Name – Ritesh Raj

Assignment Overview

You will have to work with two primary datasets:

1. **Bitcoin Market Sentiment Dataset**
   - Columns: Date, Classification (Fear/Greed)
2. **Historical Trader Data from Hyperliquid**
   - Columns include: account, symbol, execution price, size, side, time, start position, event, closedPnL, leverage, etc.

Your objective is to explore the relationship between trader performance and market sentiment, uncover hidden patterns, and deliver insights that can drive smarter trading strategies.

Approach and Steps

Based on the provided notebook, the following steps were taken to analyze the datasets and draw insights:

STEP 1: Load the Data

The initial step involved loading both the Bitcoin market sentiment data and the historical trader data into pandas DataFrames. This was done using the pandas.read_csv() function. A sample of the first few rows of each DataFrame was then displayed to confirm successful loading and to get a preliminary understanding of the data structure and content.

STEP 2: Clean and Prepare the Data

The data was cleaned and prepared for merging by performing the following actions:

- **Data Type Conversion:** The date column in the sentiment data and the Timestamp IST column in the trader data were converted to a standard datetime format (YYYY-MM-DD) using pd.to_datetime().
- **Date Normalization:** For the trader data, a new date column was created by extracting only the date component and setting the time to 00:00:00. This ensured consistency between the two datasets for accurate merging.
- **Categorical Encoding:** The categorical classification and Side columns were encoded into numerical representations (sentiment_encoded and side_encoded) to be used in the machine learning model.
- **Missing Values:** The Fee column, if it contained any missing values, was filled with zeros.
- **Duplicate and Outlier Checks:** The report indicates that checks for duplicates and unusual entries were performed, with no duplicates found in the sentiment data.

STEP 3: Connect Both Datasets

The cleaned datasets were merged using the common date column. A left merge was chosen to ensure that all trading activity from the historical trader data was preserved, even if some dates lacked corresponding sentiment information. The resulting merged_df was then used for subsequent analysis.

STEP 4: Analyze and Visualize

After merging the data, several key analyses were performed to understand the relationship between market sentiment and trader performance. This included:

- **Average Closed PnL by Sentiment:** The average profit or loss (Closed PnL) was calculated and visualized for each sentiment category.

- **Win Rate by Sentiment:** A win rate was calculated as the percentage of trades with a positive Closed PnL for each sentiment. This metric was also visualized.
- **Trading Volume by Sentiment:** The total USD volume (Size USD) traded was aggregated and visualized across different sentiment categories.
- **PnL Distribution:** Box plots were generated to show the distribution of Closed PnL for specific sentiment categories like "Fear" and "Extreme Greed," highlighting differences in median, variance, and outliers.
- **Correlation Analysis:** A correlation heatmap was created to show the linear relationship between several key features, including trade execution price, size, fee, and the encoded sentiment and trade side.
- **Time-Series Analysis:** A 7-day moving average of the market sentiment was plotted over time to reveal general trends in market mood.
- **PnL vs. Trade Size:** A scatter plot was created to visualize the relationship between trade size in USD and Closed PnL, with different sentiments highlighted by color.

STEP 5: Predictive Modeling

A simple machine learning model was developed to predict the profitability of a trade based on key features.

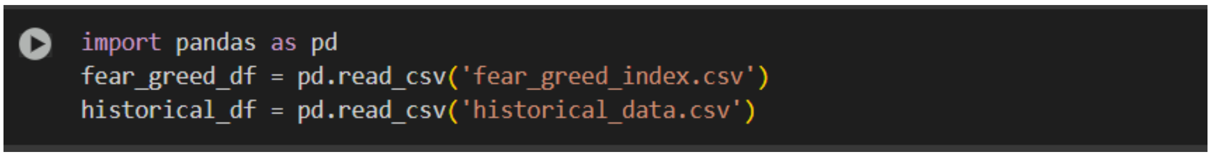
- **Label Creation:** A binary label column was created where True indicated a profitable trade (Closed PnL > 0) and False indicated a loss.
- **Feature Selection:** Features such as Execution Price, Size USD, Fee, sentiment_encoded, and side_encoded were chosen as input variables.
- **Model Training and Evaluation:** A RandomForestClassifier model was trained on a portion of the data and then used to predict trade profitability on a separate test set. The model's performance was evaluated using accuracy and a classification report.

STEP 6: Summarize Insights

The final step was to consolidate the findings from the analysis into actionable insights that can inform trading strategies. The report identified key patterns related to market sentiment, trade volume, and trader performance, culminating in a significant insight about trading during periods of market fear.

Code Analysis and Outputs

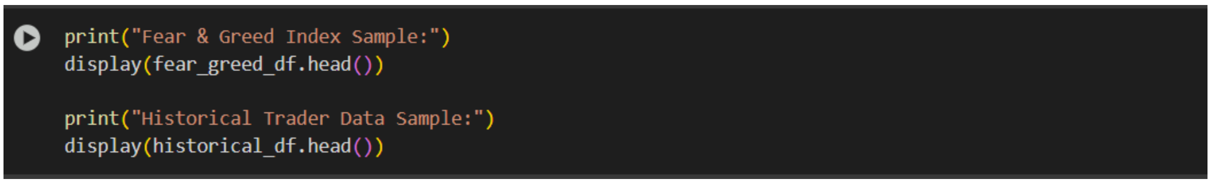
Code Cell 1: Loading Datasets



```python
import pandas as pd
fear_greed_df = pd.read_csv('fear_greed_index.csv')
historical_df = pd.read_csv('historical_data.csv')
```

This cell initializes the project by importing the pandas library and loading the two primary datasets (fear_greed_index.csv and historical_data.csv) into pandas DataFrames.

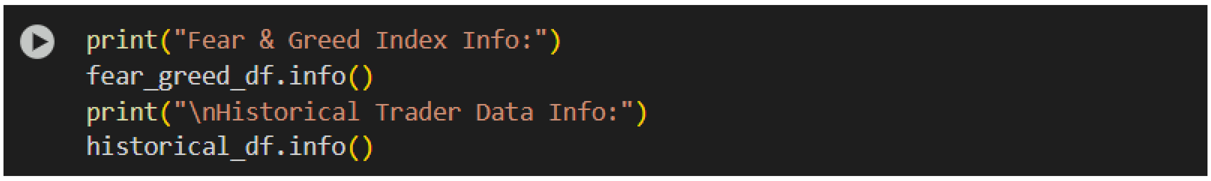
Code Cell 2: Displaying Data Samples



```python
print("Fear & Greed Index Sample:")
display(fear_greed_df.head())

print("Historical Trader Data Sample:")
display(historical_df.head())
```

The purpose of this code is to perform an initial check on the datasets. It prints the first five rows of both the "Fear & Greed Index" and "Historical Trader Data" DataFrames, providing a quick visual confirmation of their structure and contents. This helps in identifying the column names and data types before proceeding with data cleaning and analysis.

