lstm_weighted

September 19, 2023

[1]:
```
import numpy as np
import pandas as pd
from lstm_functions import *
from lost_functions import *
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import yfinance as yf
```

1 Reading and storing the Data

[2]:
```
xls = pd.ExcelFile('data/data_for_testing.xlsx')
all_data = {}
# This is too much data to load into memory at once
# for sheet in xls.sheet_names:
#     all_data[sheet] = pd.read_excel(xls, sheet_name=sheet)
for sheet in ["XOM", "SHW", "UPS", "DUK", "UNH", "JPM", "AMZN", "AAPL", "META",␣
 ↪"AMT"]:
    data = pd.read_excel(xls, sheet_name=sheet).set_index('Date')
    # Resample to monthly data as a simple way to reduce the number of data␣
 ↪points
    # Daily data is too much and take too long to train
    monthly_data = data.resample('M').last().reset_index()
    all_data[sheet] = monthly_data
```

[3]:
```
all_data['XOM'].head()
```

[3]:

	Date	Open	High	Low	Close	Adj Close	Volume	Sector	\
0	1962-01-31	0.0	1.656250	1.636719	1.656250	0.101445	1718400	Energy	
1	1962-02-28	0.0	1.757813	1.726563	1.730469	0.107163	2131200	Energy	
2	1962-03-31	0.0	1.710938	1.703125	1.707031	0.105711	809600	Energy	
3	1962-04-30	0.0	1.710938	1.671875	1.671875	0.103534	1222400	Energy	
4	1962-05-31	0.0	1.632813	1.609375	1.625000	0.101744	3190400	Energy	

	Ticker
0	XOM
1	XOM

```
2    XOM
3    XOM
4    XOM
```

[4]:
```
all_data["AAPL"].columns
```

[4]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Sector',
       'Ticker'],
      dtype='object')
```

[5]:
```
final_importance_values = {}
final_predictions = {}
# 30 is not a good number of batches, but it's a start for testing
# 60 is a good number of batches, but it takes a long time to train
time_steps = 30
features = 6
```

[12]:
```
# Loop through each stock data
for ticker, data in all_data.items():

    # Drop non-numeric columns
    data = data.drop(columns=['Sector', 'Ticker', 'Date'])  # Assuming 'Date' is the index
    lstm_model = LstmBuilder(time_step=time_steps, loss=huber_loss)
    model = lstm_model.create_model(features=features)
    scaler = MinMaxScaler()
    normalized_data = scaler.fit_transform(data)
    X, y = lstm_model.create_sequences(normalized_data)
    X_train, X_test, y_train, y_test = lstm_model.split_data(X,y)

    print("Working on: " + ticker)
    model.fit(X_train, y_train, epochs=3, batch_size=4, validation_split=0.2, verbose=0)
    '''
    Batch Size: Refers to the number of training examples utilized in one iteration. When you set batch_size=32, it means the model takes 32 sequences at a time and updates weights once after computing the loss of the entire batch.
    Input Shape (60, 6): Refers to the shape of a single input sequence.
    60 indicates the number of time steps in each sequence. In your case, each sequence contains data from 60 days.
    6 refers to the number of features ('Open', 'High', 'Low', 'Close', 'Adj Close', and 'Volume').
    So, when you train your LSTM, it takes in 32 sequences (if we consider batch_size=32) at a time, and each of those sequences contains 60 time steps with 6 features for each time step.
```

```
    In simpler terms:

    Input Shape: Shape of a single sequence that you feed into the model.
    Batch Size: Number of sequences you feed into the model at one go.
    These two are different parameters and have different roles in the training␣
 ↪process. The batch size is related to how you update the weights during␣
 ↪training, whereas the input shape is related to the structure and size of␣
 ↪your input data.
    '''

    # Predict the next day value
    last_days = normalized_data[-time_steps:].reshape(1, time_steps, features)
    prediction_next_day = model.predict(last_days)
    prediction_next_day_actual = scaler.inverse_transform(prediction_next_day)
    final_predictions[ticker] = prediction_next_day_actual.flatten()
    print(f"Predicted value for {ticker}: {prediction_next_day_actual.
 ↪flatten()}")


