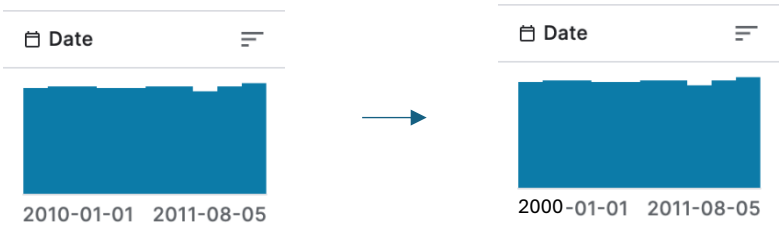
First step : Data analysis

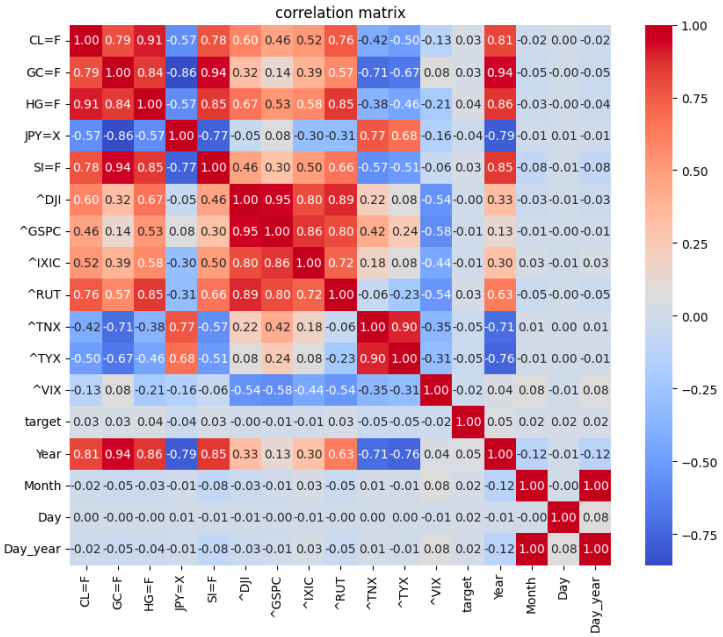
Given the initial data, we noticed that it covered a period of just over two years, which is insufficient for reliable target predictions. To improve our model's accuracy, we gathered additional data from Yahoo Finance, extending the date range back to 2000. This broader dataset provides a more comprehensive basis for making predictions.



Upon reviewing the new dataset, we identified columns that were entirely filled with NaN values. Since these columns do not contribute any useful information, we removed them. For the remaining data with intermittent NaN values, we employed interpolation to fill in the gaps, ensuring a complete dataset.

Date	CL=F	EURUSD=X	GC=F	HG=F	JPY=X	SI=F	XWD.TO	^DJI	^GSPC	^IXIC	^RUT	^TNX	^TYX	^VIX	target
2000-08-24	31.629999	NaN	NaN	NaN	106.989998	NaN	NaN	11182.740234	1508.310059	4053.280029	523.299988	5.716	5.659	17.040001	-1.233239
2000-08-25	32.049999	NaN	NaN	NaN	106.800003	NaN	NaN	11192.629883	1506.449951	4042.679932	525.109985	5.721	5.668	16.530001	5.071536
2000-08-28	32.869999	NaN	NaN	NaN	106.589996	NaN	NaN	11252.839844	1514.089966	4070.590088	526.479980	5.766	5.715	16.540001	-2.806967
2000-08-29	32.720001	NaN	NaN	NaN	106.129997	NaN	NaN	11215.099609	1509.839966	4082.169922	529.630005	5.808	5.751	16.889999	-4.801833
2000-08-30	33.400002	NaN	273.899994	0.885	106.610001	4.93	NaN	11103.009766	1502.589966	4103.810059	532.330017	5.800	5.736	17.690001	10.042718
NaN Values		834	20	19	25	19	2319	19							

With the clean data in hand, we computed the correlation matrix to understand the relationships between variables. It became clear that the target variable did not exhibit strong correlation with any of the other variables. To enhance our model's performance, we should consider integrating additional data sources or variables to improve these correlations.



Moreover, we observed that some variables were highly correlated with each other. During model training, we experimented with excluding these highly correlated variables to assess whether their removal would enhance the model's accuracy.