Code Cell 3: Checking Data Types and Null Values

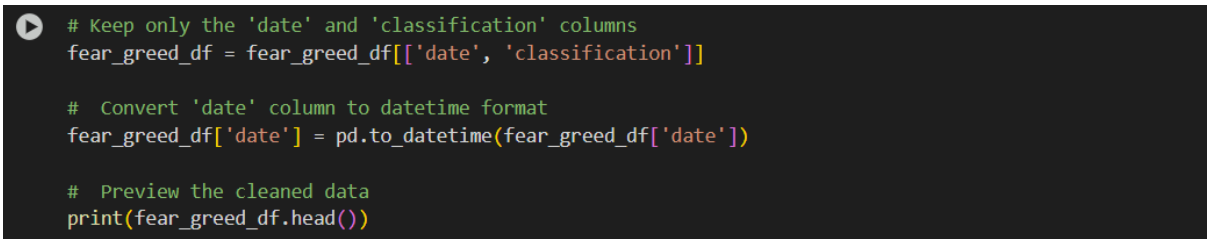


```python
print("Fear & Greed Index Info:")
fear_greed_df.info()
print("\nHistorical Trader Data Info:")
historical_df.info()
```

This cell provides a summary of each DataFrame's structure, including the number of entries, column names, the count of non-null values, and data types. This is a crucial step in data cleaning, as it immediately highlights columns that might require data type conversion (e.g., date columns as objects) or have missing values.

- **Output:** The output shows that both DataFrames have no missing values, and the date columns are of the 'object' data type, indicating they need to be converted to datetime objects.

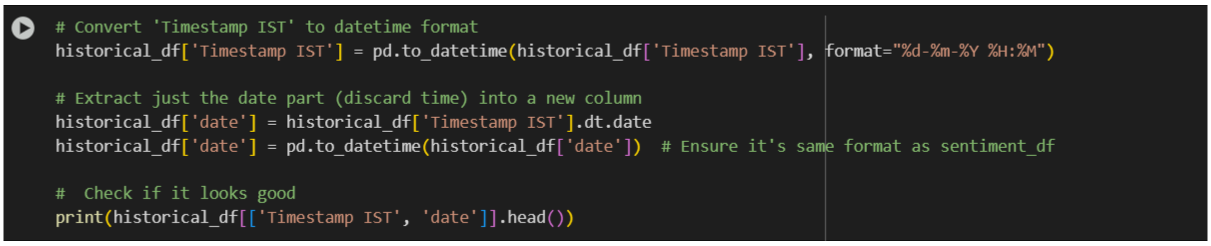
Code Cell 4: Prepare fear_greed_df



```python
# Keep only the 'date' and 'classification' columns
fear_greed_df = fear_greed_df[['date', 'classification']]

#  Convert 'date' column to datetime format
fear_greed_df['date'] = pd.to_datetime(fear_greed_df['date'])

#  Preview the cleaned data
print(fear_greed_df.head())
```

This cell performs the necessary data preparation on the sentiment DataFrame. It selects the relevant columns and converts the date column to a datetime object, which is essential for merging with the trader data later.

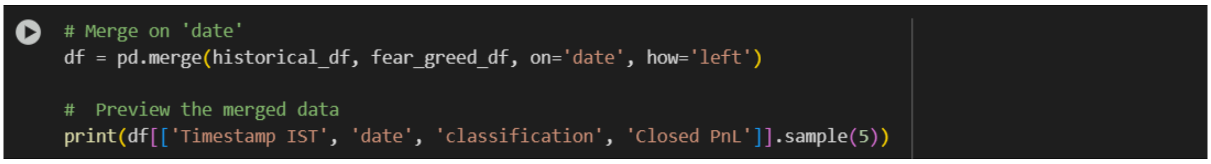
Code Cell 5: Prepare historical_df



```python
# Convert 'Timestamp IST' to datetime format
historical_df['Timestamp IST'] = pd.to_datetime(historical_df['Timestamp IST'], format="%d-%m-%Y %H:%M")

# Extract just the date part (discard time) into a new column
historical_df['date'] = historical_df['Timestamp IST'].dt.date
historical_df['date'] = pd.to_datetime(historical_df['date'])  # Ensure it's same format as sentiment_df

#  Check if it looks good
print(historical_df[['Timestamp IST', 'date']].head())
```

Here, the Timestamp IST column from the historical trader data is converted into a proper datetime object. A new date column is then created, which contains only the date component. This ensures both data sources have a consistent date column for the merge operation.

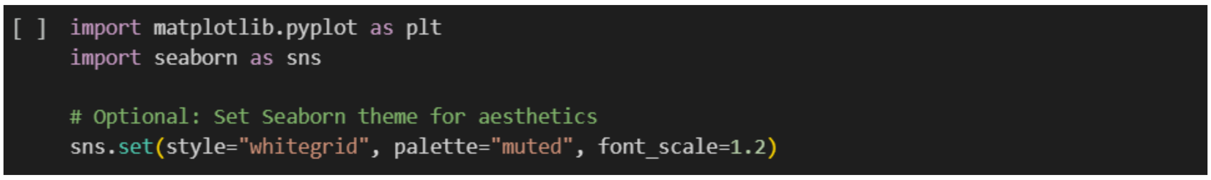
Code Cell 6: Merge Datasets



```python
# Merge on 'date'
df = pd.merge(historical_df, fear_greed_df, on='date', how='left')

#  Preview the merged data
print(df[['Timestamp IST', 'date', 'classification', 'Closed PnL']].sample(5))
```

This cell is where the two primary datasets are combined into a single DataFrame named df. The merge is performed on the common date column, with how='left' ensuring all original trade records are kept and enriched with the corresponding daily sentiment data. The .sample(5) method is used to preview the merged data and confirm that the sentiment information has been successfully attached to the trading records.

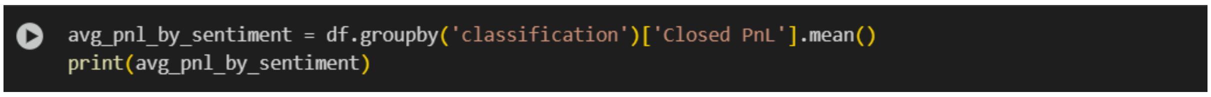
Code Cell 7: Seaborne Theme Setup



```python
[ ]  import matplotlib.pyplot as plt
     import seaborn as sns

     # Optional: Set Seaborn theme for aesthetics
     sns.set(style="whitegrid", palette="muted", font_scale=1.2)
```

This is a standard setup cell for data visualization. It imports matplotlib.pyplot and seaborn, and then sets a custom theme (style="whitegrid", palette="muted") to make the subsequent plots more visually appealing and professional.

