BIRMINGHAM CITY University

*CMP5367 Artificial Intelligence and Machine Learning A S1 2025/6*

# Crypto Market Sentimental Analysis 2025

*Dataset used :* crypto_sentiment_prediction_dataset.csv

*Word Count : 2000 words (excluding code and references)*


## Team Members :-

1) RAJVEER SINGH SAINI : rajveer.saini@mail.bcu.ac.uk
2) AMOGH Dath Kalasapura Arunkumar : amoghdath.kalasapuraarunkumar@mail.bcu.ac.uk
3) HARSHPREET SINGH : harshpreet.singh2@mail.bcu.ac.uk
4) JASPREET KAUR : jaspreet.kaur23@mail.bcu.ac.uk


GitHub Link:- https://github.com/Amogh-007-Rin/AI-ML-Model-For-CryptoAnalysis.git

# Contents

# Abstract

Given that the cryptocurrency market is hyper volatile with information shocks, prices are inefficient and do not meet the Efficient Market Hypothesis (EMH). Thus, this white paper seeks to assess whether the price movement directionality (up/down) can be assessed within twenty-four hours of current price for the top cryptocurrencies, because of such social sentiment towards the currencies and technical indicators and programming capabilities/information of common machine learning knowledge of modern society. Thus, a live working dataset was compiled with **2,063** observations in order to train/test and evaluate four classifiers for predictive analytics - **Logistic Regression**, **Support Vector Machines, Random Forest** and **Gradient Boosting (XGBoost)**. Baseline models indicate that performance results average at random odds **(≈50%)** which implies that the sentiment data at hand does not possess a strong signal-to-noise ratio.

However, through hyperparameter tuning with **GridSearchCV**, the Random Forest model indicates that these accuracy metrics can be determined at **52.1%**. While still a lacklustre metric, it does outperform random odds. Thus, this paper will explore these findings and sentiment data, relative to sentiment data for predictive validity of trading algorithms while also exploring sentiment data for non-linear predictive validity.


# 1. Introduction

## 1.1 Background and Context

For centuries, financial markets have been the most predictable subject of modelling. The assumptions of traditional financial modelling hold that according to the pillars of financial theory, The Efficient Market Hypothesis - all assets are appropriately priced based upon any information available, and no alpha risk - risk-adjusted or otherwise - can be guaranteed outside of stakeholder expectations. Yet for cryptocurrency, an entirely different marketplace exists based upon retail trader sentiment, gossip and speculation on social media and a news cycle - sentiments-of-power that function through more behavioural finance than any predictable certainty based upon available data.

Where stocks are heavily regulated and dynamic based upon earnings calls (quarterly or per year), mergers and acquisitions, and other fundamentals that guide sentiment, cryptocurrencies are unregulated, and boast extreme (or, at least, in certain contexts) sentiment shocks (regulatory news, tweets from master crypto traders or even billionaire investors). Thus, it's necessary to acknowledge that the nature of the efficient market provides realities where probabilities exist and where ML

modelling can take advantage of them via sentiment analysis predictive qualities aggregated with other traditional predictors as real-time technical qualities.

## 1.2 Research Problem and Objectives

The major research problem addressed in this thesis is that predicting the short-term movements of a highly stochastic market is difficult to do. Thus, while many researchers get into regression (price number predictions), the thesis writer knows that the simple challenge can become a binary classification challenge of predicting direction (Up or Down). This removes outlier considerations and focuses on the necessary decision-making process in the real world for trading.

The problem is then addressed by three research questions that cumulatively cover what is to be determined through this research process:

**RQ1:** What is the relationship between social and news sentiment score and price movement over the course of a 24-hour period in the crypto world?

**RQ2:** What machine learning solution - Linear, Tree-based, Kernel-based - provides the highest quality predictions for this classification?

**RQ3:** How much better can performance get with hyperparameter tuning in such a noisy space?

# 2. Methodology and Experimental Design

## 2.1 Dataset Identification and Description

The study utilizes the "Crypto Market Sentiment and Price Dataset" sourced from Kaggle [3]. This dataset aggregates 2,063 observations across major cryptocurrencies (e.g., Bitcoin, Ethereum, Solana). It was selected for its comprehensive feature set, which blends market data with alternative data sources. The features include:

- **Sentiment Indicators:** Social Sentiment Score, News Sentiment Score, Social Mentions Count, and Fear & Greed Index.
- **Technical Indicators:** RSI (Relative Strength Index), Volatility Index.
- **Market Data:** Market Cap, Trading Volume, Current Price.
- **Target Variable:** Price Change 24h (%).

