HMW 4 - Quantitative Investing

This Homework was completed by Gabriele Monti, Juan Pablo Herrera, and Fernando Garcia De Llano for the Quantitative Investing class (FBE 551) - Prof. Shane Shepherd

Imports

```
import pandas as pd
import numpy as np
from pandas.tseries.offsets import MonthEnd
import statsmodels.api as sm
```

Data Preparation

Reading the Data

```
famafrench_monthly = pd.DataFrame(pd.read_csv("FamaFrenchMonthly.csv"))
compustat_annual = pd.DataFrame(pd.read_feather("compustat_annual.feather"))
crsp_monthly = pd.DataFrame(pd.read_feather("crsp_monthly_stocks.feather"))
```

```
famafrench_monthly.head()
```

	Date	Mkt-RF	SMB	HML	RF
0	192607	2.96	-2.56	-2.43	0.22
1	192608	2.64	-1.17	3.82	0.25
2	192609	0.36	-1.40	0.13	0.23
3	192610	-3.24	-0.09	0.70	0.32
4	192611	2.53	-0.10	-0.51	0.31

Next steps: Generate code with famafrench_monthly | View recommended plots | New interactive sheet

```
compustat_annual.head()
```

	DATADATE	FYEAR	LPERMNO	AT	CEQ	CHE	LT	PSTK	SEQ	DVT	IB	SALE	CAPX
0	1970-12-31	1970.0	25881.0	33.450	10.544	1.660	22.906	0.000	10.544	0.000	1.878	45.335	2.767
1	1971-12-31	1971.0	25881.0	29.330	8.381	2.557	20.948	0.000	8.382	0.000	0.138	47.033	1.771
2	1972-12-31	1972.0	25881.0	19.907	7.021	2.027	12.886	0.000	7.021	0.000	1.554	34.362	1.254
3	1973-12-31	1973.0	25881.0	21.771	8.567	1.357	13.204	0.000	8.567	0.000	1.863	37.750	1.633
4	1974-12-31	1974.0	25881.0	25.638	9.843	1.338	15.381	0.414	10.257	0.021	1.555	50.325	1.313

```
crsp_monthly.head()
```

	PERMNO	DATE	SHRCD	EXCHCD	SICCD	PRC	VOL	RET	SPREAD	RETX	SHROUT
0	10000.0	1986-01-31	10.0	3.0	3990.0	-4.375000	1771.0	NaN	0.25000	NaN	3680.0
1	10000.0	1986-02-28	10.0	3.0	3990.0	-3.250000	828.0	-0.257143	0.25000	-0.257143	3680.0
2	10000.0	1986-03-31	10.0	3.0	3990.0	-4.437500	1078.0	0.365385	0.12500	0.365385	3680.0
3	10000.0	1986-04-30	10.0	3.0	3990.0	-4.000000	957.0	-0.098592	0.25000	-0.098592	3793.0
4	10000.0	1986-05-30	10.0	3.0	3990.0	-3.109375	1074.0	-0.222656	0.09375	-0.222656	3793.0

Cleaning CRSP data

The stocks dataframe from CRSP contains stock returns (RET), closing prices (PRC), volume (VOL), shares outstanding (SHROUT), a code describing the issue type (SHRCD), a code for the primary exchange (EXCHCD), and an industry code (SICCD).

Firms are identified by PERMNO, which remains constant over a firm's life. Data are monthly, and the date is equal to the last trading day of the month.

1. Shift the date so that it is always the last day of the month, rather than the last trading day. This will make it easier to merge in with other datasets.
2. Take the absolute value of the closing price. For shares that don't trade, CRSP sets the price equal to the closing bid-ask midpoint, but it makes the price negative as a warning about this.
3. Define market value (MV) as the product of shares outstanding and closing price.
4. Drop shares outstanding, which we won't use again, and the share code. We use the share code when we download data from WRDS. Selecting share codes of 10 or 11 means that we will be downloading common equity and not other securities (ETFs, REITS, etc.).
5. Set the index to PERMNO/DATE.
6. Sort by the index.

```
crsp_monthly['DATE'] = crsp_monthly['DATE'] + MonthEnd(0)
crsp_monthly['date'] = crsp_monthly['DATE']
crsp_monthly['PRC']  = np.abs(crsp_monthly['PRC'])
crsp_monthly['MV'] = crsp_monthly['SHROUT']*crsp_monthly['PRC']
crsp_monthly.set_index(['PERMNO','DATE'], inplace=True)
crsp_monthly.sort_index(inplace=True)
crsp_monthly.head()
```

		SHRCD	EXCHCD	SICCD	PRC	VOL	RET	SPREAD	RETX	SHROUT	date	MV
PERMNO	**DATE**											
10000.0	**1986-01-31**	10.0	3.0	3990.0	4.375000	1771.0	NaN	0.25000	NaN	3680.0	1986-01-31	16100.000000
	1986-02-28	10.0	3.0	3990.0	3.250000	828.0	-0.257143	0.25000	-0.257143	3680.0	1986-02-28	11960.000000
	1986-03-31	10.0	3.0	3990.0	4.437500	1078.0	0.365385	0.12500	0.365385	3680.0	1986-03-31	16330.000000
	1986-04-30	10.0	3.0	3990.0	4.000000	957.0	-0.098592	0.25000	-0.098592	3793.0	1986-04-30	15172.000000
	1986-05-31	10.0	3.0	3990.0	3.109375	1074.0	-0.222656	0.09375	-0.222656	3793.0	1986-05-31	11793.859375

Cleaning Compustat data

Compustat data is annual. It contains a variety of variables that are described in compustat_variables.xlsx. The one we will use here is earnings before extraordinary items (IB). For one small purpose I will also look at book equity (SEQ).

1. I have a variable LPERMNO that is equivalent to the PERMNO variable in CRSP, so we rename LPERMNO.
2. Create a new column, DATE, that is designed to represent the date that the data become known (To use this data, we must make an assumption about the first date on which this data would be available. Standard practice is to assume that by 6 months after the fiscal year end we will for sure have access to the annual report. We could either adjust the date forward by six months now, or make sure that we use something like the "shift" command to pull forward the lagged financial data. Here I'll keep the date as is, and pull it forward later in the process.) We'll make sure that the date is the last day of the month.
3. Set indexes and sort

```
#1
compustat_annual.rename(columns={"LPERMNO":"PERMNO"}, inplace=True)
#2
compustat_annual['DATE'] = compustat_annual['DATADATE'] + MonthEnd(0)
#3
compustat_annual.set_index(['PERMNO','DATE'], inplace=True)
compustat_annual.sort_index(inplace=True)

compustat_annual.head()
```

PERMNO	DATE	DATADATE	FYEAR	AT	CEQ	CHE	LT	PSTK	SEQ	DVT	IB	SALE	CAPX
10000.0	**1986-10-31**	1986-10-31	1986.0	2.115	0.418	0.348	1.697	0.0	0.418	0.000	-0.730	1.026	0.240
10001.0	**1986-06-30**	1986-06-30	1986.0	12.242	5.432	0.746	6.810	0.0	5.432	0.365	0.669	21.460	0.551
	1987-06-30	1987-06-30	1987.0	11.771	5.369	0.729	6.402	0.0	5.369	0.416	0.312	16.621	0.513
	1988-06-30	1988-06-30	1988.0	11.735	5.512	0.744	6.223	0.0	5.512	0.427	0.542	16.978	0.240
	1989-06-30	1989-06-30	1989.0	18.565	6.321	1.177	12.244	0.0	6.321	0.459	1.208	22.910	0.444

Merging the CRSP and Compustat Dataframes

We now need to merge these data. Unfortunately, the data occasionally have multiple rows with the same PERMNO and DATE. So we are eliminating duplicate PERMNO/DATE pairs.

