Notebook

December 11, 2024

1 Idiosyncratic Momentum

Paper Publication Date: January 11, 2011 (David Blitz, Joop Huij, Martin Marten) | Silver Fund Replication By : Dipesh Ghimire and Mikael Casellas

1.1 Economic Rationale

1.1.1 Underreaction Hypothesis

Idiosyncratic momentum profits are driven by investor underreaction to firm-specific news. Investors underreact to firm-specific news, causing slow information diffusion, which leads to sustained positive returns for idiosyncratic momentum over time.

1.1.2 Institutional Negligence

Institutional investors focus on total return momentum, neglecting firm-specific information diffusion, which creates mispricing opportunities for idiosyncratic momentum.

1.2 Empirical Evidence

- Idiosyncratic momentum is separate from conventional momentum and cannot be explained by it. It exists independently even when accounting for total return momentum.
- Idiosyncratic momentum generates more consistent risk-adjusted returns (higher Sharpe ratios) with lower volatility, while conventional momentum experiences higher volatility and reversals.

1.3 Construction Process

1. **Collect Data**
   - Obtain the monthly returns for each stock and the risk-free rate .
   - Get the Fama-French factors for market (mkt) , size ( ), value ( ), investment ( ), and profitability ( ).
2. **Run Time Series Regression**
   - For each stock , regress the excess stock returns on the five Fama-French factors over the past 24 months.
   - This regression will yield residuals, which represent the idiosyncratic component of the stock's return.
3. **Calculate Idiosyncratic Momentum Score**

- The idiosyncratic momentum score for stock is the volatility-adjusted mean idiosyncratic return for last year skipping last 22 days.

1.4 Importing Packages and Cleaning Data

[269]:
```python
# Importing all necessary libraries and packages

import numpy as np

import pandas as pd

import polars as pl

from joblib import Parallel, delayed

from sklearn.linear_model import LinearRegression

from numba import njit, prange

import matplotlib.pyplot as plt

import plotly.graph_objects as go

from plotly.subplots import make_subplots

import warnings
warnings.filterwarnings("ignore")
```

[32]:
```python
# Read the daily stock file using Polars

df_polars = pl.read_parquet('/home/dipesh77/SilverFund/Fall2024/Final/dataset/
 ↪combined_file.parquet')

# Drop the specified columns
columns_to_drop = ['date',␣
 ↪'obsdate','enddate','cusip','barra_ticker','issue_name','vol_crsp',
                   ␣
 ↪'retx_crsp','shrout_crsp','price_barra','mktcap_barra','ret_barra','total_risk',
                   ␣
 ↪'HistBeta','PredBeta','ncusip_crsp','ticker_crsp','shrcd_crsp',
                   'exchcd_crsp','siccd_crsp','prc_crsp']

df_polars = df_polars.drop(columns_to_drop)

# Drop null values in the 'ret_crsp' column
df_polars = df_polars.drop_nulls(subset=['ret_crsp'])
```

```python
# Convert the dataframe to a pandas dataframe
df_pandas = df_polars.to_pandas()

# Sort the dataframe by permno and date
df_pandas.sort_values(by=['permno', 'DataDate'], inplace=True)

# Reset the index of the dataframe
df_pandas.reset_index(drop=True, inplace=True)

# Rename 'DataDate' to 'date'
df_pandas.rename(columns={'DataDate': 'date'}, inplace=True)

# Read the daily fama french data
ff_daily = pd.read_csv('/home/dipesh77/QuantResearch/datasets/ff_daily.csv')

# Convert the 'date' column to datetime format
ff_daily['date'] = pd.to_datetime(ff_daily['date'])

#Merge the daily file with fama french data
merged_daily = pd.merge(df_pandas, ff_daily, on='date')

# Create excess_return column
merged_daily['excess_ret'] = merged_daily['ret_crsp'] - merged_daily['rf']

# Drop rf column
merged_daily = merged_daily.drop('rf', axis=1)

# Drop permnos with rows less than 756
merged_daily = merged_daily.groupby('permno').filter(lambda x: len(x) > 504)

# Droping duplicates for values of permno for same date
merged_daily = merged_daily.drop_duplicates(subset=['date', 'permno'])

# Dropig NaN in barrid column
merged_daily = merged_daily.dropna(subset=['barrid'])

# Droping duplicates for values of barrid for the same date
merged_daily = merged_daily.drop_duplicates(subset=['date', 'barrid'])

# Forward filling the spec_risk column
merged_daily['spec_risk'] = merged_daily.groupby('barrid')['spec_risk'].ffill()
```

