پروژه پایانی درس الگوریتم‌های معاملاتی

فرزان مسیبی، کورش تقی‌پور پاسدار

۸ بهمن ۱۴۰۳

فهرست مطالب

۱ جمع‌آوری مجموعه دادگان **۳**

۲ تخمین نوسان (Volatility Prediction) **۴**
۱.۲ پیش‌بینی نوسان با استفاده از مدل‌های آماری ۴
۲.۲ پروکسی‌های نوسان ۴

۳ روش وزن‌دهی و بهینه‌سازی پورتفولیو (Portfolio Optimization) **۶**
۱.۳ ادغام سری‌های نوسان ۶
۲.۳ بهینه‌سازی پورتفولیو برای هر دسته ۷
۱.۲.۳ Statistical Models ۷
۲.۲.۳ Proxies ۸

۴ استراتژی و استفاده از پورتفولیو **۹**

۵ تحلیل نتایج **۱۰**

۱ جمع‌آوری مجموعه دادگان

بر طبق مساله طرح شده، برای ساخت پورتفولیو، داده‌های تاریخی ۴ رمزارز BTC-USD، ETH-USD، BNB-USD و XRP-USD در بازه زمانی ۰۱ آگوست ۲۰۲۳ تا ۰۱ دسامبر ۲۰۲۴ در بازه‌های زمانی روزانه دریافت شدند و همچنین داده‌های Null نیز حذف شدند. این داده‌ها به دو بخش آموزش و تست تقسیم شدند.

بخش آمورش از تاریخ ۰۱ آگوست ۲۰۲۳ تا ۰۱ آگوست ۲۰۲۴

بخش تست از تاریخ ۰۱ آگوست ۲۰۲۴ تا ۰۱ دسامبر ۲۰۲۴

```python
coins = ("BTC-USD", "ETH-USD", "BNB-USD", "XRP-USD")

train_start = "2023-08-01"
train_end = "2024-08-01"
test_start = "2024-08-01"
test_end = "2024-12-01"

start_date = "2023-08-01"
end_date = "2024-12-01"
```

✓ 0.0s

```python
def get_coins_data(coins, start, end):
    data = yf.download(coins, start=start, end=end, interval="1d")
    data = data.dropna()
    return data
```

✓ 0.0s

```python
data = get_coins_data(coins, start_date, end_date)
data
```

✓ 0.4s

شکل ۱: دریافت داده‌ها

داده‌های دریافتی چهار رمزارز بصورت روزانه، و شامل داده‌های Open، Close، High و Low هستند.

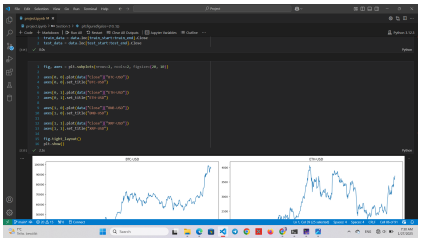
```python
data = get_coins_data(coins, start_date, end_date)
data
```

✓ 0.4s Python

```
[*********************100%***********************]  4 of 4 completed
```

Price	Adj Close				Close				High		...	Low		Open		
Ticker	BNB-USD	BTC-USD	ETH-USD	XRP-USD	BNB-USD	BTC-USD	ETH-USD	XRP-USD	BNB-USD	BTC-USD	...	ETH-USD	XRP-USD	BNB-USD	BTC-USD	ETI
Date																
2023-08-01 00:00:00+00:00	247.505020	29675.732422	1871.792114	0.707211	247.505020	29675.732422	1871.792114	0.707211	247.505020	29675.732422	...	1817.920776	0.683532	241.128052	29230.873047	18
2023-08-02 00:00:00+00:00	240.494644	29151.958984	1839.089722	0.685195	240.494644	29151.958984	1839.089722	0.685195	248.511551	29987.998047	...	1822.348267	0.680232	247.490311	29704.146484	18
2023-08-03 00:00:00+00:00	241.061523	29178.679688	1835.136475	0.664135	241.061523	29178.679688	1835.136475	0.664135	241.583252	29375.707031	...	1825.347778	0.658851	240.509979	29161.812500	18

همچنین داده‌ها را با استفاده از کتابخانه Matplotlib رسم می‌کنیم.