If there is more than one PERMNO on the same date, then the bigger one is probably more important and therefore more likely to be correct.

We are therefore going to sort the dataframe in ascending order by PERMNO, then in ascending order by DATE, and then in descending order by size (either MV or SEQ).

```
# Sort by PERMNO, DATE, and MV (or other size-related metric) to handle duplicates
crsp_monthly = crsp_monthly.sort_values(by=['PERMNO', 'DATE', 'MV'], ascending=[True, True, False])
compustat_annual = compustat_annual.sort_values(by=['PERMNO', 'DATE', 'SEQ'], ascending=[True, True, False])

# Remove duplicates by keeping the first entry for each PERMNO/DATE after sorting
crsp_monthly = crsp_monthly.groupby(['PERMNO', 'DATE']).head(1)
compustat_annual = compustat_annual.groupby(['PERMNO', 'DATE']).head(1)



# Merge CRSP with Compustat to include SEQ (Book Value), IB (Income), DVT (Dividends), and SALE (Sales)
merged_data = crsp_monthly.merge(compustat_annual, how='left', on=['PERMNO', 'DATE'])



# Define the columns we want to forward-fill and lag
compustat_metrics = ['SEQ', 'IB', 'DVT', 'SALE']

# Forward-fill Compustat metrics for each stock to provide monthly values
merged_data[compustat_metrics] = merged_data.groupby('PERMNO')[compustat_metrics].ffill()

# Apply a six-month lag to prevent look-ahead bias
merged_data[compustat_metrics] = merged_data.groupby('PERMNO')[compustat_metrics].shift(6)



# Define the columns to keep
columns_to_keep = ['PRC', 'VOL', 'RET', 'SPREAD', 'RETX',
       'SHROUT','MV', 'RET', 'SEQ', 'IB', 'DVT', 'SALE']

# Keep only the necessary columns in the merged data
merged_data = merged_data[columns_to_keep]

stocks = merged_data

stocks
```

PERMNO	DATE	PRC	VOL	RET	SPREAD	RETX	SHROUT	MV	RET	SEQ	IB	DVT	SALE
10000.0	1986-01-31	4.375000	1771.0	NaN	0.25000	NaN	3680.0	1.610000e+04	NaN	NaN	NaN	NaN	NaN
	1986-02-28	3.250000	828.0	-0.257143	0.25000	-0.257143	3680.0	1.196000e+04	-0.257143	NaN	NaN	NaN	NaN
	1986-03-31	4.437500	1078.0	0.365385	0.12500	0.365385	3680.0	1.633000e+04	0.365385	NaN	NaN	NaN	NaN
	1986-04-30	4.000000	957.0	-0.098592	0.25000	-0.098592	3793.0	1.517200e+04	-0.098592	NaN	NaN	NaN	NaN
	1986-05-31	3.109375	1074.0	-0.222656	0.09375	-0.222656	3793.0	1.179386e+04	-0.222656	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
93436.0	2023-02-28	205.710007	36239683.0	0.187565	NaN	0.187565	3164103.0	6.508876e+08	0.187565	30189.0	5519.0	0.0	53823.0
	2023-03-31	207.460007	33115078.0	0.008507	NaN	0.008507	3169314.0	6.575059e+08	0.008507	30189.0	5519.0	0.0	53823.0

Portfolio Creation Functions

Cleaning before applying the functions

```
# Drop one of the duplicate 'RET' columns
stocks = stocks.loc[:, ~stocks.columns.duplicated()]
```

Functions

```
def select_top_1000_by_metric(df, metric):
    """
    Select the top 1000 stocks each month based on the specified metric.

    Parameters:
        df (DataFrame): DataFrame containing stock data with a MultiIndex (PERMNO and DATE).
        metric (str): The column name of the metric to rank by.

    Returns:
        DataFrame: A DataFrame containing only the top 1000 stocks each month for the given metric.
    """
    top_1000_list = []

    # Group by DATE and select top 1000 stocks each month
    for date, group in df.groupby(level='DATE'):
        group = group.dropna(subset=[metric])  # Remove rows with NaN in the metric column
        top_1000 = group.nlargest(1000, metric)  # Select top 1000 by metric value
        top_1000_list.append(top_1000)

    # Combine the list into a DataFrame
    return pd.concat(top_1000_list)



def calculate_weights(top_1000_df, metric):
    """
    Calculate weights for the top 1000 stocks each month based on the given metric.

    Parameters:
        top_1000_df (DataFrame): DataFrame with the top 1000 stocks selected each month.
        metric (str): The metric used for ranking and weighting.

    Returns:
        DataFrame: DataFrame with weights calculated for each stock.
    """
    # Group by DATE and calculate weights
    top_1000_df['Weight'] = top_1000_df.groupby(level='DATE')[metric].transform(lambda x: x / x.sum())
```

```
    return top_1000_df



def lag_weights(df):
    """
    Lag the weights by one month to avoid look-ahead bias.

    Parameters:
        df (DataFrame): DataFrame containing weights.

    Returns:
        DataFrame: DataFrame with lagged weights.
    """
    # Shift weights by one month for each stock (identified by PERMNO)
    df['Lagged_Weight'] = df.groupby(level=0)['Weight'].shift(1)
    return df.dropna(subset=['Lagged_Weight'])  # Drop rows where Lagged_Weight is NaN after shifting



def calculate_portfolio_returns(df):
    """
    Calculate portfolio returns based on lagged weights.

    Parameters:
        df (DataFrame): DataFrame containing lagged weights and returns.

    Returns:
        DataFrame: Monthly portfolio returns.
    """
    # Multiply lagged weight by return and sum up by DATE to get portfolio return
    df['Weighted_Return'] = df['Lagged_Weight'] * df['RET']
    portfolio_returns = df.groupby(level='DATE')['Weighted_Return'].sum()
    return portfolio_returns



def full_portfolio_return_workflow(stocks, metric):
    """
    Full workflow to calculate monthly portfolio returns based on top 1000 stocks ranked by a metric.

    Parameters:
        stocks (DataFrame): DataFrame with stock data and MultiIndex (PERMNO, DATE).
        metric (str): Metric to rank and weight the stocks.

    Returns:
        Series: Monthly portfolio returns.
    """
    # Step 1: Select top 1000 stocks by metric each month
    top_1000_df = select_top_1000_by_metric(stocks, metric)

    # Step 2: Calculate weights
    top_1000_df = calculate_weights(top_1000_df, metric)

    # Step 3: Lag the weights
    top_1000_df = lag_weights(top_1000_df)

    # Step 4: Calculate portfolio returns
    portfolio_returns = calculate_portfolio_returns(top_1000_df)

    return portfolio_returns
```

