In [ ]:
```
import pandas as pd
from itertools import combinations
from statsmodels.tsa.stattools import coint, adfuller
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(style="whitegrid")
import statsmodels.api as sm
import numpy as np
import os
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import warnings
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegre
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from tabulate import tabulate

# Suppress SettingWithCopyWarning
warnings.filterwarnings("ignore", category=pd.errors.SettingWithCopyWarni

# Suppress FutureWarning for deprecated functions
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [ ]:
```
sp500_tickers_df = pd.read_html('https://en.wikipedia.org/wiki/List_of_S%
sp500_tickers = sp500_tickers_df['Symbol'].tolist()
```

In [ ]:
```
brim_tickers = ['ADBE', 'AMZN', 'AWR', 'BEN', 'CF', 'CNX', 'COG', 'CRM',
brim_names = ['Adobe', 'Amazon', 'American States Water', 'Franklin Resou

print(len(brim_names),len(brim_tickers))
```

```
30 30
```

In [ ]:
```
current_directory = os.getcwd()
print("Current Directory:", current_directory)
datasets = {}
```

```
test_datsets = {}
tickers = []
for t in (sp500_tickers + brim_tickers):
    #path = "/Users/rashimohta/Downloads/sp500_data/" + t + ".csv"
    path = current_directory+"/sp500_data/" + t + ".txt" #use .csv or .tx
    #print("PATH: ",path)
    df = pd.read_csv(path)
    df["DT Date"] = pd.to_datetime(df['Date']) #add a column with the dat

    if 'DT Date' in df.index.names and not 'DT Date' in df.columns:
        df['DT Date'] = df.index.copy()
        print("Added Column DT Date from Index")

    if len(df)==0:
        print("Data not available for ticker: ", t)
    #elif len(df) != 2516:
    elif len(df) < 100:
        print(f"{t} Removed from the stock universe : Fewer than 100 days
    else:
        datasets[t] = df
        test_datsets[t] = df
        tickers.append(t)
        """datasets[t] = df[:-600]
        test_datsets[t] = df[-600:]
        tickers.append(t)"""
```

```
Current Directory: /Users/rashimohta/Downloads/Pairs Trading Project
Data not available for ticker:  AMTM
Data not available for ticker:  BRK.B
Data not available for ticker:  BF.B
GEV Removed from the stock universe : Fewer than 100 days data
Data not available for ticker:  SW
SOLV Removed from the stock universe : Fewer than 100 days data
```

In [ ]:
```
current_directory = os.getcwd()
print("Current Directory:", current_directory)
datasets = {}
test_datsets = {}
tickers = []
for t in (sp500_tickers + brim_tickers):
    #path = "/Users/rashimohta/Downloads/sp500_data/" + t + ".csv"
    path = current_directory+"/sp500_data/" + t + ".txt" #use .csv or .tx
    #print("PATH: ",path)
    df = pd.read_csv(path)
    df["DT Date"] = pd.to_datetime(df['Date']) #add a column with the dat

    if 'DT Date' in df.index.names and not 'DT Date' in df.columns:
        df['DT Date'] = df.index.copy()
        print("Added Column DT Date from Index")

    if len(df)==0:
        print("Data not available for ticker: ", t)
    #elif len(df) != 2516:
    elif len(df) < 100:
```

```
        print(f"{t} Removed from the stock universe : Fewer than 100 days
    else:
        datasets[t] = df
        test_datsets[t] = df
        tickers.append(t)
        """datasets[t] = df[:-600]
        test_datsets[t] = df[-600:]
        tickers.append(t)"""
```

```
Current Directory: /Users/rashimohta/Downloads/Pairs Trading Project
Data not available for ticker:  AMTM
Data not available for ticker:  BRK.B
Data not available for ticker:  BF.B
GEV Removed from the stock universe : Fewer than 100 days data
Data not available for ticker:  SW
SOLV Removed from the stock universe : Fewer than 100 days data
```

In [ ]:

```
brim_stock_pairs = [['BEN', 'COG'], ['DISCA', 'RIG'], ['DISCK', 'RIG'], [
                    ['CNX', 'HBI'], ['AMZN', 'CRM'], ['MA', 'VFC'], ['FCX', 'G
                    ['FCX', 'HBI'], ['DISCK', 'ESV'], ['DISCK', 'NE'], ['DISCA
                    ['NBL', 'RIG'], ['CNX', 'GNW'], ['COG', 'DO'], ['HBI', 'NB
                    ['DISCA', 'MA'], ['DISCK', 'MA'], ['RIG', 'RRC'], ['CF', '
                    ['NE', 'RRC'], ['ADBE', 'RHT'], ['MA', 'RIG'], ['NBL', 'SW
                    ['AWR', 'WTR'], ['SLB', 'PFE']]
```

In [ ]:

```
def calculate_features(df):
    features = pd.DataFrame(index=df.index)

    features['Current Spread'] = df['Spread']
    features['Spread Returns'] = features['Current Spread'].pct_change()

    for days in [15, 10, 7, 5]:
        features[f'Sp Mean {days}days'] = features['Current Spread'].roll
        features[f'Sp/SpMean {days}days'] = features['Current Spread'] /

    column_order = ['Current Spread', 'Spread Returns'] + \
                   [f'Sp Mean {days}days' for days in [15, 10, 7, 5]] + \
                   [f'Sp/SpMean {days}days' for days in [15, 10, 7, 5]]

    features = features[column_order]

    return features

def prepare_data(datasets, pair):
    stock1, stock2 = pair

    df1 = datasets[stock1]
    df2 = datasets[stock2]

    if not isinstance(df1.index, pd.DatetimeIndex):
        df1['DT Date'] = pd.to_datetime(df1['DT Date'])
        df1.set_index('DT Date', inplace=True)

    if not isinstance(df2.index, pd.DatetimeIndex):
```

```
        df2['DT Date'] = pd.to_datetime(df2['DT Date'])
        df2.set_index('DT Date', inplace=True)

    # Calculate spread
    spread_df = pd.DataFrame(index=df1.index)
    spread_df['Spread'] = df1['Adj Close'] - df2['Adj Close']

    # Calculate features
    features = calculate_features(spread_df)

    # Calculate target variable (next day's spread return)
    features['Target'] = features['Spread Returns'].shift(-1)
    features['Pair'] = f"{pair[0]}_{pair[1]}"
    return features.dropna()

def prepare_all_pairs_data(datasets, pairs):
    combined_train = pd.DataFrame()

    for pair in pairs:
        try:
            pair_data = prepare_data(datasets, pair)

            # Split data into training sets
            train_data = pair_data[(pair_data.index.year < 2018) & (pair_


            # Add Pair information
            train_data['Pair'] = f"{pair[0]}_{pair[1]}"

            combined_train = pd.concat([combined_train, train_data], igno
        except:
            print("Failed for pair: ", pair)

    return combined_train

# Prepare features and target for train and test separately
def prepare_features_target(data):
    X = data.drop(['Target', 'Pair'], axis=1)
    y = data['Target']
    return X, y

# Handle infinities and large values
def handle_inf_nan(X):
    X = X.replace([np.inf, -np.inf], np.nan)
    X = X.fillna(method='ffill').fillna(method='bfill')
    return X
```

In [ ]:
```
def train_xgboost(X_train, y_train):
    '''
    param_grid = {
    'n_estimators': [100, 300],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5],
```