    # Extracting importance
    dense_weights = model.layers[-1].get_weights()[0]

    # Think about to use sum or mean and to use abs() or not
    feature_weights = dense_weights.sum(axis=0)
    weighted_importance = prediction_next_day.flatten() * feature_weights
    final_importance_value = np.sum(weighted_importance)  # Final importance as␣
 ↪a single value
    print(f"Importance value for {ticker}: {final_importance_value}")

    # Store the importance value in the dictionary
    final_importance_values[ticker] = final_importance_value

print(final_importance_values)
```

```
WARNING:tensorflow:Layer lstm_6 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Working on: XOM

2023-09-18 23:31:31.716689: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:33:47.230182: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 242ms/step
Predicted value for XOM: [-6.5087250e+02  4.8031528e+03 -1.3332198e+03
```

```
1.3049038e+04
  8.0209800e+03 -1.5735314e+09]
Importance value for XOM: -24.0202693939209
WARNING:tensorflow:Layer lstm_7 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.

2023-09-18 23:38:27.611747: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: SHW

2023-09-18 23:38:28.272546: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:39:19.200567: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 241ms/step
Predicted value for SHW: [3.5141013e+02 3.8164407e+02 2.6351364e+02
2.8656302e+02 2.8523553e+02
 1.8751348e+07]
Importance value for SHW: 5.5614213943481445
WARNING:tensorflow:Layer lstm_8 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.

2023-09-18 23:41:05.599648: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: UPS

2023-09-18 23:41:06.371764: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:41:55.870906: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

WARNING:tensorflow:5 out of the last 5 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x2f3ce75e0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 250ms/step
Predicted value for UPS: [ 1.8062826e+03 -7.7277783e+02  1.5787919e+03
```

```
1.6159189e+03
  5.7719336e+02  1.3438464e+08]
Importance value for UPS: 5.230861663818359
WARNING:tensorflow:Layer lstm_9 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.

2023-09-18 23:46:44.986558: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: DUK

2023-09-18 23:46:45.660580: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:47:35.234621: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

WARNING:tensorflow:6 out of the last 6 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x2f98134c0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 0s 245ms/step
Predicted value for DUK: [ 7.1139160e+02  1.1805482e+03  9.0841254e+02
7.9315942e+02
 -2.2108261e+02 -8.2573072e+07]
Importance value for DUK: -5.3437275886535645
WARNING:tensorflow:Layer lstm_10 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.

2023-09-18 23:49:17.867055: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: UNH

2023-09-18 23:49:18.538017: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:50:02.211735: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
```

```
1/1 [==============================] - 0s 246ms/step
Predicted value for UNH: [ 2.4961372e+02  5.6480835e+02  4.0200919e+02
4.1667096e+02
 -2.7740051e+02  1.6790776e+07]
Importance value for UNH: 0.14171358942985535
WARNING:tensorflow:Layer lstm_11 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.

2023-09-18 23:51:37.168529: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: JPM

2023-09-18 23:51:37.864359: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:52:27.500101: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 263ms/step

2023-09-18 23:54:15.884728: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Predicted value for JPM: [-1.0990291e+02  2.6118631e+02  6.9082080e+02
2.7098184e+02
  6.1224982e+02  1.9892425e+06]
Importance value for JPM: 8.93437385559082
WARNING:tensorflow:Layer lstm_12 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.
Working on: AMZN

2023-09-18 23:54:17.732835: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:54:48.319637: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 238ms/step
Predicted value for AMZN: [ 4.42947044e+01  1.07595116e+02  9.56798401e+01
1.05455544e+02
 -1.06948196e+02  7.81083350e+06]
Importance value for AMZN: 2.087344169616699
WARNING:tensorflow:Layer lstm_13 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.
```

```
2023-09-18 23:55:48.850833: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: AAPL

2023-09-18 23:55:49.587231: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-18 23:57:22.088482: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 253ms/step
Predicted value for AAPL: [-8.2657318e+00 -6.3893677e+01 -6.0852151e+00
3.0017544e+01
  9.4743820e+01  4.5861636e+07]
Importance value for AAPL: 0.1682691127061844
WARNING:tensorflow:Layer lstm_14 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.