(آ) کد مربوط به ترسیم داده‌ها

(ب) نمودار رمزارزها

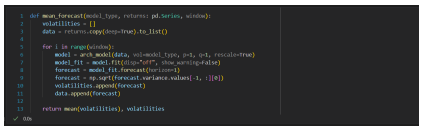
۲ تخمین نوسان (Volatility Prediction)

برای تخمین نوسان، دو راه معرفی شده است که مراحل طی شده برای هر دو توضیح داده می‌شوند.

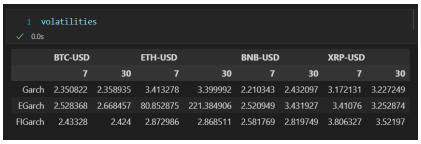
۱.۲ پیش‌بینی نوسان با استفاده از مدل‌های آماری

در این روش، با استفاده از مدل‌های آماری GARCH، EGARCH و FIGARCH نوسان را برای دو پنجره ذکر شده روی داده‌های آموزش پیش‌بینی می‌کنیم.

(ج) پیش‌بینی نوسان برای دو پنجره ذکر شده.



(د) تابعی برای محاسبه میانگین نوسان در هر پنجره

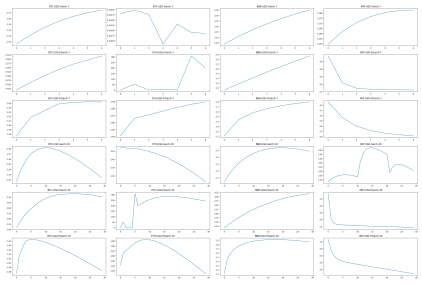


volatilities

0.0s

	BTC-USD		ETH-USD		BNB-USD		XRP-USD	
	7	30	7	30	7	30	7	30
Garch	2.350822	2.358935	3.413278	3.399992	2.210343	2.432097	3.172131	3.227249
EGarch	2.528368	2.668457	80.852875	221.384906	2.520949	3.431927	3.41076	3.252874
FIGarch	2.43328	2.424	2.872986	2.868511	2.581769	2.819749	3.806327	3.52197

(ه) میانگین نوسانات پیش‌بینی شده



(و) نمودارهای نوسانات پیش‌بینی شده

۲.۲ پروکسی‌های نوسان

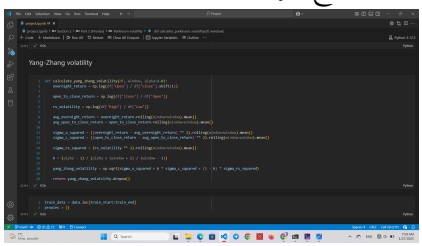
در ابتدا به پیاده‌سازی هر یک از پروکسی‌ها می‌پردازیم.

```python
def calculate_parkinson_volatility(df, window):
    log_range_squared = np.log(df['High'] / df['Low']) ** 2
    rolling_sum = log_range_squared.rolling(window=window).sum()
    ln_factor = 4 * window * np.log(2)
    parkinson_volatility = np.sqrt(rolling_sum / ln_factor)

    return parkinson_volatility.dropna()
```

Parkinson Volatility (ح)

```python
def calculate_historical_volatility(df, window):
    returns = df['Close'].pct_change()

    mean_returns = returns.rolling(window=window).mean()
    squared_deviations = (returns - mean_returns) ** 2
    rolling_sum = squared_deviations.rolling(window=window).sum()
    volatility = np.sqrt(rolling_sum / (window - 1))

    return volatility.dropna()
```

Historical Volatility (ز)