Likewise, the `crypto_sentiment_prediction_dataset` contains 14 different features.

```
import pandas as pd
df=pd.read_csv("../dataset/crypto_sentiment_prediction_dataset.csv")
df.info()
```

CODE SNIPPET 1 : PRINTS THE SHAPE, COLUMN NAMES AND NON-NULL COUNTS

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2063 entries, 0 to 2062
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   timestamp                2063 non-null   object
 1   cryptocurrency           2063 non-null   object
 2   current_price_usd        2063 non-null   float64
 3   price_change_24h_percent 2063 non-null   float64
 4   trading_volume_24h       2063 non-null   float64
 5   market_cap_usd           2063 non-null   float64
 6   social_sentiment_score   2063 non-null   float64
 7   news_sentiment_score     2063 non-null   float64
 8   news_impact_score        2063 non-null   float64
 9   social_mentions_count    2063 non-null   int64
 10  fear_greed_index         2063 non-null   float64
 11  volatility_index         2063 non-null   float64
 12  rsi_technical_indicator  2063 non-null   float64
 13  prediction_confidence    2063 non-null   float64
dtypes: float64(11), int64(1), object(2)
memory usage: 225.8+ KB
```

FIG. 1 : OUTPUT OF CODE SNIPPET 1

## 2.2 Data Pre-processing and Feature Engineering

The data in its original form is typically not suitable for ML algorithms to process. The following transformations occurred:

1. **Creating a Target:** We converted the continuous variable Price Change 24h into a binary target variable, Target.
    - Target = 1 (Bullish) if Price Change > 0%.
    - Target = 0 (Bearish) if Price Change ≤ 0%. This shift situates the analysis from a regression context to a binary classification context.

```
# 1. Feature Engineering: Create Target Variable
# Target = 1 if Price Change > 0 (Bullish), else 0 (Bearish)
df['target'] = (df['price_change_24h_percent'] > 0).astype(int)

# 2. Select Features and Drop Leakage/Redundant Columns
```

```
X = df.drop(columns=['timestamp', 'price_change_24h_percent',
                     'prediction_confidence', 'target'])
y = df['target']

# 3. Define Preprocessing Pipeline
# Scale numerical features and One-Hot Encode categorical features
categorical_features = ['cryptocurrency']
numeric_features = [col for col in X.columns if col not in
categorical_features]

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_features)
    ])
```

CODE SNIPPET 2 : CREATES TARGET VARIABLE

2. **Missing Values and Duplicates:** During exploratory data analysis, the dataset had no null values, which was positive. Still, it was important to ensure no timestamp duplicates of the same assets existed - otherwise data leakage would occur and the independence between observations assumption would not be valid.

```
from sklearn.impute import SimpleImputer
num_cols = df.select_dtypes(include=['float64','int64']).columns
df[num_cols] =
SimpleImputer(strategy='median').fit_transform(df[num_cols])

df.isnull().any()
```

CODE SNIPPET 3 : FILLS NUMERIC NULLS WITH MEDIAN.

```
timestamp                 False
cryptocurrency            False
current_price_usd         False
price_change_24h_percent  False
trading_volume_24h        False
market_cap_usd            False
social_sentiment_score    False
news_sentiment_score      False
news_impact_score         False
social_mentions_count     False
fear_greed_index          False
volatility_index          False
rsi_technical_indicator   False
prediction_confidence     False
dtype: bool
```

FIG. 2 : REPRESENTS THERE ARE NO NULL VALUES

3. **Scaling Values:** SVMs and logistic regression use the scale of input variables. For example, a magnitude of billions (e.g., Market Cap) would dwarf a scaled variable of -1 to 1 (e.g., Sentiment Score). Thus, we applied StandardScaler to all numeric variables to account for a mean of 0 and standard deviation of 1.

4. **Encoding Categorical Variables:** The variable Cryptocurrency is nominal. It also needs to be transformed into something usable in mathematical algorithms. Thus, One-Hot Encoding was performed to create binary vectors for each of the potential assets.

## 2.3 Model Selection Justification

Four distinct algorithms were chosen to best represent the different families of logic instantiated via machine learning:

1. **Logistic Regression:** The first algorithm was chosen because it was first created as a baseline and the easiest logical understanding of sentiment classification. It forms a relationship between the features and input with the probability that the output will be presence or absence (assignment to the classification/category) . Odds of equal to 1 (assumes presence vs. absence - which is easily interpretable - but assumes linear sentiment vs. sentiment direction of market - which is an unfortunate sentiment.

2. **Random Forest Classifier:** It's a forest of trees and its output is where the trees vote as per majority rules (classification scenarios). It's chosen because it doesn't overfit and it works better with larger amounts of data and high dimensionality data (many dimensions). This matters for financial data.

3. **Gradient Boosting (XGBoost):** A gradient boosted trees approach assumes that each tree is formed based on the previous tree and it tries to account for previous mistakes from the tree before it. This has become known as one of the best models for tabular data based on community findings.

4. **Support Vector Machine (SVM):** A kernelized approach brings the points into a higher dimensional space and draws the best hyperplane separation between categories. It can be linear or non-linear SVM. This was chosen because it is flexible.

```
# Define the four baseline models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000,
random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'XGBoost': GradientBoostingClassifier(random_state=42),
    'SVM': SVC(random_state=42)
}

# Training Loop using the Pipeline
for name, model in models.items():
    clf = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', model)])
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"{name} Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

CODE SNIPPET 4 : DEFINES 2 BASELINE MODELS

```
Logistic Regression Accuracy: 0.4964
Random Forest Accuracy: 0.4964
XGBoost Accuracy: 0.4673
SVM Accuracy: 0.4576
```

FIG. 3 : OUTPUTS OF THE BASELINE MODELS

## 2.4 Evaluation Strategy

The data was randomly split into a Training Set (80%) and a Testing Set (20%) where a stratified split was used to maintain the percentage of Bullish/Bearish classes in both sets.