2023-09-19 00:00:35.939604: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: META

2023-09-19 00:00:36.706082: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
2023-09-19 00:00:57.302785: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 244ms/step
Predicted value for META: [1.18332443e+02 2.13677582e+02 9.51574478e+01
1.62895233e+02
 1.01893654e+02 3.37105720e+07]
Importance value for META: 1.3775031566619873
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.

2023-09-19 00:01:39.657455: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

Working on: AMT

2023-09-19 00:01:40.519221: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
```

```
2023-09-19 00:02:33.599736: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 246ms/step
Predicted value for AMT: [2.3515135e+02 2.0364160e+02 2.1137306e+02
2.2593141e+02 2.3491467e+02
 1.6362810e+07]
Importance value for AMT: -2.9676592350006104
{'XOM': -24.02027, 'SHW': 5.5614214, 'UPS': 5.2308617, 'DUK': -5.3437276, 'UNH':
0.14171359, 'JPM': 8.934374, 'AMZN': 2.0873442, 'AAPL': 0.16826911, 'META':
1.3775032, 'AMT': -2.9676592}

2023-09-19 00:04:27.422454: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114]
Plugin optimizer for device_type GPU is enabled.
```

[13]:
```
final_importance_values
```

[13]:
```
{'XOM': -24.02027,
 'SHW': 5.5614214,
 'UPS': 5.2308617,
 'DUK': -5.3437276,
 'UNH': 0.14171359,
 'JPM': 8.934374,
 'AMZN': 2.0873442,
 'AAPL': 0.16826911,
 'META': 1.3775032,
 'AMT': -2.9676592}
```

[44]:
```
importance_values = np.array(list(final_importance_values.values()))
```

2 Run this if we want a arbitrage strategy

Each weight will be -1 to 1, the sum is 0

[45]:
```
# 1. Scale the values to [-1, 1]
arbitrage_scaled_importance = 2 * (importance_values - np.
 ↪min(importance_values)) / (np.max(importance_values) - np.
 ↪min(importance_values)) - 1

# 2. Ensure the sum is zero
arbitrage_normalized_importance = arbitrage_scaled_importance - np.
 ↪mean(arbitrage_scaled_importance)

# Convert back to dictionary
arbitrage_ticker_to_importance = dict(zip(final_importance_values.keys(),␣
 ↪arbitrage_normalized_importance))
```

