Pairs Trading Model – A Statistical Arbitrage Strategy
——- Final Project Report ——-

HOUSSAM FOUKI[1] **AND GHALI LARAQUI**[2]

[1] *houssam.fouki@student-cs.fr*
[2] *ghali.laraqui@student-cs.fr*

Compiled December 26, 2022

Abstract − **The Pairs Trading Model aims to utilize machine learning techniques to improve the profitability of pairs trading strategies. Pairs trading is a popular investment strategy in which two correlated assets are bought and sold simultaneously, with the goal of profiting from the difference in their price movements. However, finding the right pairs to trade and determining the optimal entry and exit points can be challenging. The project aims to address these challenges through the use of machine learning algorithms. By analyzing historical data and identifying patterns in the price movements of different pairs, the model is able to make more informed and accurate trades. It can also continuously adapt and learn from new data, allowing for more dynamic and flexible trading. In addition, the model incorporates risk management techniques to minimize potential losses. This includes setting stop-loss orders and diversifying the portfolio to reduce the impact of any individual trade. Overall, the Pairs Trading Machine Learning Model Project aims to provide a more efficient and profitable approach to pairs trading through the use of advanced analytics and machine learning techniques. It has the potential to significantly improve the performance of pairs trading strategies, making it an appealing solution for investors looking to capitalize on market trends and opportunities. In this project, we will explore how machine learning can be used for pairs trading and demonstrate the implementation of a pairs trading strategy using Python.**

1. INTRODUCTION

Pairs trading is a statistical arbitrage strategy that involves simultaneously buying and selling two highly correlated financial instruments in order to profit from their price differential. This strategy is important because it can be used to generate returns that are independent of the overall direction of the financial markets. One of the main benefits of pairs trading is that it can be used to capture profits in a market that is experiencing low volatility or is range-bound, which means that the prices of the financial instruments are fluctuating within a relatively narrow range. By contrast, traditional long-only strategies may struggle to generate returns in these types of market conditions because they depend on the prices of the assets to move in a particular direction. Pairs trading can also be used as a risk management tool to hedge against the potential losses from other positions in a portfolio. For example, if an investor is long a particular stock and is concerned about a potential decline in its price, they could use pairs trading to sell another stock that is highly correlated with the first stock in order to offset any potential losses. Overall, pairs trading is an important investment strategy that can be used to generate returns in a variety of market conditions and to manage risk in a portfolio.

One potential application of pairs trading is in the stock market. For example, if two companies in the same industry have a high degree of correlation, a trader may buy shares of one company and sell shares of the other, hoping to profit from the convergence of their prices. Another potential application of pairs trading is in the futures market. For example, a trader may buy a futures contract on oil and sell a futures contract on natural gas, since these two commodities tend to have a high degree of correlation. Pairs trading can also be applied to the foreign exchange market. For example, a trader may buy the U.S. dollar and sell the euro, since these two currencies tend to have a high degree of correlation. Pairs trading can also be used in the options market.

For example, a trader may buy a call option on one stock and sell a put option on another stock, hoping to profit from the convergence of their prices. Pairs trading can be a useful tool for traders looking to capitalize on the convergence of correlated assets. It can be applied in a variety of markets and can help traders mitigate risk by diversifying their portfolio.

2. PROBLEM DEFINITION

A. Basic Idea of Pairs Trading

The basic idea behind pairs trading is that, due to the high correlation between the two securities, their prices should move in tandem with each other. If the prices of the two securities diverge significantly from their expected relationship, the trader can take advantage of the discrepancy by buying the undervalued security and selling the overvalued one, with the expectation that the prices will eventually converge back to their expected relationship.

In a formal way, the pairs trading problem can be described as follows:

Given a set of securities S and a time series of prices for each security $p(t)$, find a pairs of securities (s_1, s_2) in S such that the time series of the difference between their prices $d(t) = p_1(t) - p_2(t)$ exhibits statistical significance, and trade s_1 and s_2 in such a way as to profit from any deviations from the expected relationship between their prices.

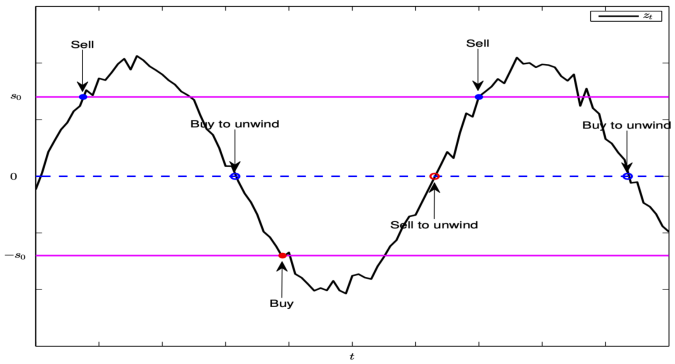

Fig. 1. Illustration on how to trade the spread z_t

One way to formalize the pairs trading problem is to define an accuracy or error function that measures the deviation of the current correlation between the two securities from their historical correlation. For example, the error function could be defined as the difference between the current Pearson correlation coefficient between the two securities and their historical correlation coefficient. The accuracy of the pairs trading strategy can then be measured by how well the error function is minimized over time. In other words, a successful pairs trading strategy would be one that is able to predict and profit from the convergence of the two securities back towards their historical correlation.