Returns

```
# List of metrics to calculate portfolio returns for
metrics = ['SEQ', 'IB', 'DVT', 'SALE', 'MV']

# Dictionary to store portfolio returns for each metric
portfolio_returns_dict = {}

# Run the workflow for each metric and store the results
```

```
for metric in metrics:
    portfolio_returns_dict[metric] = full_portfolio_return_workflow(stocks, metric)

# Combine the portfolio returns into a single DataFrame
portfolio_returns_df = pd.DataFrame(portfolio_returns_dict)

# Display the resulting DataFrame with portfolio returns for all metrics
portfolio_returns_df
```

DATE	SEQ	IB	DVT	SALE	MV
1962-02-28	NaN	NaN	NaN	NaN	0.020102
1962-03-31	NaN	NaN	NaN	NaN	-0.004759
1962-04-30	NaN	NaN	NaN	NaN	-0.063601
1962-05-31	NaN	NaN	NaN	NaN	-0.083937
1962-06-30	NaN	NaN	NaN	NaN	-0.082724
...	...	...	...	...	...
2023-02-28	-0.032509	-0.029029	-0.035920	-0.032264	-0.021617
2023-03-31	-0.014511	0.013424	0.001058	0.002412	0.034546
2023-04-30	0.014383	0.018270	0.011056	0.009112	0.011943
2023-05-31	-0.027272	-0.014010	-0.040489	-0.024559	0.008530
2023-06-30	0.062453	0.058532	0.060668	0.079240	0.068521

737 rows × 5 columns

Next steps: Generate code with portfolio_returns_df | View recommended plots | New interactive sheet

Copying the Portfolios DF for further use

```
# Create a copy of the DataFrame
portfolio_returns_copy = portfolio_returns_df.copy()
```

Question 1

Fama French Cleaned

```
# Re-importing the Fama-French data with the correct date parsing
famafrench_monthly = pd.read_csv('FamaFrenchMonthly.csv', dtype={'Date': str})

# Convert the 'Date' column to datetime format if it's in 'YYYYMM' format
famafrench_monthly['Date'] = pd.to_datetime(famafrench_monthly['Date'], format='%Y%m')

# Set 'Date' as the index
famafrench_monthly.set_index('Date', inplace=True)

# Verify the DataFrame structure
print(famafrench_monthly.head())
```

```
            Mkt-RF   SMB   HML    RF
Date
1926-07-01    2.96 -2.56 -2.43  0.22
1926-08-01    2.64 -1.17  3.82  0.25
1926-09-01    0.36 -1.40  0.13  0.23
1926-10-01   -3.24 -0.09  0.70  0.32
1926-11-01    2.53 -0.10 -0.51  0.31
```

Recreation of Table 1

1. Data Preparation: Filtered portfolio_returns_df to include dates from 1962–2004. Converted Mkt-RF and RF in famafrench_monthly from percentages to decimals.

2. Set Reference Portfolio: Used the Market Cap (MV) weighted portfolio as the Reference for comparisons.
3. Metric Calculation: For each portfolio (Reference, SEQ, IB, SALE, DVT), we calculated:

- Annualized Arithmetic Return
- Annualized Volatility
- Sharpe Ratio (using RF)
- Excess Return vs. Reference
- t-stat on Excess Return vs. Reference.

4. Output: Compiled results into a DataFrame resembling Table 1 with all metrics accurately scaled and calculated.

```
# Converting 'Mkt-RF' and 'RF' columns in famafrench_monthly to decimal form if they're in percentages
famafrench_monthly[['Mkt-RF', 'RF']] = famafrench_monthly[['Mkt-RF', 'RF']] / 100

# Filtering portfolio returns for the 1962-2004 date range
portfolio_returns_df = portfolio_returns_copy.loc['1962-01-01':'2004-12-31']

# Setting the Reference portfolio as the Market Cap (MV) portfolio
portfolio_returns_df['Reference'] = portfolio_returns_copy['MV']

# Defining the portfolios to analyze, including the new Reference (MV) portfolio
portfolios_to_analyze = ['Reference', 'SEQ', 'IB', 'SALE', 'DVT']

# Initializing a dictionary to store the results for each portfolio
results = {}

# Defining the risk-free rate as RF for Sharpe Ratio calculation
risk_free_rate = famafrench_monthly.loc['1962-01-01':'2004-12-31', 'RF']

for portfolio in portfolios_to_analyze:
    # Monthly returns for the portfolio
    monthly_returns = portfolio_returns_df[portfolio]

    # Annualized arithmetic return
    annualized_return = monthly_returns.mean() * 12

    # Annualized volatility
    annualized_volatility = monthly_returns.std() * np.sqrt(12)

    # Sharpe Ratio (using annualized return - annualized risk-free rate)
    excess_return = monthly_returns - risk_free_rate
    annualized_excess_return = excess_return.mean() * 12
    annualized_risk_free_rate = risk_free_rate.mean() * 12
    sharpe_ratio = (annualized_return - annualized_risk_free_rate) / annualized_volatility

    # Excess Return vs. Reference (using MV portfolio as the reference)
    reference_returns = portfolio_returns_df['Reference']
    excess_return_vs_reference = annualized_return - (reference_returns.mean() * 12)

    # t-stat on Excess Return vs. Reference
    difference = monthly_returns - reference_returns
    t_stat = (difference.mean() / difference.std()) * np.sqrt(len(difference))

    # Storing the results in a dictionary
    results[portfolio] = {
        'Annualized Return': annualized_return,
        'Annualized Volatility': annualized_volatility,
        'Sharpe Ratio': sharpe_ratio,
        'Excess Return vs. Reference': excess_return_vs_reference,
        't-stat on Excess Return': t_stat
    }

# Converting results to a DataFrame for display
results_df = pd.DataFrame(results).T
results_df.columns = ['Annualized Return', 'Annualized Volatility', 'Sharpe Ratio', 'Excess Return vs. Reference', 't-stat on Exc

# Displaying the results in the format of Table 1
results_df
```

```
<ipython-input-201-b3d069d7c7c2>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  portfolio_returns_df['Reference'] = portfolio_returns_copy['MV']
<ipython-input-201-b3d069d7c7c2>:41: RuntimeWarning: invalid value encountered in scalar divide
  t_stat = (difference.mean() / difference.std()) * np.sqrt(len(difference))
```

	Annualized Return	Annualized Volatility	Sharpe Ratio	Excess Return vs. Reference	t-stat on Excess Return
Reference	0.120903	0.153179	0.422342	0.000000	NaN
SEQ	0.123705	0.151353	0.445947	0.002802	-0.132138
IB	0.129579	0.147601	0.497082	0.008676	0.592840
SALE	0.136770	0.160073	0.503272	0.015867	1.326685
DVT	0.126756	0.137717	0.512257	0.005853	0.103117

Next steps: Generate code with `results_df` View recommended plots New interactive sheet