```
print(arbitrage_ticker_to_importance)
```

```
{'XOM': -1.4041877, 'SHW': 0.39110965, 'UPS': 0.37104803, 'DUK': -0.27071816,
'UNH': 0.062190354, 'JPM': 0.59581226, 'AMZN': 0.18026954, 'AAPL': 0.063801944,
'META': 0.13718969, 'AMT': -0.1265158}
```

3 Run this instead if we want a normal strategy

Each weight will be 0 to 1, the sum is 1

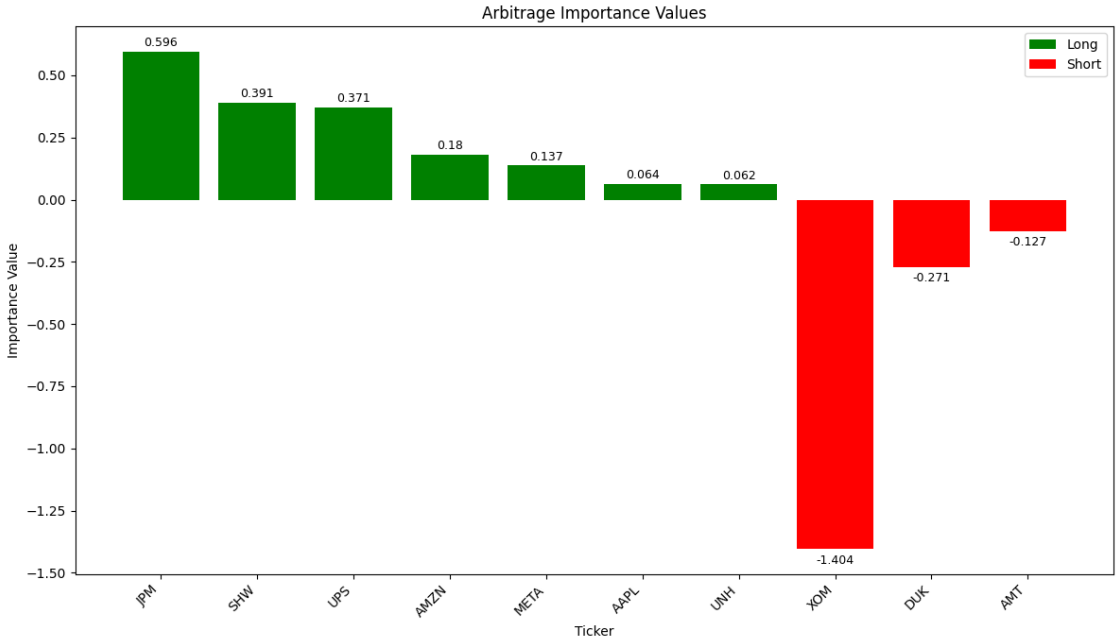
[46]:
```
def softmax(x):
    """Compute softmax values for each sets of scores in x."""
    e_x = np.exp(x - np.max(x))  # subtract max to avoid potential overflow
    return e_x / e_x.sum(axis=0)

# Convert the importance values to probabilities using softmax
probabilities = softmax(importance_values)

# Convert back to dictionary
normalized_ticker_to_importance = dict(zip(final_importance_values.keys(),␣
 ↪probabilities))

print(normalized_ticker_to_importance)
```

```
{'XOM': 4.595538e-15, 'SHW': 0.03232224, 'UPS': 0.02322422, 'DUK':
5.9354767e-07, 'UNH': 0.00014313703, 'JPM': 0.9426621, 'AMZN': 0.0010016797,
'AAPL': 0.00014698911, 'META': 0.0004925484, 'AMT': 6.3880584e-06}
```

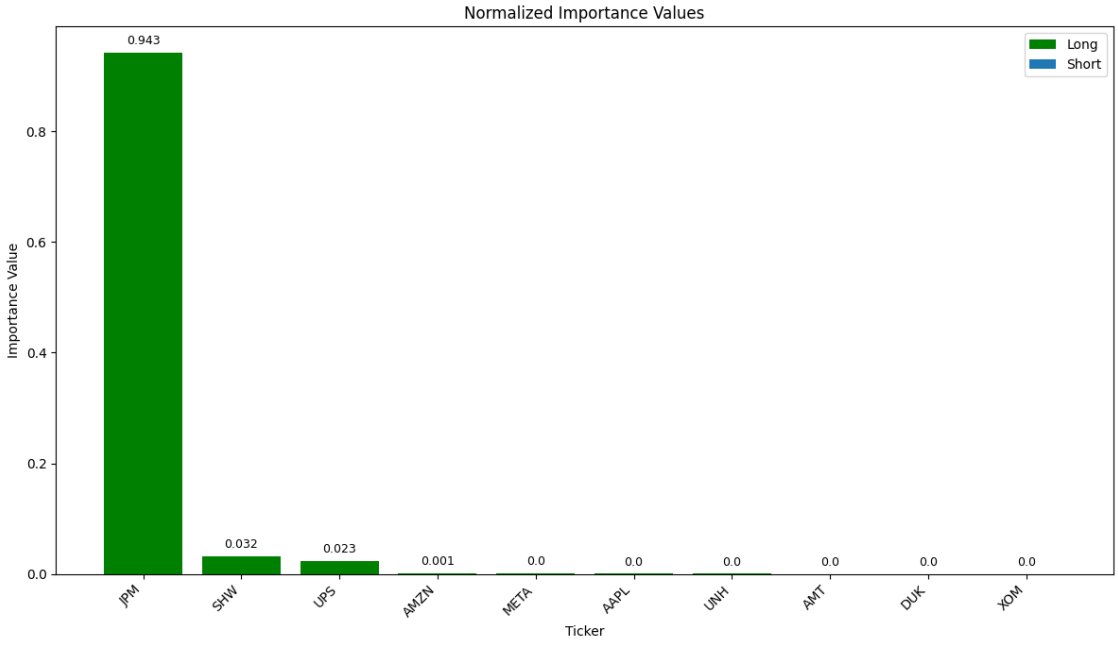
[47]:
```
def plot_importance(normalized_ticker_to_importance =␣
 ↪normalized_ticker_to_importance, title='Normalized Importance Values'):
# Split the tickers and importance values based on positive and negative values
    long_positions = {k: v for k, v in normalized_ticker_to_importance.items()␣
 ↪if v > 0}
    short_positions = {k: v for k, v in normalized_ticker_to_importance.items()␣
 ↪if v <= 0}