1.5 Calculating Idiosyncratic Momentum Signal

[35]:
```
# Create multiple jit function to calculate the rolling idiosyncratic momentum

@njit
def add_intercept(X):
    """Add an intercept (column of ones) to the feature matrix."""
    intercept = np.ones((X.shape[0], 1))
    return np.hstack((intercept, X))

@njit
def ols_beta(X, Y):
    """Compute OLS coefficients using the normal equation and np.linalg.solve.
↪"""
    X = np.ascontiguousarray(X)
    Y = np.ascontiguousarray(Y)
    X_transpose = X.T
    beta = np.linalg.solve(X_transpose @ X, X_transpose @ Y)
    return beta

@njit
def calculate_residuals(X, Y, beta):
    """Compute residuals as (Y - X @ beta)."""
    X = np.ascontiguousarray(X)
    Y = np.ascontiguousarray(Y)
    beta = np.ascontiguousarray(beta)
    predicted = X @ beta
    residuals = Y - predicted
    return residuals

@njit(parallel=True)
def calculate_rolling_momentum(x, y, intercept):
    """Calculate idiosyncratic momentum using rolling regressions."""
    n = len(x) - 504
    idio_mom = np.full(len(x), np.nan)  # Pre-allocate with NaNs

    for i in prange(n):  # Parallel loop
        x_window = np.ascontiguousarray(x[i:i + 504])
        y_window = np.ascontiguousarray(y[i:i + 504])

        # Add intercept if specified
        if intercept:
            x_window = add_intercept(x_window)
            x_window = np.ascontiguousarray(x_window)

        # Calculate OLS coefficients using np.linalg.solve for better numerical␣
↪stability
```

```python
        beta = ols_beta(x_window, y_window)

        # Calculate residuals
        residuals = calculate_residuals(x_window, y_window, beta)

        # Select the last 252 days and skip the last 22 days
        last_year_residuals = residuals[-252:]
        signal_residuals = last_year_residuals[:230]

        # Calculate the idiosyncratic momentum
        idiosyncratic_momentum = signal_residuals.mean() / signal_residuals.
↪std()

        # Store the result in the pre-allocated list
        idio_mom[504 + i] = idiosyncratic_momentum

    return idio_mom

def calculate_idio_mom(df, factors, intercept):
    """Calculate idiosyncratic momentum for a single permno group with␣
↪specified factors."""
    x = df[factors].values
    y = df[['excess_ret']].values

    # Use the optimized rolling momentum calculation with or without intercept
    idio_mom = calculate_rolling_momentum(x, y, intercept=intercept)

    return pd.Series(idio_mom, index=df.index)

def process_groups(df, factors, intercept):
    """Process each permno group in parallel with specified factors."""
    permno_groups = list(df.groupby('permno'))

    results = Parallel(n_jobs=-1, backend='loky')(
        delayed(calculate_idio_mom)(group, factors, intercept=intercept) for _,␣
↪group in permno_groups
    )
    return pd.concat(results)

# Apply the parallelized function with dynamic factors
merged_daily['idio_mom_5f'] = process_groups(merged_daily, factors= ['mktrf',␣
↪'smb', 'hml', 'cma', 'rmw'], intercept=True)

# Drop rows with NaNs in idio_mom values
merged_daily.dropna(subset=['idio_mom_5f'], inplace=True)
```