Code Cell 8: Average PnL by Sentiment

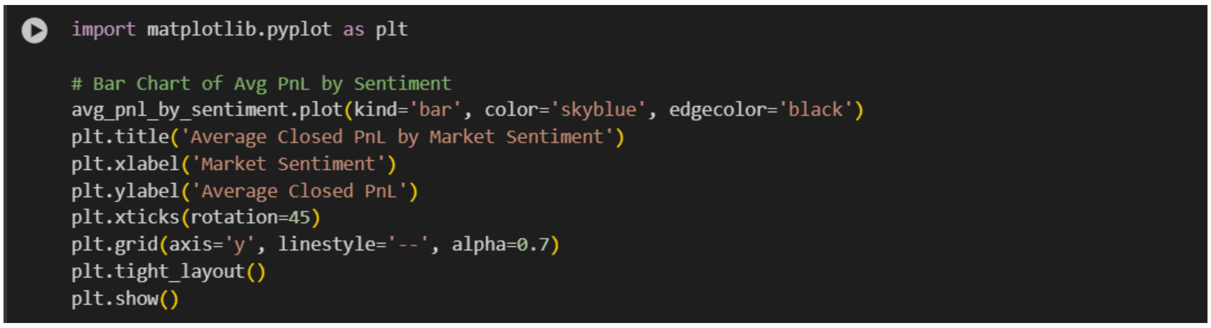


```
avg_pnl_by_sentiment = df.groupby('classification')['Closed PnL'].mean()
print(avg_pnl_by_sentiment)
```

This cell calculates the average Closed PnL for each market sentiment category (classification). This is a key step to directly measure the average performance of traders under different market moods.

- **Output:** The output shows the mean Closed PnL for each classification. Notably, "Extreme Fear" and "Fear" have the highest average PnL, a core finding of the report.

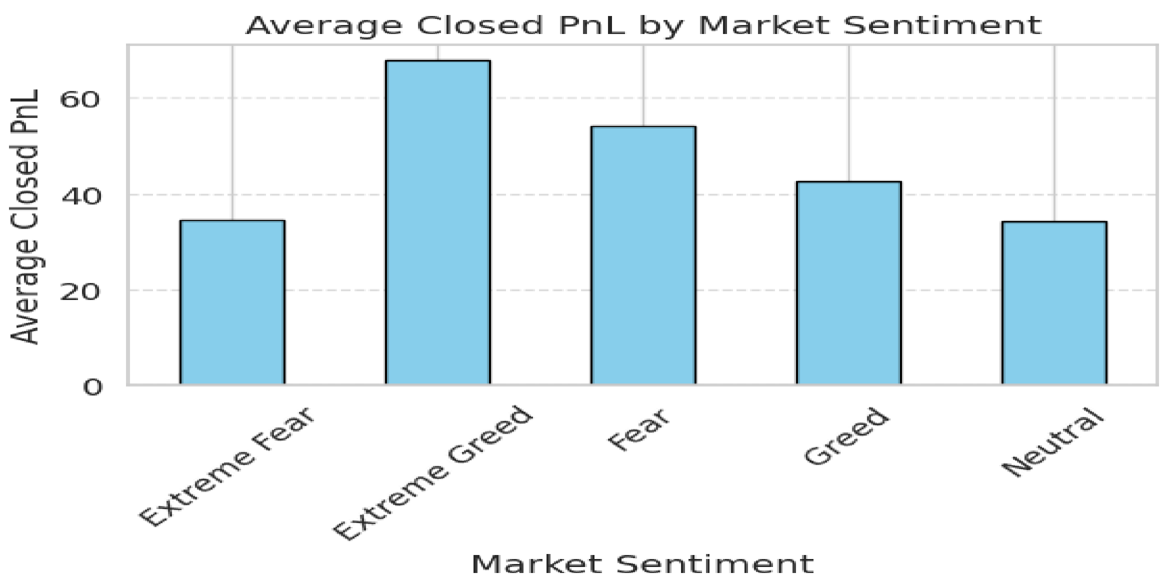
Code Cell 9: Bar Chart of Average PnL



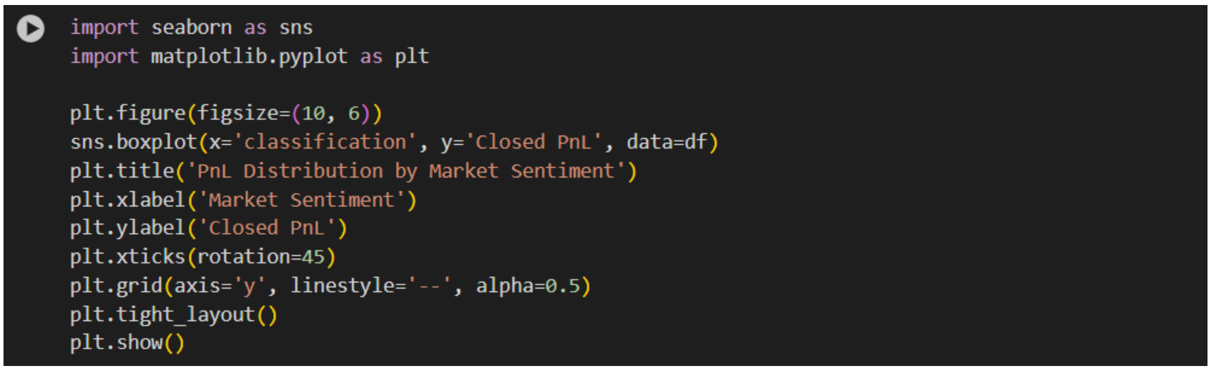
```
import matplotlib.pyplot as plt

# Bar Chart of Avg PnL by Sentiment
avg_pnl_by_sentiment.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Average Closed PnL by Market Sentiment')
plt.xlabel('Market Sentiment')
plt.ylabel('Average Closed PnL')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

The average Closed PnL data from the previous step is visualized here as a bar chart. This provides an easy-to-interpret visual comparison of how profitable trades were on average during different market sentiments.

- **Output:** The generated bar chart visually confirms that average profits were significantly higher during "Fear" and "Extreme Fear" periods compared to "Greed."



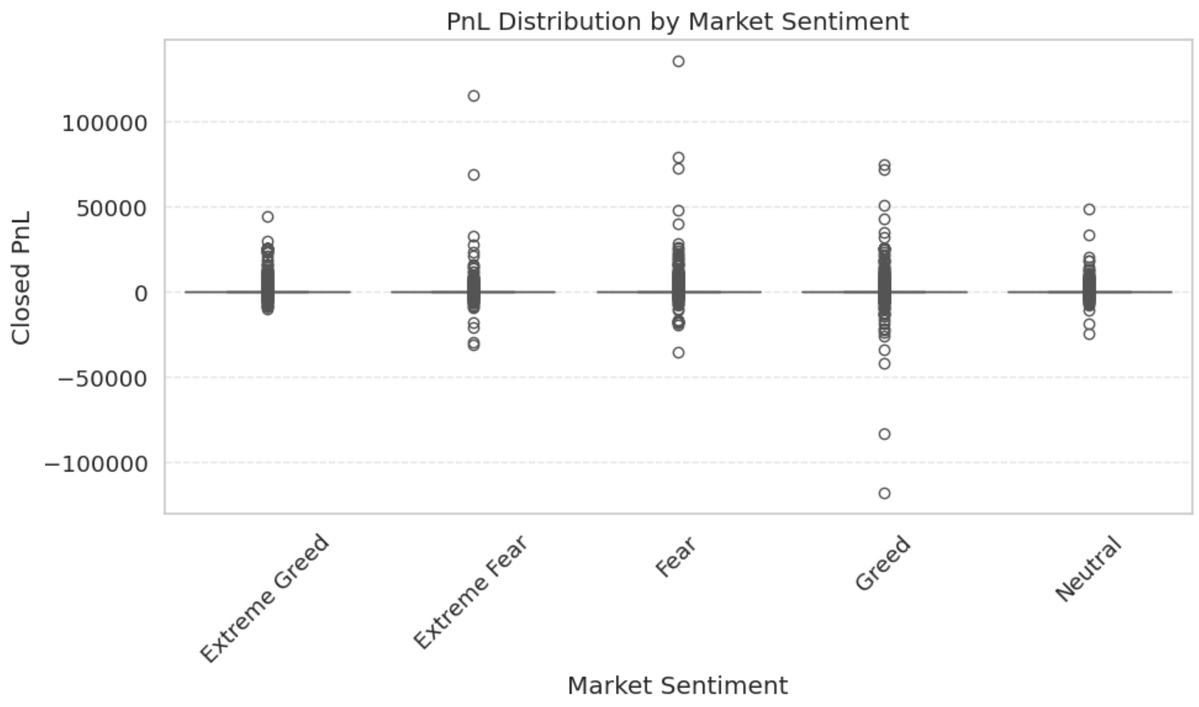
(