    # Sort the positions for better visualization
    sorted_long = dict(sorted(long_positions.items(), key=lambda item: item[1],␣
 ↪reverse=True))
    sorted_short = dict(sorted(short_positions.items(), key=lambda item:␣
 ↪item[1]))

    # Create bar charts
    fig, ax = plt.subplots(figsize=(12, 7))

    # Positive cluster
```

```
    bars_long = ax.bar(sorted_long.keys(), sorted_long.values(), color='g',␣
↪label='Long')

    # Negative cluster
    bars_short = ax.bar(sorted_short.keys(), sorted_short.values(), color='r',␣
↪label='Short')

    # Rotate x-tick labels for better readability
    plt.xticks(rotation=45, ha='right')

    # Annotate the bars
    for bar in bars_long:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2, yval + 0.01, round(yval, 3),␣
↪ha='center', va='bottom', fontsize=9)

    for bar in bars_short:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2, yval - 0.02, round(yval, 3),␣
↪ha='center', va='top', fontsize=9)

    ax.set_title(title)
    ax.set_ylabel('Importance Value')
    ax.set_xlabel('Ticker')
    ax.legend()

    plt.tight_layout()
    plt.show()
```

[48]:
```
# Plot the arbitrage importance values
plot_importance(arbitrage_ticker_to_importance, title='Arbitrage Importance␣
↪Values')
```



[49]:
```
# Plot the importance values
plot_importance(normalized_ticker_to_importance, title='Normalized Importance␣
↪Values')
```



[32]:
```
spy_data = yf.download('SPY')
spy_monthly = spy_data.resample('M').last()
spy_monthly
```

```
[**********************100%%**********************]  1 of 1 completed
```

[32]:

Date	Open	High	…	Adj Close	Volume
1993-01-31	43.968750	43.968750	…	24.941395	1003200
1993-02-28	44.437500	44.437500	…	25.207489	66200
1993-03-31	45.343750	45.468750	…	25.772102	111600
1993-04-30	44.125000	44.281250	…	25.112648	88500
1993-05-31	45.406250	45.406250	…	25.789946	79100
…	…	…	…	…	…
2023-05-31	418.279999	419.220001	…	414.840332	110811800
2023-06-30	441.440002	444.299988	…	441.721893	104921500
2023-07-31	457.410004	458.160004	…	456.180908	62040400
2023-08-31	451.649994	452.829987	…	448.767059	66084600
2023-09-30	443.049988	444.970001	…	444.635010	25487800

[369 rows x 6 columns]

[33]:
```
all_data
```

[33]:

{'XOM':

	Date	Open	…	Sector	Ticker
0	1962-01-31	0.000000	…	Energy	XOM
1	1962-02-28	0.000000	…	Energy	XOM
2	1962-03-31	0.000000	…	Energy	XOM
3	1962-04-30	0.000000	…	Energy	XOM
4	1962-05-31	0.000000	…	Energy	XOM
..	…	…	…	…	…
736	2023-05-31	102.290001	…	Energy	XOM
737	2023-06-30	107.320000	…	Energy	XOM
738	2023-07-31	105.190002	…	Energy	XOM
739	2023-08-31	111.120003	…	Energy	XOM
740	2023-09-30	117.550003	…	Energy	XOM

[741 rows x 9 columns],

'SHW':

	Date	Open	…	Sector	Ticker
0	1980-03-31	0.000000	…	Materials	SHW
1	1980-04-30	0.000000	…	Materials	SHW
2	1980-05-31	0.000000	…	Materials	SHW
3	1980-06-30	0.000000	…	Materials	SHW
4	1980-07-31	0.000000	…	Materials	SHW
..	…	…	…	…	…
518	2023-05-31	228.550003	…	Materials	SHW
519	2023-06-30	262.500000	…	Materials	SHW