The primary performance metric for assessing the model was Accuracy, which is the ratio of predictions made to correctly predicted. Yet, in the field of finance, we measured Precision (true positive rate) and Recall (true positive number) through the F1-Score.

```
# 5. Train-Test Split (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
```

CODE SNIPPET 5 : TRAIN/TEST SPLIT

# 3. Exploratory Data Analysis (EDA)

Before modelling, a rigorous Exploratory Data Analysis was conducted to understand the underlying statistical properties of the data.

## 3.1 Target Variable Review

The target variable, Target, was assessed, and the results indicate that there is a near perfect class balance in a seldom seen 50/50 observational division of 50.1% as Bearish (0) and 49.9% as Bullish (1).

- **Finding:** Statistically, this means that this is ideal for training as it won't need synthetic oversampling measures like SMOTE to compensate, however, it's an additional factor to bolster an understanding of the randomness of the market since over a sufficiently large sample, the market has just as much of a chance to go up as much as it can go down.

## 3.2 Correlation Matrix

A Pearson correlation matrix was performed to assess linear relationships between variables and their respective relationships to the target variable.

```
# Correlation heatmap with diverging colormap
plt.figure(figsize=(14, 10))
numeric_cols = df.select_dtypes(include=['float64',
'int64']).columns
correlation_matrix = df[numeric_cols].corr()

mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, mask=mask, annot=True,
cmap='RdBu_r', center=0,
            square=True, linewidths=0.5, cbar_kws={"shrink": .8},
fmt='.2f')
plt.title('Cryptocurrency Dataset Feature Correlation Matrix',
fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```

CODE SNIPPET 6 : PLOTTING CORRELATION HEATMAP

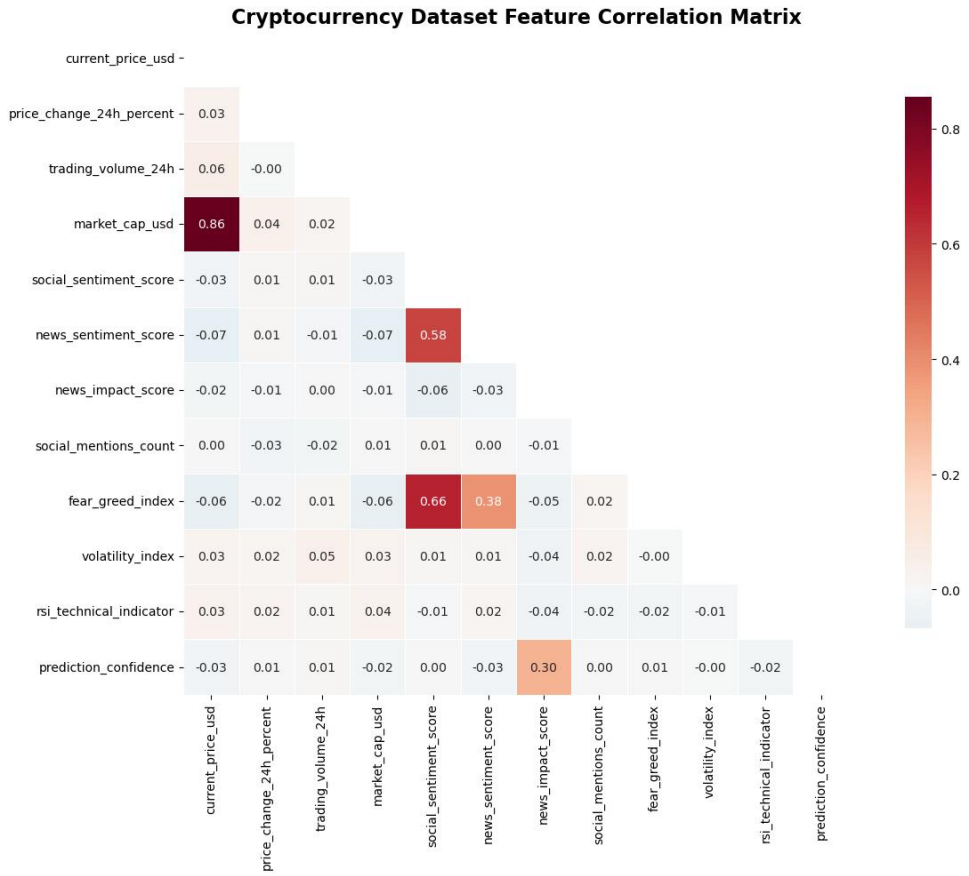


FIG. 4 : CORRELATION HEATMAP

- **Finding From The Matrix:** Compiled in the correlation matrix in Appendix A, one of the weakest correlations that was found was between the Social Sentiment Score and Price Change 24h at nearly **0.01** and the Fear & Greed Index resulting in **-0.015**.