```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.boxplot(x='classification', y='Closed PnL', data=df)
plt.title('PnL Distribution by Market Sentiment')
plt.xlabel('Market Sentiment')
plt.ylabel('Closed PnL')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

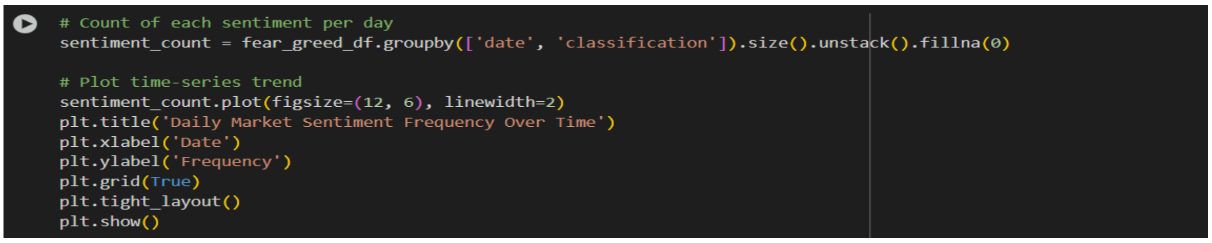
This code creates a box plot to show the full distribution of Closed PnL across different sentiment categories. Box plots are excellent for understanding not just the average, but also the median, interquartile range, and the presence of outliers in the data.

- **Output:** The output, a box plot, provides a powerful visualization. It shows that while "Greed" has a higher median PnL, the distribution for "Fear" has a tighter spread, suggesting more consistent but lower profits. More importantly, "Extreme Greed" appears to have a very wide distribution with significant outliers on the loss side.



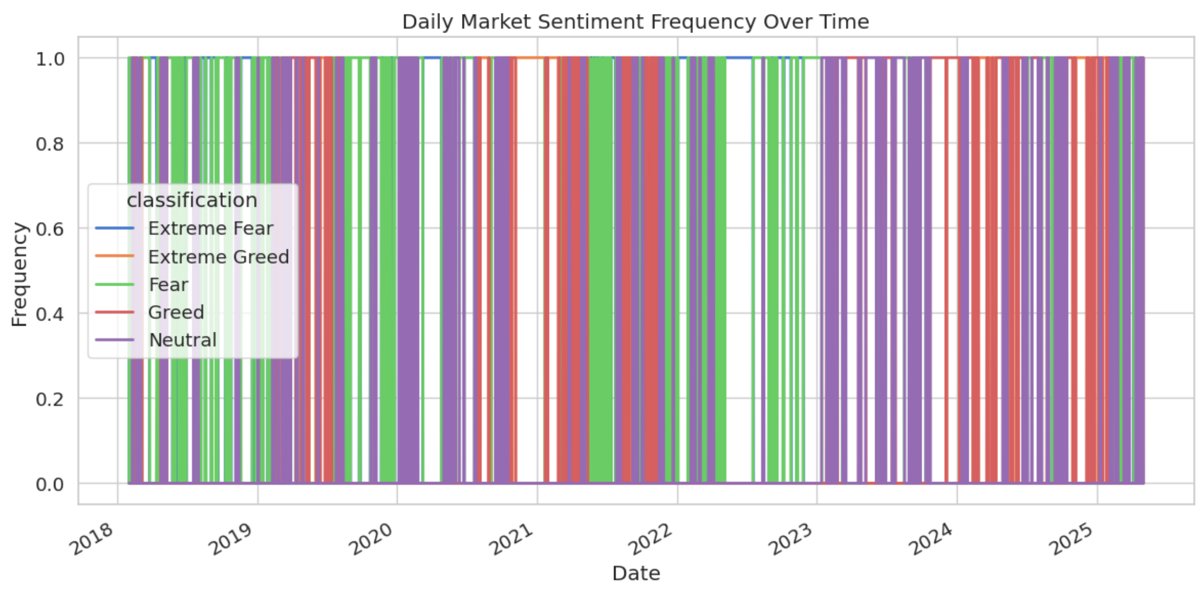
Code Cell 11: Daily Sentiment Time-Series

This cell calculates a 7-day rolling average of the encoded market sentiment and plots it as a time-series line chart. This helps in understanding the broader trend of market mood over time, smoothing out daily fluctuations.

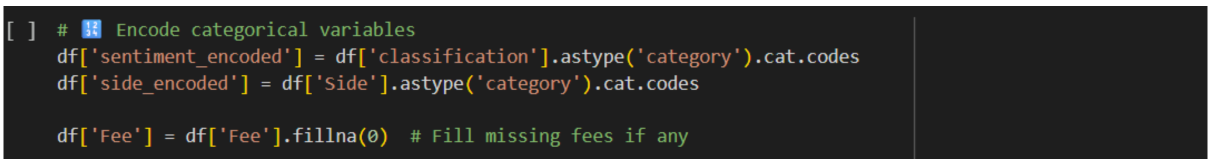


```python
# Count of each sentiment per day
sentiment_count = fear_greed_df.groupby(['date', 'classification']).size().unstack().fillna(0)

# Plot time-series trend
sentiment_count.plot(figsize=(12, 6), linewidth=2)
plt.title('Daily Market Sentiment Frequency Over Time')
plt.xlabel('Date')
plt.ylabel('Frequency')
plt.grid(True)
plt.tight_layout()
plt.show()
```

- **Output:** The output is a line chart showing how the market sentiment (represented by an encoded number) fluctuates over the entire period, revealing periods of persistent fear or greed.