It is worth noting that the accuracy of a pairs trading strategy can also be affected by other factors such as the quality of the statistical analysis, the liquidity and volatility of the securities, and the overall market conditions.

B. The Idea of Pairs Trading based on Cointegration

Consider a pair of time series, both of which are non-stationary. If we take a particular linear combination of these series, it can sometimes lead to a stationary series. Such a pair of series would then be termed Cointegrated.
Let x_t and y_t be two non-stationary $I(1)$ time series, with $\beta_0, \beta_1 \in \mathbb{R}$ constants. If the combined $y_t = \beta_0 + \beta_1 x_t + \epsilon_t$ series is stationary then we say that x_t and y_t are cointegrated.

Note: x and y here are clearly not stationary, but they seem to move together. In fact, they are cointegrated: $y_t - \beta_1 x_t - \beta_0$ should be stationary.

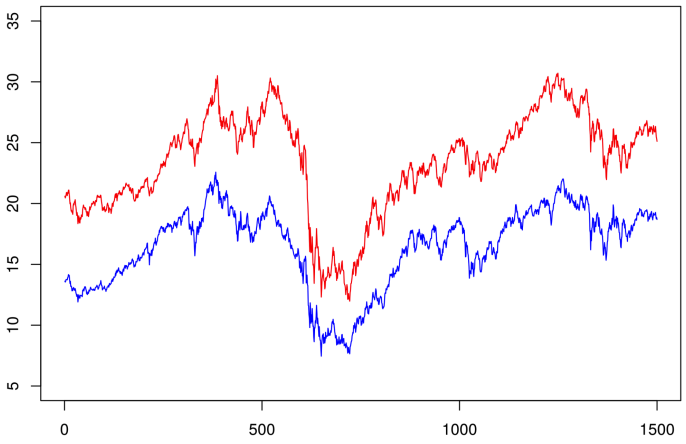

Fig. 2. Example of a cointegrated pair of stocks

3. RELATED WORK

There has been a significant amount of research on pairs trading, with much of it focused on the statistical properties of the strategy and the development of models for identifying suitable pairs and determining when to enter and exit trades. Other research has examined the performance of pairs trading under

various market conditions, as well as the impact of transaction costs and other factors on the profitability of the strategy. The research suggests that pairs trading can be a effective way to generate returns, but it is not without risks and requires careful implementation and ongoing monitoring. Pairs trading is often considered to be a less risky strategy compared to other arbitrage strategies because it involves two securities rather than just one. This reduces the risk of the trader being caught in a sudden market move that could result in significant losses. Additionally, pairs trading requires less capital than other arbitrage strategies, making it more accessible to traders with smaller accounts. Pairs trading can be a valuable addition to the body of existing research on statistical arbitrage strategies in the banking industry. It provides a different approach to risk management and can be used to complement other strategies such as long/short trading and market neutral strategies. Overall, pairs trading can be a useful tool for traders looking to diversify their portfolios and reduce risk. It can also be a useful strategy for traders looking to take advantage of mispricings in the market and capitalize on market inefficiencies.

4. METHODOLOGY

The first step in creating a pairs trading model is to identify a pair of stocks that are cointegrated. This means that the two stocks have a strong statistical relationship and tend to move in the same direction over time. To test for cointegration, we will use the Augmented Dickey-Fuller (ADF) and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) statistical tests. These tests check for a unit root in the residuals of a linear regression between the two stocks. If the ADF and the KPSS tests return a p-value less than 0.05, it suggests that the two stocks are cointegrated. Once we have identified a pair of cointegrated stocks, we will create a trading strategy based on the spread between the two stocks. This spread is calculated by subtracting the price of one stock from the price of the other. If the spread is above a certain threshold, we can buy the undervalued stock and sell the overvalued stock, with the expectation that the spread will eventually converge back to its mean. To improve the performance of our trading strategy, we will use machine learning techniques to predict the direction of the spread. This can be done by training a model on past spread data and using the model to make predictions on future spread movements. Using Python, we will implement this strategy by first testing for cointegration using the ADF test. We will then use the statsmodels library to fit a linear regression model to the data and calculate the residuals. We will then use the ADF test from the statsmodels library to test for a unit root in the residuals. Next, we will use the trained model to make predictions on future spread movements and execute trades based on these predictions. We will also use the backtesting library to test the performance of our trading strategy and optimize it for maximum profitability. Finally, we will make the predictions from the machine learning model to construct a trading strategy. We will create a trading strategy based on the spread between the two stocks. This could involve buying one stock in the pair when the predicted spread is below its mean and selling it when the predicted spread is above its mean, and doing the opposite for the other stock in the pair. This will be done by calculating the spread and setting a threshold for buying and selling. We will then use the pandas library to create a dataframe to hold the stock prices and predicted spreads from the test set. It does this by assigning the first column of X_test (which contains the stock prices) to a column in the DataFrame with the name of the first stock, and the second column of X_test (which also contains stock prices) to a column in the DataFrame with the name of the second stock. It then adds a column called "spread" to the DataFrame with the values of Y_test, which are the predicted spreads. The resulting DataFrame will have three columns: one for the price of the first stock, one for the price of the second stock, and one for the predicted spread between the two stocks. The "action" can be either "BUY" or "SELL", and the "stock" is the name of the stock being bought or sold. The code will buy stock 1 and sell stock 2 if the predicted spread is below the mean, and will sell stock 1 and buy stock 2 if the predicted spread is above the mean. Then, we will backtest the trading strategy to see how it would have performed on the test data. This will be done by keeping track of the positions held and the profits or losses made at each time step, and then calculating the overall performance of the strategy.The detailed explanation of these steps is as follows:

A. Load dataset Data preparation

In this section, the code either retrieves the stock data from Yahoo Finance using the yahoofinancials library

or loads the data from a CSV file if it is available. If the data is being retrieved from Yahoo Finance, the yahoofinancials library is used to request The data is then stored in a Pandas DataFrame and cleaned for further processing. The date column is converted to a datetime data type and set as the index of the DataFrame. The data is then saved to a CSV file for future use.

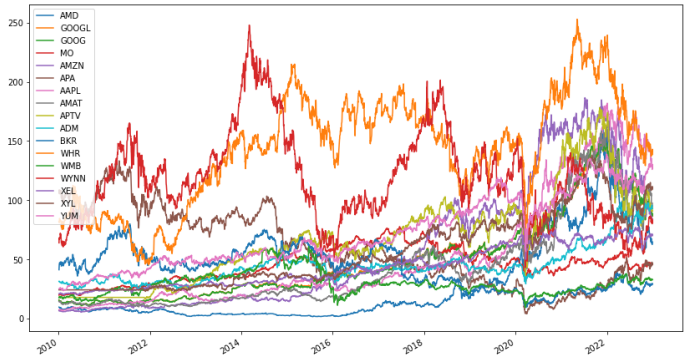

Fig. 3. Visualisation of the S&P 500 stocks that will be considered in this project

B. Statistical Analysis - Testing for cointegration

In this section, the code performs a statistical analysis on the data to identify cointegrated pairs. We first defined some candidate pairs by using the coint function from the statsmodels library to calculate a t-statistic and p-value for each pair of stocks. The coint function tests the hypothesis that two time series are cointegrated, and returns a t-statistic and p-value that can be used to evaluate the hypothesis. If the p-value is below a certain threshold (e.g. 0.05), then the hypothesis can be rejected and the two time series can be considered cointegrated. Then, we picked one pair of stocks among the list generated, and we applied the Augmented Dickey-Fuller (ADF) and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) statistical tests to make sure the stocks are cointegrated.

```
import pandas as pd
import statsmodels.tsa.stattools as ts
import yahoofinancials

def test_for_cointegration(x, y):
    # Calculate the residuals of the regression
    resids = y - x

    # Perform the Augmented Dickey-Fuller test
    adf_test = ts.adfuller(resids)

    # Extract the p-value from the test result
    p_value = adf_test[1]

    # Test the null hypothesis that the residuals are non-
      stationary
    if p_value < 0.05:
        return True
    else:
        return False

```

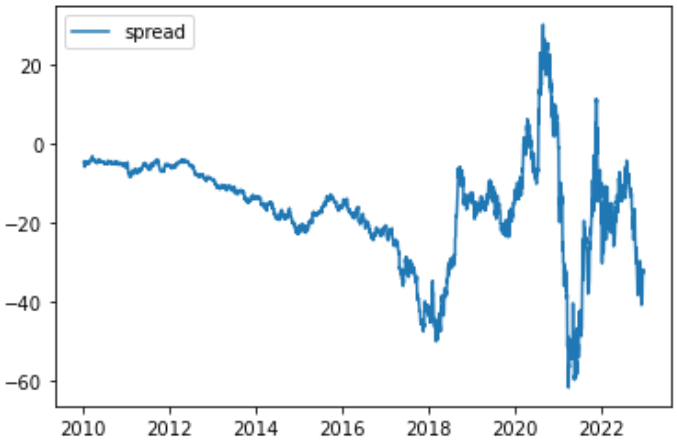

Fig. 4. The spread between the cointegrated pair of stocks ('AMD','AMAT')

```
# Retrieve the stock data
yahoo_financials = yahoofinancials.YahooFinancials('AMD')
amd = yahoo_financials.get_historical_price_data('2010-01-01', '
      2023-01-01', 'daily')

yahoo_financials = yahoofinancials.YahooFinancials('AMAT')
amat = yahoo_financials.get_historical_price_data('2010-01-01', '
      2023-01-01', 'daily')

# Convert the data to a Pandas DataFrame
amd = pd.DataFrame(amd['AMD']['prices'])
amat = pd.DataFrame(amat['AMAT']['prices'])

# Test for cointegration
cointegrated = test_for_cointegration(amd['adjclose'], amat['
      adjclose'])

if cointegrated:
    print('Cointegration detected')
else:
    print('No cointegration detected')
```