Question 2

```
# Re-importing the Fama-French data with the correct date parsing
famafrench_monthly = pd.read_csv('FamaFrenchMonthly.csv', dtype={'Date': str})

# Convert the 'Date' column to datetime format if it's in 'YYYYMM' format
famafrench_monthly['Date'] = pd.to_datetime(famafrench_monthly['Date'], format='%Y%m')

# Set 'Date' as the index
famafrench_monthly.set_index('Date', inplace=True)

# Verify the DataFrame structure
print(famafrench_monthly.head())
```

```
            Mkt-RF   SMB   HML    RF
Date
1926-07-01    2.96 -2.56 -2.43  0.22
1926-08-01    2.64 -1.17  3.82  0.25
1926-09-01    0.36 -1.40  0.13  0.23
1926-10-01   -3.24 -0.09  0.70  0.32
1926-11-01    2.53 -0.10 -0.51  0.31
```

```
# Converting 'Mkt-RF' and 'RF' columns in famafrench_monthly to decimal form if they're in percentages
famafrench_monthly[['Mkt-RF', 'RF']] = famafrench_monthly[['Mkt-RF', 'RF']] / 100

# Filtering portfolio returns for the 1962-2004 date range
portfolio_returns_df = portfolio_returns_copy.loc['2004-12-31':'2023-06-30']

# Setting the Reference portfolio as the Market Cap (MV) portfolio
portfolio_returns_df['Reference'] = portfolio_returns_copy['MV']

# Defining the portfolios to analyze, including the new Reference (MV) portfolio
portfolios_to_analyze = ['Reference', 'SEQ', 'IB', 'SALE', 'DVT']

# Initializing a dictionary to store the results for each portfolio
results = {}

# Defining the risk-free rate as RF for Sharpe Ratio calculation
risk_free_rate = famafrench_monthly.loc['2004-12-31':'2023-06-30', 'RF']

for portfolio in portfolios_to_analyze:
    # Monthly returns for the portfolio
    monthly_returns = portfolio_returns_df[portfolio]

    # Annualized arithmetic return
    annualized_return = monthly_returns.mean() * 12

    # Annualized volatility
    annualized_volatility = monthly_returns.std() * np.sqrt(12)

    # Sharpe Ratio (using annualized return - annualized risk-free rate)
    excess_return = monthly_returns - risk_free_rate
    annualized_excess_return = excess_return.mean() * 12
```

```
        annualized_risk_free_rate = risk_free_rate.mean() * 12
        sharpe_ratio = (annualized_return - annualized_risk_free_rate) / annualized_volatility

        # Excess Return vs. Reference (using MV portfolio as the reference)
        reference_returns = portfolio_returns_df['Reference']
        excess_return_vs_reference = annualized_return - (reference_returns.mean() * 12)

        # t-stat on Excess Return vs. Reference
        difference = monthly_returns - reference_returns
        t_stat = (difference.mean() / difference.std()) * np.sqrt(len(difference))

        # Storing the results in a dictionary
        results[portfolio] = {
            'Annualized Return': annualized_return,
            'Annualized Volatility': annualized_volatility,
            'Sharpe Ratio': sharpe_ratio,
            'Excess Return vs. Reference': excess_return_vs_reference,
            't-stat on Excess Return': t_stat
        }

# Converting results to a DataFrame for display
results_df_2 = pd.DataFrame(results).T
results_df_2.columns = ['Annualized Return', 'Annualized Volatility', 'Sharpe Ratio', 'Excess Return vs. Reference', 't-stat on

# Displaying the results in the format of Table 1
results_df_2
```

```
<ipython-input-203-efebf1a98f40>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  portfolio_returns_df['Reference'] = portfolio_returns_copy['MV']
<ipython-input-203-efebf1a98f40>:41: RuntimeWarning: invalid value encountered in scalar divide
  t_stat = (difference.mean() / difference.std()) * np.sqrt(len(difference))
```

	Annualized Return	Annualized Volatility	Sharpe Ratio	Excess Return vs. Reference	t-stat on Excess Return
Reference	0.114697	0.152592	0.669865	0.000000	NaN
SEQ	0.100814	0.179956	0.490859	-0.013883	-1.002498
IB	0.107114	0.164606	0.574905	-0.007583	-0.911151
SALE	0.114682	0.176352	0.579527	-0.000015	-0.001230
DVT	0.102778	0.161504	0.559099	-0.011919	-0.897354

Next steps: Generate code with `results_df_2` | View recommended plots | New interactive sheet

Comparison between Question 1 (1962-2005) and Question 2 (2005-Present) Values

```
print(results_df)
print(results_df_2)
```

```
           Annualized Return  Annualized Volatility  Sharpe Ratio  \
Reference           0.120903               0.153179      0.422342
SEQ                 0.123705               0.151353      0.445947
IB                  0.129579               0.147601      0.497082
SALE                0.136770               0.160073      0.503272
DVT                 0.126756               0.137717      0.512257

           Excess Return vs. Reference  t-stat on Excess Return
Reference                     0.000000                      NaN
SEQ                           0.002802                -0.132138
IB                            0.008676                 0.592840
SALE                          0.015867                 1.326685
DVT                           0.005853                 0.103117
           Annualized Return  Annualized Volatility  Sharpe Ratio  \
Reference           0.114697               0.152592      0.669865
SEQ                 0.100814               0.179956      0.490859
IB                  0.107114               0.164606      0.574905
SALE                0.114682               0.176352      0.579527
DVT                 0.102778               0.161504      0.559099

           Excess Return vs. Reference  t-stat on Excess Return
Reference                     0.000000                      NaN
SEQ                          -0.013883                -1.002498
```

```
IB          -0.007583          -0.911151
SALE        -0.000015          -0.001230
DVT         -0.011919          -0.897354
```

From 1962-2004 to 2005-2023, notable shifts are evident across annualized return, volatility, Sharpe ratio, and excess returns. In the earlier period, annualized returns were generally higher for all portfolios, with SALE and IB portfolios particularly outperforming, showing excess returns of 0.0159 and 0.0087 over the reference, respectively. In contrast, from 2005 onward, all portfolios underperformed relative to the reference (MV) portfolio, indicated by negative excess returns. Additionally, Sharpe ratios in the latter period for portfolios like SEQ, IB, SALE, and DVT generally decreased, reflecting a reduced risk-adjusted return, except for the Reference portfolio, which maintained a higher Sharpe ratio. This shift implies that while the reference market portfolio continued to offer stable returns with manageable volatility, other factor-based portfolios faced increased volatility and diminished returns, highlighting a more challenging out-of-sample period with weaker relative performance. These changes may reflect significant market events such as the 2008 financial crisis and increased market efficiency, impacting factor-based strategies.