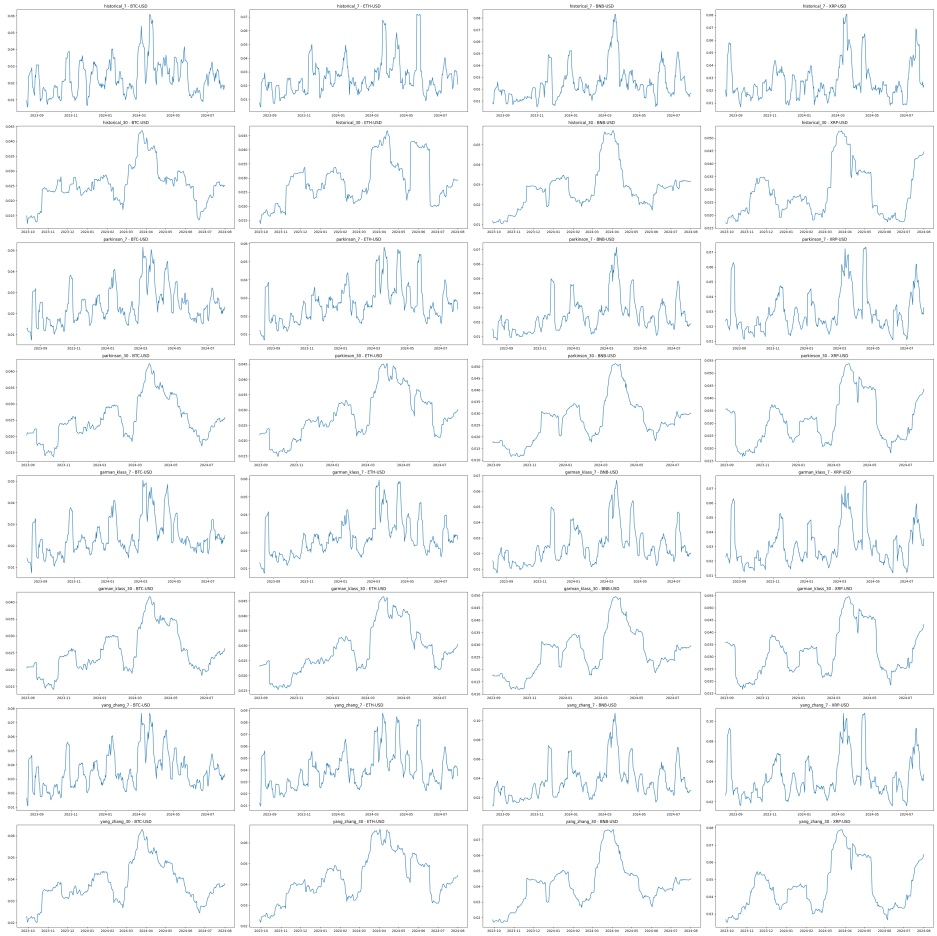


Yang-Zhang Volatility (ی)

```python
def calculate_garman_klass_volatility(df, window):
    high_low_term = np.log(df['High'] / df['Low']) ** 2
    close_open_term = np.log(df['Close'] / df['Open']) ** 2

    first_sum = high_low_term.rolling(window=window).sum() / (2 * window)
    second_sum = close_open_term.rolling(window=window).sum() * (2 * np.log(2) - 1) / window

    volatility = np.sqrt(first_sum - second_sum)

    return volatility.dropna()
```

Garman-Klass Volatility (ط)

سپس به محاسبه تخمین نوسانات دو دو پنجره زمانی ۷ و ۳۰ روزه می‌پردازیم.



شکل ۲: نوسانات تخمین زده شده برای رمزارزهای مختلف

۳ روش وزن‌دهی و بهینه‌سازی پورتفولیو (Portfolio Optimization)

۱.۳ ادغام سری‌های نوسان

در این بخش قصد داریم تا میانگین سری‌ها را محاسبه کنیم.

```
mean_volatilities = volatilities.T.groupby(level=0).mean().T
print(mean_volatilities)
```

✓ 0.0s

```
          BNB-USD   BTC-USD     ETH-USD   XRP-USD
Garch     2.32122  2.354879    3.406635   3.19969
EGarch   2.976438  2.598413  151.118891  3.331817
FIGarch  2.700759   2.42864    2.870749  3.664149
```

(آ) Statistical Models

(ج) نمودارهای میانگین‌های سری نوسانات برای Proxies

```
historical_mean = (proxies["parkinson_30"] + proxies["parkinson_7"]) / 2
mean_proxies["historical"] = historical_mean
```

✓ 0.0s

```
parkinson_mean = (proxies["parkinson_30"] + proxies["parkinson_7"]) / 2
mean_proxies["parkinson"] = parkinson_mean
```