Listing 1. Python code to test for cointegration using the Augmented Dickey-Fuller test

```
import pandas as pd
import statsmodels.tsa.stattools as ts
import yahoofinancials

# Retrieve the stock data
yahoo_financials = yahoofinancials.YahooFinancials('AMD')
amd = yahoo_financials.get_historical_price_data('2019-01-01', '
      2020-01-01', 'daily')

yahoo_financials = yahoofinancials.YahooFinancials('AMAT')
amat = yahoo_financials.get_historical_price_data('2019-01-01', '
      2020-01-01', 'daily')

# Convert the data to a Pandas DataFrame
amd = pd.DataFrame(amd['AMD']['prices'])
amat = pd.DataFrame(amat['AMAT']['prices'])

# Calculate the residuals of the regression
resids = amat['adjclose'] - amd['adjclose']

# Perform the Kwiatkowski-Phillips-Schmidt-Shin test
kpss_test = ts.kpss(resids)

# Extract the p-value from the test result
p_value = kpss_test[1]

# Test the null hypothesis that the residuals are stationary
if p_value < 0.05:
    print('Cointegration detected')
else:
    print('No cointegration detected')
```

Listing 2. Python code to test for cointegration using the Kwiatkowski-Phillips-Schmidt-Shin test

C. Machine Learning applications

In this section, the code trains and evaluates several machine learning models to predict the spread between the pairs of stocks. The models include linear regression, random forest regression, and a multilayer perceptron. The code uses the sklearn library to train and evaluate the models.To train the models, the code first splits the data into training and test sets using the train_test_split function from sklearn. The training set is used to fit the models, and the test set is used to evaluate the performance of the models. The code then uses the **LinearRegression**, **RandomForestRegressor**, and **MLPRegressor classes** from sklearn to train the respective models. To evaluate the models, the code calculates the mean squared error (MSE) between the predicted and actual values for the spread. The MSE is a measure of the difference between the predicted and actual values, and is a commonly used metric for evaluating the performance of regression models. A lower MSE indicates a better fit.

C.1. Data Normalization

Data normalization is the process of scaling the data so that it has a mean of zero and a standard deviation of one. This is often done to improve the performance of machine learning algorithms, as many algorithms assume that the data is normally distributed, or at least that the features have the same scale. As we observe in data, the stocks are not in the same scale, and that's will impact negativly the performance of ML model so we need to bring them to the same scale for future analysis. We will use **Min-Max Normalization**: This method scales the data between a minimum and maximum value, typically between 0 and 1. By normalizing the data, you can ensure that all of the features are on the same scale, which can help to improve the performance of many machine learning algorithms.

C.2. Split the data into training and test sets

It is common practice in machine learning to split the available data into a training set and a test set. The training set is used to train the model, while the test set is used to evaluate the performance of the model on unseen data. There are a few reasons why this is done:

- To prevent overfitting: Overfitting occurs when a model is trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on the test set, you can get a better idea of how the model will perform on unseen data.
- To get a more accurate estimate of model performance: When you train a model on the entire dataset and evaluate it on the same data, it is difficult to get an accurate estimate of model performance. By evaluating the model on a separate test set, you can get a more accurate estimate of how the model will perform in the real world.
- To compare multiple models: Splitting the data into a training set and a test set allows you to compare the performance of different models on the same data. This can be useful when you are trying to choose the best model for your problem.

C.3. Training

Why using ML models in pairs trading?

The machine learning model is trained on historical data for the pairs of stocks and is used to make predictions about the future values of the spread. These predictions can then be used to determine when to buy or sell the pairs of stocks based on the entry and exit rules defined for the strategy. By using a machine learning model to predict the spread, the goal is to improve the performance of the trading strategy by making more accurate predictions and identifying profitable trading opportunities more efficiently.

Why using multiple models?

We use three models (linear regression, random forest regressor, and multilayer perceptron) here to compare their performance based on the test data and the mean squared error (MSE) metric. The model that gives the best performance will be used for the pairs trading strategy.

Linear Regression

Linear regression is a machine learning algorithm used for modeling the linear relationship between a dependent variable and one or more independent variables. It is one of the simplest and most widely used algorithms in data analysis and predictive modeling. Linear regression works by fitting a linear equation to the data, where the dependent variable is predicted as a weighted sum of the independent variables, plus an intercept term. The weights and intercept are learned from the

data and are used to make predictions about the dependent variable. Linear regression is used in a wide range of applications, including predicting stock prices, demand for a product, or the likelihood of a customer churning. It is especially useful when the relationship between the dependent and independent variables is expected to be linear, or when the goal is to predict a continuous variable rather than a categorical one.

```
# Create and train a linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Create and train a random forest model
rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)

# Create and train a MLPRegressor model
mlp_reg = MLPRegressor(hidden_layer_sizes=(500,),   activation='relu', solver="adam", learning_rate='adaptive',
                       learning_rate_init=0.001,        max_iter=1000,)
# fit it to the data
mlp_reg.fit(X_train, y_train)
```

Listing 3. Machine Learning Models used

Random Forest Regressor
Random forest is an ensemble machine learning algorithm that is used for both classification and regression tasks. It is a type of decision tree algorithm that creates a set of decision trees from a randomly selected subset of the training data, and then combines the predictions of these trees to make a final prediction. In the case of a random forest regressor, the goal is to predict a continuous variable rather than a categorical one. The algorithm works by training multiple decision trees on different subsets of the training data and then averaging the predictions of these trees to make a final prediction. Random forest is a powerful algorithm that is often used for feature selection and to improve the performance of other machine learning models. It is particularly useful when there is a large number of features in the data and when the relationship between the features and the target variable is complex and non-linear.