Second step: Sentiment data collecting and predicting

Since the added sentiment data was incomplete and the previous dataset lacked sufficient information, we decided to collect additional data. Our data collection process was divided into two phases:

- Data Collection with GNews:

Initially, we used the GNews API to gather more headlines related to those in our original dataset. This process was time-consuming, taking over 4 hours to generate a single headline for each existing headline in the dataset. Additionally, we noticed that the generated headlines did not span the entire date range needed, which would reduce the dataset's length if merged without further expansion. Consequently, we needed a more efficient method to obtain more headlines.

```
google_news = GNews()
google_news.start_date = (2010, 1, 4)
google_news.end_date = (2011, 12, 30)
```

```
#Searching for new titles using the titles in the dataset
for title in tqdm(data_sentiment['Title']):
    res = google_news.get_news(title)
    for resultat in res[:1]:
        pub_date = pd.to_datetime(resultat['published date'])
        dict [pub_date]= resultat['title']
```

```
100%|          | 8139/8139 [4:14:30<00:00,  1.88s/it]
```

To address the time issue, we implemented parallel processing using concurrent.futures.ThreadPoolExecutor(). This approach significantly reduced the data collection time from 4 hours to 15 minutes while allowing us to gather more headlines.

```
list_years= []
google_news = GNews(language='en',country='US',max_results=5)
for year in range(2000,2010):
    google_news.start_date = (year,1, 1)
    google_news.end_date = (year, 12, 30)
    def fetch_related_news(title):
        news = google_news.get_news(title)
        return news[:5]

    # Function to parallelize fetching news
    def parallel_fetch_news(titles):
        with concurrent.futures.ThreadPoolExecutor() as executor:
            results = list(executor.map(fetch_related_news, titles))
        return results

    #Fetch related news in parallel
    list_years.append(parallel_fetch_news(data_sentiment['Title']))
```

- Supplemental Data from Google Search:

In our search for more comprehensive data, we discovered a dataset containing daily headlines from 2008 to 2011. Although this dataset did not entirely resolve our issue of

needing daily headlines dating back to 2000, it substantially improved our coverage from 2008 onwards.

After collecting the new data, we proceeded with sentiment analysis. We needed to convert the titles into a numerical value using vectorizer.

```
#vectorisation
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(df_combined['Title'])
y = df['Sentiment'][:8139]
y_bis = df['val_Sentiment']
X_train, X_test, y_train, y_test, y_bis_train,y_bis_test = train_test_split(X[:8139], y,y_bis, test_size=0.2, random_state=42)
```

We then tested several models to predict sentiment:

- RandomForestClassifier achieved an accuracy of 0.78.

```
#Random forest classifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

predictions = rf.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7807125307125307
```

- DecisionTreeClassifier achieved an accuracy of 0.70.

```
#decision tree
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7039312039312039
```

- Multinomial Naive Bayes Classifier achieved an accuracy of 0.73.

```
#bayesian classification
model = MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7352579852579852
```

- XGBClassifier achieved an accuracy of 0.76.

```
#XGBoost classifier
xgb = XGBClassifier()
xgb.fit(X_train, y_bis_train)
predictions = xgb.predict(X_test)
accuracy = accuracy_score(y_bis_test, predictions)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.769041769041769
```

- FinBERT achieved F1 score of 0.88.

```
# Training
seed_test = 2022
random.seed(seed_test)
np.random.seed(seed_test)
torch.manual_seed(seed_test)
torch.cuda.manual_seed_all(seed_test)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

for epoch in tqdm(range(1, epochs+1)):
    model.train()
    loss_train_total = 0
    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch), leave=False, disable=False)
    for batch in progress_bar:
        model.zero_grad()
        batch = tuple(b.to(device) for b in batch)
        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        outputs = model(**inputs)
        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
        progress_bar.set_postfix({'training_loss': '{:.3f}'.format(loss.item()/len(batch))})
    torch.save(model.state_dict(), f'finetuned_finBERT_epoch_{epoch}.model')
    tqdm.write(f'\nEpoch {epoch}')

    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f'Training loss: {loss_train_avg}')
    val_loss, predictions, true_vals = evaluate(dataloader_validation)
    val_f1 = f1_score_func(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')
    tqdm.write(f'F1 Score (Weighted): {val_f1}')
```

```
Epoch 1
Training loss: 0.478417780343174
 50%|                                                  | 1/2 [28:43<28:43, 1723.32s/it]
Validation loss: 0.4112782022830251
F1 Score (Weighted): 0.8843301633354589
```

Given that FinBERT achieved the highest accuracy with a score of 0.83, it was chosen for making further sentiment predictions. We utilized FinBERT to process the collected headlines and generate sentiment scores, completing the sentiment data for our dataset. With this comprehensive sentiment analysis, we created a new column in our training data that captured the sentiment scores derived from FinBERT. This enriched dataset, now including sentiment information, provided a more robust foundation for our subsequent predictive modeling efforts.

