی مدل بلک-لیترمن

مدل بلک-لیترمن یک رویکرد پیشرفته برای بهینه‌سازی سبد سرمایه است که داده‌های تاریخی بازار را با دیدگاه‌های سرمایه‌گذاران ترکیب می‌کند تا تخصیص بهینه دارایی‌ها را تعیین کند. این مدل با گنجاندن دیدگاه‌های ذهنی سرمایه‌گذاران در یک چارچوب ریاضی سازگار، بهینه‌سازی میانگین-واریانس سنتی را ارتقا می‌دهد.

اجزای کلیدی مدل بلک-لیترمن:

۱. **بازده‌های تعادلی بازار:**

- فرض می‌کند که قیمت‌های بازار تعادلی بین عرضه و تقاضا را منعکس می‌کنند.
- بازده‌های تعادلی بازار بر اساس ریسک‌گریزی بازار و داده‌های تاریخی محاسبه می‌شوند.

۲. **دیدگاه‌های سرمایه‌گذاران:**

- به سرمایه‌گذاران اجازه می‌دهد نظرات ذهنی خود را در مورد بازده دارایی‌ها (مانند بازده مورد انتظار بر اساس نوسان) معرفی کنند.
- این دیدگاه‌ها با استفاده از ماتریس‌ها بیان می‌شوند تا بر تخصیص نهایی تأثیر بگذارند.

۳. **تنظیم بیزی:**

- بازده‌های تعادلی بازار را با دیدگاه‌های سرمایه‌گذاران با استفاده از استنتاج بیزی ترکیب می‌کند تا مجموعه‌ای جدید از بازده‌های پسین تولید کند.

تجزیه و تحلیل کد:

گام ۱: آماده‌سازی تخمین‌های میانگین نوسان
کد ابتدا تخمین‌های میانگین نوسان را در دو پنجره زمانی (۷ روزه و ۳۰ روزه) محاسبه می‌کند:

```
methods = ['historical', 'parkinson', 'garman_klass', 'yang_zhang',
'garch', 'egarch', 'figarch']
means = {method: ((volatility_results['window_7'][method] +
volatility_results['window_30'][method]) / 2).astype(float).dropna()
         for method in methods}
```

- این ترکیب دیکشنری میانگین تخمین نوسان در دو پنجره زمانی را محاسبه می‌کند.
- اطمینان می‌دهد که داده‌ها تمیز هستند (dropna مقادیر گمشده را حذف می‌کند).
- نتیجه دیکشنری‌ای از نوسان دارایی‌ها برای هر روش تخمین است.

گام ۲: پیاده‌سازی بهینه‌ساز بلک-لیترمن

۲.۱. مقداردهی اولیه

```
def __init__(self, means, returns_df, risk_free_rate=0.03, tau=0.05):
    self.means = means
    self.returns = returns_df
```

```
    self.rf = risk_free_rate
    self.tau = tau
    self.n_assets = len(returns_df.columns)
```

- **ورودی‌ها:**
  - `means`: دیکشنری تخمین‌های میانگین نوسان از مدل‌های مختلف.
  - `returns_df`: بازده‌های تاریخی دارایی‌ها برای تخمین ریسک و بازده.
  - `risk_free_rate`: نرخ بدون ریسک سالانه (پیش‌فرض: ۳٪).
  - `tau`: پارامتری که سطح اطمینان در تخمین‌های قبلی را نشان می‌دهد (مقدار کمتر = اطمینان بیشتر به داده‌های بازار).
  - `n_assets`: تعداد دارایی‌ها در سبد سرمایه.

۲.۲. محاسبه وزن‌های بازار برابر

```
def calculate_equal_market_weights(self):
    return np.array([1/self.n_assets] * self.n_assets)
```

- وزن‌های برابر برای هر دارایی در سبد سرمایه بازمی‌گرداند (مثلاً اگر ۴ دارایی وجود داشته باشد، هر کدام ۲۵٪ وزن خواهند داشت).