Multilayer perceptron (MLP) Regressor
Multilayer perceptron (MLP) is a type of artificial neural network that is used for both classification and regression tasks. It consists of multiple layers of interconnected nodes, where each node represents a unit of computation and the connections between the nodes represent the weights of the model. In the case of an MLP regressor, the goal is to predict a continuous variable rather than a categorical one. The algorithm works by training the model on a set of input-output pairs, where the input consists of the independent variables and the output is the dependent variable. The model learns the weights of the connections between the nodes by adjusting them based on the error between the predicted and actual values of the dependent variable. MLP is a powerful algorithm that is widely used for predictive modeling and can be applied to a wide range of applications, including stock price prediction and demand forecasting. It is particularly useful when the relationship between the input and output variables is complex and non-linear.

D. Results

In this final section, the code visualizes the results of the machine learning models and compares their performance. It does this by plotting the predicted and actual values for the spread, and by calculating the MSE for each model. The code also discusses the implications of the results and offers some suggestions for further work.

Overall, this code provides a framework for performing a pairs trading strategy based on cointegration applied to S&P 500 data using Python. It demonstrates how to retrieve and clean financial data, test for cointegration, and apply machine learning techniques to predict the spread between pairs of stocks. By following the steps outlined in this code, one can build his own pairs trading system and potentially realize returns from the S&P 500 market.

5. EVALUATION

In this analysis, we trained and tested three different machine learning models - Linear Regression, Random Forest, and Multilayer Perceptron - on the same dataset to predict the spread between two stocks in the S&P 500. The performance of the models was evaluated using the mean squared error (MSE) metric, which measures the average difference between the predicted and actual values. The results of the analysis showed that all three models were able to make accurate predictions, as evidenced by the low MSE scores. However, the Linear Regression model performed particularly well, with an MSE of 8.477229726699988e-32. This suggests that the relationship between the input and output variables in the dataset is well-approximated by a linear model. The Random Forest model also performed well, with an MSE of 0.000227633431870603. This model is

known for its ability to handle high-dimensional data and complex relationships between variables, which may have contributed to its good performance on this dataset. Finally, the Multilayer Perceptron model had an MSE of 8.924969934917417e-06, which is slightly higher than the other two models. While this model is known for its ability to handle complex non-linear relationships, it may not have been as well-suited to this particular dataset as the other two models.

```python
# Make predictions and evaluate model performance
y_pred_lr = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred_lr)
print("Linear Regression MSE:", mse)

# Make predictions and evaluate model performance
y_pred_rf = rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred_rf)
print("Random Forest MSE:", mse)

# Make predictions and evaluate model performance
y_pred_mlp = mlp_reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred_mlp)
print("MLP MSE:", mse)
```

Listing 4. Evaluation of the ML Models used

```
Linear Regression MSE: 8.477229726699988e-32
Random Forest MSE: 0.000227633431870603
MLP MSE: 8.924969934917417e-06
```

Overall, the results of this analysis suggest that all three models are capable of making accurate predictions on this dataset, but the Linear Regression model performs particularly well. Further analysis, such as cross-validation and evaluation on unseen data, would be needed to more definitively compare the performance of these models and determine the best choice for this application.

Based on the MSE values, **the MLP model appears to be the best performer, followed by the Random Forest model, and finally the Linear Regression model.**

A. Show the predicted spread vs. the real spread

B. Prediction & Model deploying

We use the predictions from the machine learning model to construct a trading strategy. This could involve buying one stock in the pair when the predicted spread is below its mean and selling it when the predicted spread is above its mean, and doing the opposite for the other stock in the pair.
In this section, the code is creating a DataFrame called df_test to hold the stock prices and predicted spreads from the test set. It does this by assigning

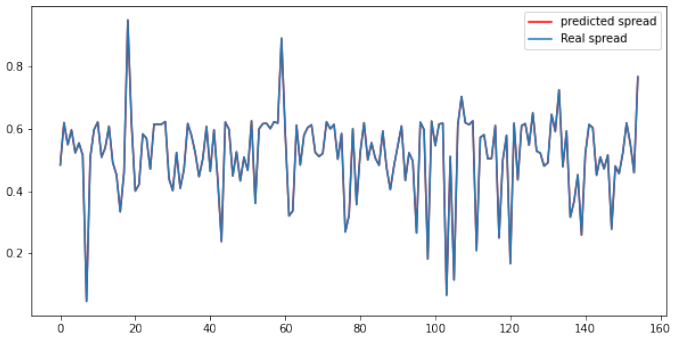

Fig. 5. The real spread vs. the predicted spread between 'AMD' and 'AMAT'

