# -\*- coding: utf-8 -\*-

"""Untitled21.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1JYfwwJI8Ev4aVWwRef7GLpTrz3BZCoIp

"""

# 📦 Install missing libraries

!pip install ta --quiet

# 📚 Imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# For technical indicators

import ta

# Statistical tools

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

# Deep learning

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# Machine learning

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# General setup

import warnings

warnings.filterwarnings("ignore")

np.random.seed(42)

from google.colab import files

import io

import pandas as pd

# Upload again (if needed)

uploaded = files.upload()

# Print filenames to verify

print("Uploaded files:", uploaded.keys())

# Adjust names based on actual keys

# This is a safe way to load the files dynamically

for filename in uploaded.keys():

if 'bitcoin' in filename.lower():

btc\_df = pd.read\_csv(io.StringIO(uploaded[filename].decode('utf-8')))

print(f"✅ Loaded Bitcoin data from: {filename}")

elif 'tesla' in filename.lower():

tsla\_df = pd.read\_csv(io.StringIO(uploaded[filename].decode('utf-8')))

print(f"✅ Loaded Tesla data from: {filename}")

# For date format like "13/07/2010" → dd/mm/yyyy with slashes

btc\_df['Date'] = pd.to\_datetime(btc\_df['Date'], dayfirst=True)

tsla\_df['Date'] = pd.to\_datetime(tsla\_df['Date'], dayfirst=True)

# Set date as index

btc\_df.set\_index('Date', inplace=True)

tsla\_df.set\_index('Date', inplace=True)

# Preview to confirm success

print("✅ Bitcoin date format fixed:")

display(btc\_df.head())

print("✅ Tesla date format fixed:")

display(tsla\_df.head())

import ta

import ta # If not already installed, run: !pip install ta

def add\_technical\_indicators(df):

# Create a copy to avoid modifying original

df = df.copy()

# --- VOLATILITY ---

df['Volatility'] = df['Close'].rolling(window=30).std()

# --- CONTROL VARIABLE: Volume ---

df['Volume\_log'] = np.log1p(df['Volume'])

# --- SMA (Simple Moving Average) ---

df['SMA12'] = ta.trend.sma\_indicator(df['Close'], window=12)

df['SMA26'] = ta.trend.sma\_indicator(df['Close'], window=26)

# --- EMA (Exponential Moving Average) ---

df['EMA12'] = ta.trend.ema\_indicator(df['Close'], window=12)

df['EMA26'] = ta.trend.ema\_indicator(df['Close'], window=26)

# --- RSI (Relative Strength Index) ---

df['RSI'] = ta.momentum.rsi(df['Close'], window=14)

# --- MACD (Moving Average Convergence Divergence) ---

df['MACD'] = ta.trend.macd\_diff(df['Close'])

# --- Bollinger Bands ---

bb = ta.volatility.BollingerBands(close=df['Close'], window=20, window\_dev=2)

df['Bollinger\_Upper'] = bb.bollinger\_hband()

df['Bollinger\_Lower'] = bb.bollinger\_lband()

# Drop rows with missing values from rolling windows

df.dropna(inplace=True)

return df

btc\_df = add\_technical\_indicators(btc\_df)

tsla\_df = add\_technical\_indicators(tsla\_df)

print("✅ BTC indicators:")

display(btc\_df.head())

print("✅ TSLA indicators:")

display(tsla\_df.head())

# Save to CSV in Colab's file system

btc\_df.to\_csv('Bitcoin\_with\_indicators.csv')

tsla\_df.to\_csv('Tesla\_with\_indicators.csv')

# Download to your computer

from google.colab import files

files.download('Bitcoin\_with\_indicators.csv')

files.download('Tesla\_with\_indicators.csv')

from statsmodels.tsa.stattools import adfuller

def check\_stationarity(series, name='Series'):

result = adfuller(series.dropna())

print(f"--- ADF Test for {name} ---")

print(f"ADF Statistic : {result[0]:.4f}")

print(f"p-value : {result[1]:.4f}")

if result[1] > 0.05:

print("⚠️ Series is likely NON-stationary (p > 0.05)")

else:

print("✅ Series is likely STATIONARY (p <= 0.05)")

# Run for Close prices

check\_stationarity(btc\_df['Close'], 'Bitcoin Close')

check\_stationarity(tsla\_df['Close'], 'Tesla Close')

from statsmodels.tsa.stattools import adfuller

def adf\_test\_report(series, name='Time Series'):

print(f"\n--- Augmented Dickey-Fuller Test: {name} ---")

result = adfuller(series.dropna(), autolag='AIC')

print(f"ADF Statistic : {result[0]:.4f}")

print(f"p-value : {result[1]:.4f}")

print(f"# Lags Used : {result[2]}")

print(f"# Observations : {result[3]}")

print("Critical Values :")