```
    'min_child_weight': [1, 3],
    'gamma': [0, 0.1],
    'subsample': [0.9],
    'lambda': [1, 2]  # L2 regularization
    }

    model = XGBRegressor(random_state=42)
    random_search = RandomizedSearchCV(
        estimator=model,
        param_distributions=param_grid,
        scoring='neg_mean_squared_error',
        cv=3,
        verbose=1,
        random_state=42
    )

    random_search.fit(X_train, y_train)

    print("Best Hyperparameters:", random_search.best_params_)
    #Best Hyperparameters: {'subsample': 0.9, 'n_estimators': 100, 'min_c
    '''
    model = XGBRegressor(
        n_estimators=100,
        learning_rate=0.05,
        max_depth=3,
        min_child_weight=1,
        subsample=0.9,
        colsample_bytree=0.8,
        random_state=42,
        gamma = 0,
    )
    model.fit(X_train, y_train)
    return model

def evaluate_model(model, X_test, y_test):
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    return mse, r2, predictions

def implement_trading_strategy(predictions, test_features, tolerance=0.00
    positions = np.zeros(len(predictions))
    positions[predictions > tolerance] = 1
    positions[predictions < -tolerance] = -1

    returns = positions * test_features['Spread Returns'].values
    cumulative_returns = np.cumprod(1 + returns) - 1
    return cumulative_returns, positions
```

In [ ]:

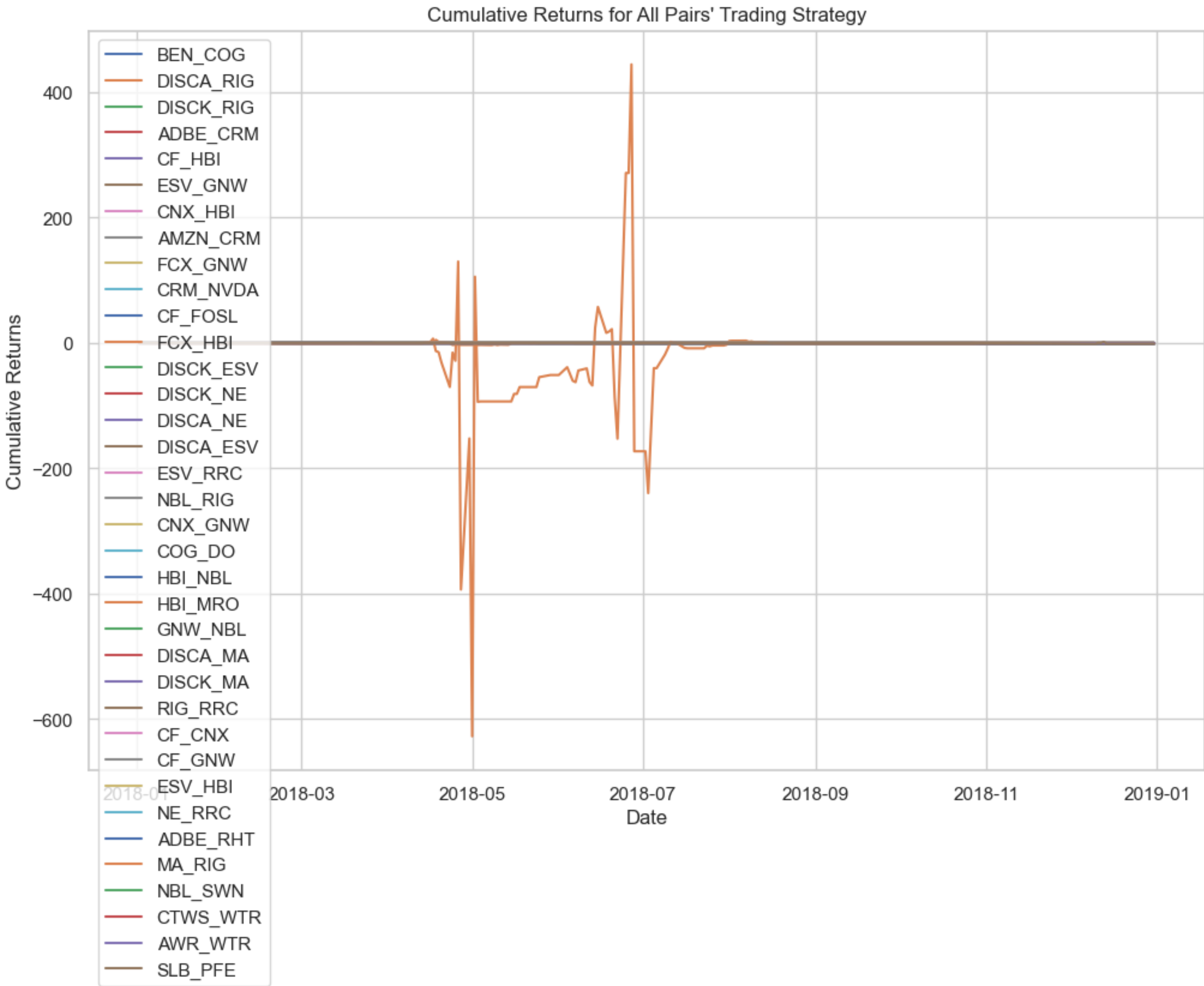
```
combined_train = prepare_all_pairs_data(datasets, brim_stock_pairs)

X_train, y_train = prepare_features_target(combined_train)
```

```
# Handle infinities and large values
X_train = handle_inf_nan(X_train)

def clean_target_data(X, y):
    mask = y.notna() & ~np.isinf(y)  # Create mask to filter out NaN and
    return X[mask], y[mask]

X_train, y_train = clean_target_data(X_train, y_train)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Train the model
model = train_xgboost(X_train_scaled, y_train)

cumulative_returns_dict = {}
results = []

plt.figure(figsize=(12, 8))
for pair in brim_stock_pairs:
    try:
        pair_data = prepare_data(datasets, pair)

        test_data = pair_data[pair_data.index.year == 2018]
        X_test, y_test = prepare_features_target(test_data)

        # Handle infinities and large values
        X_test = handle_inf_nan(X_test)
        X_test_scaled = scaler.transform(X_test)

        mse, r2, predictions = evaluate_model(model, X_test_scaled, y_tes

        print(f"Pair: {pair}, Mean Squared Error: {mse}, R-squared: {r2}"

        # Implement trading strategy
        cumulative_returns, positions = implement_trading_strategy(predic

        cumulative_returns_dict[pair[0]+pair[1]] = cumulative_returns
        results.append([pair[0], pair[1], mse, r2, cumulative_returns[-1]

        plt.plot(test_data.index, cumulative_returns, label=f"{pair[0]}_{

    except:
        print("failed for pair: ", pair)

# Finalize and show the plot
plt.title("Cumulative Returns for All Pairs' Trading Strategy")
plt.xlabel("Date")
plt.ylabel("Cumulative Returns")
plt.legend(loc="upper left")
plt.show()
```

```
Failed for pair:  ['MA', 'VFC']
```

```
Pair: ['BEN', 'COG'], Mean Squared Error: 0.00451371349363909, R-squared: -0.00026810014327161014
Pair: ['DISCA', 'RIG'], Mean Squared Error: 0.001973425179299559, R-squared: -0.0031005352485402593
Pair: ['DISCK', 'RIG'], Mean Squared Error: 0.0022371665971717583, R-squared: -0.0032409391077670158
Pair: ['ADBE', 'CRM'], Mean Squared Error: 0.0009721693843248306, R-squared: -0.0027555952005899886
Pair: ['CF', 'HBI'], Mean Squared Error: 0.0016747755748894892, R-squared: -0.0029470894992782437
Pair: ['ESV', 'GNW'], Mean Squared Error: 0.001516347826730432, R-squared: -0.0006441598065316523
Pair: ['CNX', 'HBI'], Mean Squared Error: 0.12420939573188185, R-squared: -0.35479007476139124
Pair: ['AMZN', 'CRM'], Mean Squared Error: 0.0004666383504514382, R-squared: -0.005883572034045992
failed for pair:  ['MA', 'VFC']
Pair: ['FCX', 'GNW'], Mean Squared Error: 0.001411615078856414, R-squared: -0.007590062015828236
Pair: ['CRM', 'NVDA'], Mean Squared Error: 0.0004768313584557387, R-squared: -0.007661109832311386
Pair: ['CF', 'FOSL'], Mean Squared Error: 0.0029678608300975212, R-squared: -0.0001652199540649324
Pair: ['FCX', 'HBI'], Mean Squared Error: 1.1685075135682383, R-squared: 0.4602091136913551
Pair: ['DISCK', 'ESV'], Mean Squared Error: 0.002576803341838784, R-squared: -0.0010491975351158978
Pair: ['DISCK', 'NE'], Mean Squared Error: 0.0007791434745997887, R-squared: -0.0032454049497929738
Pair: ['DISCA', 'NE'], Mean Squared Error: 0.0007845826742430618, R-squared: -0.003544479508166054
Pair: ['DISCA', 'ESV'], Mean Squared Error: 0.0027463758954903435, R-squared: -0.0024229656030931856
Pair: ['ESV', 'RRC'], Mean Squared Error: 0.0017591181907027987, R-squared: -1.898209434081366e-07
Pair: ['NBL', 'RIG'], Mean Squared Error: 0.0009181409687401804, R-squared: -0.0010505409766095042
Pair: ['CNX', 'GNW'], Mean Squared Error: 0.0011388236692157902, R-squared: -0.001245347539460706
Pair: ['COG', 'DO'], Mean Squared Error: 0.012672665141832372, R-squared: -0.0004250098941913638
Pair: ['HBI', 'NBL'], Mean Squared Error: 0.005846642261410964, R-squared: -0.0014558739615091465
Pair: ['HBI', 'MRO'], Mean Squared Error: 1.2656450589321255, R-squared: -0.3990049358816612
Pair: ['GNW', 'NBL'], Mean Squared Error: 0.0008294961365566999, R-squared: -0.0016159289064052729
Pair: ['DISCA', 'MA'], Mean Squared Error: 0.0004116082904857544, R-squared: -0.005828939077023199
Pair: ['DISCK', 'MA'], Mean Squared Error: 0.00040039160447767825, R-squared: -0.006019416706309455
Pair: ['RIG', 'RRC'], Mean Squared Error: 0.018031267071124398, R-squared: 0.05513578849055534
```