1.6 Creating Naive Long Short Decile Portfolio

[36]:

```
# Create bins for the idio_mom values
merged_daily['bins'] = merged_daily.groupby('date')['idio_mom_5f'].
 ↪transform(lambda x: pd.qcut(x, 10, labels=False, duplicates='drop'))

# Group by date and bins and calculate the mean return of each bin for each date
port = merged_daily.groupby(['date', 'bins'])['ret_crsp'].mean().unstack()

#Create a long short portfolio
port['long_short'] = port[9] - port[0]

# Calculate the long-short portfolio mean return (estimation for 21 trading␣
 ↪days)
ls_mean = (port['long_short'].mean()*100*21)

# Calculate the annualized sharpe ratio
sharpe = ( port['long_short'].mean() / port['long_short'].std() ) * np.sqrt(252)

# Merge fama french data with portfolio data
port = port.merge(ff_daily, on='date', how='left')

# Set up OLS regression to calculate alpha on the Fama-French 5 factor model +␣
 ↪Momentum
X = port[['mktrf', 'hml', 'smb', 'cma', 'rmw', 'umd']]
y = port['long_short']
reg = LinearRegression().fit(X, y)

# Calculate alpha on the Fama-French 5 factor model + Momentum
alpha = reg.intercept_

# Make alpha monthly and in percentage format (estimation for 21 trading days)
alpha = alpha * 21 * 100

#Beta to umd factor
beta = reg.coef_[0]
beta_hml = reg.coef_[1]
beta_smb = reg.coef_[2]
beta_cma = reg.coef_[3]
beta_rmw = reg.coef_[4]
beta_umd = reg.coef_[5]

print(f'Long-Short Portfolio Monthly Mean Return: {ls_mean:.2f}%')
print(f'Annualized Sharpe Ratio: {sharpe:.2f}')
print(f'Monthly Alpha to 5-Factor Model + Momentum: {alpha:.2f}%')
print(f'Beta to mktrf: {beta:.2f}')
print(f'Beta to hml: {beta_hml:.2f}')
```

```
print(f'Beta to smb: {beta_smb:.2f}')
print(f'Beta to cma: {beta_cma:.2f}')
print(f'Beta to rmw: {beta_rmw:.2f}')
print(f'Beta to umd: {beta_umd:.2f}')
```

```
Long-Short Portfolio Monthly Mean Return: 0.62%
Annualized Sharpe Ratio: 0.66
Monthly Alpha to 5-Factor Model + Momentum: 0.41%
Beta to mktrf: 0.08
Beta to hml: 0.20
Beta to smb: -0.02
Beta to cma: -0.16
Beta to rmw: 0.03
Beta to umd: 0.46
```

1.7 PnL Plots on Naive Long Short Decile Portfolio

[37]:
```
#Calculating drawdown
cumulative = (port['long_short'] + 1).cumprod()
previous_peaks = np.maximum.accumulate(cumulative)
drawdown = (cumulative - previous_peaks) / previous_peaks

# Create subplots: 1 row, 3 columns
fig = make_subplots(rows=1, cols=3, subplot_titles=("PnL Cumulative Sum", "PnL␣
↪Cumulative Product", "Drawdown Plot"))

# Add traces to the subplots
fig.add_trace(go.Scatter(x=port['date'], y=port['long_short'].cumsum(),␣
↪name='PnL Cumulative Sum'), row=1, col=1)
fig.add_trace(go.Scatter(x=port['date'], y=(port['long_short']+1).cumprod(),␣
↪name='PnL Cumulative Product'), row=1, col=2)
fig.add_trace(go.Scatter(x=port['date'], y=drawdown, name='Drawdown Plot'),␣
↪row=1, col=3)

# Update layout
fig.update_layout(title_text='Long-Short Portfolio: PnL and Drawdown Plots',␣
↪xaxis_title='Date', yaxis_title='PnL')

# Show plot
fig.show()
```

1.8 Factor Attribution on the Naive Long Short Decile Portfolio

[38]:
```
X  = port[['mktrf', 'hml', 'smb', 'cma', 'rmw', 'umd']]
y = port['long_short']
reg = LinearRegression().fit(X, y)
```

```
#Get the residuals
residuals = y - X @ reg.coef_

#Ret driven by factors
mktrf = port['mktrf'] * reg.coef_[0]
hml = port['hml'] * reg.coef_[1]
smb = port['smb'] * reg.coef_[2]
cma = port['cma'] * reg.coef_[3]
rmw = port['rmw'] * reg.coef_[4]
umd = port['umd'] * reg.coef_[5]

decomposition = pd.DataFrame({'residuals': residuals, 'mktrf': mktrf, 'hml':␣
↪hml, 'smb': smb, 'cma': cma, 'rmw': rmw, 'umd': umd})

#Calculating drawdown
cumulative = (decomposition['residuals'] + 1).cumprod()
previous_peaks = np.maximum.accumulate(cumulative)
drawdown = (cumulative - previous_peaks) / previous_peaks

# Create subplots: 1 row, 3 columns
fig = make_subplots(rows=1, cols=3, subplot_titles=("Cumulative prod of␣
↪decomposed returns", "Cumulative sum of decomposed returns", "Drawdown of␣
↪Residuals Portfolio"))

# Add traces to the subplots
for column in decomposition.columns:
    fig.add_trace(go.Scatter(x=port['date'], y=(decomposition[column] + 1).
↪cumprod(), mode='lines', name=column), row=1, col=1)
    fig.add_trace(go.Scatter(x=port['date'], y=decomposition[column].cumsum(),␣
↪mode='lines', name=column), row=1, col=2)

fig.add_trace(go.Scatter(x=port['date'], y=drawdown, name='Drawdown'), row=1,␣
↪col=3)

# Update layout
fig.update_layout(title_text='Return Decomposition', xaxis_title='Date',␣
↪yaxis_title='Cumulative Returns')

# Show plot
fig.show()
```