✓ 0.0s

```
garman_klass_mean = (proxies["garman_klass_30"] + proxies["garman_klass_7"]) / 2
mean_proxies["garman_klass"] = garman_klass_mean
```

✓ 0.0s

```
yang_zhang_mean = (proxies["yang_zhang_30"] + proxies["yang_zhang_7"]) / 2
mean_proxies["yang_zhang"] = yang_zhang_mean
```

✓ 0.0s

(ب) Proxies

۲.۳ بهینه‌سازی پورتفولیو برای هر دسته

حال به بهینه‌سازی پورتفولیو با توجه به نوسانات می‌پردازیم.

۱.۲.۳ Statistical Models

در ابتدا Market Cap را بدست می‌آوریم.

```
market_caps = pd.Series()

for coin in coins:
    stock = yf.Ticker(coin)
    market_cap = stock.info["marketCap"]
    market_caps.loc[coin] = market_cap
print(market_caps)
```

✓ 1.2s

```
BTC-USD    2038155706368
ETH-USD     390719635456
BNB-USD      94889828352
XRP-USD     174795177984
dtype: int64
```

سپس به محاسبه کواریانس مقادیر و risk-aversion می‌پردازیم تا وزن‌های اولیه‌ی پورتفولیو بدست آیند. این وزن‌ها سپس توسط مدل تغییر خواهند کرد.

```python
cov_matrix = risk_models.sample_cov(data.loc[train_start:train_end].Close)
delta = black_litterman.market_implied_risk_aversion(data.loc[train_start:train_end].Close)
market_prior = black_litterman.market_implied_prior_returns(
    market_caps, delta, cov_matrix
)

print("Initial Market Prior:\n", market_prior)
```

✓ 0.0s

```
Initial Market Prior:
 Ticker
BNB-USD    0.364445
BTC-USD    0.595768
ETH-USD    0.343019
XRP-USD    0.000935
dtype: float64
```

حال به محاسبه وزن‌های بهینه‌شده توسط مدل می‌پردازیم. این وزن‌ها براساس نمودار Efficient Frontier و برای متریک Max Sharpe Ratio بدست آمده اند.

```python
weights_dict = {}

for model in mean_volatilities.index:
    bl = BlackLittermanModel(
        cov_matrix,
        absolute_views=mean_volatilities.loc[model],
        pi=market_prior,
    )

    bl_returns = bl.bl_returns()
    ef = EfficientFrontier(bl_returns, cov_matrix)
    ef.add_constraint(lambda w: w >= 0.1)
    ef.add_constraint(lambda w: w <= 0.5)
    weights = ef.max_sharpe()
    weights = ef.clean_weights()

    bl_weights = pd.DataFrame(
        list(weights.items()), columns=["symbol", "weight"]
    ).set_index("symbol")

    print("\n", model)
    performance = ef.portfolio_performance(verbose=True)
    weights_dict[model] = weights
```

✓ 0.0s

```python
# Print optimized weights
statistical_weights = pd.DataFrame(weights_dict).T
print(statistical_weights)
```

✓ 0.0s

```
         BNB-USD  BTC-USD  ETH-USD  XRP-USD
Garch    0.15995  0.48228  0.21322  0.14456
EGarch   0.10000  0.30000  0.50000  0.10000
FIGarch  0.22170  0.46679  0.10000  0.21150
```