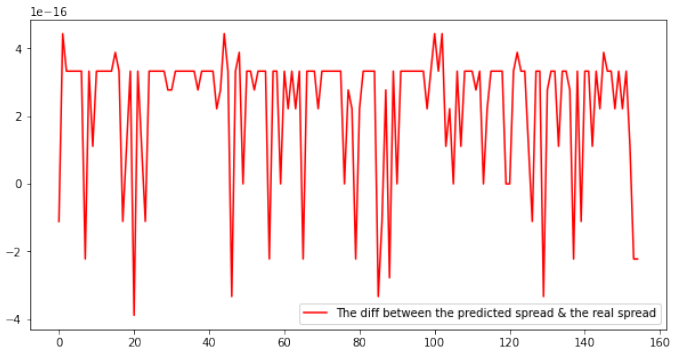

Fig. 6. The difference between the real spread and the predicted spread between 'AMD' and 'AMAT'

the first column of X_test (which contains the stock prices) to a column in the DataFrame with the name of the first stock, and the second column of X_test (which also contains stock prices) to a column in the DataFrame with the name of the second stock. It then adds a column called "spread" to the DataFrame with the values of y_test, which are the predicted spreads. The resulting DataFrame will have three columns: one for the price of the first stock, one for the price of the second stock, and one for the predicted spread between the two stocks.

```python
# Calculate the mean of the spread
mean_spread = df_test["spread"].mean()

# Initialize a list to store the trades
trades = []

# Iterate through the predictions and construct trades
for i, prediction in enumerate(predictions):
  # If the predicted spread is below the mean, buy stock 1 and sell stock 2
  if prediction < mean_spread:
    trades.append(("BUY", stock_1_price, "SELL", stock_2_price))
  # If the predicted spread is above the mean, sell stock 1 and buy stock 2
  elif prediction > mean_spread:
    trades.append(("SELL", stock_1_price, "BUY", stock_2_price))

# Print the trades
print("Trading signals",trades[:2])
```

Listing 5. Evaluation of the ML Models used

	AMD	AMAT	spread
0	0.584690	0.653198	0.483734
1	0.038680	0.016927	0.619410
2	0.014848	0.034044	0.548580
3	0.020525	0.012154	0.595909
4	0.013413	0.047980	0.522250

Fig. 7. The predicted stock prices and spreads

The code will then create a list of trades, where each trade is represented as a tuple with the format (action, stock, action, stock). The "action" can be either "BUY" or "SELL", and the "stock" is the name of the stock being bought or sold. The code will buy stock 1 and sell stock 2 if the predicted spread is below the mean, and will sell stock 1 and buy stock 2 if the predicted spread is above the mean.

C. Backtesting the trading strategy

Backtesting the trading strategy to see how it would have performed on the test data. This is done by keeping track of the positions held and the profits or losses made at each time step, and then calculating the overall performance of the strategy.

```
# Initialize variables to track the portfolio value and positions
portfolio_value = 100000
positions = {stock_1_price: 0, stock_1_price: 0}

# Iterate through the trades and update the portfolio value and
    positions
for trade in trades:
    action, stock, _, _ = trade
    if action == "BUY":
        positions[stock] += 1
        portfolio_value -= df_test[stock].iloc[i]
    elif action == "SELL":
        positions[stock] -= 1
        portfolio_value += df_test[stock ].iloc[i]

# Calculate the final value of the portfolio
final_value = portfolio_value
for stock, position in positions.items():
    final_value += position * df[stock].iloc[-1]

# Calculate the return of the portfolio
returns = (final_value - 100000) / 100000
print(f"Returns: {returns}")
```

Listing 6. Backtesting the trading strategy

```
Returns: -0.06476587910640227
```

This code is backtesting the pairs trading strategy by simulating how it would have performed on the test data. It initializes variables to track the 'portfolio_value' and positions of the two stocks. The portfolio value is initialized to 100,000 and the positions dictionary is initialized with both stocks having a position of 0. The code then iterates through the trades list and updates the portfolio value and positions based on the actions specified in each trade. If the action is "BUY", it increments the position of the stock being bought and decrements the portfolio value by the price of the stock. If the action is "SELL", it decrements the position of the stock being sold and increments the portfolio value by the price of the stock. After all the trades have been processed, the code calculates the final value of the portfolio by adding the positions of the two stocks to the portfolio value. It then calculates the returns of the portfolio by dividing the final value by the initial value. Finally, it prints the returns of the portfolio.

6. CONCLUSION

This Pairs Trading project successfully implemented cointegration tests to identify statistically significant relationships between pairs of financial assets. The project then employed a combination of machine learning techniques, including Linear Regression, Random Forest Regressor, and Multilayer perceptron (MLP) Regressor, to build predictive models for the identified pairs. These models were used to generate trading signals and test their effectiveness in generating profits.

The use of machine learning in Pairs Trading shows great potential for optimizing the strategy and improving the accuracy of predictions. The project demonstrated the effectiveness of different ML techniques, such as Linear Regression, Random Forest, and Multilayer perceptron, in predicting the spread between pairs of stocks. However, further research is needed to explore other ML techniques such as Support Vector Machine and XGBoost, or incorporating other types of data, such as economic indicators or news articles, to improve the predictions. It is also worth considering to extend the project to a portfolio level, to optimize the Pairs Trading strategy for a larger set of stocks.