1.9 Creating the Grinold-Kahn Alpha Forecast

[41]:
```
# Drop columns with no spec_risk
merged_daily = merged_daily.dropna(subset=['spec_risk'])
```

[58]:
```
# Setting information coffecient
ic = 0.05

# Cross-sectional z-score for the idiosyncratic momentum signal
merged_daily['zscore'] = merged_daily.groupby('date')['idio_mom_5f'].
↪transform(lambda x: (x-x.mean())/x.std())

# Creating the alpha forecast
merged_daily['alpha'] = ic * merged_daily['spec_risk'] * merged_daily['zscore']

# Convert the alphas into pivot table (format for backtester)
pivoted_data = merged_daily.pivot(index='date', columns='barrid',␣
↪values='alpha').dropna(axis=1, how='all')
```

[67]:
```
#Saving alphas to parquet file
pivoted_data.to_parquet('/home/dipesh77/SilverFund/Fall2024/Final/dataset/alpha.
↪parquet')
```

1.10 Loading the optimal portfolio weights from the backtester

[243]:
```
# Load the optimal portfolio weights
port_weights = pd.read_parquet('/home/dipesh77/SilverFund/Fall2024/Final/
↪dataset/portfolio.parquet')

port_weights = pd.DataFrame(port_weights.unstack()).reset_index()

port_weights.columns = ['barrid', 'date', 'weights']

# Get the required columns from the merged daily data
daily_data = merged_daily[['date', 'barrid', 'ret_crsp', 'mktrf', 'smb', 'hml',␣
↪'rmw', 'cma', 'umd']]

# Merge the daily data with the port weights
daily_data = pd.merge(daily_data, port_weights, on=['date', 'barrid'],␣
↪how='left')

# Fill NaN weights with 0
daily_data['weights'] = daily_data['weights'].fillna(0)

# Scaling the weights so that long book and short book both sums to 1 while␣
↪keeping the dollar neutral, ie. gearing is 1
daily_data['abs_weights'] = np.abs(daily_data['weights'])
abs_weights = pd.DataFrame(daily_data.groupby('date')['abs_weights'].sum()).
↪reset_index()
daily_data.drop(columns='abs_weights', inplace=True)
daily_data = pd.merge(daily_data, abs_weights, on='date', how='left')
daily_data['scale'] = 2/daily_data['abs_weights']
```

```
daily_data['scaled_weights'] = daily_data['weights'] * daily_data['scale']
```

1.11 PnL Plots on the Optimal Mean-Variance Dollar Neutral Portfolio

[244]:
```
# Calculating the weighted returns for each stock
daily_data['weighted_ret'] = daily_data['scaled_weights'] *␣
 ↪daily_data['ret_crsp']

# Group by date and calculate the mean portfolio return
portfolio_return = daily_data.groupby('date')['weighted_ret'].sum()

# Converting the portfolio return to a dataframe
portfolio_return = pd.DataFrame(portfolio_return).
 ↪rename(columns={'weighted_ret': 'ret'})

#Calculating drawdown
cumulative = (portfolio_return['ret'] + 1).cumprod()
previous_peaks = np.maximum.accumulate(cumulative)
drawdown = (cumulative - previous_peaks) / previous_peaks

# Create subplots: 1 row, 3 columns
fig = make_subplots(rows=1, cols=3, subplot_titles=("PnL Cumulative Sum", "PnL␣
 ↪Cumulative Product", "Drawdown Plot"))

# Add traces to the subplots
fig.add_trace(go.Scatter(x=portfolio_return.index, y=portfolio_return['ret'].
 ↪cumsum(), name='PnL Cumulative Sum'), row=1, col=1)
fig.add_trace(go.Scatter(x=portfolio_return.index,␣
 ↪y=(portfolio_return['ret']+1).cumprod(), name='PnL Cumulative Product'),␣
 ↪row=1, col=2)
fig.add_trace(go.Scatter(x=portfolio_return.index, y=drawdown, name='Drawdown␣
 ↪Plot'), row=1, col=3)

# Update layout
fig.update_layout(title_text='Optimal Mean-Variance Dollar Neutral Portfolio:␣
 ↪PnL and Drawdown Plots', xaxis_title='Date', yaxis_title='PnL')

# Show plot
fig.show()
```