After predicting the sentiment for each headline, we faced the challenge of aggregating multiple sentiments for days with more than one title. To address this, we developed several strategies to create a daily sentiment signal, representing the overall sentiment for each day. The strategies we explored included/

- Risk-Taking Strategy: This approach emphasized more extreme sentiment values, giving higher weight to strongly positive sentiments.

$$\frac{\lambda \sum S_{positive} - (1-\lambda) \sum S_{negative}}{\sum S_{negative} + \sum S_{positive} + \sum S_{indecisive}} \qquad 1 > \lambda > 0{,}5$$

- Neutral Strategy: This strategy averaged the sentiments for each day, providing a balanced view that smooths out extreme values. It aimed to reflect the general market sentiment without leaning towards high-risk interpretations.

$$\frac{\sum S_{positive} - \sum S_{negative}}{\sum S_{negative} + \sum S_{positive} + \sum S_{indecisive}}$$

- Non Risk-Taking Strategy: This conservative approach prioritized moderate sentiments, minimizing the impact of positive sentiments.

$$\frac{\lambda \sum S_{positive} - (1-\lambda) \sum S_{negative}}{\sum S_{negative} + \sum S_{positive} + \sum S_{indecisive}} \qquad 0.5 > \lambda > 0$$

We incorporated each of these daily sentiment signals into our models to evaluate their impact on prediction accuracy. After extensive testing, we found that the neutral strategy produced the best results. This approach, by averaging the daily sentiments, offered a reliable and consistent signal that improved the model's performance.

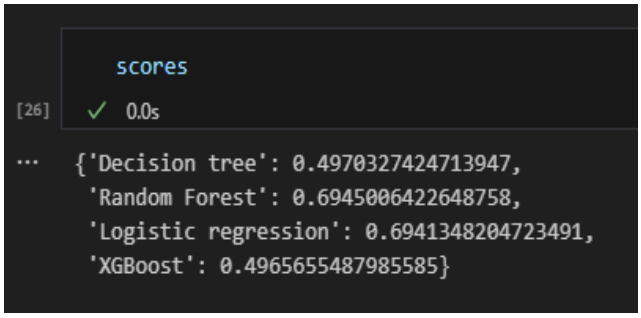
Third step : Model training

1- Classic machine learning methods

We tried some of the most widely used machine learning algorithms: Decision tree, random forest, xgboost and the logistic regression.

To assess the performance of these models:

- We split the original training set into a training set (80%) and a validation set (20%).
- We selected the best hyperparameters for each model using the Optuna library. It automatically searches for the best hyperparameters of a model by using sophisticated algorithms to efficiently navigate the hyperparameter space, often resulting in faster and more accurate parameter selection compared to traditional methods like grid search or random search.
- Once the best parameters were found, we performed a k-fold cross validation with k=5 and calculated the average score of each model
- We selected the best two models: Logistic regression and random forest



2- Neural networks

For the neural networks, the data must be transformed into two dimensions before passing it into the network.

```
x_train_1 = x_train_1.reshape(x_train_1.shape[0],1,x_train_1.shape[1])
```

We used two regressive neural network structures:

The first structure has 4 LSTM layers of 50 units each, followed by a dense layer of 1 neuron. The loss function is the Mean Squared Error.

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Dense

model = Sequential()
model.add(LSTM(units=50,return_sequences=True,input_shape=(1, X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam',loss='mean_squared_error')
model.fit(x_ip_1_tensor,y_op_1_tensor,epochs=10,batch_size=32);
```

This gave an insample F1 score of about 51%

The second structure is much more complicated, as it has 2 LSTM layers of 120 units going forwards and backwards respectively, then 2 GRU layers of 80 units going forwards and backwards.

This is followed by 3 dense layers of 40, 30 and 20 neurons respectively. Lastly there is a dense layer of 1 neuron

Model: "regressive_model_2"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 1, 5)	0
lstm_10 (LSTM)	(None, 1, 120)	60,480
lstm_11 (LSTM)	(None, 1, 120)	115,680
gru_2 (GRU)	(None, 1, 80)	48,480
gru_3 (GRU)	(None, 80)	38,880
dense_6 (Dense)	(None, 40)	3,240
dense_7 (Dense)	(None, 30)	1,230
dense_8 (Dense)	(None, 20)	620
dense_9 (Dense)	(None, 1)	21

Total params: 268,631 (1.02 MB)
Trainable params: 268,631 (1.02 MB)
Non-trainable params: 0 (0.00 B)

The second structure gave an F1 Score of about 62%

We also tried a binary classification network with 2 neurons, but the loss was much worse than that of regressive networks.