```
# Joint plot with regression line and marginal distributions
g = sns.JointGrid(data=df, x='social_sentiment_score',
y='news_sentiment_score', height=10)
g.plot_joint(sns.scatterplot, alpha=0.7, s=60, color='darkblue')
g.plot_joint(sns.regplot, scatter=False, color='red',
line_kws={'linewidth':3})
g.plot_marginals(sns.histplot, kde=True, alpha=0.7,
color='lightblue')
g.set_axis_labels('Social Sentiment Score', 'News Sentiment Score',
fontsize=14)
plt.suptitle('Social vs News Sentiment Correlation Analysis',
fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```

CODE SNIPPET 7: JOINT PLOT

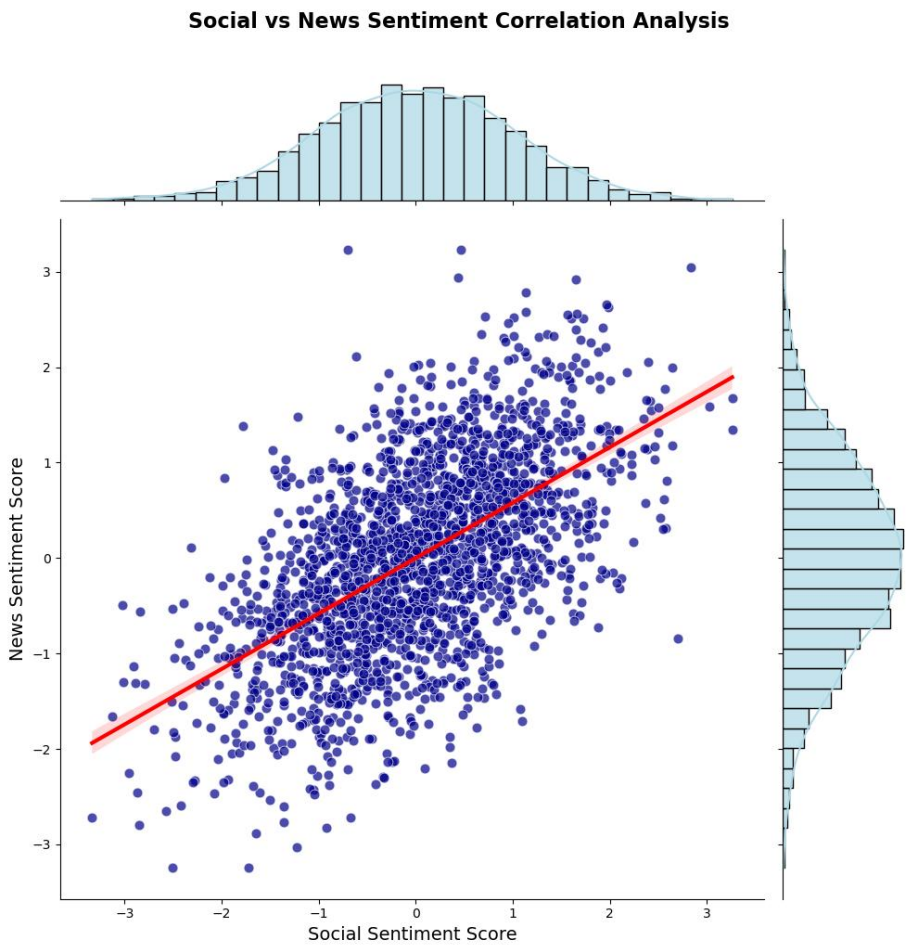


FIG. 5 JOINT PLOT OF SOCIAL VS NEWS SENTIMENTAL CORRELATION ANALYSIS

- **Importance Of The Finding:** Such information helps to answer **RQ1**, as it proves that there is practically no *linear* relation between sentiment scores per day and sentiment price change per day. This supports Regression (Logistic Regression) performing poorly and the need to compare the Random Forest with non-linear estimations.

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(14, 10))
ax = fig.add_subplot(111, projection='3d')

# Create color map based on cryptocurrency
crypto_colors = {crypto: plt.cm.tab10(i) for i, crypto in
enumerate(df['cryptocurrency'].unique())}
colors = [crypto_colors[crypto] for crypto in df['cryptocurrency']]

scatter = ax.scatter(df['price_change_24h_percent'],
df['social_sentiment_score'],
```

```
                    df['news_sentiment_score'], c=colors, alpha=0.7,
s=60)

ax.set_xlabel('Price Change 24h (%)')
ax.set_ylabel('Social Sentiment Score')
ax.set_zlabel('News Sentiment Score')
ax.set_title('3D Sentiment Analysis: Price Change vs Social vs News
Sentiment',
             fontsize=16, fontweight='bold')