```
Pair: ['CF', 'CNX'], Mean Squared Error: 0.0011050683498695993, R-squared:
-0.0008084589430712441
Pair: ['CF', 'GNW'], Mean Squared Error: 0.0006406255833040104, R-squared:
-0.0002880193141667764
Pair: ['ESV', 'HBI'], Mean Squared Error: 0.002056426710537815, R-squared:
1.4162895762059868e-05
Pair: ['NE', 'RRC'], Mean Squared Error: 0.0014078256262157335, R-squared:
-0.001270143779948718
Pair: ['ADBE', 'RHT'], Mean Squared Error: 0.005653497953014792, R-square
d: -0.04248712632337526
Pair: ['MA', 'RIG'], Mean Squared Error: 0.0003392074676467846, R-squared:
-0.007404052009366335
Pair: ['NBL', 'SWN'], Mean Squared Error: 0.0007551887186818223, R-square
d: -0.0013313752361685527
failed for pair:  ['CTWS', 'AWR']
Pair: ['CTWS', 'WTR'], Mean Squared Error: 0.001485423671635775, R-square
d: -0.008006100181691123
Pair: ['AWR', 'WTR'], Mean Squared Error: 0.0007874220880120775, R-square
d: -0.013233981845165443
Pair: ['SLB', 'PFE'], Mean Squared Error: 0.5267609816691273, R-squared: -
0.5583297274057162
```



In [ ]:
```python
results_df = pd.DataFrame(results, columns=["Stock 1", "Stock 2", "MSE",
print(results_df)
```

```
   Stock 1 Stock 2       MSE     R-squared  Final Return
0      BEN     COG  0.004514 -2.681001e-04    -69.602278
1    DISCA     RIG  0.001973 -3.100535e-03    -26.438180
2    DISCK     RIG  0.002237 -3.240939e-03    -14.968153
3     ADBE     CRM  0.000972 -2.755595e-03      0.000000
4       CF     HBI  0.001675 -2.947089e-03    -21.584651
5      ESV     GNW  0.001516 -6.441598e-04    -17.322765
6      CNX     HBI  0.124209 -3.547901e-01    -99.999947
7     AMZN     CRM  0.000467 -5.883572e-03      0.000000
8      FCX     GNW  0.001412 -7.590062e-03    -22.036002
9      CRM    NVDA  0.000477 -7.661110e-03      0.000000
10      CF    FOSL  0.002968 -1.652200e-04    -62.163850
11     FCX     HBI  1.168508  4.602091e-01    -99.687298
12   DISCK     ESV  0.002577 -1.049198e-03    -67.246877
13   DISCK      NE  0.000779 -3.245405e-03      0.000000
14   DISCA      NE  0.000785 -3.544480e-03      0.000000
15   DISCA     ESV  0.002746 -2.422966e-03    -69.890783
16     ESV     RRC  0.001759 -1.898209e-07    -39.783494
17     NBL     RIG  0.000918 -1.050541e-03      0.000000
18     CNX     GNW  0.001139 -1.245348e-03      0.000000
19     COG      DO  0.012673 -4.250099e-04    -99.237835
20     HBI     NBL  0.005847 -1.455874e-03    -92.251316
21     HBI     MRO  1.265645 -3.990049e-01    -99.993541
22     GNW     NBL  0.000829 -1.615929e-03     -5.929722
23   DISCA      MA  0.000412 -5.828939e-03      0.000000
24   DISCK      MA  0.000400 -6.019417e-03      0.000000
25     RIG     RRC  0.018031  5.513579e-02   -100.036212
26      CF     CNX  0.001105 -8.084589e-04    -11.124088
27      CF     GNW  0.000641 -2.880193e-04      0.000000
28     ESV     HBI  0.002056  1.416290e-05    -38.919603
29      NE     RRC  0.001408 -1.270144e-03      0.000000
30    ADBE     RHT  0.005653 -4.248713e-02    -91.495293
31      MA     RIG  0.000339 -7.404052e-03      0.000000
32     NBL     SWN  0.000755 -1.331375e-03      0.000000
33    CTWS     WTR  0.001485 -8.006100e-03      0.000000
34     AWR     WTR  0.000787 -1.323398e-02      0.000000
35     SLB     PFE  0.526761 -5.583297e-01   -100.664589
```

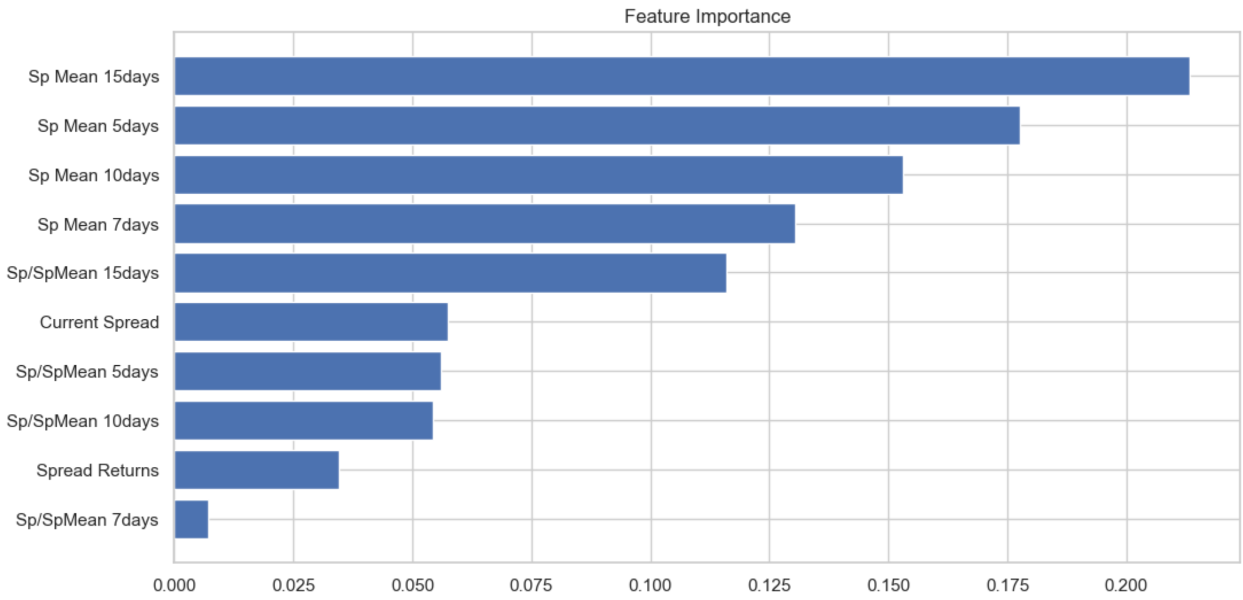
In [ ]:

```
print(results_df[results_df['Final Return']>0].reset_index())
```

```
Empty DataFrame
Columns: [index, Stock 1, Stock 2, MSE, R-squared, Final Return]
Index: []
```

In [ ]:

```
# Feature importance
feature_importance = model.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(12, 6))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X_train.columns[sorted_idx])
plt.title('Feature Importance')
plt.show()
```



Other models

In [ ]:
```
def prepare_all_pairs_data(datasets, pairs, start_year, end_year):
    combined_data = pd.DataFrame()

    for pair in pairs:
        try:
            pair_data = prepare_data(datasets, pair)

            # Filter data for the specified year range
            pair_data = pair_data[(pair_data.index.year >= start_year) &

            # Add Pair information
            pair_data['Pair'] = f"{pair[0]}_{pair[1]}"

            combined_data = pd.concat([combined_data, pair_data], ignore_
        except Exception as e:
            print(f"Failed for pair: {pair}. Error: {e}")

    return combined_data


def train_and_evaluate_model(model, X_train, y_train, X_test, y_test, tes
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    cumulative_returns, positions = implement_trading_strategy(prediction
    return mse, r2, cumulative_returns, positions
```

In [ ]:
```
# Prepare combined training data (2014-2017)
combined_train = prepare_all_pairs_data(datasets, brim_stock_pairs, 2014,
X_train, y_train = prepare_features_target(combined_train)

# Handle infinities and large values
```

```
X_train = handle_inf_nan(X_train)
X_train, y_train = clean_target_data(X_train, y_train)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
Failed for pair: ['MA', 'VFC']. Error: 'VFC'
```

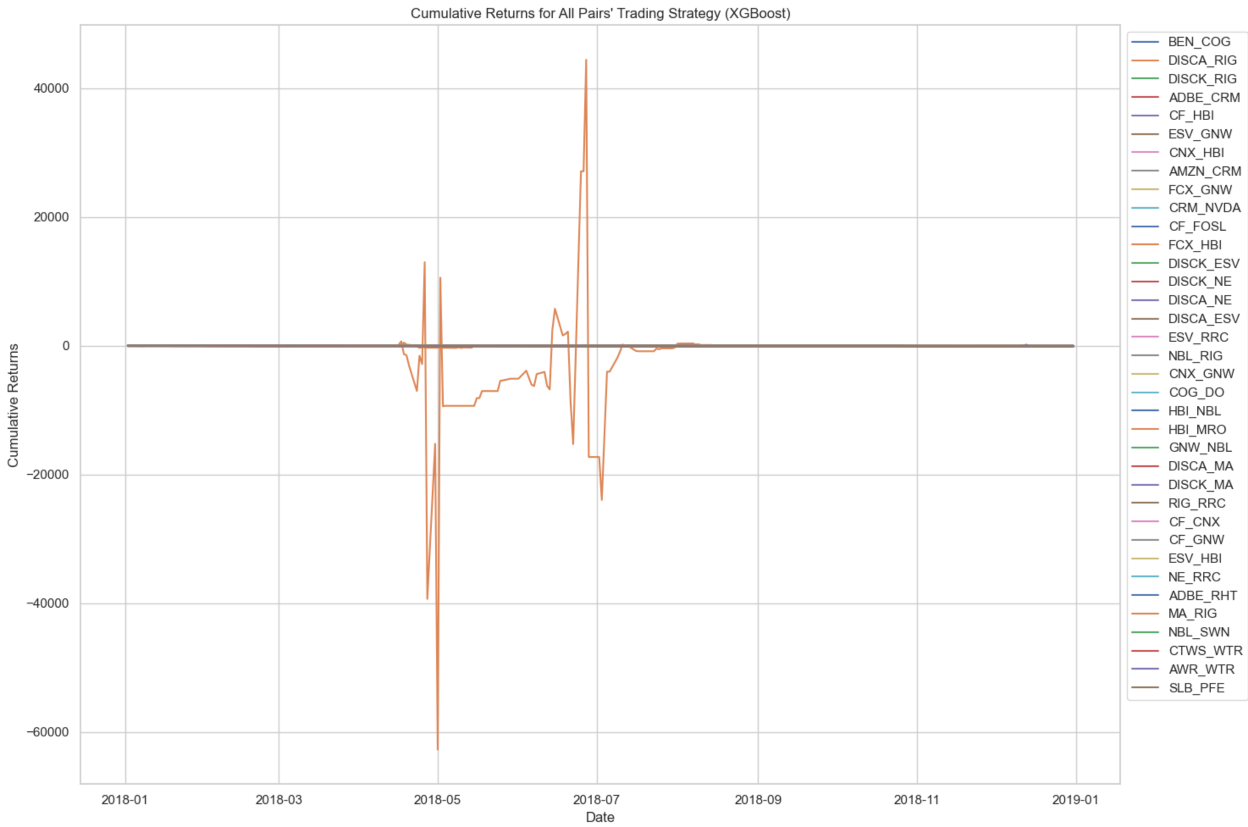
In [ ]:
```
models = {
    'XGBoost': xgb.XGBRegressor(n_estimators=100, learning_rate=0.05, max
                                min_child_weight=1, subsample=0.9, colsam
                                random_state=42, gamma=0),
                                'Linear Regression': LinearRegression()
}
#'Linear Regression': LinearRegression()
trained_models = {name: model.fit(X_train_scaled, y_train) for name, mode
```

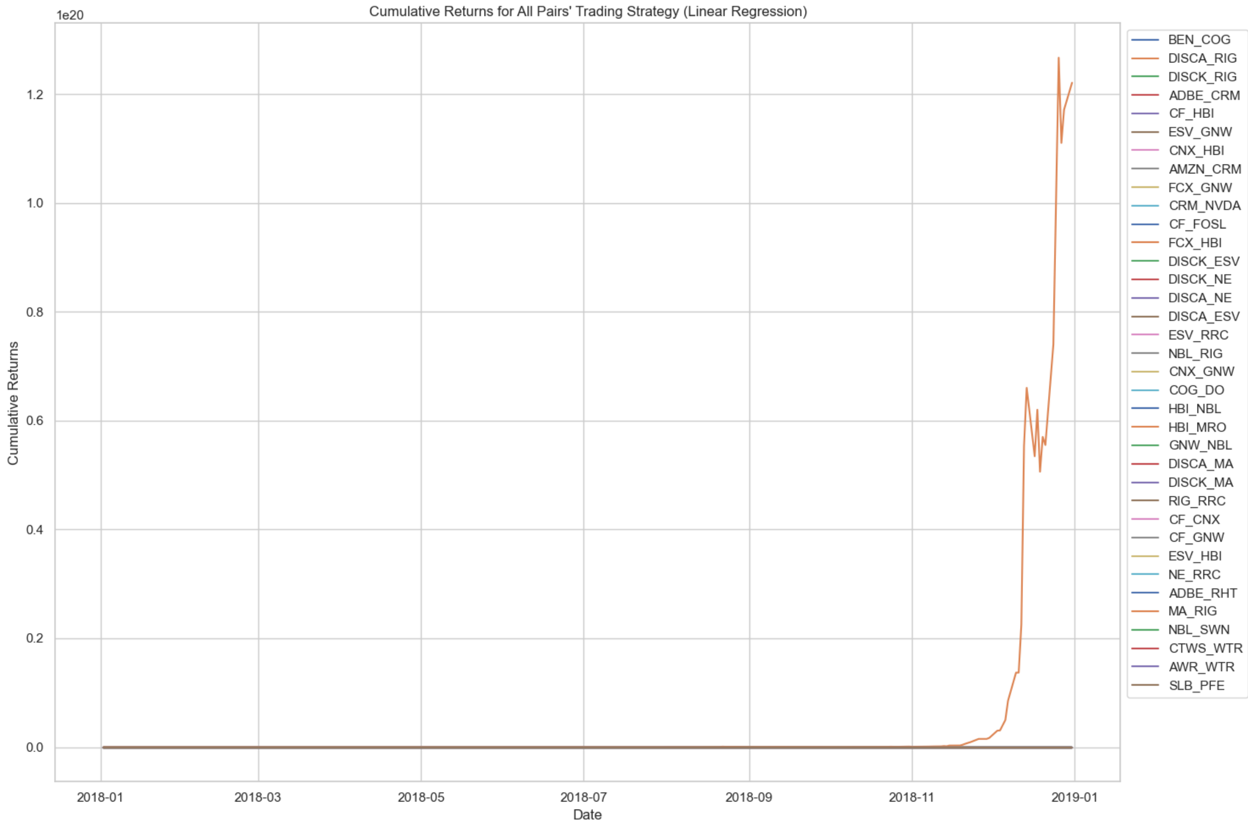
In [ ]:
```
results = {model_name: {'results': [], 'cumulative_returns': {}} for mode

for pair in brim_stock_pairs:
    try:
        pair_data = prepare_data(datasets, pair)
        test_data = pair_data[pair_data.index.year == 2018]
        X_test, y_test = prepare_features_target(test_data)

