**APPENDIX**

1. Data Loading

Bitcoin price data

import yfinance as yf

# Define the Bitcoin ticker and date range

btc\_ticker = "BTC-USD"

start\_date = "2016-01-01"

end\_date = "2023-12-31"

btc\_df = yf.download(btc\_ticker, start=start\_date, end=end\_date)

Bitcoin Tweets

from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

twitter\_df = pd.read\_csv('/content/drive/MyDrive/bitcoin\_tweets.csv')

1. Data Preprocessing - bitcoin price data

print(btc\_data.isnull().sum())

btc\_df = btc\_df.fillna(method='ffill')

def detect\_outliers(data, column):

Q1 = data[column].quantile(0.25)

Q3 = data[column].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

return ((data[column] < lower\_bound) | (data[column] > upper\_bound))

btc\_df['price\_outlier'] = detect\_outliers(btc\_df, 'Close')

btc\_df['Returns'] = btc\_data['Close'].pct\_change()

btc\_df['MA7'] = btc\_data['Close'].rolling(window=7).mean()

btc\_df['MA30'] = btc\_data['Close'].rolling(window=30).mean()

btc\_df['Volatility7'] = btc\_data['Returns'].rolling(window=7).std()

btc\_df['Volatility30'] = btc\_data['Returns'].rolling(window=30).std()

from statsmodels.tsa.stattools import adfuller

def adf\_test(series):

result = adfuller(series.dropna())

print('ADF Statistic: %f' % result[0])

print('p-value: %f' % result[1])

print('Critical Values:')

for key, value in result[4].items():

print('\t%s: %.3f' % (key, value))

return result

result = adf\_test(btc\_df['Close'])

if result[1] > 0.05:

btc\_df['Diff\_close'] = btc\_df['Close'].diff()

print("Performing ADF test on differenced series:")

new\_result = adf\_test(btc\_df['Diff\_close'])

1. Bitcoin Price Series and First Difference

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

data = yf.download('BTC-USD', start='2016-01-01', end='2023-12-31')

data['First Difference'] = data['Close'].diff()

fig, axs = plt.subplots(2, 1, figsize=(12, 8))

axs[0].plot(data['Close'], label='Bitcoin Price')

axs[0].set\_title('Bitcoin Prices')

axs[0].set\_ylabel('Price in USD')

axs[0].legend()

axs[1].plot(data['First Difference'], label='First Difference', color='orange')

axs[1].set\_title('First Difference of Bitcoin Prices')

axs[1].set\_xlabel('Date')

axs[1].set\_ylabel('Difference in USD')

axs[1].legend()

plt.tight\_layout()

plt.show()

1. Data Processing - Bitcoin Tweets

import pandas as pd

print(twitter\_df['date'].unique())

twitter\_data['date'] = pd.to\_datetime(twitter\_df['date'], format='%d-%m-%Y %H:%M', errors='coerce')

print(twitter\_df[twitter\_df['date'].isna()])

twitter\_df = twitter\_df[~twitter\_data['date'].isna()]

twitter\_df = twitter\_df.sort\_values('date')

import re

def clean\_tweet(tweet):

tweet = tweet.lower()

tweet = re.sub(r'http\S+|www\S+|https\S+', '', tweet, flags=re.MULTILINE)

tweet = re.sub(r'\@\w+|\#', '', tweet)

tweet = re.sub(r'[^\w\s]', '', tweet)

tweet = re.sub(r'\s+', ' ', tweet).strip()

return tweet

twitter\_df['cleaned\_tweet'] = twitter\_df['text'].apply(clean\_tweet)

import nltk

try:

nltk.data.find('tokenizers/punkt')

except LookupError:

nltk.download('punkt')

from nltk.tokenize import word\_tokenize

twitter\_df['tokens'] = twitter\_df['cleaned\_tweet'].apply(word\_tokenize)

import nltk

try:

nltk.data.find('corpora/stopwords')

except LookupError:

nltk.download('stopwords')

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('english'))

def remove\_stopwords(tokens):

return [word for word in tokens if word not in stop\_words]

twitter\_df['tokens\_without\_stopwords'] = twitter\_df['tokens'].apply(remove\_stopwords)

1. Sentiment analysis

!pip install vaderSentiment

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()

def get\_sentiment\_scores(tweet):

return sid.polarity\_scores(tweet)

twitter\_data['sentiment\_scores'] = twitter\_data['cleaned\_tweet'].apply(get\_sentiment\_scores)

twitter\_data['compound\_score'] = twitter\_data['sentiment\_scores'].apply(lambda x: x['compound'])

daily\_sentiment = twitter\_data.groupby(twitter\_data['date'].dt.date).agg({

'compound\_score': 'mean',

'text': 'count'