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

print(f" {key}: {value:.4f}")

def save\_adf\_results(series, name='Time Series', filename='adf\_results.txt'):

result = adfuller(series.dropna(), autolag='AIC')

with open(filename, 'a') as f:

f.write(f"\n--- Augmented Dickey-Fuller Test: {name} ---\n")

f.write(f"ADF Statistic : {result[0]:.4f}\n")

f.write(f"p-value : {result[1]:.4f}\n")

f.write(f"# Lags Used : {result[2]}\n")

f.write(f"# Observations : {result[3]}\n")

f.write("Critical Values :\n")

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

f.write(f" {key}: {value:.4f}\n")

if result[1] > 0.05:

f.write("⚠️ Series is likely NON-STATIONARY (p > 0.05)\n")

else:

f.write("✅ Series is likely STATIONARY (p <= 0.05)\n")

# Run and save for BTC and TSLA

save\_adf\_results(btc\_df['Close'], name='Bitcoin Close', filename='adf\_results.txt')

save\_adf\_results(tsla\_df['Close'], name='Tesla Close', filename='adf\_results.txt')

# After differencing

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

tsla\_df['Close\_diff'] = tsla\_df['Close'].diff()

save\_adf\_results(btc\_df['Close\_diff'], name='Bitcoin Close (1st Difference)', filename='adf\_results.txt')

save\_adf\_results(tsla\_df['Close\_diff'], name='Tesla Close (1st Difference)', filename='adf\_results.txt')

from google.colab import files

files.download('adf\_results.txt')

# First-order differencing

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

tsla\_df['Close\_diff'] = tsla\_df['Close'].diff()

# Check stationarity again after differencing

check\_stationarity(btc\_df['Close\_diff'], 'Bitcoin Close (1st Diff)')

check\_stationarity(tsla\_df['Close\_diff'], 'Tesla Close (1st Diff)')

import matplotlib.pyplot as plt

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

# Ensure differencing is done

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

# Plot ACF and PACF

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

plot\_acf(btc\_df['Close\_diff'].dropna(), ax=axes[0], lags=30)

axes[0].set\_title('Bitcoin - ACF (After First Differencing)')

plot\_pacf(btc\_df['Close\_diff'].dropna(), ax=axes[1], lags=30)

axes[1].set\_title('Bitcoin - PACF (After First Differencing)')

plt.tight\_layout()

plt.show()

fig.savefig("Bitcoin\_ACF\_PACF.png", dpi=300)

from google.colab import files

files.download("Bitcoin\_ACF\_PACF.png")

# Ensure differencing is done

tsla\_df['Close\_diff'] = tsla\_df['Close'].diff()

# Plot ACF and PACF

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

plot\_acf(tsla\_df['Close\_diff'].dropna(), ax=axes[0], lags=30)

axes[0].set\_title('Tesla - ACF (After First Differencing)')

plot\_pacf(tsla\_df['Close\_diff'].dropna(), ax=axes[1], lags=30)

axes[1].set\_title('Tesla - PACF (After First Differencing)')

plt.tight\_layout()

plt.show()

fig.savefig("Tesla\_ACF\_PACF.png", dpi=300)

from google.colab import files

files.download("Tesla\_ACF\_PACF.png")

# Plot Tesla differenced trend

plt.figure(figsize=(12, 5))

plt.plot(tsla\_df['Close\_diff'], label='Differenced Close', color='darkgreen')

plt.title('Tesla - Close Price After First Differencing')

plt.xlabel('Date')

plt.ylabel('Price Change')

plt.grid(True)

plt.legend()

plt.tight\_layout()

# Save and trigger download

plt.savefig("Tesla\_Differenced\_Trend.png", dpi=300)

from google.colab import files

files.download("Tesla\_Differenced\_Trend.png")

# Plot Bitcoin differenced trend

plt.figure(figsize=(12, 5))

plt.plot(btc\_df['Close\_diff'], label='Differenced Close', color='darkred')

plt.title('Bitcoin - Close Price After First Differencing')

plt.xlabel('Date')

plt.ylabel('Price Change')

plt.grid(True)

plt.legend()

plt.tight\_layout()

# Save and trigger download

plt.savefig("Bitcoin\_Differenced\_Trend.png", dpi=300)

from google.colab import files

files.download("Bitcoin\_Differenced\_Trend.png")