Question 3

```
# Re-importing the Fama-French data with the correct date parsing
famafrench_monthly = pd.read_csv('FamaFrenchMonthly.csv', dtype={'Date': str})

# Convert the 'Date' column to datetime format if it's in 'YYYYMM' format
famafrench_monthly['Date'] = pd.to_datetime(famafrench_monthly['Date'], format='%Y%m')

# Set 'Date' as the index
famafrench_monthly.set_index('Date', inplace=True)

# Verify the DataFrame structure
print(famafrench_monthly.head())
```

```
            Mkt-RF   SMB   HML    RF
Date
1926-07-01    2.96 -2.56 -2.43  0.22
1926-08-01    2.64 -1.17  3.82  0.25
1926-09-01    0.36 -1.40  0.13  0.23
1926-10-01   -3.24 -0.09  0.70  0.32
1926-11-01    2.53 -0.10 -0.51  0.31
```

```
portfolio_returns_copy
```

DATE	SEQ	IB	DVT	SALE	MV
1962-02-28	NaN	NaN	NaN	NaN	0.020102
1962-03-31	NaN	NaN	NaN	NaN	-0.004759
1962-04-30	NaN	NaN	NaN	NaN	-0.063601
1962-05-31	NaN	NaN	NaN	NaN	-0.083937
1962-06-30	NaN	NaN	NaN	NaN	-0.082724
...	...	...	...	...	...
2023-02-28	-0.032509	-0.029029	-0.035920	-0.032264	-0.021617
2023-03-31	-0.014511	0.013424	0.001058	0.002412	0.034546
2023-04-30	0.014383	0.018270	0.011056	0.009112	0.011943
2023-05-31	-0.027272	-0.014010	-0.040489	-0.024559	0.008530
2023-06-30	0.062453	0.058532	0.060668	0.079240	0.068521

737 rows × 5 columns

Next steps: Generate code with portfolio_returns_copy | View recommended plots | New interactive sheet

Setting up the data

```
# Convert Fama-French dates to end of the month
famafrench_monthly.index = pd.to_datetime(famafrench_monthly.index, format="%Y-%m-%d") + pd.offsets.MonthEnd(0)
```

```
# Converting 'Mkt-RF' and 'RF' columns in famafrench_monthly to decimal form if they're in percentages
famafrench_monthly[['Mkt-RF', 'RF']] = famafrench_monthly[['Mkt-RF', 'RF']] / 100

# Filter Fama-French data for the 1962-2004 period
famafrench_period_1962_2004 = famafrench_monthly.loc['1962-02-28':'2004-12-31']


# Create a copy of the portfolio returns DataFrame
portfolio_returns_df2 = portfolio_returns_copy.copy()

# Set the Reference portfolio as the Market Cap (MV) portfolio
portfolio_returns_df2['Reference'] = portfolio_returns_copy['MV']

# Filter portfolio returns for the 1962-2004 period
portfolio_returns_period = portfolio_returns_df2.loc['1962-02-28':'2004-12-31']
```

CAPM Beta and Alpha for 1962-2004

```
# Define the portfolios to analyze
portfolios_to_analyze = ['Reference', 'SEQ', 'IB', 'SALE', 'DVT']

# Define a dictionary to store the results
capm_results_1962_2004 = {}

# Risk-free rate and market excess return for the period
risk_free_rate_period = famafrench_period_1962_2004['RF']
market_excess_return_period = famafrench_period_1962_2004['Mkt-RF']

# Loop over each portfolio and calculate CAPM metrics
for portfolio in portfolios_to_analyze:
    # Monthly returns for the portfolio
    monthly_returns = portfolio_returns_period[portfolio]

    # Calculate excess returns (portfolio return - risk-free rate)
    excess_returns = (monthly_returns - risk_free_rate_period).dropna()

    # Align market excess return with excess returns
    market_excess_return_aligned = market_excess_return_period.reindex(excess_returns.index).dropna()
    excess_returns = excess_returns.reindex(market_excess_return_aligned.index)

    # Calculate Annualized Arithmetic Return
    annualized_return = monthly_returns.mean() * 12

    # Check that excess returns and market returns align after reindexing
    if not excess_returns.empty and not market_excess_return_aligned.empty:
        # CAPM regression: excess returns vs. market excess return
        X = sm.add_constant(market_excess_return_aligned)
        capm_model = sm.OLS(excess_returns, X, missing='drop').fit()

        capm_alpha = capm_model.params['const'] * 12  # Annualize alpha
        capm_beta = capm_model.params['Mkt-RF']
        t_stat_alpha = capm_model.tvalues['const']

        # Store the results in the dictionary
        capm_results_1962_2004[portfolio] = {
            'Annualized Return': annualized_return,
            'CAPM Beta vs. MKT-RF': capm_beta,
            'CAPM Alpha': capm_alpha,
            't-stat on CAPM Alpha': t_stat_alpha
        }
    else:
        # Handle case where data is missing
        capm_results_1962_2004[portfolio] = {
            'Annualized Return': annualized_return,
            'CAPM Beta vs. MKT-RF': np.nan,
            'CAPM Alpha': np.nan,
            't-stat on CAPM Alpha': np.nan
        }

# Display the results for the 1962-2004 period
capm_results_df_1962_2004 = pd.DataFrame(capm_results_1962_2004).T
capm_results_df_1962_2004
```

	Annualized Return	CAPM Beta vs. MKT-RF	CAPM Alpha	t-stat on CAPM Alpha
Reference	0.120903	0.981743	0.009616	6.138082
SEQ	0.123705	0.930235	0.011847	1.596202
IB	0.129579	0.916747	0.017456	2.742821
SALE	0.136770	0.980327	0.020797	2.603566
DVT	0.126756	0.823356	0.020288	2.497101

Next steps: Generate code with capm_results_df_1962_2004 View recommended plots New interactive sheet

These results provide insight into each portfolio's performance characteristics over the 1962–2004 period, measured against the Fama-French MKT-RF factor.

- Annualized Return:

The portfolios show a range of returns, with SALE having the highest at 13.68% and Reference the lowest at 12.09%. This suggests that fundamental indexing metrics (like Sales or Dividends) delivered higher returns than the cap-weighted market reference.

- CAPM Beta vs. MKT-RF:

All portfolios have betas close to 1, indicating a level of market sensitivity similar to the MKT-RF benchmark. However, DVT shows a notably lower beta of 0.823, suggesting less volatility and exposure to market movements compared to other portfolios.

- CAPM Alpha:

SALE and DVT portfolios show the highest CAPM alphas (2.08% and 2.03% annualized, respectively), indicating they achieved returns above the market benchmark. This suggests these portfolios provided additional value not explained by market exposure alone.

- t-stat on CAPM Alpha:

The Reference portfolio has a high t-statistic of 6.14, making its alpha statistically significant. SALE, IB, and DVT also show significant t-statistics above 2.0, reinforcing that these portfolios' alphas are likely not due to random variation. The lower t-stat of SEQ suggests its alpha might be less reliable in terms of statistical significance.

In summary, fundamental portfolios (especially SALE and DVT) outperformed the cap-weighted reference both in returns and alpha, with relatively stable market exposure. These findings highlight the potential value of fundamental indexing metrics in delivering returns beyond traditional market exposure.