}).rename(columns={'text': 'tweet\_count'})

1. ARIMAX model

Model identification

!pip install pmdarima

from pmdarima import auto\_arima

y = btc\_df['Close']

exog = btc\_df['MA7', 'MA30', 'Volatility7', 'Volatility30']]

model = auto\_arima(y, exogenous=exog, start\_p=0, start\_q=0, max\_p=5, max\_q=5, m=1,

seasonal=False, d=None, trace=True, error\_action='ignore',

suppress\_warnings=True, stepwise=True)

print(model.order)

Model Fitting

import pandas as pd

import numpy as np

exog = btc\_df[['MA7', 'MA30', 'Volatility7', 'Volatility30']]

print("Missing values in each column:\n", exog.isnull().sum())

print("Infinite values in each column:\n", np.isinf(exog).sum())

exog = exog.fillna(exog.median())

exog.replace([np.inf, -np.inf], np.nan, inplace=True)

exog = exog.fillna(exog.median())

from statsmodels.tsa.statespace.sarimax import SARIMAX

order = model.order

arimax\_model = SARIMAX(y, exog=exog, order=order)

arimax\_results = arimax\_model.fit()

print(arimax\_results.summary())

Diagnostic Checking

import matplotlib.pyplot as plt

arimax\_results.plot\_diagnostics(figsize=(15, 12))

plt.savefig('arimax\_diagnostics new.png')

plt.show()

from statsmodels.stats.diagnostic import acorr\_ljungbox

lb\_test = acorr\_ljungbox(arimax\_results.resid, lags=[10, 20, 30])

print("Ljung-Box Test Results:", lb\_test)

Forecast

import matplotlib.pyplot as plt

train\_size = int(len(y) \* 0.8)

train, test = y[:train\_size], y[train\_size:]

exog\_train, exog\_test = exog[:train\_size], exog[train\_size:]

arimax\_model\_train = SARIMAX(train, exog=exog\_train, order=order)

arimax\_results\_train = arimax\_model\_train.fit()

forecast = arimax\_results\_train.get\_forecast(steps=len(test), exog=exog\_test)

forecast\_mean = forecast.predicted\_mean

plt.figure(figsize=(12, 6))

plt.plot(train.index, train, label='Training Data', color='blue')

plt.plot(test.index, test, label='Test Data', color='green')

plt.plot(test.index, forecast\_mean, label='Forecast', color='red')

plt.fill\_between(test.index,

forecast.conf\_int().iloc[:, 0],

forecast.conf\_int().iloc[:, 1], color='pink', alpha=0.3)

plt.title('ARIMAX Model Forecast vs Actuals')

plt.xlabel('Date')

plt.ylabel('Values')

plt.legend()

plt.show()

1. Hybrid model

Data Alignment

merged\_data = pd.merge(btc\_data, daily\_sentiment, left\_index=True, right\_index=True, how='left')

merged\_data['compound\_score'] = merged\_data['compound\_score'].fillna(method='ffill')

merged\_data['tweet\_count'] = merged\_data['tweet\_count'].fillna(0)

Lagged Sentiment features

for lag in [1, 2, 3, 7]:

merged\_data[f'sentiment\_lag\_{lag}'] = merged\_data['compound\_score'].shift(lag)

merged\_data[f'tweet\_count\_lag\_{lag}'] = merged\_data['tweet\_count'].shift(lag)

Model Refit

import numpy as np

import pandas as pd

from statsmodels.tsa.statespace.sarimax import SARIMAX

y\_hybrid = merged\_data['Close']

exog\_hybrid = merged\_data[['MA7', 'MA30', 'Volatility7', 'Volatility30', 'compound\_score', 'tweet\_count', 'sentiment\_lag\_1', 'sentiment\_lag\_2', 'sentiment\_lag\_3', 'sentiment\_lag\_7', 'tweet\_count\_lag\_1', 'tweet\_count\_lag\_2', 'tweet\_count\_lag\_3', 'tweet\_count\_lag\_7']]

exog\_hybrid.replace([np.inf, -np.inf], np.nan, inplace=True)

for column in exog\_hybrid.columns:

exog\_hybrid[column].fillna(exog\_hybrid[column].median(), inplace=True)

assert not exog\_hybrid.isnull().any().any(), "NaNs remain in the dataset"

assert not np.isinf(exog\_hybrid.values).any(), "Infs remain in the dataset"

train\_size = int(len(y\_hybrid) \* 0.8)

train\_hybrid, test\_hybrid = y\_hybrid.iloc[:train\_size], y\_hybrid.iloc[train\_size:]

exog\_train\_hybrid, exog\_test\_hybrid = exog\_hybrid.iloc[:train\_size], exog\_hybrid.iloc[train\_size:]

hybrid\_model = SARIMAX(train\_hybrid, exog=exog\_train\_hybrid, order=order)

hybrid\_results = hybrid\_model.fit()

print(hybrid\_results.summary())