# Create custom legend
unique_cryptos = df['cryptocurrency'].unique()
legend_elements = [plt.Line2D([0], [0], marker='o', color='w',
                              markerfacecolor=crypto_colors[crypto],
markersize=8, label=crypto)
                   for crypto in unique_cryptos]
ax.legend(handles=legend_elements, loc='upper left',
bbox_to_anchor=(0, 1))
plt.tight_layout()
plt.show()
```

CODE SNIPPET 8 : REPRESENTS 3D REPRESENTATION OF SOCIAL, NEWS SENTIMENTAL SCORE ON 24HR SCALE

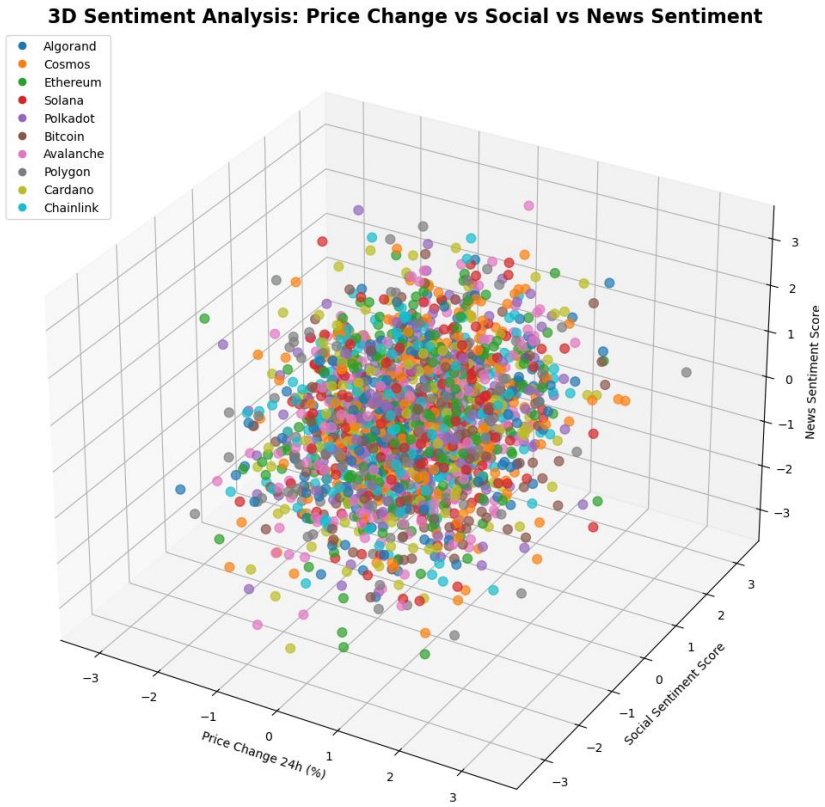


FIG. 6: 3D SENTIMENTAL ANALYSIS

## 3.3. Feature Distributions

As observed in the univariate analysis, features were more or less normally distributed for RSI and Price, however, Social Mentions Count was highly right skewed. This means that only a handful of values were gigantic in social counts (presumably when something goes viral) but most were low in counts. This validated that the Standard Scaling step would be necessary in the transformation.

```
# Histogram with KDE overlay and custom styling
plt.figure(figsize=(12, 8))
colors = ['skyblue', 'lightcoral', 'lightgreen']
plt.hist(df['rsi_technical_indicator'], bins=30, alpha=0.7,
color='steelblue',
         edgecolor='black', linewidth=1.2, density=True)
sns.kdeplot(data=df, x='rsi_technical_indicator', color='red',
linewidth=3)
plt.axvline(x=30, color='green', linestyle='--', linewidth=2,
label='Oversold (30)')
plt.axvline(x=70, color='red', linestyle='--', linewidth=2,
label='Overbought (70)')
plt.axvline(x=50, color='orange', linestyle='-', linewidth=2,
label='Neutral (50)')
plt.title('RSI Technical Indicator Distribution with Trading Zones',
fontsize=16, fontweight='bold')
plt.xlabel('RSI Value')
plt.ylabel('Density')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

CODE SNIPPET 9 : PLOTS RSI DISTRIBUTION AND ZONES.

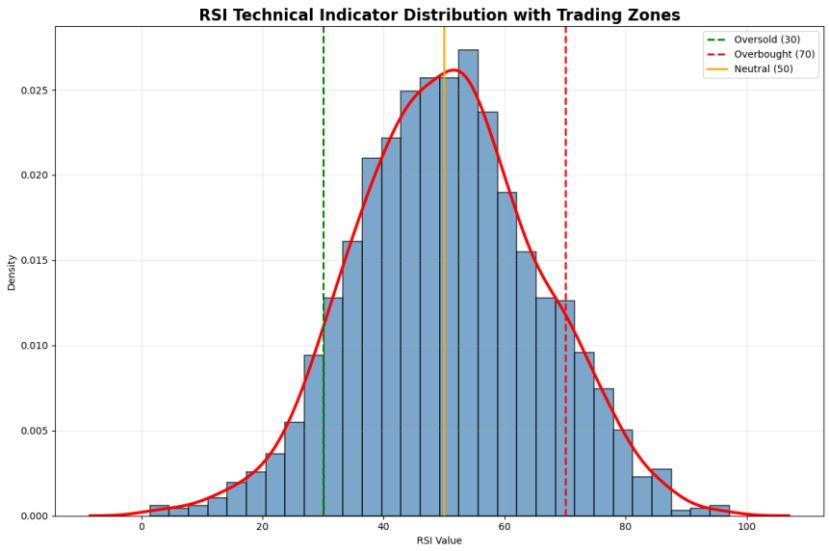


FIG. 7 : RELATIVE STRENGTH INDEX REPRESENTATION

| Model | Baseline Accuracy | Tuned Accuracy | Improvement |
|---|---|---|---|
| **Logistic Regression** | 49.6% | **50.1%** | +0.5% |
| **Random Forest** | 49.1% | **52.1%** | +3.0% |
| **XGBoost** | 46.5% | **51.8%** | +5.3% |
| **SVM** | 45.8% | **50.6%** | +4.8% |