۲.۳. محاسبه بازده‌های تعادلی (مبتنی بر CAPM)

```
def calculate_equilibrium_returns(self, market_weights, risk_aversion=2.5):
    cov_matrix = self.returns.cov() * 252  # ماتریس کوواریانس سالانه
    return risk_aversion * cov_matrix.dot(market_weights)
```

- از اصول مدل قیمت‌گذاری دارایی سرمایه (CAPM) برای تخمین بازده‌های مورد انتظار استفاده می‌کند:
  - ماتریس کوواریانس سالانه بازده‌ها.
  - فرض می‌کند سطح ریسک‌گریزی برابر با ۲.۵ است (پیش‌فرض).

۲.۴. تولید دیدگاه‌های سرمایه‌گذاران

```
def generate_views(self):
    assets = self.returns.columns
    n_assets = len(assets)
    P = np.eye(n_assets)
    Q = np.zeros(n_assets)
    vol_ranks = {method: {asset: vol[asset].mean() for asset in assets} for
method, vol in self.means.items()}
    agg_ranks = {asset: sum(vol_ranks[method][asset] for method in
vol_ranks) / len(vol_ranks) for asset in assets}
    for i, asset in enumerate(assets):
        Q[i] = agg_ranks[asset]
    return P, Q
```

- **P (ماتریس دیدگاه‌ها):** نشان می‌دهد که دیدگاه‌ها چگونه در سراسر دارایی‌ها اعمال می‌شوند (ماتریس هویتی برای دیدگاه‌های مطلق).
- **Q (بازده‌های دیدگاه‌ها):** بازده مورد انتظار بر اساس رتبه‌بندی‌های نوسان محاسبه شده را نشان می‌دهد.

۲.۵. بهینه‌سازی وزن‌های سبد سرمایه

```
def optimize_weights(self, min_weight=0.05, max_weight=0.8):
    # فرمول بلک-لیترمن
    constraints = [
        {'type': 'eq', 'fun': lambda x: np.sum(x) - 1},
        {'type': 'ineq', 'fun': lambda x: x - min_weight},
        {'type': 'ineq', 'fun': lambda x: max_weight - x}
    ]
    result = sco.minimize(neg_sharpe, initial_weights, method='SLSQP',
bounds=bounds, constraints=constraints)
    optimal_weights = pd.Series(result.x, index=self.returns.columns)
    return optimal_weights
```

مراحل کلیدی:

۱. محاسبه بازده‌های قبلی با استفاده از .CAPM
۲ . ترکیب دیدگاه‌ها با داده‌های قبلی با استفاده از آمار بیزی.
۳. بهینه‌سازی نسبت شارپ با محدودیت‌های وزنی.
۴. بازگرداندن وزن‌های بهینه برای سبد سرمایه.

گام ۳: اجرای بهینه‌سازی برای هر مدل نوسان

```
for method in means.keys():
    optimizer = BlackLittermanOptimizer(method_means, returns_df)
    weights, metrics = optimizer.optimize_weights()
    weights_dict[method] = weights
    metrics_dict[method] = metrics
```

- بهینه‌سازی بلک-لیترمن را برای هر روش تخمین نوسان جداگانه اجرا می‌کند.
- وزن‌ها و معیارهای عملکرد حاصل را ذخیره می‌کند.

گام ۴: تجسم و گزارش‌دهی

نمایش وزن‌های بهینه:

```
def plot_optimal_weights():
    weights_df = pd.DataFrame(weights_dict)
    weights_df.plot(kind='bar', figsize=(12, 6), width=0.8)
    plt.title('Optimal Portfolio Weights by Volatility Method')
    plt.show()
```

- نمایش تأثیر هر روش تخمین نوسان بر تخصیص سبد سرمایه.

گام ۵: بهینه‌سازی ترکیبی سبد سرمایه:

```
combined_optimizer = BlackLittermanOptimizer(means, returns_df)
combined_weights, combined_metrics = combined_optimizer.optimize_weights()
```

- تمام تخمین‌های نوسان را برای ایجاد یک سبد سرمایه بهینه نهایی ترکیب می‌کند.

نتیجه‌گیری:

این کد یک رویکرد جامع برای بهینه‌سازی سبد سرمایه با استفاده از مدل بلک-لیترمن و تکنیک‌های مختلف

تخمین نوسان اعمال می‌کند. این روش تنوع سبد سرمایه را تضمین کرده و نسبت شارپ را حداکثر می‌کند و در عین حال دیدگاه‌های سرمایه‌گذاران و داده‌های بازار را متعادل می‌کند.