CAPM Beta and Alpha for 2005-Present

```
# Re-importing the Fama-French data with the correct date parsing
famafrench_monthly = pd.read_csv('FamaFrenchMonthly.csv', dtype={'Date': str})

# Convert the 'Date' column to datetime format if it's in 'YYYYMM' format
famafrench_monthly['Date'] = pd.to_datetime(famafrench_monthly['Date'], format='%Y%m')

# Set 'Date' as the index
famafrench_monthly.set_index('Date', inplace=True)

# Verify the DataFrame structure
print(famafrench_monthly)
```

```
            Mkt-RF   SMB   HML    RF
Date
1926-07-01    2.96 -2.56 -2.43  0.22
1926-08-01    2.64 -1.17  3.82  0.25
1926-09-01    0.36 -1.40  0.13  0.23
1926-10-01   -3.24 -0.09  0.70  0.32
1926-11-01    2.53 -0.10 -0.51  0.31
...            ...   ...   ...   ...
2024-03-01    2.83 -2.51  4.21  0.43
2024-04-01   -4.67 -2.39 -0.52  0.47
2024-05-01    4.34  0.78 -1.66  0.44
2024-06-01    2.77 -3.06 -3.31  0.41
2024-07-01    1.24  6.84  5.70  0.45

[1177 rows x 4 columns]
```

```
# Convert Fama-French dates to end of the month
famafrench_monthly.index = pd.to_datetime(famafrench_monthly.index, format="%Y-%m-%d") + pd.offsets.MonthEnd(0)
```

```
# Converting 'Mkt-RF' and 'RF' columns in famafrench_monthly to decimal form if they're in percentages
famafrench_monthly[['Mkt-RF', 'RF']] = famafrench_monthly[['Mkt-RF', 'RF']] / 100

# Filter Fama-French data for the 2005-2023 period
famafrench_period_2005_2023 = famafrench_monthly.loc['2004-12-31':'2023-06-30']


# Create a copy of the portfolio returns DataFrame
portfolio_returns_df3 = portfolio_returns_copy.copy()

# Set the Reference portfolio as the Market Cap (MV) portfolio
portfolio_returns_df3['Reference'] = portfolio_returns_copy['MV']

# Filter portfolio returns for the 2005-2023 period
portfolio_returns_period = portfolio_returns_df3.loc['2004-12-31':'2023-06-30']


# Define the portfolios to analyze
portfolios_to_analyze = ['Reference', 'SEQ', 'IB', 'SALE', 'DVT']

# Define a dictionary to store the results
capm_results_2005_2023 = {}

# Risk-free rate and market excess return for the period
risk_free_rate_period = famafrench_period_2005_2023['RF']
market_excess_return_period = famafrench_period_2005_2023['Mkt-RF']

# Loop over each portfolio and calculate CAPM metrics
for portfolio in portfolios_to_analyze:
    # Monthly returns for the portfolio
    monthly_returns = portfolio_returns_period[portfolio]

    # Calculate excess returns (portfolio return - risk-free rate)
    excess_returns = (monthly_returns - risk_free_rate_period).dropna()

    # Align market excess return with excess returns
    market_excess_return_aligned = market_excess_return_period.reindex(excess_returns.index).dropna()
    excess_returns = excess_returns.reindex(market_excess_return_aligned.index)

    # Calculate Annualized Arithmetic Return
    annualized_return = monthly_returns.mean() * 12

    # Check that excess returns and market returns align after reindexing
    if not excess_returns.empty and not market_excess_return_aligned.empty:
        # CAPM regression: excess returns vs. market excess return
        X = sm.add_constant(market_excess_return_aligned)
        capm_model = sm.OLS(excess_returns, X, missing='drop').fit()

        capm_alpha = capm_model.params['const'] * 12  # Annualize alpha
        capm_beta = capm_model.params['Mkt-RF']
        t_stat_alpha = capm_model.tvalues['const']

        # Store the results in the dictionary
        capm_results_2005_2023[portfolio] = {
            'Annualized Return': annualized_return,
            'CAPM Beta vs. MKT-RF': capm_beta,
            'CAPM Alpha': capm_alpha,
            't-stat on CAPM Alpha': t_stat_alpha
        }
    else:
        # Handle case where data is missing
        capm_results_2005_2023[portfolio] = {
            'Annualized Return': annualized_return,
            'CAPM Beta vs. MKT-RF': np.nan,
            'CAPM Alpha': np.nan,
            't-stat on CAPM Alpha': np.nan
        }

# Display the results for the 1962-2004 period
capm_results_df_2005_2023 = pd.DataFrame(capm_results_2005_2023).T
capm_results_df_2005_2023
```

	Annualized Return	CAPM Beta vs. MKT-RF	CAPM Alpha	t-stat on CAPM Alpha
Reference	0.114697	0.976937	0.011975	8.268432
SEQ	0.100814	1.099246	-0.013202	-1.023753
IB	0.107114	1.032249	-0.000716	-0.088897
SALE	0.114682	1.088819	0.001628	0.145315
DVT	0.102778	0.970145	0.000683	0.051140

Next steps: Generate code with capm_results_df_2005_2023 View recommended plots New interactive sheet

- Annualized Return:

The Reference portfolio achieves the highest return at 11.47%, with SALE very close at 11.47%, suggesting that market cap weighting and sales-based indexing provided comparable returns during this period. Other portfolios, such as SEQ and DVT, have lower annualized returns, around 10-10.3%, indicating slightly weaker performance compared to the cap-weighted reference.

- CAPM Beta vs. MKT-RF:

Most portfolios have betas near 1, indicating similar market exposure to the MKT-RF benchmark. However, SEQ and SALE have betas slightly above 1 (1.10 and 1.09, respectively), suggesting they are more sensitive to market movements.

- CAPM Alpha:

The Reference portfolio shows a positive CAPM alpha of 1.20%, which is statistically significant, with a high t-stat of 8.27. Other portfolios exhibit near-zero or negative alphas. Notably, SEQ has a negative alpha of -1.32%, though it is not statistically significant (t-stat of -1.02), while IB and DVT also have near-zero alphas, indicating they performed similarly to market expectations.

- t-stat on CAPM Alpha:

The Reference portfolio stands out with a highly significant t-statistic of 8.27, confirming that its alpha is statistically significant. The other portfolios' t-stats are low, indicating that their alphas are not statistically significant, with values close to zero (e.g., DVT at 0.05), suggesting little additional value beyond market exposure.

In summary, the Reference portfolio delivered the strongest and statistically significant CAPM alpha, while other fundamental indexing metrics showed limited or no excess return relative to market exposure, with minimal statistical significance in their alphas.