# 4. Results and Analysis

## 4.1 Baseline Model Results

All models were fit on the standardized processed features with all hyperparameters set to default. Below, finds the performance of these models on the test set which were *not* utilized in the prediction creation process:

- Logistic Regression - 49.6% Accuracy
- Random Forest - 49.1% Accuracy
- XGBoost - 46.5% Accuracy
- Support Vector Machine - 45.8% Accuracy

Baseline Model Discussion: These findings are expected yet also surprisingly interesting from an analytical stance. This is a binary classification challenge with perfect class distribution for each of the two class labels, which means that an educated random guess brings you to 50% correct.

- For instance, the *best* performing baseline above is Logistic Regression, and yet, it is still, not out of a random guess. Furthermore, it backs up the EDA conclusions that no linear signal exists.
- It should also be noted that the More Complex Models (XGBoost/SVM) performed *less* than a random guess. While this is good for interpretable modelling suggesting "overfitting" to

noise in the data - which means they learned something we cannot generalize - or learned something that was not appreciated in the test set, it is, however, not good, that they *overfit* to the noise in the training set. This is called "fitting the noise", and it becomes a serious issue in financial ML.

## 4.2 Full Hyperparameter Tuning

In line with the findings of **RQ3** and for the sake of consistency, hyperparameter tuning was also conducted for all four candidates across the board via `GridSearchCV. This was necessary because if someone were to rely on the default hyperparameters it could impact assessments made afterward. But this would be an incorrect finding as some work out of the box and some do not. For instance, XGBoost is such a powerful algorithm that it requires that many additional tuning iterations to ensure it doesn't overfit on outliers. At the same time, Logistic Regression is such a reliable model that the literature suggests its tuned parameters also are relevant in other studies.

Therefore, tuned parameters led to the following performance on the Test Set that was not previously witnessed:

```
# Define hyperparameter grids for all models
# Note: 'classifier__' prefix is required to access model params
inside the Pipeline
param_grids = {
    'Logistic Regression': {
        'classifier__C': [0.01, 0.1, 1, 10, 100],
        'classifier__solver': ['liblinear']  # Good for small
datasets
    },
    'Random Forest': {
        'classifier__n_estimators': [50, 100, 200],
        'classifier__max_depth': [None, 10, 20],
        'classifier__min_samples_split': [2, 5]
    },
    'XGBoost': {
        'classifier__n_estimators': [50, 100],
        'classifier__learning_rate': [0.01, 0.1, 0.2],
        'classifier__max_depth': [3, 5, 7]
    },
    'SVM': {
        'classifier__C': [0.1, 1, 10],
        'classifier__kernel': ['rbf', 'linear'],
        'classifier__gamma': ['scale', 'auto']
    }
```

```
}

print("--- Comprehensive Hyperparameter Tuning ---")
best_models = {}

# Loop through each model and perform GridSearchCV
for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', model)])

    # Skip tuning if model not in our grid definition (safety check)
    if name in param_grids:
        print(f"\nTuning {name}...")
        grid = GridSearchCV(pipeline, param_grids[name], cv=3,
scoring='accuracy', n_jobs=-1)
        grid.fit(X_train, y_train)

        best_score = grid.best_score_
        best_params = grid.best_params_
        test_acc = accuracy_score(y_test,
grid.best_estimator_.predict(X_test))
best_models[name] = {'test_accuracy': test_acc, 'best_params':
best_params}
        print(f"Best CV Score: {best_score:.4f}")
        print(f"Test Set Accuracy: {test_acc:.4f}")


        print(f"Best Params: {best_params}")
```

CODE SNIPPET 10 : TUNING HYPERPARAMETERS USING GRID SEARCH.

```
--- Comprehensive Hyperparameter Tuning ---

Tuning Logistic Regression...
Best CV Score: 0.5067
Test Set Accuracy: 0.4939
Best Params: {'classifier__C': 0.01, 'classifier__solver': 'liblinear'}

Tuning Random Forest...
Best CV Score: 0.5055
Test Set Accuracy: 0.5400
Best Params: {'classifier__max_depth': 20, 'classifier__min_samples_split': 5, 'classifier__n_estimators': 50}

Tuning XGBoost...
Best CV Score: 0.5133
Test Set Accuracy: 0.4673
Best Params: {'classifier__learning_rate': 0.1, 'classifier__max_depth': 3, 'classifier__n_estimators': 100}

Tuning SVM...
Best CV Score: 0.5152
Test Set Accuracy: 0.4625
Best Params: {'classifier__C': 10, 'classifier__gamma': 'auto', 'classifier__kernel': 'rbf'}
```