```
520 2023-07-31  281.480011  …  Materials     SHW
521 2023-08-31  270.739990  …  Materials     SHW
522 2023-09-30  269.459991  …  Materials     SHW

[523 rows x 9 columns],
'UPS':           Date        Open  …       Sector  Ticker
0   1999-11-30   67.375000  …  Industrials     UPS
1   1999-12-31   68.875000  …  Industrials     UPS
2   2000-01-31   63.000000  …  Industrials     UPS
3   2000-02-29   55.000000  …  Industrials     UPS
4   2000-03-31   59.562500  …  Industrials     UPS
..         …          …  …          …     …
282 2023-05-31  169.059998  …  Industrials     UPS
283 2023-06-30  176.589996  …  Industrials     UPS
284 2023-07-31  187.809998  …  Industrials     UPS
285 2023-08-31  172.029999  …  Industrials     UPS
286 2023-09-30  159.759995  …  Industrials     UPS

[287 rows x 9 columns],
'DUK':           Date       Open  …     Sector  Ticker
0   1980-03-31   0.000000  …  Utilities     DUK
1   1980-04-30   0.000000  …  Utilities     DUK
2   1980-05-31   0.000000  …  Utilities     DUK
3   1980-06-30   0.000000  …  Utilities     DUK
4   1980-07-31   0.000000  …  Utilities     DUK
..         …        …  …        …     …
518 2023-05-31  88.300003  …  Utilities     DUK
519 2023-06-30  89.010002  …  Utilities     DUK
520 2023-07-31  93.739998  …  Utilities     DUK
521 2023-08-31  90.260002  …  Utilities     DUK
522 2023-09-30  94.650002  …  Utilities     DUK

[523 rows x 9 columns],
'UNH':           Date        Open  …      Sector  Ticker
0   1984-10-31    0.000000  …  Healthcare     UNH
1   1984-11-30    0.000000  …  Healthcare     UNH
2   1984-12-31    0.000000  …  Healthcare     UNH
3   1985-01-31    0.000000  …  Healthcare     UNH
4   1985-02-28    0.000000  …  Healthcare     UNH
..         …          …  …         …     …
463 2023-05-31  478.119995  …  Healthcare     UNH
464 2023-06-30  478.000000  …  Healthcare     UNH
465 2023-07-31  503.000000  …  Healthcare     UNH
466 2023-08-31  492.359985  …  Healthcare     UNH
467 2023-09-30  482.630005  …  Healthcare     UNH

[468 rows x 9 columns],
```

```
'JPM':            Date        Open  …      Sector  Ticker
0   1980-03-31    0.000000  …  Financials     JPM
1   1980-04-30    0.000000  …  Financials     JPM
2   1980-05-31    0.000000  …  Financials     JPM
3   1980-06-30    0.000000  …  Financials     JPM
4   1980-07-31    0.000000  …  Financials     JPM
..         …         …  …         …     …
518 2023-05-31  136.729996  …  Financials     JPM
519 2023-06-30  144.600006  …  Financials     JPM
520 2023-07-31  157.179993  …  Financials     JPM
521 2023-08-31  148.259995  …  Financials     JPM
522 2023-09-30  147.839996  …  Financials     JPM