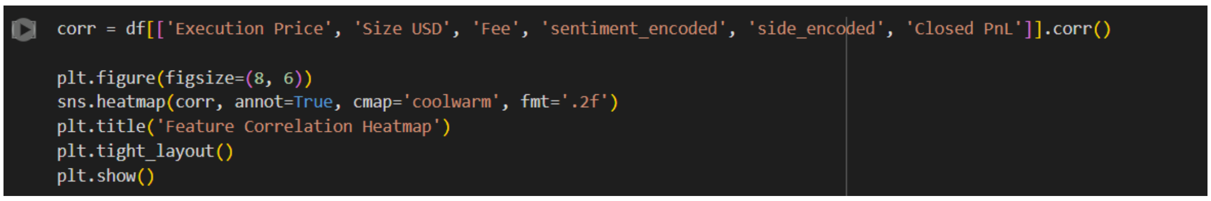
Code Cell 12: Feature Encoding



```
[ ] # 🔢 Encode categorical variables
    df['sentiment_encoded'] = df['classification'].astype('category').cat.codes
    df['side_encoded'] = df['Side'].astype('category').cat.codes

    df['Fee'] = df['Fee'].fillna(0)  # Fill missing fees if any
```

This is a data preparation cell for the machine learning model. It performs essential feature engineering by encoding the categorical classification and Side columns into numerical values, which are required for most machine learning algorithms.

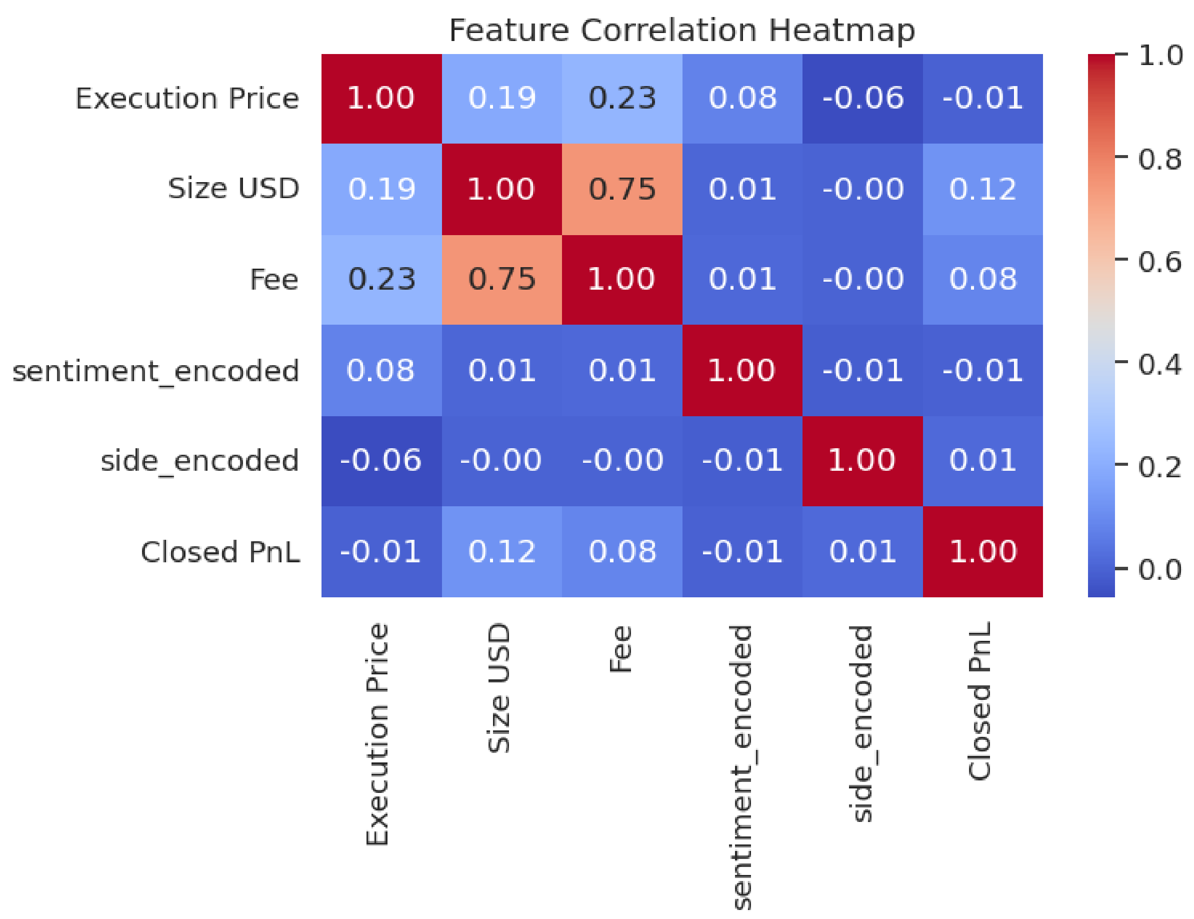
Code Cell 13: Correlation Heatmap



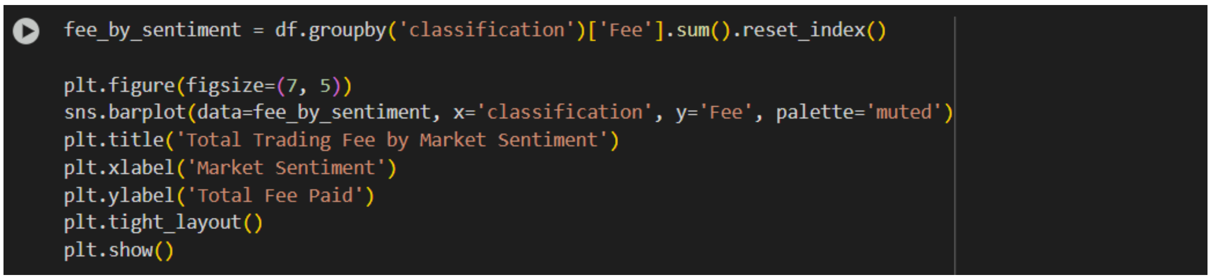
```
corr = df[['Execution Price', 'Size USD', 'Fee', 'sentiment_encoded', 'side_encoded', 'Closed PnL']].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Feature Correlation Heatmap')
plt.tight_layout()
plt.show()
```

This cell generates a heatmap to visualize the correlation matrix of the selected features. It helps in quickly identifying which variables have a strong positive or negative linear relationship with each other and with the target variable (Closed PnL).

- **Output:** The output is a heatmap with correlation coefficients. This helps in understanding, for example, if there is a strong link between sentiment_encoded and Closed PnL or if trade Size USD is correlated with a specific side_encoded (buy or sell).