شکل ۳: وزن‌های نهایی بدست آمده برای مدل‌های آماری

۲.۲.۳ Proxies

برای پروکسی‌ها هم مشابه مدل‌های آماری، به محاسبه Market Cap، Covariance Matrix و Risk-Aversion می‌پردازیم تا وزن‌های اولیه بدست آیند. سپس به محاسبه وزن‌های بهینه می‌پردازیم.

```python
print(80 * '=')
print(f"\n{model} Model:")
mu = mean_historical_return(mean_proxies[model])
S = CovarianceShrinkage(mean_proxies[model]).ledoit_wolf()

num_assets = len(mean_proxies[model].columns)
random_weights = np.random.random(num_assets)
random_weights /= random_weights.sum()

print("\nRandom Weights (Initial Guess):")
for asset, weight in zip(mean_proxies[model].columns, random_weights):
    print(f"{asset}: {weight:.2%}")

random_portfolio_return = np.dot(random_weights, mu)
random_portfolio_volatility = np.sqrt(np.dot(random_weights.T, np.dot(S, random_weights)))
random_sharpe_ratio = (random_portfolio_return - 0.01) / random_portfolio_volatility

print(f"\nRandom Portfolio Return: {random_portfolio_return:.2%}")
print(f"Random Portfolio Volatility: {random_portfolio_volatility:.2%}")
print(f"Random Portfolio Sharpe Ratio: {random_sharpe_ratio:.2f}")


market_weights = random_weights.copy()
market_prior = S @ market_weights

historical_returns = train_data.pct_change() * 100
views = historical_returns.mean()
view_confidences = np.diag(1/ historical_returns.var())

bl = BlackLittermanModel(S, pi=market_prior, absolute_views=views, omega=view_confidences)
bl_returns = bl.bl_returns()
bl_cov_matrix = bl.bl_cov()

ef = EfficientFrontier(bl_returns, bl_cov_matrix)
optimized_weights = ef.max_sharpe(risk_free_rate=0.01)
```

```python
weights_df = pd.DataFrame(weights_from_proxies)
weights_df
```

✓ 0.0s

	historical	parkinson	garman_klass	yang_zhang
BNB-USD	0.56371	0.41119	0.54706	0.25078
BTC-USD	0.26667	0.34394	0.21560	0.36937
ETH-USD	0.02864	0.19878	0.23734	0.37985
XRP-USD	0.14098	0.04609	0.00000	0.00000

(ب) وزن‌های محاسبه شده

```
==========================================================================

historical Model:

Random Weights (Initial Guess):
BNB-USD: 43.16%
BTC-USD: 19.80%
ETH-USD: 8.92%
XRP-USD: 28.12%

Random Portfolio Return: 22.70%
Random Portfolio Volatility: 96.93%
Random Portfolio Sharpe Ratio: 0.22
Expected annual return: 51.9%
Annual volatility: 99.8%
Sharpe Ratio: 0.51

Optimized Weights (Black-Litterman):
BNB-USD: 56.37%
BTC-USD: 26.67%
ETH-USD: 2.86%
XRP-USD: 14.10%
==========================================================================
```

(آ) نمونه‌ای از یک خروجی برای Historical Proxy

۴ استراتژی و استفاده از پورتفولیو

در این بخش، در ابتدا به پیاده‌سازی استراتژی و متریک‌های ذکرشده می‌پردازیم.

```python
def calculate_daily_returns(portfolio_values):
  return portfolio_values.pct_change().dropna()

def calculate_maximum_drawdown(daily_returns):
  cumulative_returns = (daily_returns + 1).cumprod()
  peak = cumulative_returns.cummax()
  drawdown = (cumulative_returns - peak) / peak
  max_drawdown = drawdown.min()

  return max_drawdown

def calculate_sharpe(daily_returns, risk_free = 0.01):
  expected_return = daily_returns.mean() * 252
  std_dev = daily_returns.std()
  annualized_volatility = std_dev * np.sqrt(252)

  sharpe = (expected_return - risk_free) / annualized_volatility
  return sharpe

def calculate_net_profit(portfolio_values):
  return portfolio_values.iloc[-1] - portfolio_values.iloc[0]
```

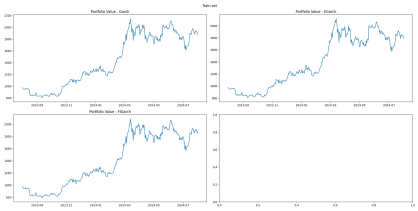
(د) متریک‌های Sharpe Ratio، Net Profit و Max Drawdown

```python
def buy_and_hold(symbols, data, initial_capital, weights, trans_fee=0.02):

  opt_investment_amounts = initial_capital * weights
  initial_prices = data.iloc[0]

  opt_shares = (opt_investment_amounts * (1 - trans_fee)) / initial_prices

  final_prices = data.iloc[-1]
  opt_final_portfolio_value = np.sum(opt_shares * final_prices)


  opt_portfolio_values = (data * opt_shares).sum(axis = 1)

  print("Optimal Weights:")
  for w, symbol in zip(weights, symbols):
    print(f"{symbol}: {w:.4f}")
  print(50 * '-')

  print("Shares bought:")
  for i in opt_shares.index:
    print(f"{i}: {opt_shares[i]:.4f}")
  print(50 * '-')

  print("Optimal final portfolio value: $[:.2f]".format(opt_final_portfolio_value))

  return opt_portfolio_values
```