Forecast

hybrid\_forecast = hybrid\_results.get\_forecast(steps=len(test\_hybrid), exog=exog\_test\_hybrid)

hybrid\_forecast\_mean = hybrid\_forecast.predicted\_mean

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

plt.plot(test\_hybrid, label='Actual', color='blue') # Actual data

plt.plot(hybrid\_forecast\_mean.index, hybrid\_forecast\_mean, label='Forecasted', color='red') # Forecasted data

plt.title('Hybrid Model Forecast vs Actual')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.grid(True)

plt.show()

Directional Accuracy

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np

def calculate\_metrics(actual, predicted):

mae = mean\_absolute\_error(actual, predicted)

rmse = np.sqrt(mean\_squared\_error(actual, predicted))

mape = np.mean(np.abs((actual - predicted) / actual)) \* 100

direction\_actual = np.sign(np.diff(actual))

direction\_predicted = np.sign(np.diff(predicted))

directional\_accuracy = np.mean(direction\_actual == direction\_predicted) \* 100

return mae, rmse, mape, directional\_accuracy

arimax\_mae, arimax\_rmse, arimax\_mape, arimax\_da = calculate\_metrics(test, forecast\_mean)

hybrid\_mae, hybrid\_rmse, hybrid\_mape, hybrid\_da = calculate\_metrics(test\_hybrid, hybrid\_forecast\_mean)

print("ARIMAX Model Metrics:")

print(f"MAE: {arimax\_mae:.2f}")

print(f"RMSE: {arimax\_rmse:.2f}")

print(f"MAPE: {arimax\_mape:.2f}%")

print(f"Directional Accuracy: {arimax\_da:.2f}%")

print("\nHybrid Model Metrics:")

print(f"MAE: {hybrid\_mae:.2f}")

print(f"RMSE: {hybrid\_rmse:.2f}")

print(f"MAPE: {hybrid\_mape:.2f}%")

print(f"Directional Accuracy: {hybrid\_da:.2f}%")

Diebold-Mariano Test

actual = test\_hybrid

forecast1 = forecast\_mean

forecast2 = hybrid\_forecast\_mean

dm\_statistic, dm\_pvalue = diebold\_mariano\_test(actual, forecast1, forecast2)

print(f"DM Statistic: {dm\_statistic}")

print(f"p-value: {dm\_pvalue}")

import numpy as np

import scipy.stats as stats

import pandas as pd

def diebold\_mariano\_test(actual, forecast1, forecast2, h=1):

d = np.power(actual - forecast1, 2) - np.power(actual - forecast2, 2)

mean\_d = np.mean(d)

var\_d = np.var(d, ddof=1) / len(d)

dm\_stat = mean\_d / np.sqrt(var\_d)

dm\_stat\_normalized = dm\_stat \* np.sqrt(len(d))

p\_value = 2 \* stats.norm.cdf(-np.abs(dm\_stat\_normalized))

return dm\_stat\_normalized, p\_value

actual = np.array(test\_hybrid)

forecast1 = np.array(forecast\_mean)

forecast2 = np.array(hybrid\_forecast\_mean)

dm\_statistic, dm\_pvalue = diebold\_mariano\_test(actual, forecast1, forecast2)

print(f"DM Statistic: {dm\_statistic}")

print(f"p-value: {dm\_pvalue}")

1. Time Series Plot

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 8))

plt.plot(test.index, test, label='Actual')

plt.plot(test.index, forecast\_mean, label='ARIMAX Forecast')

plt.plot(test.index, hybrid\_forecast\_mean, label='Hybrid Forecast')

plt.title('Bitcoin Price: Actual vs Forecasts')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.savefig('price\_forecasts\_comparison.png')

plt.close()

Residual plots

def plot\_residuals(actual, predicted, model\_name):

residuals = actual - predicted

plt.figure(figsize=(15, 8))

plt.plot(actual.index, residuals)

plt.title(f'{model\_name} Model Residuals')

plt.xlabel('Date')

plt.ylabel('Residual')

plt.savefig(f'{model\_name.lower()}\_residuals.png')

plt.close()

plot\_residuals(test, forecast\_mean, 'ARIMAX')

plot\_residuals(test\_hybrid, hybrid\_forecast\_mean, 'Hybrid')