        # Handle infinities and large values
        X_test = handle_inf_nan(X_test)
        X_test_scaled = scaler.transform(X_test)

        for model_name, model in trained_models.items():
            mse, r2, cumulative_returns, positions = train_and_evaluate_m
                model, X_train_scaled, y_train, X_test_scaled, y_test, te
            )

            results[model_name]['results'].append([pair[0], pair[1], mse,
            results[model_name]['cumulative_returns'][f"{pair[0]}_{pair[1

    except Exception as e:
        print(f"Failed for pair: {pair}. Error: {e}")
```

```
Failed for pair: ['MA', 'VFC']. Error: 'VFC'
Failed for pair: ['CTWS', 'AWR']. Error: Input contains infinity or a valu
e too large for dtype('float64').
```

In [ ]:
```
# Convert results to DataFrames
for model_name in results:
    results[model_name]['results_df'] = pd.DataFrame(
        results[model_name]['results'],
        columns=["Stock 1", "Stock 2", "MSE", "R-squared", "Final Return"
    )
for model_name, model_results in results.items():
    cumulative_returns = model_results['cumulative_returns']
```

```
    plt.figure(figsize=(15, 10))

    for pair, returns in cumulative_returns.items():
        plt.plot(test_data.index, returns, label=f"{pair}")

    plt.title(f"Cumulative Returns for All Pairs' Trading Strategy ({mode
    plt.xlabel("Date")
    plt.ylabel("Cumulative Returns")
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.tight_layout()
    plt.show()
```





In [ ]:
```
# Print results for all models
for model_name, model_results in results.items():
    print(f"\nResults for {model_name}:")
    print(model_results['results_df'])
    print(f"\nPositive returns for {model_name}:")
    print(model_results['results_df'][model_results['results_df']['Final
```

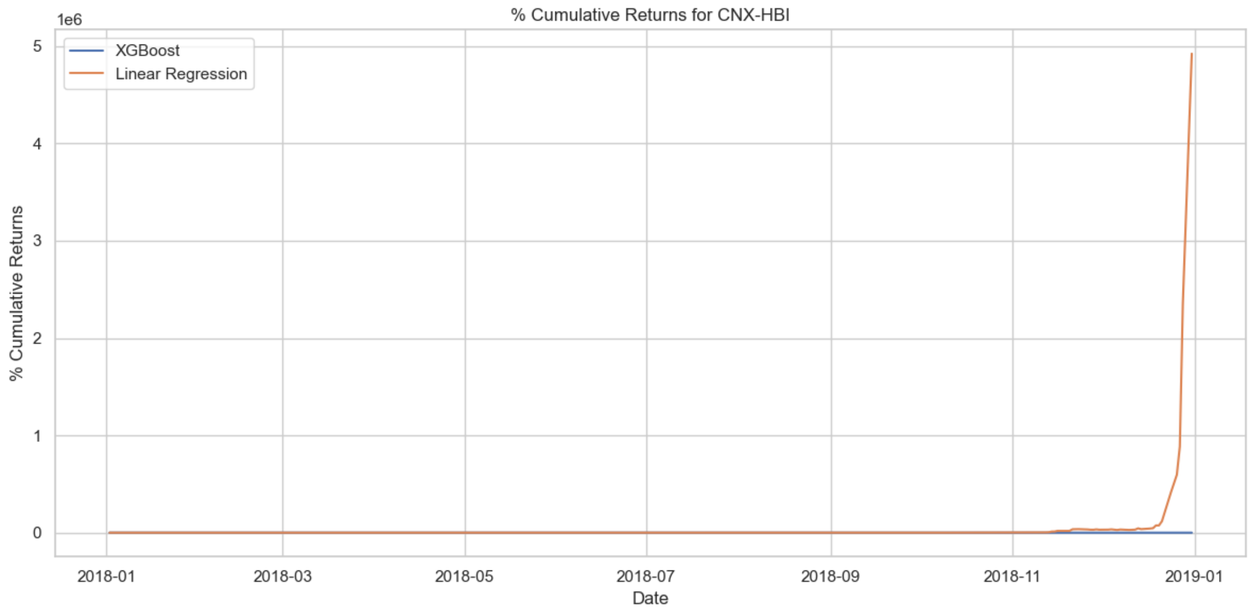
Results for XGBoost:

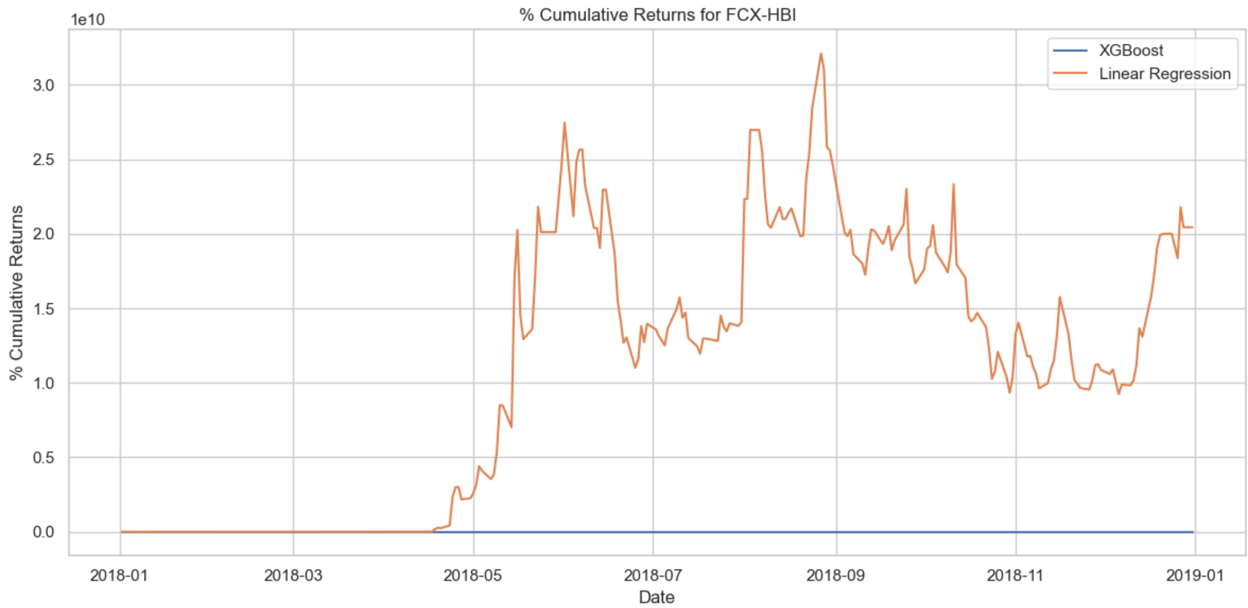
	Stock 1	Stock 2	MSE	R-squared	Final Return
0	BEN	COG	0.004514	-2.681001e-04	-69.602278
1	DISCA	RIG	0.001973	-3.100535e-03	-26.438180
2	DISCK	RIG	0.002237	-3.240939e-03	-14.968153
3	ADBE	CRM	0.000972	-2.755595e-03	0.000000
4	CF	HBI	0.001675	-2.947089e-03	-21.584651
5	ESV	GNW	0.001516	-6.441598e-04	-17.322765
6	CNX	HBI	0.124209	-3.547901e-01	-99.999947
7	AMZN	CRM	0.000467	-5.883572e-03	0.000000
8	FCX	GNW	0.001412	-7.590062e-03	-22.036002
9	CRM	NVDA	0.000477	-7.661110e-03	0.000000
10	CF	FOSL	0.002968	-1.652200e-04	-62.163850
11	FCX	HBI	1.168508	4.602091e-01	-99.687298
12	DISCK	ESV	0.002577	-1.049198e-03	-67.246877
13	DISCK	NE	0.000779	-3.245405e-03	0.000000
14	DISCA	NE	0.000785	-3.544480e-03	0.000000
15	DISCA	ESV	0.002746	-2.422966e-03	-69.890783
16	ESV	RRC	0.001759	-1.898209e-07	-39.783494
17	NBL	RIG	0.000918	-1.050541e-03	0.000000
18	CNX	GNW	0.001139	-1.245348e-03	0.000000
19	COG	DO	0.012673	-4.250099e-04	-99.237835
20	HBI	NBL	0.005847	-1.455874e-03	-92.251316

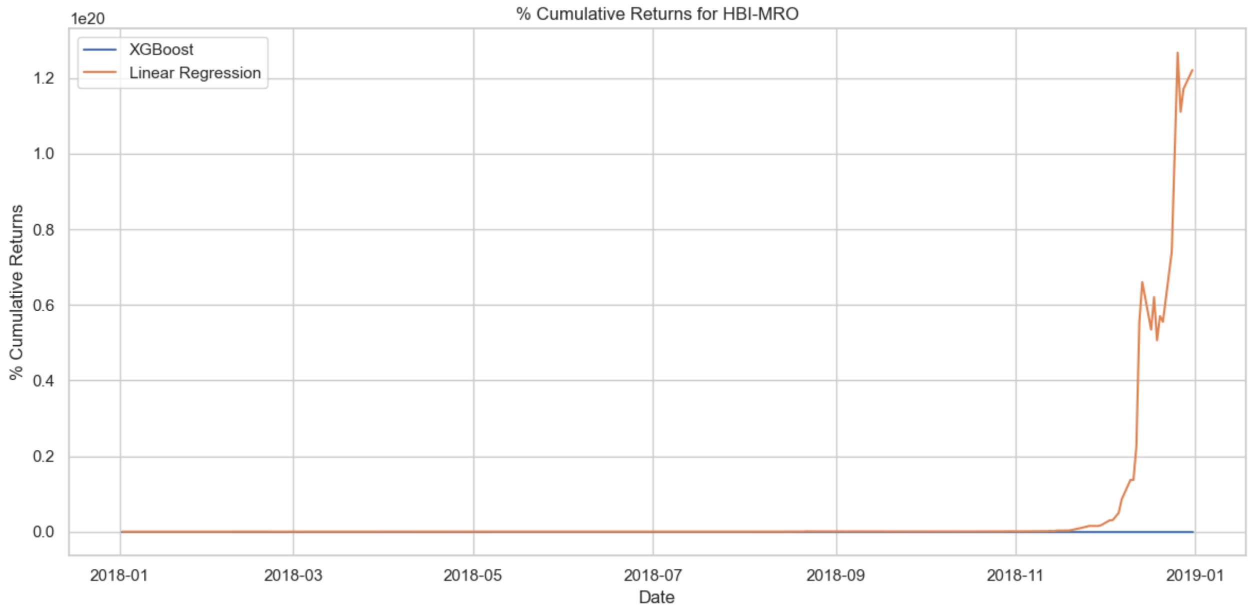
```
21      HBI     MRO  1.265645 -3.990049e-01    -99.993541
22      GNW     NBL  0.000829 -1.615929e-03     -5.929722
23    DISCA      MA  0.000412 -5.828939e-03      0.000000
24    DISCK      MA  0.000400 -6.019417e-03      0.000000
25      RIG     RRC  0.018031  5.513579e-02   -100.036212
26       CF     CNX  0.001105 -8.084589e-04    -11.124088
27       CF     GNW  0.000641 -2.880193e-04      0.000000
28      ESV     HBI  0.002056  1.416290e-05    -38.919603
29       NE     RRC  0.001408 -1.270144e-03      0.000000
30     ADBE     RHT  0.005653 -4.248713e-02    -91.495293
31       MA     RIG  0.000339 -7.404052e-03      0.000000
32      NBL     SWN  0.000755 -1.331375e-03      0.000000
33     CTWS     WTR  0.001485 -8.006100e-03      0.000000
34      AWR     WTR  0.000787 -1.323398e-02      0.000000
35      SLB     PFE  0.526761 -5.583297e-01   -100.664589