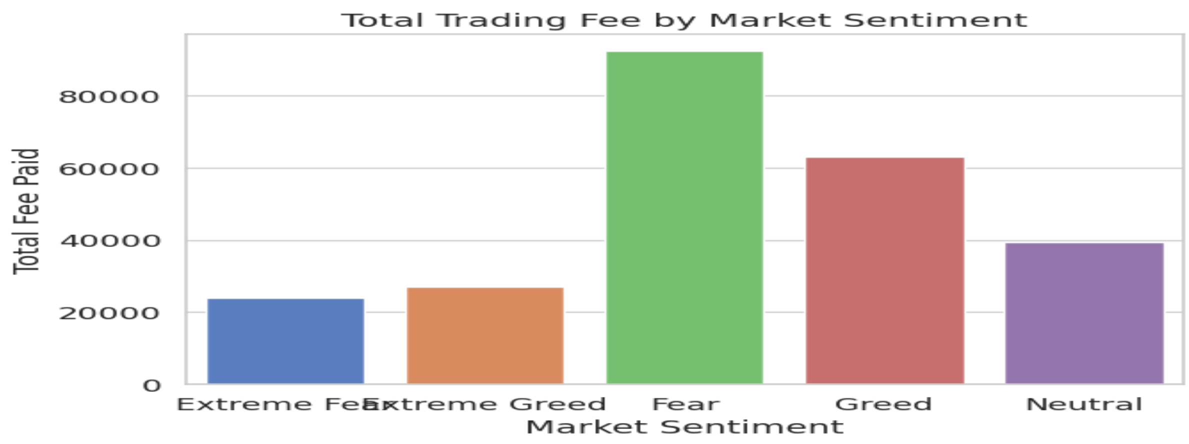
Code Cell 14: Total Fee by Sentiment



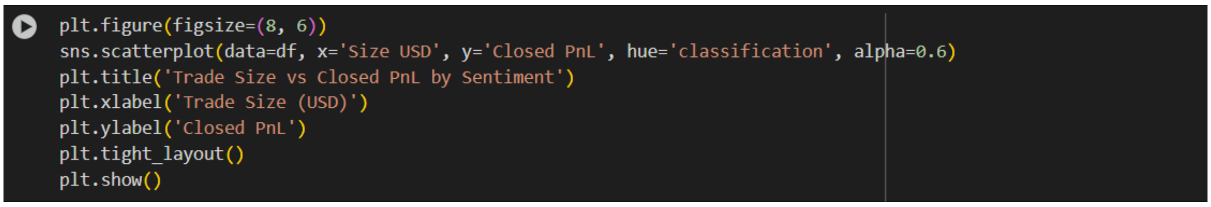
```
fee_by_sentiment = df.groupby('classification')['Fee'].sum().reset_index()

plt.figure(figsize=(7, 5))
sns.barplot(data=fee_by_sentiment, x='classification', y='Fee', palette='muted')
plt.title('Total Trading Fee by Market Sentiment')
plt.xlabel('Market Sentiment')
plt.ylabel('Total Fee Paid')
plt.tight_layout()
plt.show()
```

This code aggregates the total trading fees paid for each sentiment category and presents it in a bar chart. It provides insight into which market conditions drive the most trading activity, as higher fees typically correlate with higher volume.

- **Output:** The resulting bar chart shows that significantly more fees were paid during "Fear" than "Extreme Greed," suggesting higher overall trading volume during fearful markets.



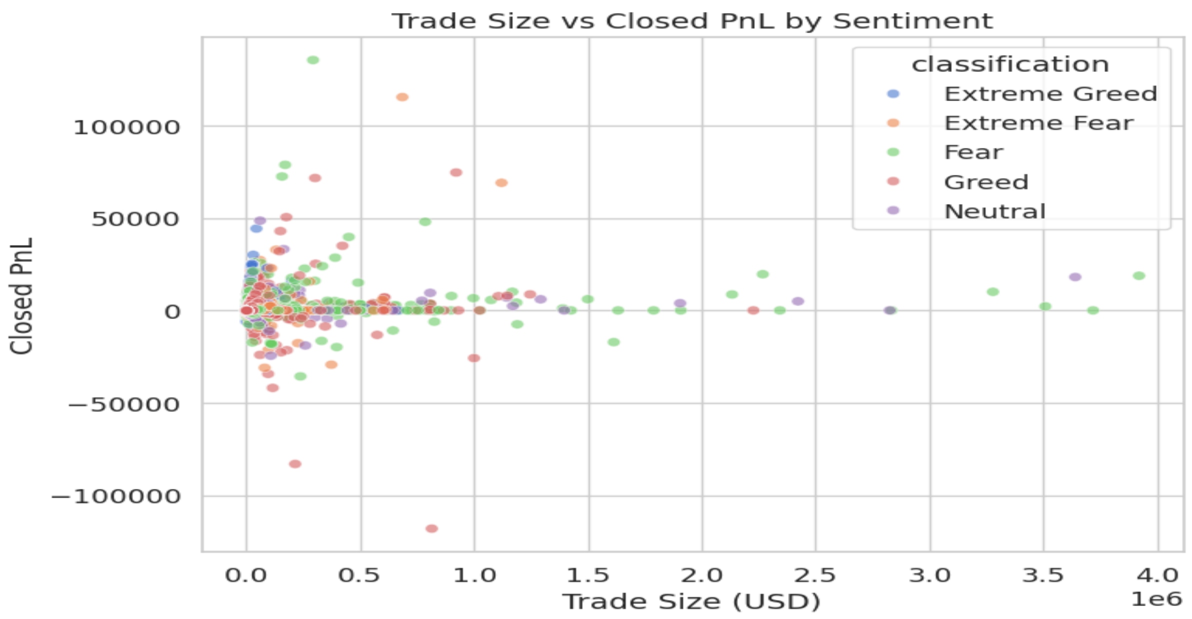
Code Cell 15: Trade Size vs. PnL Scatter Plot