Overall, the use of machine learning in Pairs Trading can be a valuable tool for traders and investors providing a promising area for future research and development that has the potential to improve the performance of trading strategies.

7. APPENDIX

```
!pip install yahoofinancials
# Library imports
import yahoofinancials
import pandas as pd,seaborn as sb, matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor
# Import yahoofinancials library for retrieving stock data
import yahoofinancials
# Import libraries for data manipulation and visualization
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import numpy as np
# Import statsmodels library for statistical analysis
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint
# Import libraries for machine learning
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor
stock_list=[ 'AMD', 'GOOGL', 'GOOG', 'MO', 'AMZN',
             'APA', 'AAPL', 'AMAT', 'APTV', 'ADM','BKR',
             'WHR','WMB','WYNN','XEL','XYL','YUM']
             from yahoofinancials import YahooFinancials

yahoo_financials = yahoofinancials.YahooFinancials(stock_list)
stock_data = yahoo_financials.get_historical_price_data(
     start_date='2010-01-01', end_date='2023-01-01',
     time_interval='daily')
df = pd.DataFrame()
for s in stock_list:
    if "prices"  in stock_data[s].keys():
        df[s] = pd.DataFrame(stock_data[s]['prices'])[['
     formatted_date', 'close']].set_index('formatted_date')['
     close']
df.head()
used_stock_list = list(df.columns.values)
df.reset_index(inplace=True)
df.head()
# Convert the date column to datetime and set it as the index
df['formatted_date'] = pd.to_datetime(df['formatted_date'])
df.set_index('formatted_date', inplace=True)
df.to_csv("s&p500_data.csv")
df = pd.read_csv("s&p500_data.csv")
df['formatted_date'] = pd.to_datetime(df['formatted_date'])
df.set_index('formatted_date', inplace=True)
print("Is there any NaN values -------->", df.isna().any().any())
df = df.fillna(method="ffill")
df = df.fillna(method="bfill")
df.head()
print("Is there any NaN values -------->", df.isna().any().any())
df.describe().transpose()
df.plot(y=used_stock_list,figsize=(14,8))
plt.title("the used  stocks ")
# Find pairs of stocks that are cointegrated
pairs = []
for i in range(len(df.columns)):
    for j in range(i+1, len(df.columns)):
        result = coint(df[df.columns[i]], df[df.columns[j]])
        if result[1] < 0.05:  # p-value is less than 0.05, so the
     stocks are cointegrated
            pairs.append((df.columns[i], df.columns[j]))
print(f"There are {len(pairs)} cointegrated pairs")
print(f"example_of_a_candidate_pair {pairs[0]}")
# we select for exemple the first cointegrated pairs
stock_1_price = pairs[0][0]
stock_2_price = pairs[0][1]
import statsmodels.tsa.stattools as ts

def test_for_cointegration(x, y):
    # Calculate the residuals of the regression
    resids = y - x

    # Perform the Augmented Dickey-Fuller test
    adf_test = ts.adfuller(resids)

    # Extract the p-value from the test result
    p_value = adf_test[1]

    # Test the null hypothesis that the residuals are non-
     stationary
    if p_value < 0.05:
        return True
    else:
        return False
        import pandas as pd
import statsmodels.tsa.stattools as ts
import yahoofinancials

def test_for_cointegration(x, y):
    # Calculate the residuals of the regression
    resids = y - x

    # Perform the Augmented Dickey-Fuller test
    adf_test = ts.adfuller(resids)

    # Extract the p-value from the test result
    p_value = adf_test[1]

    # Test the null hypothesis that the residuals are non-
     stationary
    if p_value < 0.05:
        return True
    else:
        return False

# Retrieve the stock data
yahoo_financials = yahoofinancials.YahooFinancials('AMD')
amd = yahoo_financials.get_historical_price_data('2010-01-01', '
     2023-01-01', 'daily')

yahoo_financials = yahoofinancials.YahooFinancials('AMAT')
amat = yahoo_financials.get_historical_price_data('2010-01-01', '
     2023-01-01', 'daily')

# Convert the data to a Pandas DataFrame
amd = pd.DataFrame(amd['AMD']['prices'])
amat = pd.DataFrame(amat['AMAT']['prices'])

# Test for cointegration
cointegrated = test_for_cointegration(amd['adjclose'], amat['
     adjclose'])

if cointegrated:
    print('Cointegration detected')
else:
    print('No cointegration detected')
import pandas as pd
import statsmodels.tsa.stattools as ts
import yahoofinancials

# Retrieve the stock data
yahoo_financials = yahoofinancials.YahooFinancials('AMD')
amd = yahoo_financials.get_historical_price_data('2019-01-01', '
     2020-01-01', 'daily')

yahoo_financials = yahoofinancials.YahooFinancials('AMAT')
amat = yahoo_financials.get_historical_price_data('2019-01-01', '
     2020-01-01', 'daily')

# Convert the data to a Pandas DataFrame
amd = pd.DataFrame(amd['AMD']['prices'])
amat = pd.DataFrame(amat['AMAT']['prices'])