1.12 Factor Attribution on the Optimal Mean-Variance Dollar Neutral Portfolio

[245]:
```
portfolio_return = pd.merge(portfolio_return, ff_daily, on='date', how='left')
X  = portfolio_return[['mktrf', 'hml', 'smb', 'cma', 'rmw', 'umd']]
y = portfolio_return['ret']
reg = LinearRegression().fit(X, y)
```

```
#Get the residuals
residuals = y - X @ reg.coef_

#Ret driven by factors
mktrf = portfolio_return['mktrf'] * reg.coef_[0]
hml = portfolio_return['hml'] * reg.coef_[1]
smb = portfolio_return['smb'] * reg.coef_[2]
cma = portfolio_return['cma'] * reg.coef_[3]
rmw = portfolio_return['rmw'] * reg.coef_[4]
umd = portfolio_return['umd'] * reg.coef_[5]

decomposition = pd.DataFrame({'residuals': residuals, 'mktrf': mktrf, 'hml':␣
↪hml, 'smb': smb, 'cma': cma, 'rmw': rmw, 'umd': umd})

#Calculating drawdown
cumulative = (decomposition['residuals'] + 1).cumprod()
previous_peaks = np.maximum.accumulate(cumulative)
drawdown = (cumulative - previous_peaks) / previous_peaks

# Create subplots: 1 row, 3 columns
fig = make_subplots(rows=1, cols=3, subplot_titles=("Cumulative prod of␣
↪decomposed returns", "Cumulative sum of decomposed returns", "Drawdown of␣
↪Residuals Portfolio"))

# Add traces to the subplots
for column in decomposition.columns:
    fig.add_trace(go.Scatter(x=port['date'], y=(decomposition[column] + 1).
↪cumprod(), mode='lines', name=column), row=1, col=1)
    fig.add_trace(go.Scatter(x=port['date'], y=decomposition[column].cumsum(),␣
↪mode='lines', name=column), row=1, col=2)

fig.add_trace(go.Scatter(x=port['date'], y=drawdown, name='Drawdown'), row=1,␣
↪col=3)

# Update layout
fig.update_layout(title_text='Return Decomposition', xaxis_title='Date',␣
↪yaxis_title='Cumulative Returns')

# Show plot
fig.show()
```

1.13 Portfolio Statistics

[246]:
```python
# Calculate the long-short portfolio mean return (estimation for 21 trading␣
↪days)
ls_mean = (portfolio_return['ret'].mean()*100*21)

# Calculate the annualized sharpe ratio
sharpe = ( portfolio_return['ret'].mean() / portfolio_return['ret'].std() ) *␣
↪np.sqrt(252)

# Set up OLS regression to calculate alpha on the Fama-French 5 factor model +␣
↪Momentum
X = portfolio_return[['mktrf', 'hml', 'smb', 'cma', 'rmw', 'umd']]
y = portfolio_return['ret']
reg = LinearRegression().fit(X, y)

# Calculate alpha on the Fama-French 5 factor model + Momentum
alpha = reg.intercept_

# Make alpha monthly and in percentage format (estimation for 21 trading days)
alpha = alpha * 21 * 100

#Beta to umd factor
beta = reg.coef_[0]
beta_hml = reg.coef_[1]
beta_smb = reg.coef_[2]
beta_cma = reg.coef_[3]
beta_rmw = reg.coef_[4]
beta_umd = reg.coef_[5]

print(f'Long-Short Portfolio Monthly Mean Return: {ls_mean:.2f}%')
print(f'Annualized Sharpe Ratio: {sharpe:.2f}')
print(f'Monthly Alpha to 5-Factor Model + Momentum: {alpha:.2f}%')
print(f'Beta to mktrf: {beta:.2f}')
print(f'Beta to hml: {beta_hml:.2f}')
print(f'Beta to smb: {beta_smb:.2f}')
print(f'Beta to cma: {beta_cma:.2f}')
print(f'Beta to rmw: {beta_rmw:.2f}')
print(f'Beta to umd: {beta_umd:.2f}')
```

```
Long-Short Portfolio Monthly Mean Return: 0.55%
Annualized Sharpe Ratio: 2.25
Monthly Alpha to 5-Factor Model + Momentum: 0.54%
Beta to mktrf: 0.02
Beta to hml: 0.01
Beta to smb: -0.01
Beta to cma: -0.02
Beta to rmw: 0.01
```