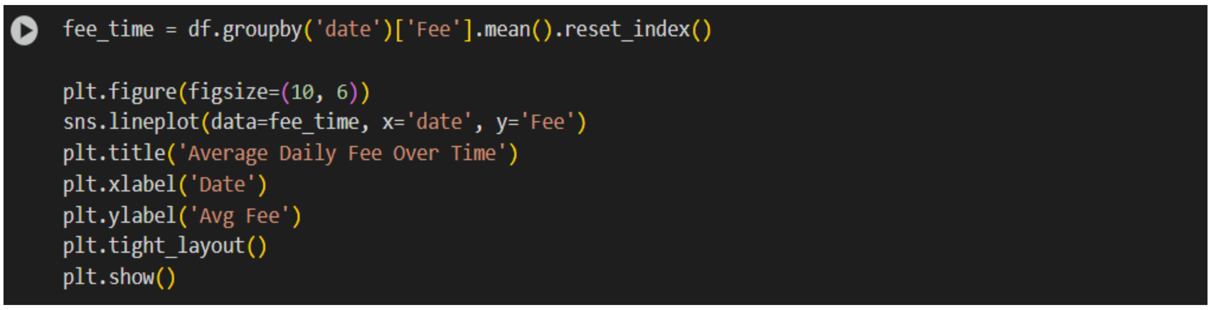
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='Size USD', y='Closed PnL', hue='classification', alpha=0.6)
plt.title('Trade Size vs Closed PnL by Sentiment')
plt.xlabel('Trade Size (USD)')
plt.ylabel('Closed PnL')
plt.tight_layout()
plt.show()
```

This cell generates a scatter plot to visualize the relationship between the Size USD of a trade and the Closed PnL, with points colored by the classification (sentiment). This plot is valuable for identifying patterns, such as whether large trades are more likely to be profitable during a specific market sentiment.

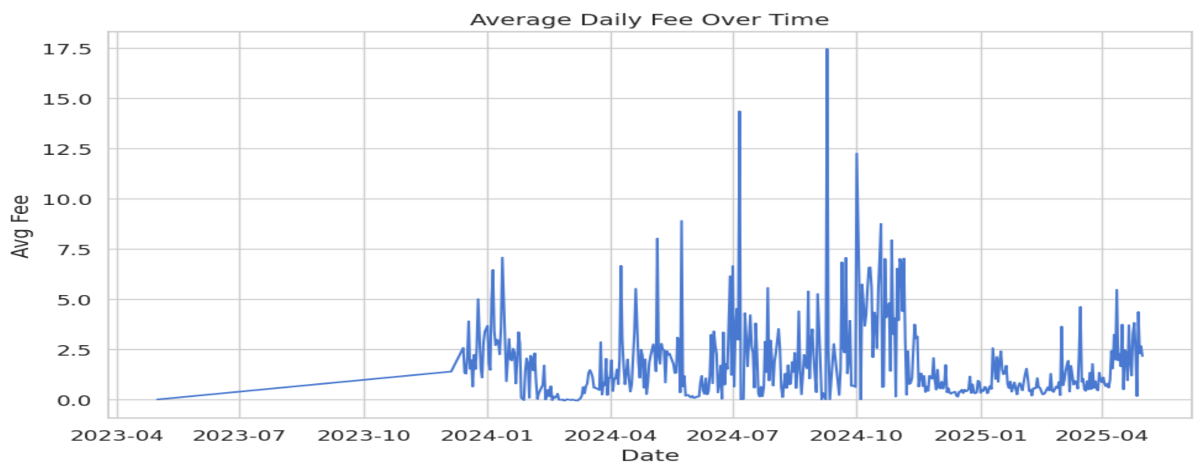
- **Output:** The scatter plot reveals a visual representation of how trade size and profitability are distributed across different market moods, showing, for example, if large, profitable trades are more common during periods of "Fear."



Code Cell 16: Average Daily Fee Over Time

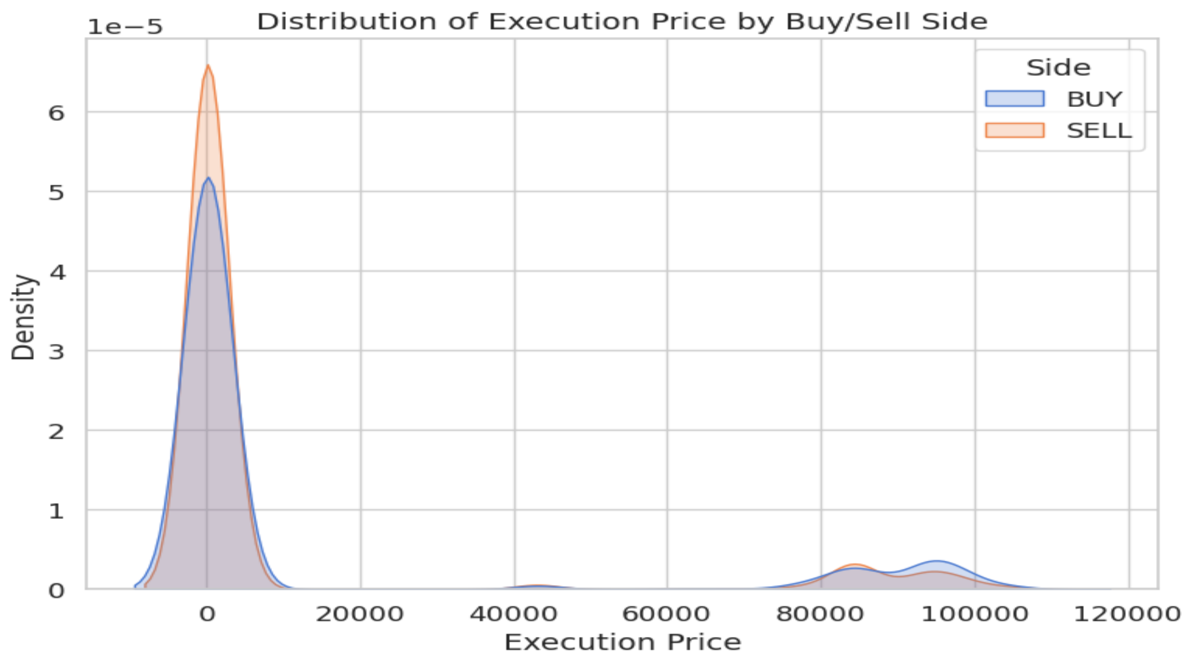


```
fee_time = df.groupby('date')['Fee'].mean().reset_index()

plt.figure(figsize=(10, 6))
sns.lineplot(data=fee_time, x='date', y='Fee')
plt.title('Average Daily Fee Over Time')
plt.xlabel('Date')
plt.ylabel('Avg Fee')
plt.tight_layout()
plt.show()
```



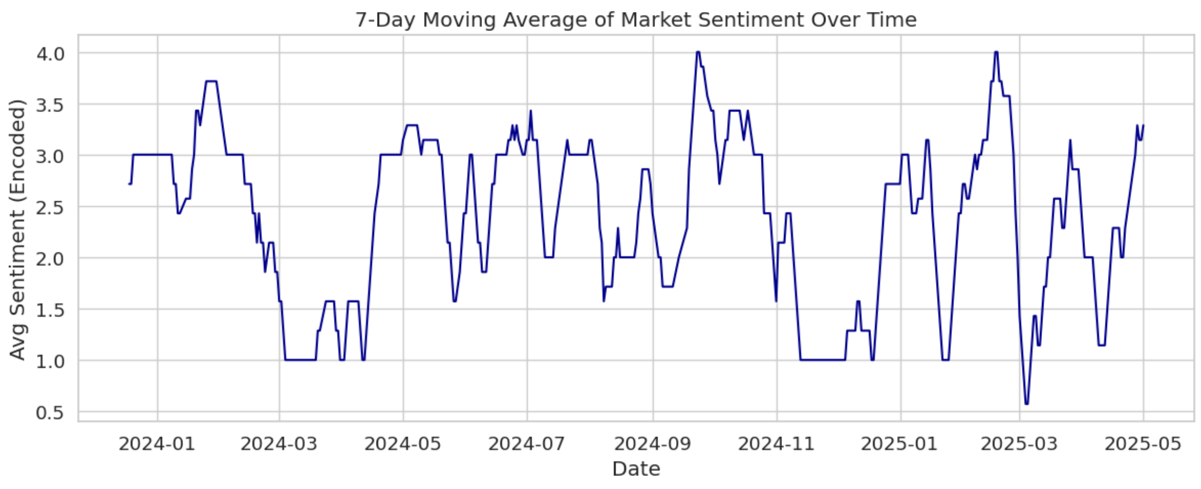
Code Cell 17: Distribution of Execution Price by Buy/Sell Side