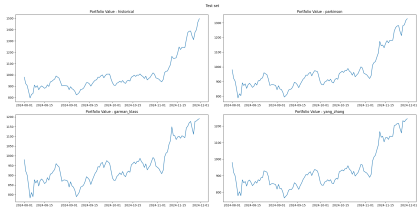
(ج) استراتژی Buy and Hold

سپس به اجرای استراتژی روی وزن‌های بدست آمده از روش‌های مختلف می‌پردازیم. نمودارهای قیمتی در ادامه خواهند آمد و مفادیر وزن‌ها نیز از نوت‌بوک قابل دسترسی اند.

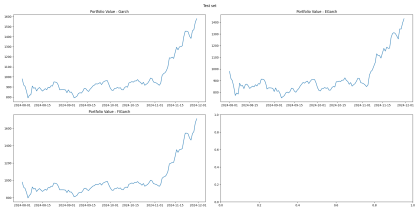
(و) اجرای استراتژی برای وزن‌های پروکسی‌ها روی داده آموزش



(ه) اجرای استراتژی برای وزن‌های مدل‌های آماری روی داده آموزش



(ح) اجرای استراتژی برای وزن‌های پروکسی‌ها روی داده تست

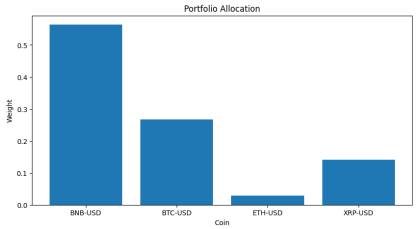


(ز) اجرای استراتژی برای وزن‌های مدل‌های آماری روی داده تست

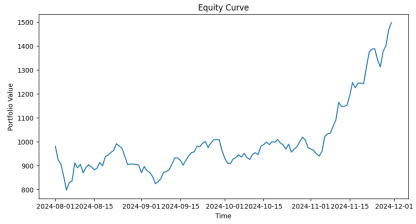
۵ تحلیل نتایج

با توجه به معیارهای گزارش شده (در نوت‌بوک) برای هر دسته وزن و تخمین‌گر نوسان، سری نوسانی FIGarch عملکرد بهتری نسبت به بقیه دارد زیرا هم نسبت Sharpe Ratio بهتری نسبت به بقیه دارد و هم اینکه مقدار Net Profit آن بیشتر است.

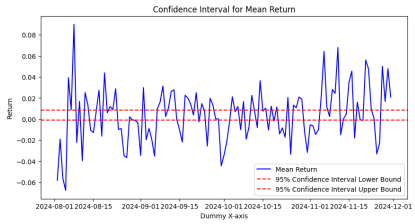
حال به رسم نمودارهای ذکر شده برای دسته با خروجی بهینه می‌پردازیم.



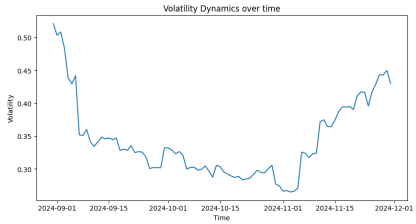
(ی) نمودار Portfolio Allocation



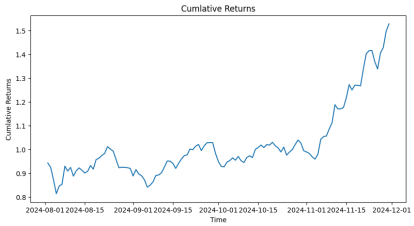
(ط) نمودار Equity



(ل) نمودار Confidence Intervals of Portfolio Estimates



(ک) نمودار Volatility Dynamics over time



(م) نمودار Cumulative Return