[523 rows x 9 columns],
'AMZN':           Date        Open  …                   Sector  Ticker
0   1997-05-31    0.075000  …  Consumer Discretionary    AMZN
1   1997-06-30    0.075521  …  Consumer Discretionary    AMZN
2   1997-07-31    0.121875  …  Consumer Discretionary    AMZN
3   1997-08-31    0.118229  …  Consumer Discretionary    AMZN
4   1997-09-30    0.200000  …  Consumer Discretionary    AMZN
..         …         …  …                     …     …
312 2023-05-31  121.449997  …  Consumer Discretionary    AMZN
313 2023-06-30  129.470001  …  Consumer Discretionary    AMZN
314 2023-07-31  133.199997  …  Consumer Discretionary    AMZN
315 2023-08-31  135.059998  …  Consumer Discretionary    AMZN
316 2023-09-30  145.080002  …  Consumer Discretionary    AMZN

[317 rows x 9 columns],
'AAPL':           Date        Open  …                   Sector  Ticker
0   1980-12-31    0.152902  …  Information Technology    AAPL
1   1981-01-31    0.127232  …  Information Technology    AAPL
2   1981-02-28    0.118304  …  Information Technology    AAPL
3   1981-03-31    0.110491  …  Information Technology    AAPL
4   1981-04-30    0.126674  …  Information Technology    AAPL
..         …         …  …                     …     …
509 2023-05-31  177.330002  …  Information Technology    AAPL
510 2023-06-30  191.630005  …  Information Technology    AAPL
511 2023-07-31  196.059998  …  Information Technology    AAPL
512 2023-08-31  187.839996  …  Information Technology    AAPL
513 2023-09-30  174.000000  …  Information Technology    AAPL

[514 rows x 9 columns],
'META':           Date        Open  …                   Sector  Ticker
0   2012-05-31   28.549999  …  Communication Services    META
1   2012-06-30   31.920000  …  Communication Services    META
2   2012-07-31   23.370001  …  Communication Services    META
3   2012-08-31   18.680000  …  Communication Services    META
```

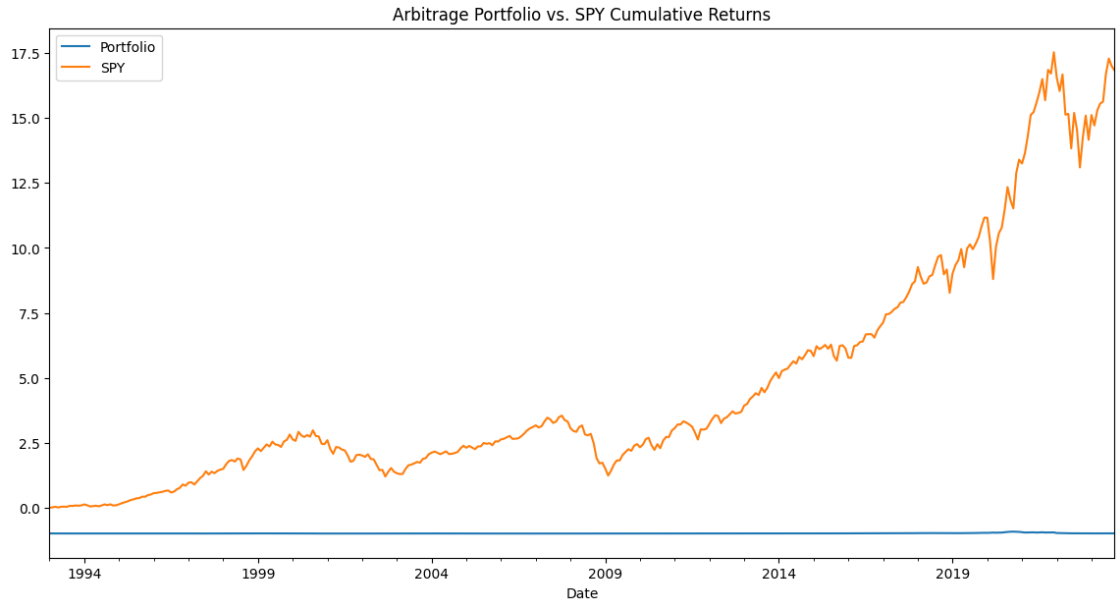
```
4   2012-09-30   20.570000  …  Communication Services    META
..         …          …  …                       …     …
132 2023-05-31  260.000000  …  Communication Services    META
133 2023-06-30  284.760010  …  Communication Services    META
134 2023-07-31  323.690002  …  Communication Services    META
135 2023-08-31  295.799988  …  Communication Services    META
136 2023-09-30  306.739990  …  Communication Services    META