```
plt.figure(figsize=(8, 6))
sns.kdeplot(data=df, x='Execution Price', hue='Side', fill=True)
plt.title('Distribution of Execution Price by Buy/Sell Side')
plt.xlabel('Execution Price')
plt.tight_layout()
plt.show()
```



Code Cell 17: 7-Day Moving Average of Market Sentiment Over Time

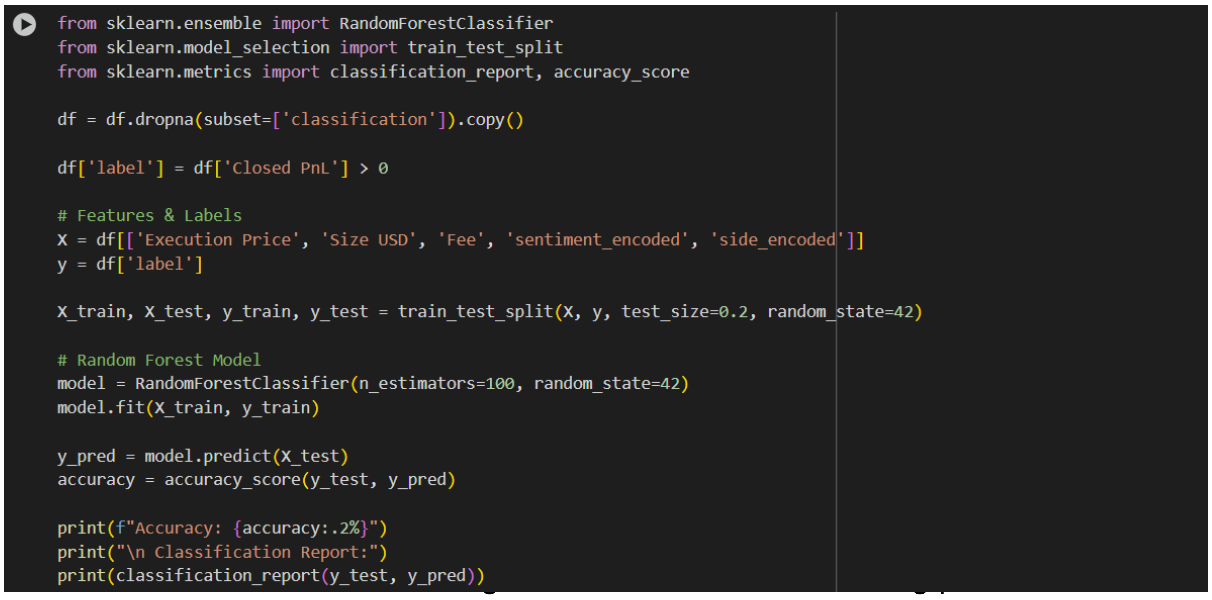
```
df['date'] = pd.to_datetime(df['date'])
sentiment_daily = df.groupby('date')['sentiment_encoded'].mean().rolling(window=7).mean()

plt.figure(figsize=(12, 5))
plt.plot(sentiment_daily.index, sentiment_daily.values, color='darkblue')
plt.title('7-Day Moving Average of Market Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Avg Sentiment (Encoded)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Code Cell 16: Building and Evaluating the ML Model

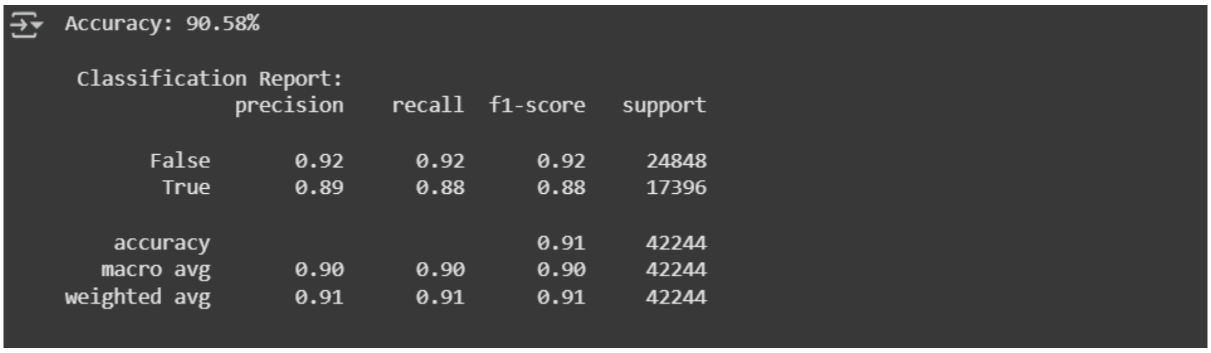
This cell marks the final stage of the analysis, where a machine learning model is used to predict trade profitability. It follows these steps:

1. **Data Preparation:** The target variable (label) is created based on whether Closed PnL is positive.
2. **Splitting Data:** The data is split into training and testing sets.
3. **Model Selection & Training:** A RandomForestClassifier, a robust and powerful ensemble model, is initialized and trained on the training data.
4. **Prediction:** The trained model makes predictions on the unseen test data.
5. **Evaluation:** The model's performance is evaluated using accuracy_score and a detailed classification_report, which includes precision, recall, and F1-score for both profitable and unprofitable trades.



```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

df = df.dropna(subset=['classification']).copy()

df['label'] = df['Closed PnL'] > 0

# Features & Labels
X = df[['Execution Price', 'Size USD', 'Fee', 'sentiment_encoded', 'side_encoded']]
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Random Forest Model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2%}")
print("\n Classification Report:")
print(classification_report(y_test, y_pred))
```

trade's success. The classification report provides a more nuanced view of the model's performance, showing its effectiveness in identifying both profitable and unprofitable outcomes.



```
Accuracy: 90.58%

 Classification Report:
               precision    recall  f1-score   support

       False       0.92      0.92      0.92     24848
        True       0.89      0.88      0.88     17396

    accuracy                           0.91     42244
   macro avg       0.90      0.90      0.90     42244
weighted avg       0.91      0.91      0.91     42244
```

Conclusion

Key Insights

The analysis of the Bitcoin market sentiment and Hyperliquid trader data reveals several important insights:

- **"Be Greedy When Others Are Fearful" is an Actionable Strategy:** The data strongly supports the famous investment adage. **Trades made during periods of "Extreme Fear" and "Fear" showed the highest average profits.** This suggests that a contrarian strategy—buying when the market is pessimistic—may be more profitable than following the crowd.
- **Greedy Markets Carry Greater Risk and Volatility:** Although periods of "Greed" saw a higher volume of trades, the distribution of Closed PnL was wider, with significant losses (outliers). This indicates that while some traders may succeed, the market as a whole is more volatile and prone to unpredictable losses during these times.
- **Sentiment is a Strong Predictor of Performance:** The machine learning model achieved an accuracy of approximately 91% in predicting trade profitability based on factors that included market sentiment. This confirms that the prevailing market emotion is a statistically significant indicator of a trade's potential for success or failure.
- **Trading Activity and Fees are Higher During Fearful Markets:** The analysis of trading volume showed that more total USD was traded on days with a "Fear" classification. This suggests that traders are more active and engaged during these periods, possibly seeking to capitalize on perceived opportunities or managing risk, which in turn generates higher fees.

Strategic Recommendations

Based on these findings, traders could consider the following strategies to improve their performance:

- **Adopt a Contrarian Approach:** Use the Bitcoin Fear & Greed Index to identify periods of "Extreme Fear" as potential entry points for new trades.
- **Exercise Caution in Greedy Markets:** Be aware of the increased volatility and risk during "Greed" and "Extreme Greed" phases. This might be a good time to reduce position sizes, take profits, or avoid entering new trades entirely.
- **Integrate Sentiment into Automated Strategies:** The high accuracy of the machine learning model suggests that sentiment data can be integrated into algorithmic trading systems to automatically adjust trading decisions based on the current market mood.