Beta to umd: 0.00

1.14 Strategy Turnover

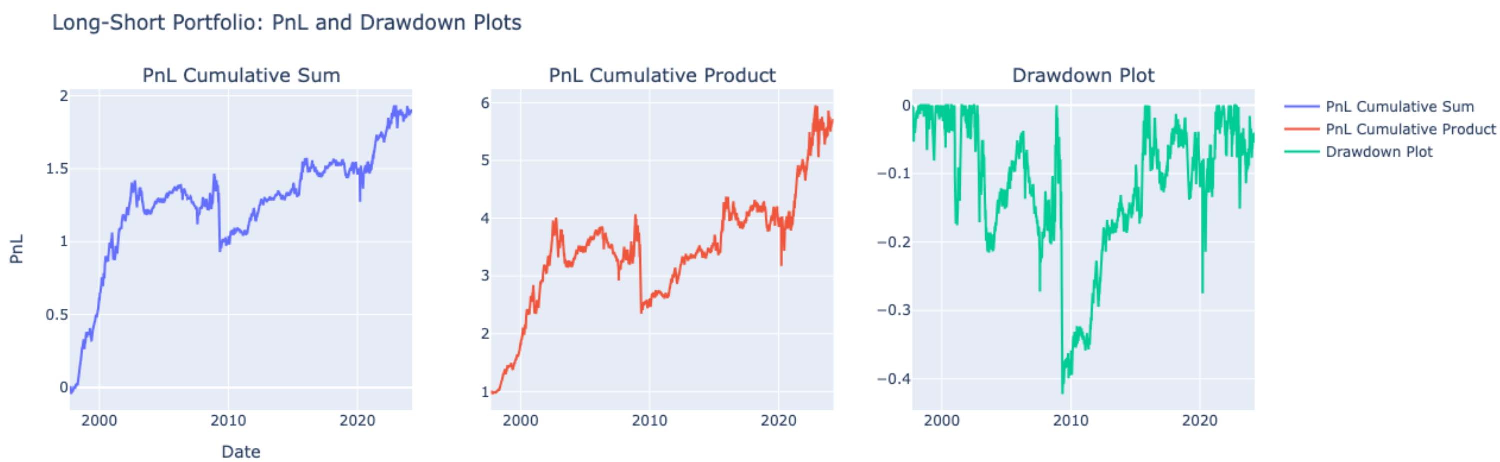
[259]:
```
# Calculating the absolute change in weights
daily_data['abs_change_weights'] = daily_data.
 ↪groupby('barrid')['scaled_weights'].diff().abs()

# Calculating the turnover
turnover = daily_data.groupby('date')['abs_change_weights'].sum()

# Calculating the average turnover
avg_turnover = turnover.mean()

# Plotting the turnover in plotly with mean turnover line
fig = go.Figure()
fig.add_trace(go.Scatter(x=turnover.index, y=turnover, mode='lines',␣
 ↪name='Turnover'))
fig.add_trace(go.Scatter(x=turnover.index, y=[turnover.mean()]*len(turnover),␣
 ↪mode='lines', name='Mean Turnover', line=dict(dash='dash')))
fig.update_layout(title='Turnover', xaxis_title='Date', yaxis_title='Turnover')
fig.show()

# Displaying the average turnover
print(f'Average Turnover: {avg_turnover:.2f}')
```