Positive returns for XGBoost:
Empty DataFrame
Columns: [Stock 1, Stock 2, MSE, R-squared, Final Return]
Index: []

Results for Linear Regression:
   Stock 1 Stock 2       MSE  R-squared  Final Return
0      BEN     COG  0.004511   0.000289  2.436705e+01
1    DISCA     RIG  0.001988  -0.010529 -6.000638e+01
2    DISCK     RIG  0.002252  -0.009949 -6.304569e+01
3     ADBE     CRM  0.000987  -0.017856 -1.749857e+00
4       CF     HBI  0.001684  -0.008622 -4.296583e+01
5      ESV     GNW  0.001513   0.001642  1.981945e+02
6      CNX     HBI  0.092709  -0.011203  4.922263e+06
7     AMZN     CRM  0.000480  -0.034224 -3.816226e+01
8      FCX     GNW  0.001403  -0.001276  9.610367e+01
9      CRM    NVDA  0.000486  -0.027352  8.802548e+00
10      CF    FOSL  0.002975  -0.002636  5.935695e+01
11     FCX     HBI  2.177121  -0.005719  2.044167e+10
12   DISCK     ESV  0.002582  -0.003177 -6.743625e+00
13   DISCK      NE  0.000786  -0.012492 -3.308104e+01
14   DISCA      NE  0.000792  -0.012781 -3.364055e+01
15   DISCA     ESV  0.002748  -0.002949 -1.103411e+01
16     ESV     RRC  0.001757   0.001293  1.825549e+02
17     NBL     RIG  0.000916   0.001064  4.829557e+01
18     CNX     GNW  0.001138  -0.000181  2.786207e+01
19     COG      DO  0.012780  -0.008902  2.704250e+02
20     HBI     NBL  0.005851  -0.002220 -1.123061e+01
21     HBI     MRO  0.904465   0.000233  1.221062e+20
22     GNW     NBL  0.000830  -0.001839  4.973311e+01
23   DISCA      MA  0.000425  -0.038245 -2.792092e+01
24   DISCK      MA  0.000414  -0.039504 -2.788354e+01
25     RIG     RRC  0.019146  -0.003279  1.215596e+03
26      CF     CNX  0.001113  -0.007997 -2.717591e+01
27      CF     GNW  0.000644  -0.005875 -2.602832e+00
28     ESV     HBI  0.002053   0.001591  2.390695e+02
29      NE     RRC  0.001407  -0.001035  2.633550e+01
```

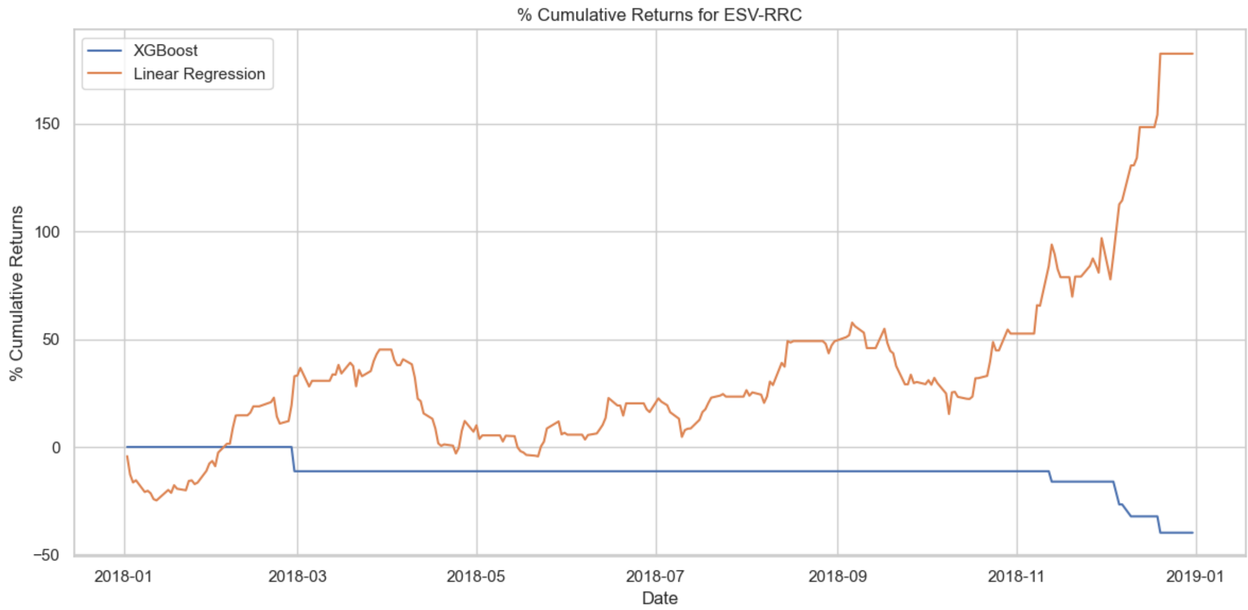
```
30     ADBE     RHT  0.005420   0.000524  2.427129e+02
31       MA     RIG  0.000347  -0.031217  7.110779e+00
32      NBL     SWN  0.000754   0.000522  2.682372e+01
33     CTWS     WTR  0.001502  -0.019102 -3.317890e+01
34      AWR     WTR  0.000804  -0.034022 -5.343680e+01
35      SLB     PFE  0.337668   0.001067  1.627211e+07