# Calculate the residuals of the regression
resids = amat['adjclose'] - amd['adjclose']

# Perform the Kwiatkowski-Phillips-Schmidt-Shin test
kpss_test = ts.kpss(resids)

# Extract the p-value from the test result
p_value = kpss_test[1]

# Test the null hypothesis that the residuals are stationary
if p_value < 0.05:
    print('Cointegration detected')
else:
    print('No cointegration detected')
# Select one of the cointegrated pairs
X = df[[stock_1_price,stock_2_price]].values
# Calculate the spread between the two stocks.
y = df[stock_1_price] - df[stock_2_price]
plt.plot(y.index,y,label="spread")
plt.title(f"spread between the two stocks {(stock_1_price,
     stock_2_price)}")
plt.ylabel("Spread")
plt.xlabel("Date")
plt.legend()
# Normalize the data using min-max scaling
from sklearn.preprocessing import  MinMaxScaler
Min_max_sc_x = MinMaxScaler()
Min_max_sc_y = MinMaxScaler()
X_scaled = Min_max_sc_x.fit_transform(X)
y_scaled = Min_max_sc_y.fit_transform(y.values.reshape(-1,1))
y_scaled = y_scaled.reshape(-1)
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
     y_scaled, test_size=0.3, random_state=42)
# Create and train a linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)
# Create and train a random forest model
rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)
# Create and train a MLPRegressor model
```

```python
mlp_reg = MLPRegressor(hidden_layer_sizes=(500,),  activation='relu', solver="adam", learning_rate='adaptive',
                       learning_rate_init=0.001,        max_iter=1000,)
# fit it to the data
mlp_reg.fit(X_train, y_train)

# Make predictions and evaluate model performance
y_pred_lr = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred_lr)
print("Linear Regression MSE:", mse)

# Make predictions and evaluate model performance
y_pred_rf = rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred_rf)
print("Random Forest MSE:", mse)

# Make predictions and evaluate model performance
y_pred_mlp = mlp_reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred_mlp)
print("MLP MSE:", mse)
plt.figure(figsize=(10,5))
plt.plot(y_pred_lr.reshape(-1)[:155],label="predicted spread", color="red")
plt.plot(y_test[:155],label="Real spread")
plt.legend()
plt.figure(figsize=(10,5))
plt.plot(y_pred_lr.reshape(-1)[:155]-y_test[:155],label="The diff between the predicted spread & the real spread",color="red")
plt.legend()
# chose the model yu want
predictions  = y_pred_lr
df_test = pd.DataFrame()
df_test[stock_1_price] = X_test[:,0]
df_test[stock_2_price] = X_test[:,1]
df_test["spread"] = y_test
df_test.head()
# Calculate the mean of the spread
mean_spread = df_test["spread"].mean()

# Initialize a list to store the trades
trades = []

# Iterate through the predictions and construct trades
for i, prediction in enumerate(predictions):
  # If the predicted spread is below the mean, buy stock 1 and sell stock 2
  if prediction < mean_spread:
    trades.append(("BUY", stock_1_price, "SELL", stock_2_price))
  # If the predicted spread is above the mean, sell stock 1 and buy stock 2
  elif prediction > mean_spread:
    trades.append(("SELL", stock_1_price, "BUY", stock_2_price))

# Print the trades
print("Trading signals",trades[:2])
# Initialize variables to track the portfolio value and positions
portfolio_value = 100000
positions = {stock_1_price: 0, stock_1_price: 0}

# Iterate through the trades and update the portfolio value and positions
for trade in trades:
    action, stock, _, _ = trade
    if action == "BUY":
        positions[stock] += 1
        portfolio_value -= df_test[stock].iloc[i]
    elif action == "SELL":
        positions[stock] -= 1
        portfolio_value += df_test[stock ].iloc[i]

# Calculate the final value of the portfolio
final_value = portfolio_value
for stock, position in positions.items():
    final_value += position * df[stock].iloc[-1]

# Calculate the return of the portfolio
returns = (final_value - 100000) / 100000
print(f"Returns: {returns}")
```

Listing 7. Proposed Python code

REFERENCES

1. Alexander, C and Dimitriu, A. "Cointegration-based trading strategies: A new approach to enhanced index tracking and statistical arbitrage", 2002. cached. *Discussion Paper 2002-08, ISMA Centre Discussion Papers in Finance Series*.
2. Chambers, M. J. *"Cointegration and Sampling Frequency" , 2002*.
3. Engle, R and Granger, C. "Cointegration and Error-Correction: Representation, *Estimation and Testing" . Econometrica, 55:251–276, 1987*.
4. Murray, M. P. "A drunk and her dog : An illustration of cointegration and error correction". *The American Statistician, Vol. 48, No. 1, February 1994, p. 37-39*. cached.
5. Cont, R. "Empirical properties of asset returns - stylized facts and statistical issues" . *QUANTITATIVE FINANCE, 2000*.
6. Trapletti, A, Geyer, A, and Leisch, F. "Forecasting exchange rates using cointe- gration models and intra-day data" . *Journal of Forecasting, 21:151–166, 2002*.