بررسی استراتژی خرید و نگهداری (Buy-and-Hold)

۱. مرور کلی استراتژی خرید و نگهداری

استراتژی خرید و نگهداری یک روش سرمایه‌گذاری منفعلانه است که در آن سرمایه‌گذار یک سبد دارایی خریداری کرده و آن را برای مدت طولانی نگه می‌دارد، بدون توجه به نوسانات کوتاه‌مدت بازار. این استراتژی بر اساس معیارهای عملکرد کلیدی مانند نسبت شارپ، سود خالص، و افت سرمایه ارزیابی می‌شود.

---

۲. تجزیه و تحلیل کد

گام ۱: کلاس BuyAndHoldStrategy

۱.۱. مقداردهی اولیه

```
def __init__(self, prices, weights, initial_capital=1000,
transaction_cost=0.02):
```

- **ورودی‌ها:**
  - `prices`: داده‌های تاریخی قیمت دارایی‌ها.
  - `weights`: وزن‌های تخصیص سبد سرمایه (بر اساس بهینه‌سازی در مراحل قبل).
  - `initial_capital`: سرمایه اولیه (پیش‌فرض: ۱۰۰۰ دلار).
  - `transaction_cost`: هزینه تراکنش برای خرید دارایی‌ها (پیش‌فرض: ۲٪).
- **هدف:**
  تخصیص سرمایه اولیه به طور متناسب با وزن‌های سبد و در نظر گرفتن هزینه تراکنش‌ها.

۱.۲. اجرای استراتژی

```
def run_strategy(self):
```

تخصیص اولیه سبد سرمایه:

```
 initial_positions = self.initial_capital * self.weights
initial_costs = np.sum(initial_positions) * self.transaction_cost
positions = initial_positions * (1 - self.transaction_cost)
```

- سرمایه بر اساس وزن‌های تخصیص‌یافته میان دارایی‌ها توزیع می‌شود.
- هزینه تراکنش از سرمایه اولیه کسر می‌شود.

محاسبه ارزش سبد سرمایه:

```
 portfolio_values = pd.Series(index=self.prices.index)
for date in self.prices.index:
    current_prices = self.prices.loc[date]
    portfolio_values[date] = np.sum(positions * current_prices /
self.prices.iloc[0])
```

- ارزش سبد سرمایه از ضرب دارایی‌ها در قیمت‌های روزانه آن‌ها محاسبه می‌شود.

محاسبه معیارهای عملکرد:

```
 daily_returns = portfolio_values.pct_change().dropna()
metrics = {
    'Sharpe Ratio': self.calculate_sharpe_ratio(daily_returns),
    'Net Profit': portfolio_values[-1] - self.initial_capital,
    'Net Profit (%)': ((portfolio_values[-1] / self.initial_capital) - 1) *
100,
    'Max Drawdown': self.calculate_max_drawdown(portfolio_values),
    'Max Drawdown (%)': self.calculate_max_drawdown(portfolio_values) *
100,
    'Final Value': portfolio_values[-1]
}
```

- **نسبت شارپ:** اندازه‌گیری بازده تعدیل‌شده نسبت به ریسک.
- **سود خالص:** تفاوت بین ارزش نهایی و سرمایه اولیه.
- **افت سرمایه:** بیشترین کاهش ارزش سبد از اوج تا کف.

۱.۳. محاسبه نسبت شارپ

```
def calculate_sharpe_ratio(self, returns, rf=0.03):
    excess_returns = returns - rf/252
    return np.sqrt(252) * excess_returns.mean() / returns.std()
```

- بازده‌ها با در نظر گرفتن نرخ بدون ریسک سالانه (۳٪) تنظیم می‌شوند.
- نسبت شارپ سالانه‌سازی می‌شود.

۱.۴. محاسبه افت سرمایه (Max Drawdown)

```
def calculate_max_drawdown(self, portfolio_values):
    rolling_max = portfolio_values.expanding().max()
    drawdowns = portfolio_values / rolling_max - 1
    return drawdowns.min()
```

- بیشترین کاهش ارزش سبد نسبت به حداکثر مقدار آن محاسبه می‌شود.