Positive returns for Linear Regression:
   Stock 1 Stock 2       MSE  R-squared  Final Return
0      BEN     COG  0.004511   0.000289  2.436705e+01
1      ESV     GNW  0.001513   0.001642  1.981945e+02
2      CNX     HBI  0.092709  -0.011203  4.922263e+06
3      FCX     GNW  0.001403  -0.001276  9.610367e+01
4      CRM    NVDA  0.000486  -0.027352  8.802548e+00
5       CF    FOSL  0.002975  -0.002636  5.935695e+01
6      FCX     HBI  2.177121  -0.005719  2.044167e+10
7      ESV     RRC  0.001757   0.001293  1.825549e+02
8      NBL     RIG  0.000916   0.001064  4.829557e+01
9      CNX     GNW  0.001138  -0.000181  2.786207e+01
10     COG      DO  0.012780  -0.008902  2.704250e+02
11     HBI     MRO  0.904465   0.000233  1.221062e+20
12     GNW     NBL  0.000830  -0.001839  4.973311e+01
13     RIG     RRC  0.019146  -0.003279  1.215596e+03
14     ESV     HBI  0.002053   0.001591  2.390695e+02
15      NE     RRC  0.001407  -0.001035  2.633550e+01
16    ADBE     RHT  0.005420   0.000524  2.427129e+02
17      MA     RIG  0.000347  -0.031217  7.110779e+00
18     NBL     SWN  0.000754   0.000522  2.682372e+01
19     SLB     PFE  0.337668   0.001067  1.627211e+07
```

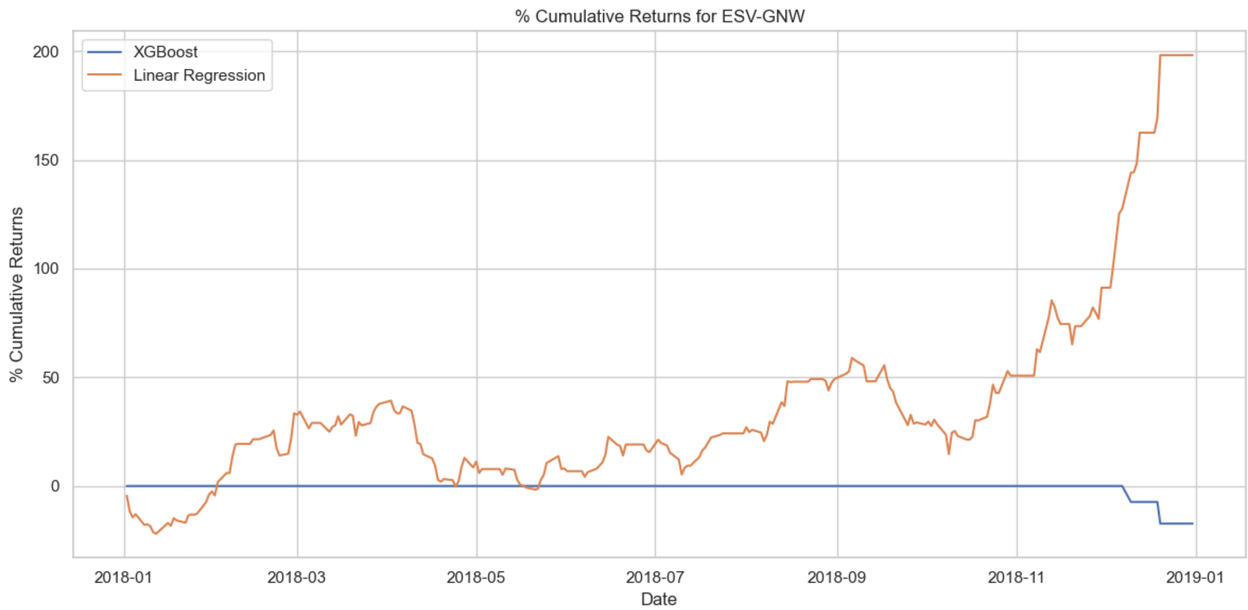
In [ ]:
```
# Separate plots for pairs in the paper
specific_pairs = [
    ('CNX', 'HBI'), ('FCX', 'HBI'), ('HBI', 'MRO'),
    ('ESV', 'RRC'), ('ESV', 'GNW')
]

for pair in specific_pairs:
    plt.figure(figsize=(12, 6))
    pair_key = f"{pair[0]}_{pair[1]}"

    for model_name, model_results in results.items():
        if pair_key in model_results['cumulative_returns']:
            returns = model_results['cumulative_returns'][pair_key]
            plt.plot(test_data.index, returns, label=model_name)