Sentiment vs price movement

plt.figure(figsize=(15, 8))

plt.scatter(merged\_data['compound\_score'], merged\_data['Returns'], alpha=0.5)

plt.title('Sentiment Score vs Bitcoin Returns')

plt.xlabel('Compound Sentiment Score')

plt.ylabel('Daily Returns')

plt.savefig('sentiment\_vs\_returns.png')

plt.close()

Rolling window analysis

def rolling\_window\_forecast(data, window\_size, model\_func):

forecasts = []

for i in range(len(data) - window\_size):

train = data[i:i+window\_size]

test = data[i+window\_size:i+window\_size+1]

forecast = model\_func(train, test)

forecasts.append(forecast)

return pd.Series(forecasts, index=data.index[window\_size+1:])

def arimax\_forecast(train, test):

model = SARIMAX(train, order=order, exog=exog[train.index])

results = model.fit()

return results.forecast(steps=1, exog=exog.loc[test.index])

def hybrid\_forecast(train, test):

model = SARIMAX(train, order=order, exog=exog\_hybrid.loc[train.index])

results = model.fit()

return results.forecast(steps=1, exog=exog\_hybrid.loc[test.index])

window\_size = 252 # Approximately one year of trading days

arimax\_rolling = rolling\_window\_forecast(y, window\_size, arimax\_forecast)

hybrid\_rolling = rolling\_window\_forecast(y\_hybrid, window\_size, hybrid\_forecast)

rolling\_metrics = pd.DataFrame({

'ARIMAX\_MAE': calculate\_metrics(y[window\_size+1:], arimax\_rolling)[0],

'Hybrid\_MAE': calculate\_metrics(y\_hybrid[window\_size+1:], hybrid\_rolling)[0]

})

plt.figure(figsize=(15, 8))

rolling\_metrics.plot()

plt.title('Rolling MAE Comparison')

plt.xlabel('Date')

plt.ylabel('MAE')

plt.legend(['ARIMAX', 'Hybrid'])

plt.show()

Sensitivity Analysis

def sensitivity\_analysis(data, lags):

results = {}

for lag in lags:

exog = data[['MA7', 'MA30', 'Volatility7', 'Volatility30',

'compound\_score', 'tweet\_count'] +

[f'sentiment\_lag\_{i}' for i in range(1, lag+1)] +

[f'tweet\_count\_lag\_{i}' for i in range(1, lag+1)]]

train, test = y\_hybrid[:train\_size], y\_hybrid[train\_size:]

exog\_train, exog\_test = exog[:train\_size], exog[train\_size:]

model = SARIMAX(train, exog=exog\_train, order=order)

results\_fit = model.fit()

forecast = results\_fit.get\_forecast(steps=len(test), exog=exog\_test)

mae, rmse, mape, da = calculate\_metrics(test, forecast.predicted\_mean)

results[lag] = {'MAE': mae, 'RMSE': rmse, 'MAPE': mape, 'DA': da}

return pd.DataFrame(results).T

lags\_to\_test = [1, 3, 7, 14, 30]

sensitivity\_results = sensitivity\_analysis(merged\_data, lags\_to\_test)

plt.figure(figsize=(15, 8))

sensitivity\_results[['MAE', 'RMSE']].plot(marker='o')

plt.title('Sensitivity Analysis: Impact of Sentiment Lags')

plt.xlabel('Number of Lags')

plt.ylabel('Error')

plt.legend()

plt.show()

Cross-Validation

from sklearn.model\_selection import TimeSeriesSplit

def time\_series\_cv(data, model\_func, n\_splits=5):

tscv = TimeSeriesSplit(n\_splits=n\_splits)

cv\_scores = []

for train\_index, test\_index in tscv.split(data):

train = data.iloc[train\_index]

test = data.iloc[test\_index]

forecast = model\_func(train, test)

mae, \_, \_, \_ = calculate\_metrics(test, forecast)

cv\_scores.append(mae)

return cv\_scores

arimax\_cv\_scores = time\_series\_cv(y, arimax\_forecast)

hybrid\_cv\_scores = time\_series\_cv(y\_hybrid, hybrid\_forecast)

print("ARIMAX Cross-Validation MAE Scores:", arimax\_cv\_scores)

print("Hybrid Cross-Validation MAE Scores:", hybrid\_cv\_scores)

plt.figure(figsize=(10, 6))

plt.boxplot([arimax\_cv\_scores, hybrid\_cv\_scores], labels=['ARIMAX', 'Hybrid'])

plt.title('Cross-Validation MAE Comparison')

plt.ylabel('MAE')

plt.show()