Question 4

```
# Re-importing the Fama-French data with the correct date parsing
famafrench_monthly = pd.read_csv('FamaFrenchMonthly.csv', dtype={'Date': str})

# Convert the 'Date' column to datetime format if it's in 'YYYYMM' format
famafrench_monthly['Date'] = pd.to_datetime(famafrench_monthly['Date'], format='%Y%m')

# Set 'Date' as the index
famafrench_monthly.set_index('Date', inplace=True)

# Verify the DataFrame structure
print(famafrench_monthly.head())
```

```
            Mkt-RF   SMB   HML    RF
Date
1926-07-01    2.96 -2.56 -2.43  0.22
1926-08-01    2.64 -1.17  3.82  0.25
1926-09-01    0.36 -1.40  0.13  0.23
1926-10-01   -3.24 -0.09  0.70  0.32
1926-11-01    2.53 -0.10 -0.51  0.31
```

Data Cleaning and Processing

```
# Convert Fama-French dates to end of the month
famafrench_monthly.index = pd.to_datetime(famafrench_monthly.index, format="%Y-%m-%d") + pd.offsets.MonthEnd(0)

# Converting 'Mkt-RF' and 'RF' columns in famafrench_monthly to decimal form if they're in percentages
famafrench_monthly[['Mkt-RF', 'RF']] = famafrench_monthly[['Mkt-RF', 'RF']] / 100
```

```
# Filter Fama-French data for the 1962-2004 period
famafrench_period_1962_2004 = famafrench_monthly.loc['1962-02-28':'2004-12-31']

# Create a copy of the portfolio returns DataFrame
portfolio_returns_df_1962_2004 = portfolio_returns_copy.copy()
portfolio_returns_df_1962_2004['Reference'] = portfolio_returns_copy['MV']
portfolio_returns_period = portfolio_returns_df_1962_2004.loc['1962-02-28':'2004-12-31']
```

FF3 Metrics for 1962-2004 Period

```
# Define portfolios to analyze
portfolios_to_analyze = ['Reference', 'SEQ', 'IB', 'SALE', 'DVT']

# Define a dictionary to store the results
ff3_results_1962_2004 = {}

# Fama-French factors for the period
risk_free_rate_period = famafrench_period_1962_2004['RF']
market_excess_return_period = famafrench_period_1962_2004['Mkt-RF']
smb_period = famafrench_period_1962_2004['SMB']
hml_period = famafrench_period_1962_2004['HML']

# Loop over each portfolio
for portfolio in portfolios_to_analyze:
    # Monthly returns for the portfolio
    monthly_returns = portfolio_returns_period[portfolio]

    # Calculate excess returns
    excess_returns = (monthly_returns - risk_free_rate_period).dropna()

    # Align the Fama-French factors with excess returns
    market_excess_aligned = market_excess_return_period.reindex(excess_returns.index).dropna()
    smb_aligned = smb_period.reindex(excess_returns.index).dropna()
    hml_aligned = hml_period.reindex(excess_returns.index).dropna()
    excess_returns = excess_returns.reindex(market_excess_aligned.index)

    # Combine factors into one DataFrame for regression
    ff_factors = pd.DataFrame({
        'Mkt-RF': market_excess_aligned,
        'SMB': smb_aligned,
        'HML': hml_aligned
    })

    # Annualized Arithmetic Return
    annualized_return = monthly_returns.mean() * 12

    # Check that all data aligns for regression
    if not excess_returns.empty and not ff_factors.empty:
        # FF3 regression: excess returns vs. Mkt-RF, SMB, and HML
        X = sm.add_constant(ff_factors)
        ff3_model = sm.OLS(excess_returns, X, missing='drop').fit()

        # Retrieve regression coefficients and t-stats
        ff3_alpha = ff3_model.params['const'] * 12  # Annualize alpha
        beta_mkt_rf = ff3_model.params['Mkt-RF']
        beta_smb = ff3_model.params['SMB']
        beta_hml = ff3_model.params['HML']
        t_stat_alpha = ff3_model.tvalues['const']

        # Store the results in the dictionary
        ff3_results_1962_2004[portfolio] = {
            'Annualized Return': annualized_return,
            'FF3 Beta MKT-RF': beta_mkt_rf,
            'FF3 Beta SMB': beta_smb,
            'FF3 Beta HML': beta_hml,
            'FF3 Alpha': ff3_alpha,
            't-stat on FF3 Alpha': t_stat_alpha
        }
    else:
        # Handle case where data is missing
        ff3_results_1962_2004[portfolio] = {
            'Annualized Return': annualized_return,
            'FF3 Beta MKT-RF': np.nan,
            'FF3 Beta SMB': np.nan,
            'FF3 Beta HML': np.nan,
```

```
            'FF3 Alpha': np.nan,
            't-stat on FF3 Alpha': np.nan
        }

# Convert results to a DataFrame for display
ff3_results_df_1962_2004 = pd.DataFrame(ff3_results_1962_2004).T
ff3_results_df_1962_2004
```

	Annualized Return	FF3 Beta MKT-RF	FF3 Beta SMB	FF3 Beta HML	FF3 Alpha	t-stat on FF3 Alpha
Reference	0.120903	0.996623	-0.000844	-0.000107	0.011678	15.228012
SEQ	0.123705	1.030383	-0.000715	0.003451	-0.011537	-2.126390
IB	0.129579	1.004096	-0.001060	0.002676	-0.000064	-0.013502
SALE	0.136770	1.065250	0.000638	0.004016	-0.009024	-1.524097
DVT	0.126756	0.945274	-0.001259	0.003922	-0.005862	-1.092001

Next steps: Generate code with ff3_results_df_1962_2004 View recommended plots New interactive sheet

- Annualized Return:

SALE achieved the highest annualized return at 13.68%, followed by IB at 12.96% and DVT at 12.68%, indicating that certain fundamental factors (such as sales and dividends) provided higher returns during this period. The Reference portfolio, with a return of 12.09%, had slightly lower returns than some of the alternative metrics.

- FF3 Beta MKT-RF:

All portfolios have betas close to 1, indicating similar sensitivity to the MKT-RF factor. SALE has the highest beta (1.065), showing slightly greater market sensitivity, while DVT has the lowest at 0.945, suggesting slightly less exposure to market movements.

- FF3 Beta SMB and HML:

SMB (Small Minus Big): All portfolios have nearly zero or slightly negative SMB betas, indicating minimal sensitivity to size factor (large-cap vs. small-cap). HML (High Minus Low): The portfolios also exhibit near-zero HML betas, suggesting minimal exposure to the value vs. growth factor during this period.

- FF3 Alpha:

The Reference portfolio shows a positive FF3 alpha of 1.17%, with a highly significant t-statistic of 15.23, indicating a strong excess return beyond what is explained by market, size, and value factors. Other portfolios have negative alphas. Notably, SEQ and SALE have alphas of -1.15% and -0.90%, respectively, with SEQ showing statistical significance (t-stat of -2.13), suggesting underperformance relative to the FF3 model expectations.

- t-stat on FF3 Alpha:

The Reference portfolio's t-statistic of 15.23 confirms its alpha is statistically significant. SEQ has a t-statistic of -2.13, making its negative alpha significant, suggesting underperformance. Other portfolios have low t-stats, indicating that their alphas are not statistically significant.

In summary, during the 1962-2004 period, the Reference portfolio provided a statistically significant positive alpha, outperforming alternative fundamental metrics. Portfolios based on fundamental factors like SEQ and SALE exhibited slightly higher market sensitivity but delivered lower, and in some cases negative, alphas relative to the FF3 model.