گام ۲: ارزیابی استراتژی‌ها

ارزیابی بر روی داده‌های آموزشی و تست

```
def evaluate_strategies(train_prices, test_prices, weights_dict):
    strategy = BuyAndHoldStrategy(train_prices, weights)
    metrics, values = strategy.run_strategy()
```

- استراتژی‌ها به طور جداگانه بر روی داده‌های آموزشی (۸۰٪) و تست (۲۰٪) ارزیابی می‌شوند.
- نتایج شامل:
  - **معیارهای عملکرد:** مانند نسبت شارپ، سود خالص، و افت سرمایه.
  - **ارزش‌های سبد:** سری زمانی رشد سبد سرمایه.

گام ۳: تجسم عملکرد

مقایسه ارزش سبد سرمایه

```
def plot_performance_comparison(results, title):
```

- محور x: زمان (تاریخ‌ها).
- محور y: ارزش سبد سرمایه.
- legend: روش‌های مختلف تخمین نوسان.

خلاصه‌سازی عملکرد

```
def create_summary_table(results):
```

- جدولی شامل معیارهای کلیدی هر استراتژی تولید می‌کند.
- نتایج بر اساس نسبت شارپ مرتب می‌شوند.

گام ۴: تجسم توازن ریسک-بازده

نمودار پراکندگی ریسک-بازده

```
def plot_risk_return_scatter(results, title):
```

- محور x: افت سرمایه حداکثر (%)، نشان‌دهنده ریسک.
- محور y: سود خالص (%)، نشان‌دهنده بازده.
- هر نقطه: یک استراتژی.
- بینشی درباره توازن بین بازده بالقوه و مواجهه با ریسک فراهم می‌کند.

گام ۵: تحلیل همبستگی

```
def calculate_strategy_correlations(results):
```

- درصد تغییر روزانه هر استراتژی محاسبه می‌شود.
- نقشه حرارتی (Heatmap) برای نمایش همبستگی‌ها تولید می‌شود.

گام ۶: اجرای تحلیل‌ها

تقسیم داده‌ها به آموزش و تست:

```
strategy_results = evaluate_strategies(price_df[:int(len(price_df)*0.8)],
price_df[int(len(price_df)*0.8)::], weights_dict)
```

مقایسه عملکرد:

```
plot_performance_comparison(strategy_results['train'], 'Training Set')
plot_performance_comparison(strategy_results['test'], 'Test Set')
```

نمایش نتایج کلیدی:

```
print(create_summary_table(strategy_results['train']).round(2))
print(create_summary_table(strategy_results['test']).round(2))
```

---

۳. تفسیر نتایج

معیارهای عملکرد:

- **نسبت شارپ بالاتر = بازده تعدیل‌شده بهتر.**
- **افت سرمایه کمتر = ریسک پایین‌تر.**
- **ارزش نهایی بالاتر = رشد کلی بهتر.**

مقایسه مدل‌های نوسان:

- برخی مدل‌ها (مانند GARCH) ممکن است عملکرد بهتری نسبت به دیگران داشته باشند.
- نتایج ممکن است بین مجموعه‌های آموزشی و تست متفاوت باشد (به دلیل بیش‌برازش).

توازن ریسک-بازده:

- به سرمایه‌گذاران کمک می‌کند تا تعادلی بین بازده بالقوه و مواجهه با ریسک پیدا کنند.

بینش‌های همبستگی:

- استراتژی‌هایی با همبستگی پایین می‌توانند برای کاهش ریسک سبد ترکیب شوند.

---

۴. نتیجه‌گیری

این کد با موفقیت ارزیابی استراتژی خرید و نگهداری را پیاده‌سازی می‌کند:

- **ارزیابی عملکرد سبد سرمایه** با استفاده از استراتژی‌های مختلف مبتنی بر نوسان.
- **تجسم داده‌های کلیدی** برای درک بهتر عملکرد.
- **تولید معیارهای کلیدی عملکرد** برای تصمیم‌گیری.
- **تحلیل ریسک و همبستگی** برای آگاهی‌بخشی به تنوع سبد سرمایه.