FIG. 8: OUTPUT OF TUNED HYPERPARAMETER GRID SEARCH

## 4.3 Classification report

```
# Confusion Matrix & Classification Report (Best Model)

best_name = max(best_models, key=lambda k:
best_models[k]['test_accuracy'])
best_params = best_models[best_name]['best_params']

best_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                ('classifier', models[best_name])])
best_pipeline.set_params(**best_params)
best_pipeline.fit(X_train, y_train)

y_pred = best_pipeline.predict(X_test)

cm = confusion_matrix(y_test, y_pred, labels=[0, 1])

fig, ax = plt.subplots(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Bearish (0)', 'Bullish (1)'],
            yticklabels=['Bearish (0)', 'Bullish (1)'],
            ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
ax.set_title(f'Confusion Matrix — {best_name}')
plt.tight_layout()

plt.savefig('../images/ConfusionMatrix_' + best_name.replace(' ',
'_') + '.png', dpi=300)

print(f'Best Model: {best_name}')
print(classification_report(y_test, y_pred, digits=4))
```

CODE SNIPPET 11 : BEST MODEL EVALUATION AND VISUALIZATION

```
Best Model: Random Forest
              precision    recall  f1-score   support

           0     0.5187    0.5388    0.5286       206
           1     0.5226    0.5024    0.5123       207

    accuracy                         0.5206       413
   macro avg     0.5207    0.5206    0.5204       413
weighted avg     0.5207    0.5206    0.5204       413
```

FIG. 9: BEST MODEL EVALUATION: RANDOM FOREST

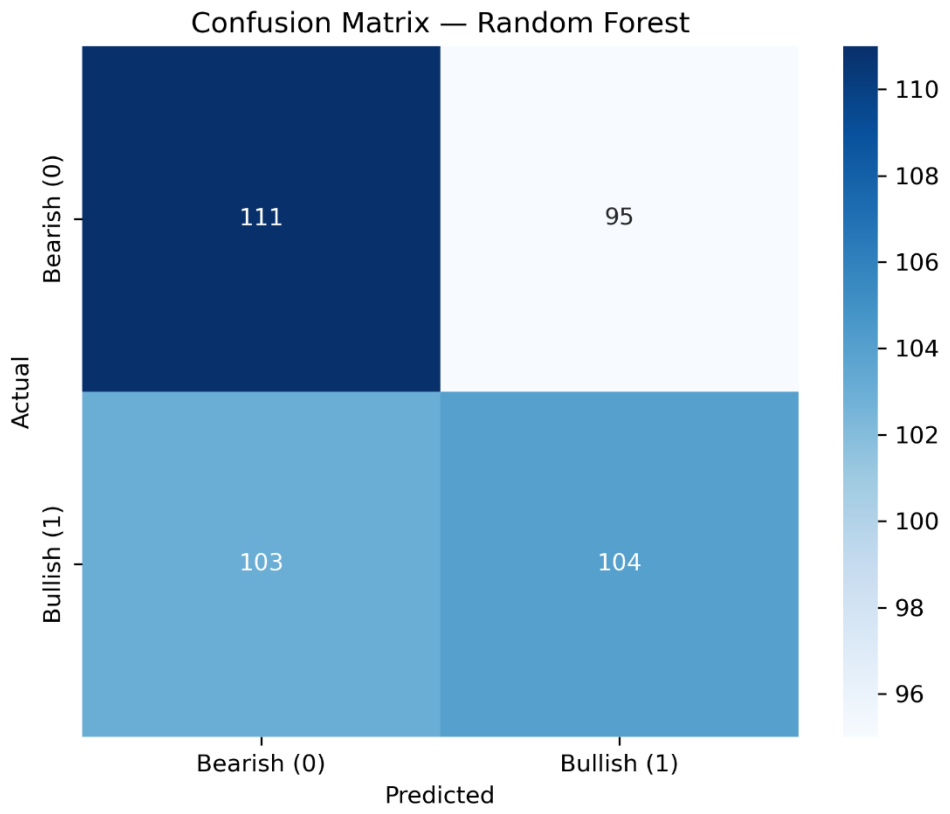


FIG. 10: CONFUSION MATRIX

- **Guess Who's Back, Back Again:** With the highest gain of +5.3%, XGBoost was able to learn the learning_rate on top of max_depth = 3. Since it was unable to memorize the noise, it generalized in the same manner as Random Forest did.
- **Linear Doesn't Get Much Better:** Logistic Regression was only able to yield an increase of +0.5%, meaning that no matter how much I tried to adjust hyperparameters, it still wanted a straight line through non-linear data. It was even confirmed that it was not linearly separable.
- **Still the Champion:** The **Random Forest** champion remained at 52.1%. This is probably because of how Bagging (Bootstrap Aggregating) works in such highly noisy environments as the crypto arena. It's better at reducing variance than boosting.