FF3 Metrics for 2005-2023 Period

```
# Filter data for the 2005-present period
famafrench_period_2005_present = famafrench_monthly.loc['2005-01-31':]
portfolio_returns_df_2005_present = portfolio_returns_copy.copy()
portfolio_returns_df_2005_present['Reference'] = portfolio_returns_copy['MV']
portfolio_returns_period_2005 = portfolio_returns_df_2005_present.loc['2005-01-31':]

# Initialize dictionary to store results for 2005-present
ff3_results_2005_present = {}

# Fama-French factors for the period
risk_free_rate_2005 = famafrench_period_2005_present['RF']
market_excess_return_2005 = famafrench_period_2005_present['Mkt-RF']
smb_2005 = famafrench_period_2005_present['SMB']
hml_2005 = famafrench_period_2005_present['HML']

# Calculate FF3 metrics for each portfolio
```

```
for portfolio in portfolios_to_analyze:
    # Monthly returns
    monthly_returns = portfolio_returns_period_2005[portfolio]
    excess_returns = (monthly_returns - risk_free_rate_2005).dropna()

    # Align factors
    market_excess_aligned = market_excess_return_2005.reindex(excess_returns.index).dropna()
    smb_aligned = smb_2005.reindex(excess_returns.index).dropna()
    hml_aligned = hml_2005.reindex(excess_returns.index).dropna()
    excess_returns = excess_returns.reindex(market_excess_aligned.index)

    # FF factors for regression
    ff_factors_2005 = pd.DataFrame({
        'Mkt-RF': market_excess_aligned,
        'SMB': smb_aligned,
        'HML': hml_aligned
    })

    # Calculate Annualized Arithmetic Return
    annualized_return = monthly_returns.mean() * 12

    # FF3 regression
    if not excess_returns.empty and not ff_factors_2005.empty:
        X = sm.add_constant(ff_factors_2005)
        ff3_model = sm.OLS(excess_returns, X, missing='drop').fit()

        ff3_alpha = ff3_model.params['const'] * 12
        beta_mkt_rf = ff3_model.params['Mkt-RF']
        beta_smb = ff3_model.params['SMB']
        beta_hml = ff3_model.params['HML']
        t_stat_alpha = ff3_model.tvalues['const']

        ff3_results_2005_present[portfolio] = {
            'Annualized Return': annualized_return,
            'FF3 Beta MKT-RF': beta_mkt_rf,
            'FF3 Beta SMB': beta_smb,
            'FF3 Beta HML': beta_hml,
            'FF3 Alpha': ff3_alpha,
            't-stat on FF3 Alpha': t_stat_alpha
        }
    else:
        ff3_results_2005_present[portfolio] = {
            'Annualized Return': annualized_return,
            'FF3 Beta MKT-RF': np.nan,
            'FF3 Beta SMB': np.nan,
            'FF3 Beta HML': np.nan,
            'FF3 Alpha': np.nan,
            't-stat on FF3 Alpha': np.nan
        }

# Display results
ff3_results_df_2005_present = pd.DataFrame(ff3_results_2005_present).T
ff3_results_df_2005_present
```

	Annualized Return	FF3 Beta MKT-RF	FF3 Beta SMB	FF3 Beta HML	FF3 Alpha	t-stat on FF3 Alpha
Reference	0.113277	0.990795	-0.000672	-0.000145	0.010724	18.071917
SEQ	0.099205	1.056831	-0.000095	0.004081	-0.003310	-0.452343
IB	0.105487	1.025431	-0.000875	0.002095	0.003144	0.545381
SALE	0.113187	1.044235	0.000678	0.002986	0.009969	1.247593
DVT	0.101317	0.952838	-0.001273	0.003744	0.008122	0.910979

Next steps: Generate code with `ff3_results_df_2005_present` View recommended plots New interactive sheet

- Annualized Return:

Reference and SALE portfolios have the highest annualized returns, at 11.33% and 11.32%, respectively, indicating that traditional market cap weighting and sales-based indexing performed comparably during this period. SEQ and DVT had lower annualized returns of around 9.9-10.1%, suggesting weaker performance relative to the Reference and SALE portfolios.

- FF3 Beta MKT-RF:

All portfolios have betas close to 1, with SEQ and SALE showing slightly higher betas (1.057 and 1.044), indicating slightly more sensitivity to the market factor. DVT has the lowest beta (0.953), suggesting it experienced less exposure to market movements.

- FF3 Beta SMB and HML:

SMB (Small Minus Big): All portfolios show near-zero or slightly negative exposure to SMB, indicating minimal sensitivity to the size factor. HML (High Minus Low): The HML betas are also close to zero, suggesting that these portfolios had minimal exposure to the value vs. growth factor.

- FF3 Alpha:

Reference has the highest FF3 alpha at 1.07%, with a highly significant t-statistic of 18.07, showing a strong excess return beyond what is explained by the FF3 factors. SALE and DVT also show positive alphas of 0.99% and 0.81%, respectively, though with lower statistical significance. SEQ has a slightly negative alpha of -0.33%, while IB has a small positive alpha of 0.31%, neither of which are statistically significant.

- t-stat on FF3 Alpha:

The Reference portfolio's alpha is highly statistically significant (t-stat of 18.07), reinforcing its consistent excess return. SALE and DVT have moderate t-stats of 1.25 and 0.91, indicating some positive alpha but without strong significance. The other portfolios, SEQ and IB, have very low t-stats, confirming that their alphas are not statistically significant.

In summary, during the 2005-2023 period, the Reference portfolio provided a statistically significant positive alpha, outperforming fundamental indexing metrics. While SALE and DVT displayed positive, albeit less significant, alphas, other metrics like SEQ and IB showed minimal excess returns and low statistical significance, suggesting they performed similarly to the FF3 model expectations without notable outperformance.

Question 5

Comparison of FF3 Alpha and CAPM Alpha:

Across both periods (1962-2004 and 2005-2023), the FF3 alpha for the Reference portfolio generally remains positive and significant, similar to the CAPM alpha results. However, in the FF3 model, some of the fundamental portfolios exhibit lower or even negative alphas, indicating that the inclusion of size (SMB) and value (HML) factors helps explain portions of the excess return that were attributed to alpha in the simpler CAPM model. Specifically, portfolios like SEQ and SALE show lower or negative alphas under FF3 compared to CAPM, suggesting that their excess returns relative to the Reference may be partly explained by exposure to size and value factors rather than purely outperforming the market.

- Factors Explaining "Excess Return vs. Reference":

Size and Value Exposure: Many fundamental portfolios show non-zero (though often small) loadings on the SMB and HML factors, indicating some sensitivity to size and value. For instance, SEQ and DVT portfolios exhibit a slight positive sensitivity to the HML factor, which may indicate a tilt toward value stocks, partially explaining their returns relative to the Reference. Market Sensitivity: Portfolios like SALE show a beta slightly greater than 1 on the MKT-RF factor, suggesting higher sensitivity to overall market movements. This heightened exposure may contribute to periods of higher returns during strong market performance, though it also implies greater risk during downturns.

- Out-of-Sample FF3 Results:

In the 2005-2023 period, the Reference portfolio maintained a significant positive alpha, indicating it continued to deliver returns above what is explained by the FF3 factors. However, fundamental portfolios like SEQ and IB showed either insignificant or negative alphas out-of-sample, contrasting with some positive alpha observed in the 1962-2004 period. This shift suggests that fundamental indexing strategies faced challenges in achieving significant outperformance in recent years, as the explanatory power of FF3 factors appears to have strengthened or the advantages of fundamental metrics have diminished in comparison to cap-weighted indexes.

- Implications for "Fundamental Indexation":

The analysis indicates that while fundamental indexation strategies (e.g., weighting by sales, book value, or dividends) can sometimes yield