    plt.title(f"% Cumulative Returns for {pair[0]}-{pair[1]}")
    plt.xlabel("Date")
    plt.ylabel("% Cumulative Returns")
    plt.legend()
    plt.tight_layout()
    plt.show()
```


% Cumulative Returns for CNX-HBI
1e6
XGBoost
Linear Regression
% Cumulative Returns
Date
0
1
2
3
4
5
2018-01
2018-03
2018-05
2018-07
2018-09
2018-11
2019-01

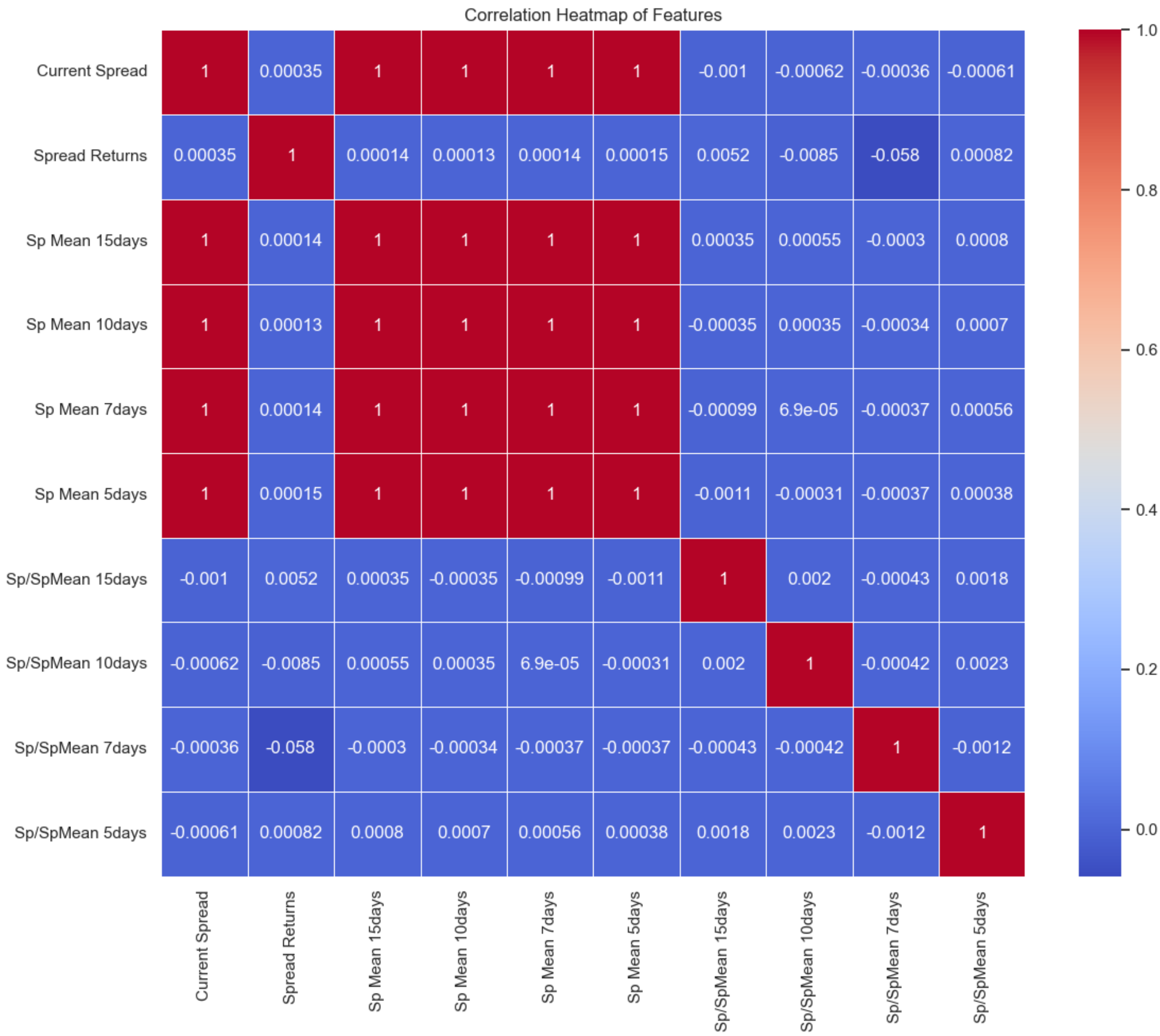

% Cumulative Returns for FCX-HBI
1e10
XGBoost
Linear Regression
% Cumulative Returns
Date
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2018-01
2018-03
2018-05
2018-07
2018-09
2018-11
2019-01


% Cumulative Returns for HBI-MRO
1e20
XGBoost
Linear Regression
% Cumulative Returns
Date
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2018-01
2018-03
2018-05
2018-07
2018-09
2018-11
2019-01



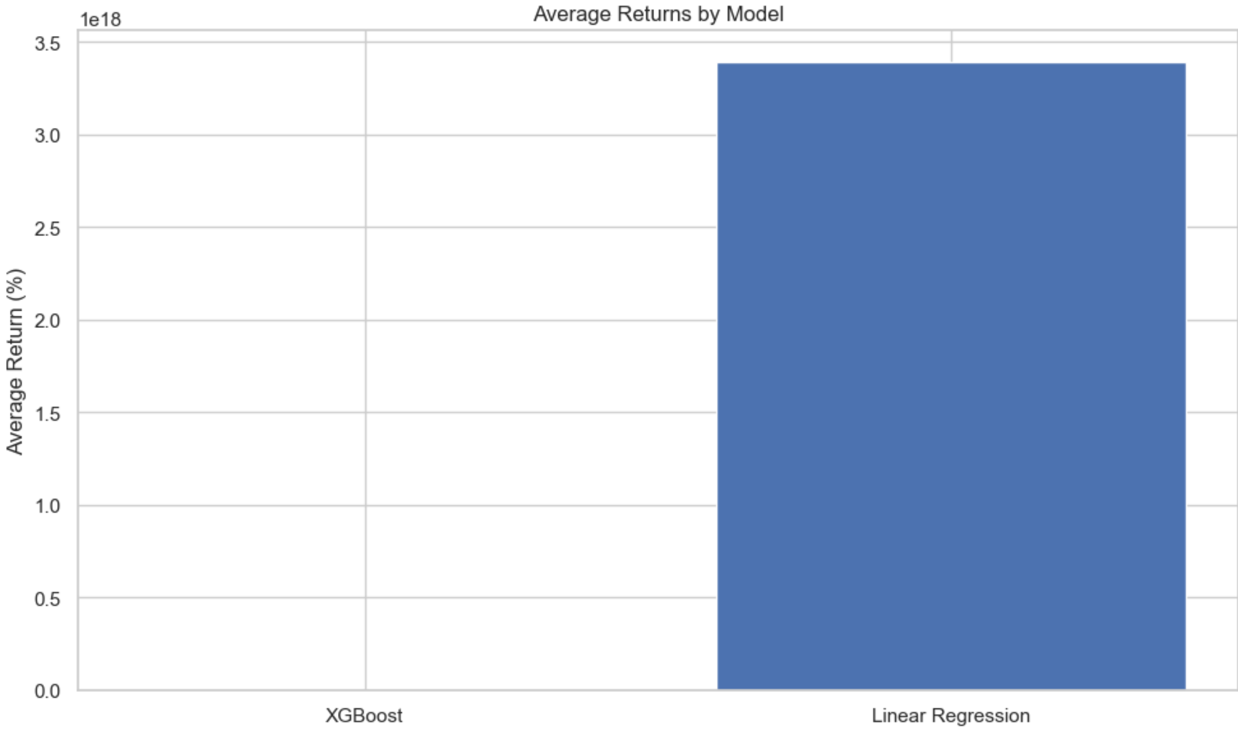


In [ ]:
```
# Heatmap of correlation between features
plt.figure(figsize=(12, 10))
sns.heatmap(X_train.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap of Features')
plt.tight_layout()
plt.show()
```



In [ ]:

```python
# Bar plot of average returns by model
avg_returns = {model: results[model]['results_df']['Final Return'].mean()
plt.figure(figsize=(10, 6))
plt.bar(avg_returns.keys(), avg_returns.values())
plt.title('Average Returns by Model')
plt.ylabel('Average Return (%)')
plt.tight_layout()
plt.show()
```



In [ ]:
```
# Table of top 5 performing pairs for each model
for model_name, model_results in results.items():
    top_5 = model_results['results_df'].nlargest(5, 'Final Return')
    print(f"\nTop 5 performing pairs for {model_name}:")
    print(tabulate(top_5, headers='keys', tablefmt='pretty', floatfmt=".2
```

Top 5 performing pairs for XGBoost:

	Stock 1	Stock 2	MSE	R-squared	Final Return
3	ADBE	CRM	0.0009721693843248306	-0.0027555952005899886	0.0
7	AMZN	CRM	0.0004666383504514382	-0.005883572034045992	0.0
9	CRM	NVDA	0.0004768313584557387	-0.007661109832311386	0.0
13	DISCK	NE	0.0007791434745997887	-0.0032454049497929738	0.0
14	DISCA	NE	0.0007845826742430618	-0.003544479508166054	0.0

Top 5 performing pairs for Linear Regression:

	Stock 1	Stock 2	MSE	R-squared	Final Return
21	HBI	MRO	0.9044646688384439	0.00023270576098088913	1.2210619658254595e+20
11	FCX	HBI	2.177120673630198	-0.0057187346883840195	20441670124.75649
35	SLB	PFE	0.3376684340646472	0.0010673206239936173	16272107.207067419
6	CNX	HBI	0.09270880925149369	-0.011203491304254465	4922263.355409072
25	RIG	RRC	0.01914602861346223	-0.003279312432044712	1215.5955451655107