# Period splits for Bitcoin

btc\_period1 = btc\_df[(btc\_df.index >= '2014-09-17') & (btc\_df.index <= '2019-12-31')]

btc\_period2 = btc\_df[(btc\_df.index >= '2020-01-01') & (btc\_df.index <= '2022-12-31')]

btc\_period3 = btc\_df.copy()

# Period splits for Tesla

tsla\_period1 = tsla\_df[(tsla\_df.index >= '2010-07-01') & (tsla\_df.index <= '2019-12-31')]

tsla\_period2 = tsla\_df[(tsla\_df.index >= '2020-01-01') & (tsla\_df.index <= '2022-12-31')]

tsla\_period3 = tsla\_df.copy()

from statsmodels.tsa.arima.model import ARIMA

import pandas as pd

import numpy as np

# Define p and q ranges

p\_range = range(0, 11)

q\_range = range(0, 11)

# Period and dataset mapping

periods = {

'Bitcoin': {

'Period 1': btc\_period1,

'Period 2': btc\_period2,

'Period 3': btc\_period3

},

'Tesla': {

'Period 1': tsla\_period1,

'Period 2': tsla\_period2,

'Period 3': tsla\_period3

}

}

# Store results

aic\_results = []

# Fit ARIMA(p,1,q) for each period and asset

for asset\_name, period\_dict in periods.items():

for period\_name, df in period\_dict.items():

best\_aic = np.inf

best\_order = None

for p in p\_range:

for q in q\_range:

try:

model = ARIMA(df['Close'], order=(p, 1, q))

fitted\_model = model.fit()

aic = fitted\_model.aic

aic\_results.append({

'Asset': asset\_name,

'Period': period\_name,

'ARIMA\_Order': f"({p},1,{q})",

'AIC': round(aic, 2)

})

except Exception as e:

continue # Skip invalid configurations

# Create a DataFrame

aic\_df = pd.DataFrame(aic\_results)

# Get the best (lowest AIC) model per Period + Asset

best\_models\_df = aic\_df.sort\_values(by='AIC').groupby(['Asset', 'Period']).first().reset\_index()

# Display best models and export

best\_models\_df.to\_csv("Best\_ARIMA\_Models\_Per\_Period.csv", index=False)

from google.colab import files

files.download("Best\_ARIMA\_Models\_Per\_Period.csv")

# Save it again (after recreating it)

best\_models\_df.to\_csv("Best\_ARIMA\_Models\_Per\_Period.csv", index=False)

from google.colab import files

files.download("Best\_ARIMA\_Models\_Per\_Period.csv")

import pandas as pd

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.stats.diagnostic import acorr\_ljungbox

from scipy.stats import jarque\_bera

# 👉 Recreate your already split data (adjust as needed)

# btc\_period1, btc\_period2, btc\_period3 = ...

# tsla\_period1, tsla\_period2, tsla\_period3 = ...

# Define best ARIMA orders per period and asset

model\_specs = {

'Bitcoin': {

'Period 1': {'data': btc\_period1, 'order': (9, 1, 9)},

'Period 2': {'data': btc\_period2, 'order': (9, 1, 6)},

'Period 3': {'data': btc\_period3, 'order': (10, 1, 9)}

},

'Tesla': {

'Period 1': {'data': tsla\_period1, 'order': (3, 1, 3)},

'Period 2': {'data': tsla\_period2, 'order': (8, 1, 8)},

'Period 3': {'data': tsla\_period3, 'order': (7, 1, 9)}

}

}

diagnostics = []

for asset, periods in model\_specs.items():

for period, spec in periods.items():

try:

model = ARIMA(spec['data']['Close'], order=spec['order'])

fitted = model.fit()

residuals = fitted.resid.dropna()

# Diagnostics

loglik = round(fitted.llf, 3)

aic = round(fitted.aic, 3)

bic = round(fitted.bic, 3)

lb\_test = acorr\_ljungbox(residuals, lags=[1], return\_df=True)

lb\_p = round(lb\_test['lb\_pvalue'].iloc[0], 4)

jb\_stat, \_ = jarque\_bera(residuals)

jb\_stat = round(jb\_stat, 2)

diagnostics.append({

'Asset': asset,

'Period': period,

'Best Model': str(spec['order']),

'Log Likelihood': loglik,

'AIC': aic,

'BIC': bic,

'Ljung-Box (L1) (Q)': lb\_p,

'Jarque-Bera (JB)': jb\_stat

})

except Exception as e:

print(f"Error for {asset} {period}: {e}")

continue

# Create DataFrame

diag\_df = pd.DataFrame(diagnostics)

# Save to CSV

diag\_df.to\_csv("ARIMA\_Diagnostics\_All\_Periods.csv", index=False)

from google.colab import files

files.download("ARIMA\_Diagnostics\_All\_Periods.csv")