[137 rows x 9 columns],
'AMT':           Date        Open  …       Sector  Ticker
0   1998-02-28   17.375000  …  Real Estate     AMT
1   1998-03-31   19.875000  …  Real Estate     AMT
2   1998-04-30   22.500000  …  Real Estate     AMT
3   1998-05-31   20.500000  …  Real Estate     AMT
4   1998-06-30   22.937500  …  Real Estate     AMT
..         …          …  …          …     …
303 2023-05-31  182.259995  …  Real Estate     AMT
304 2023-06-30  194.270004  …  Real Estate     AMT
305 2023-07-31  189.339996  …  Real Estate     AMT
306 2023-08-31  183.080002  …  Real Estate     AMT
307 2023-09-30  178.630005  …  Real Estate     AMT

[308 rows x 9 columns]}
```

[51]:
```python
# Construct the Portfolio and Backtest
def␣
 ↪build_portfolio(normalized_ticker_to_importance=normalized_ticker_to_importance,␣
 ↪strategy='Normal'):
    portfolio_returns = pd.DataFrame()
    for ticker, importance in normalized_ticker_to_importance.items():
        data = all_data[ticker].set_index('Date')
        data['Returns'] = data['Adj Close'].pct_change().fillna(0)
        portfolio_returns[ticker] = data['Returns'] * importance
    portfolio_returns['Portfolio'] = portfolio_returns.sum(axis=1)
    spy_monthly['SPY Returns'] = spy_monthly['Adj Close'].pct_change().fillna(0)
    # Cumulative Returns
    portfolio_returns['Cumulative Portfolio'] = (portfolio_returns['Portfolio']␣
 ↪+ 1).cumprod() - 1
    spy_monthly['Cumulative SPY'] = (spy_monthly['SPY Returns'] + 1).cumprod()␣
 ↪- 1
    combined = pd.concat([portfolio_returns['Cumulative Portfolio'],␣
 ↪spy_monthly['Cumulative SPY']], axis=1).dropna()
    print(combined)
    # Plot
    plt.figure(figsize=(14,7))
    combined['Cumulative Portfolio'].plot(label="Portfolio")
    combined['Cumulative SPY'].plot(label="SPY")
```

```
    plt.legend()
    plt.title(strategy + " Portfolio vs. SPY Cumulative Returns")
    plt.show()
```

[53]:
```
# Build the portfolio for arbitrage strategy
build_portfolio(arbitrage_ticker_to_importance, strategy='Arbitrage')
```

```
            Cumulative Portfolio  Cumulative SPY
Date
1993-01-31             -0.992259        0.000000
1993-02-28             -0.992964        0.010669
1993-03-31             -0.993273        0.033306
1993-04-30             -0.993537        0.006866
1993-05-31             -0.993566        0.034022
…                            …               …
2023-05-31             -0.988667       15.632604
2023-06-30             -0.987674       16.710393
2023-07-31             -0.986406       17.290112
2023-08-31             -0.988293       16.992861
2023-09-30             -0.989570       16.827191

[369 rows x 2 columns]
```



[52]:
```
# Build the portfolio for the Normal strategy
build_portfolio(normalized_ticker_to_importance, strategy='Normal')
```

```
            Cumulative Portfolio  Cumulative SPY
```

Date		
1993-01-31	4.170809	0.000000
1993-02-28	4.167075	0.010669
1993-03-31	4.194334	0.033306
1993-04-30	3.930977	0.006866
1993-05-31	3.940744	0.034022
…	…	…
2023-05-31	136.002065	15.632604
2023-06-30	146.247137	16.710393
2023-07-31	159.591522	17.290112
2023-08-31	148.041704	16.992861
2023-09-30	150.686342	16.827191

[369 rows x 2 columns]