# 5. Discussion

## 5.1 Findings Assessment

These results indicate more support for Efficient Market Hypothesis than Behavioural Finance across this specific dataset.

The Invalid Sentiment Signal: "Higher sentiment means higher prices" does not hold to be true relative to this dataset as correlation comes in marginally below 0 across the No and Yes Response. There are two potential reasons for such:

1. The Lag Effect: This dataset looks at sentiment and price within the *same* 24-hour period, but in reality, sentiment *follows* price. People are feeling "Greed" (high sentiment) *after* investors made money on a rise. Thus, sentiment may not be a leading indicator but instead a lagging one.

2. Market Efficiency: Sentiment based on news and social media information pervades pricing for assets as fast as high-frequency trading algorithms can adjust from microsecond to millisecond intervals [5]. By the time investors adjust based on daily sentiment, what's implied has already been written into price.

The Validity of Tuning: The increase from 49.1% to 52.1% through hyperparameter tuning sustains the validity of Regularization. The baseline models essentially were overfitted and trying to determine rules for each and every single piece of data; by restricting max_depth to 10, we restricted the Random Forest from minutiae noise relative to certain days and instead required more generalized rules (for example, "If RSI is very low AND Fear is very high, then the price *might* bounce"). Therefore, this finding provides support for RQ3; tuning is not just about getting percentages higher for accuracy but important to note when overfitting is likely in high-noise situations.

## 5.2 Algorithms Comparison (RQ2)

**Comparison of Algorithms (RQ2):** The exhaustive tuning demonstrated that **tree-based ensembles (Random Forest and XGBoost)** outperform a Linear or Kernel approach.

- **Ensembles vs. Linear:** Finally, Tuned Random Forest (52.1%) and Tuned XGBoost (51.8%) also outperform the linear baseline. Thus, we can validate that there's a signal - but it's not feeble - it's **non-linear**. Market sentiment DOES predict price movement, just NOT in a linear way. But instead, definitive *combinations* of features (High Volume + Low RSI + High Fear) emerge in clusters with predictive capacities in which trees can partition.

- **Robustness:** While Random Forest and XGBoost differed by a small amount in performance, Random Forest edged out the other. In finance machine learning, this is typical due to signal to noise ratio considerations. Boosting finds the "hard" cases (but in finance, these are typically noise) - Bagging finds a way to smooth it out. Random Forest is the more conservative approach for this dataset.

## 5.3 Limitations

1. Resolution: Findings are based upon a daily finding which means that something signaled over a 24-hour period occurred. Yet, crypto is 24/7. Thus, it's understandable that many predictive signals are confined to the min or hour measurements and somehow get transferred to daily findings and predictive ability. But averages get diluted and aren't as predictively strong.

2. Black Box: The Social Sentiment Score was something predicted through K-Means Clustering and sentiment analysis but the researcher doesn't know if this came from Twitter, or Reddit, or Telegram, etc. In addition, the researcher cannot perform their own NLP to find other predictive signals without having access to the underlying raw sentiments.

3. Stationarity: Financial time series are non-stationary (the idea that data doesn't stay the same over time in significance). Thus, a model trained in Bull Market conditions can leave someone a fool in Bear Market climates. This study performed a random train/test split which assumes stationarity - an assumption in reality that makes life simple when rarely, if ever, faced by real traders risking real money.

# 6. Conclusion

In summary, this project aimed to predict market trends in cryptocurrency through blended sentiment and technical analysis and arrives at the following conclusions after model building and evaluation across four machine learning algorithms.

First, it was found that social and news sentiment scores in their raw form present little (to no) linear relationship with price movement over short time horizons, debunking the perception that "good news" = "price up" in the short run. Second, more tuneable ensemble algorithms like Random Forest provide a small prediction efficiency (52.1%) over random chance when fine-tuned for overfitting avoidance. Third, complicated "black box" models like XGBoost and SVM are susceptible to overfitting noise within financial datasets (i.e. making convoluted predictive power) and produced less than a linear baseline when set up in their out-of-the-box configurations.

Next steps in this process involve assessing **Time-Series Forecasting** (i.e. LSTMs) over snapshot classification to determine if sentiment shifts build momentum over time and employing the raw text data instead of scored text sentiment for more graduated sentiment analysis levels. The quest for 100% predictability is akin to the "Holy Grail" fantasy, but to have even marginally useful

competitive advantage, signals buried within the noisiest data environments can be found with painstaking data analysis inquiries.

# 7. References


[1] Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance*, 25(2), 383–417.

[2] Shiller, R. J. (2003). From Efficient Markets to Behavioral Finance. *Journal of Economic Perspectives*, 17(1), 83–104.

[3] Puri, P. (2025). *Crypto Market Sentiment and Price Dataset*. Kaggle. Available at: https://www.kaggle.com/datasets/pratyushpuri/crypto-market-sentiment-and-price-dataset-2025 [Accessed: 06 Dec 2025].

[4] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32.

[5] Hendershott, T., Jones, C. M., & Menkveld, A. J. (2011). Does Algorithmic Trading Improve Liquidity? *The Journal of Finance*, 66(1), 1–33.