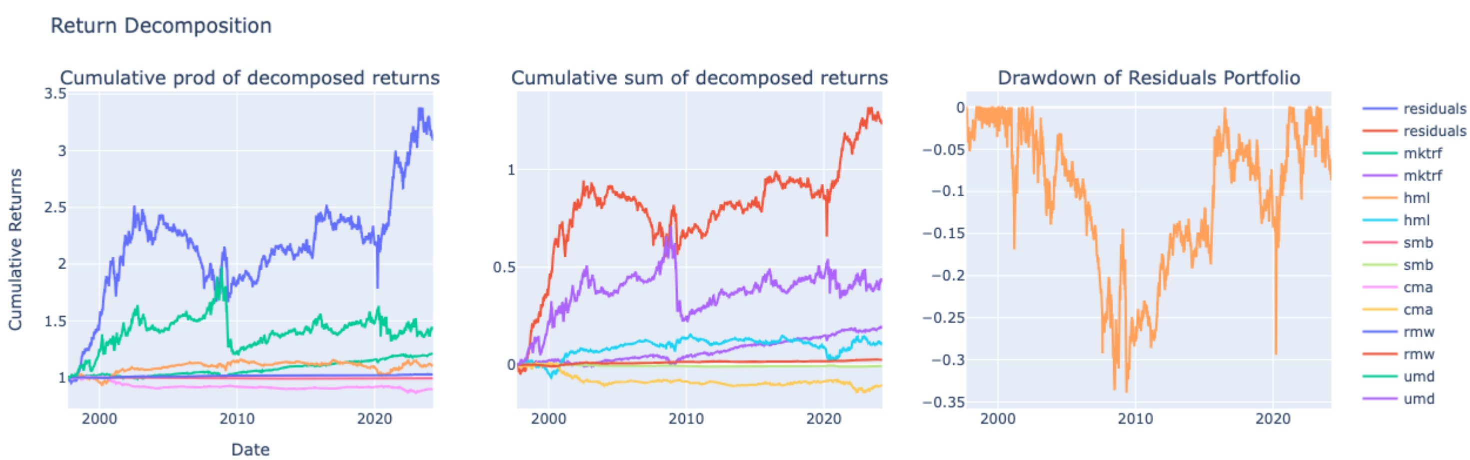
```
Average Turnover: 0.41
```

1.15 Alpha Decay

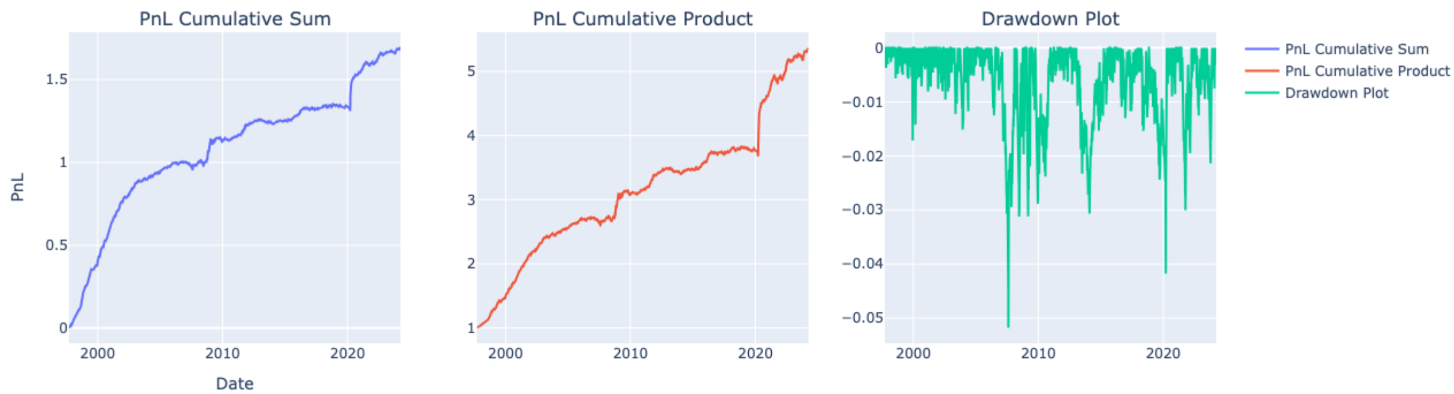
[273]:
```
# Function to calculate Sharpe ratio for a given lag
def calculate_sharpe_ratio(lag, daily_data):
    # Create a copy of the dataframe to avoid modifying the original data
    data_copy = daily_data.copy()

    # Lagging the weights
    data_copy['lag_weights'] = data_copy.groupby('barrid')['scaled_weights'].
 ↪shift(lag)

    # Calculating the weighted returns for each stock
    data_copy['weighted_ret'] = data_copy['lag_weights'] * data_copy['ret_crsp']

    # Group by date and calculate the mean portfolio return
    portfolio_return = data_copy.groupby('date')['weighted_ret'].sum()

    # Calculate mean and std for the Sharpe ratio
    mean_return = portfolio_return.mean()
```

```
    std_return = portfolio_return.std()

    # Sharpe ratio
    sharpe = (mean_return / std_return) * np.sqrt(252)

    return sharpe

# Parallel computation of Sharpe ratios for different lags
sharpe_ratios = Parallel(n_jobs=-1, backend='loky')(
    delayed(calculate_sharpe_ratio)(i, daily_data) for i in range(0, 42)
)

# Plotting the Sharpe ratios using plotly
fig = go.Figure()
fig.add_trace(go.Scatter(x=list(range(len(sharpe_ratios))), y=sharpe_ratios,␣
 ↪mode='lines+markers', name='Sharpe Ratio'))
fig.update_layout(title='Alpha Decay', xaxis_title='Lag', yaxis_title='Sharpe␣
 ↪Ratio')
fig.show()
```

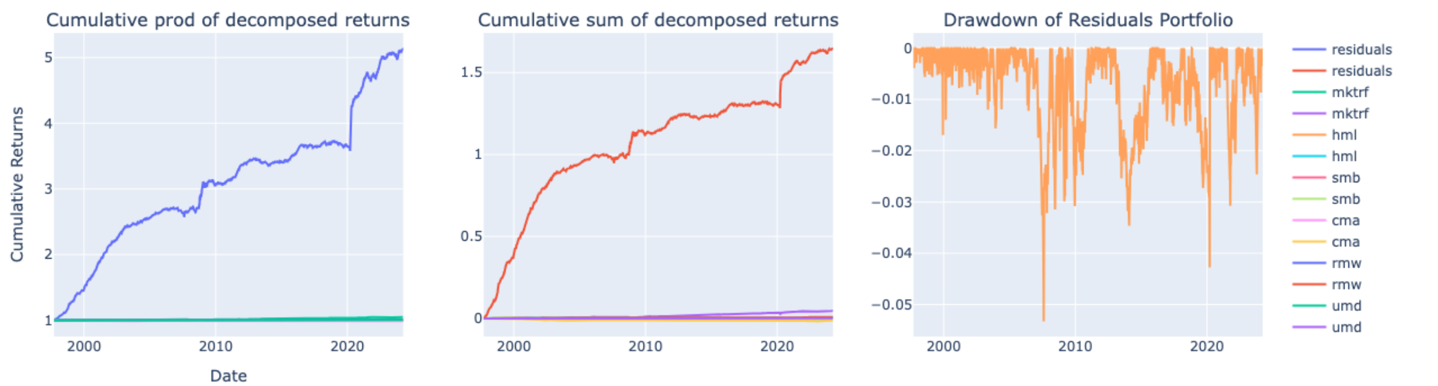




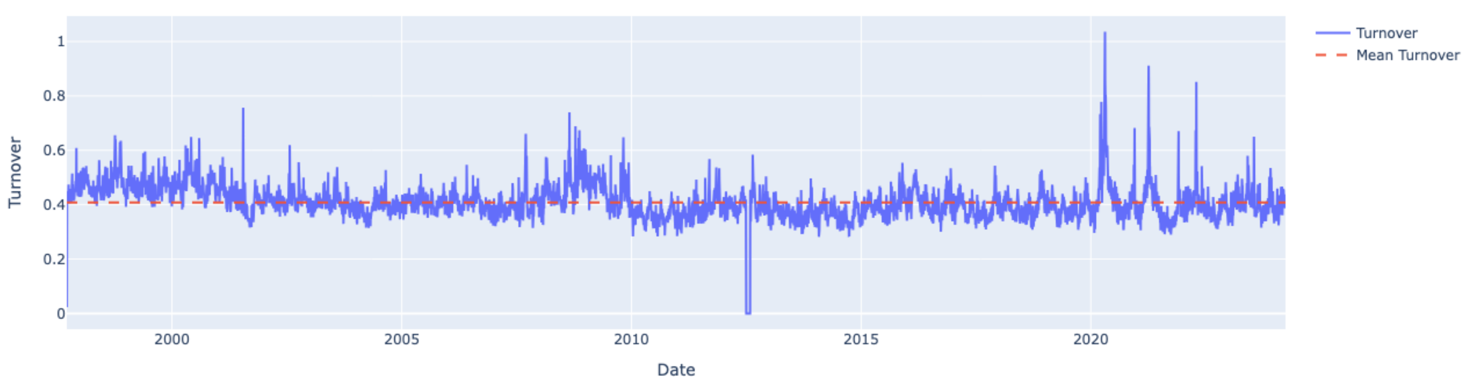
Optimal Mean-Variance Dollar Neutral Portfolio: PnL and Drawdown Plots



Return Decomposition



Turnover



Alpha Decay