```
#  Import Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import TimeSeriesSplit, train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import xgboost as xgb
import joblib
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

plt.style.use('ggplot')   # safe built-in style
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)

# Upload dataset
```

```
from google.colab import files
print("Please upload your CSV file (from Excel export).")
uploaded = files.upload()   # choose your CSV file

# get filename
for fn in uploaded.keys():
   filename = fn
print("Uploaded:", filename)

# Load CSV and Initial Cleaning
df = pd.read_csv(filename, parse_dates=['Date'], dayfirst=False, low_memory=False)
df = df.sort_values('Date').reset_index(drop=True)

# normalize column names and map Price/Close ambiguity
df.columns = [c.strip() for c in df.columns]
rename_map = {}
for c in df.columns:
   lc = c.lower()
   if 'price' in lc or 'close' in lc:
      rename_map[c] = 'Close'
   if 'open' in lc:
      rename_map[c] = 'Open'
   if 'high' in lc:
      rename_map[c] = 'High'
   if 'low' in lc:
      rename_map[c] = 'Low'
```

```
    if 'vol' in lc and 'vol' in c.lower():
        rename_map[c] = 'Volume'
    if 'change' in lc:
        rename_map[c] = 'ChangePct'
df = df.rename(columns=rename_map)

# Clean numeric columns (remove commas and %)
for col in ['Close','Open','High','Low']:
    if col in df.columns:
        df[col] = df[col].astype(str).str.replace(',','').str.replace(' ','')
        df[col] = pd.to_numeric(df[col], errors='coerce')

# Volume '107.50K' handling
def vol_to_num(x):
    if pd.isna(x): return np.nan
    s = str(x).strip().replace(',','')
    try:
        if s[-1] in ['K','k']: return float(s[:-1]) * 1e3
        if s[-1] in ['M','m']: return float(s[:-1]) * 1e6
        return float(s)
    except:
        return np.nan

if 'Volume' in df.columns:
    df['Volume'] = df['Volume'].apply(vol_to_num)
```

```python
if 'ChangePct' in df.columns:
    df['ChangePct'] = df['ChangePct'].astype(str).str.replace('%','').str.replace(',','')
    df['ChangePct'] = pd.to_numeric(df['ChangePct'], errors='coerce')

# Drop rows with no Close
df = df.dropna(subset=['Close']).reset_index(drop=True)
print("Data shape after basic cleaning:", df.shape)
df.head()

# EDA Plots
print(df.info())
print(df.describe().T)

plt.figure(figsize=(12,5))
plt.plot(df['Date'], df['Close'])
plt.title('Gold Close Price over Time')
plt.xlabel('Date'); plt.ylabel('Price')
plt.tight_layout()
plt.savefig('gold_trend.png', dpi=300, bbox_inches='tight')
plt.show()

#Correlation Heatmap (EDA)
corr_cols = [c for c in ['Close','Open','High','Low','Volume','ChangePct'] if c in df.columns]
plt.figure(figsize=(8,6))
corr = df[corr_cols].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
```

```python
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.savefig('correlation_heatmap.png', dpi=300, bbox_inches='tight')
plt.show()

# Feature Engineering
# Moving averages
df['MA_7'] = df['Close'].rolling(7).mean()
df['MA_21'] = df['Close'].rolling(21).mean()
df['MA_50'] = df['Close'].rolling(50).mean()
df['MA_200'] = df['Close'].rolling(200).mean()

# Returns & log returns
df['LogReturn'] = np.log(df['Close']).diff()
df['Return_1d'] = df['Close'].pct_change()

# Volatility
df['Vol_21'] = df['Return_1d'].rolling(21).std()

# RSI
def compute_rsi(series, window=14):
    delta = series.diff()
    up = delta.clip(lower=0)
    down = -1*delta.clip(upper=0)
    ma_up = up.ewm(alpha=1/window, adjust=False).mean()
    ma_down = down.ewm(alpha=1/window, adjust=False).mean()
```

```
    rs = ma_up / (ma_down + 1e-10)
    rsi = 100 - (100 / (1 + rs))
    return rsi
df['RSI_14'] = compute_rsi(df['Close'], 14)


# MACD
ema12 = df['Close'].ewm(span=12, adjust=False).mean()
ema26 = df['Close'].ewm(span=26, adjust=False).mean()
df['MACD'] = ema12 - ema26
df['MACD_Signal'] = df['MACD'].ewm(span=9, adjust=False).mean()


df = df.dropna().reset_index(drop=True)


# Create Target Column
df['Target'] = df['Close'].shift(-1)
df = df.dropna().reset_index(drop=True)


features = ['Close','Open','High','Low','Volume',
            'MA_7','MA_21','MA_50','MA_200',
            'Vol_21','RSI_14','MACD','MACD_Signal','Return_1d','LogReturn']
features = [f for f in features if f in df.columns]
```

```
# Train/Test Split
split_idx = int(len(df)*0.8)
train_df = df.iloc[:split_idx]
test_df = df.iloc[split_idx:]

X_train = train_df[features].values
```

```
y_train = train_df['Target'].values
X_test = test_df[features].values
y_test = test_df['Target'].values
```

```
# Feature Scaling (for LSTM)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

joblib.dump(scaler, 'feature_scaler.joblib')


# RandomForest Model


from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import matplotlib.pyplot as plt

import numpy as np

import joblib


# RandomForest Regressor

rf = RandomForestRegressor(n_estimators=200, random_state=42)

rf.fit(X_train, y_train)


# Predictions

pred_rf = rf.predict(X_test)


# Metrics Function (Updated)
```

```python
def print_metrics(y_true, y_pred, name='Model'):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)  # manual RMSE calculation
    r2 = r2_score(y_true, y_pred)
    print(f"{name} -> MAE: {mae:.4f}, RMSE: {rmse:.4f}, R2: {r2:.4f}")

# Print RandomForest metrics
print_metrics(y_test, pred_rf, 'RandomForest')


# Plot Actual vs Predicted
plt.figure(figsize=(12,5))
plt.plot(test_df['Date'], y_test, label='Actual Price')
plt.plot(test_df['Date'], pred_rf, label='RF Predicted Price')
plt.xlabel('Date')
plt.ylabel('Gold Price')
plt.title('RandomForest: Actual vs Predicted Gold Price')
plt.legend()
plt.show()

# Save Model


joblib.dump(rf, 'rf_gold_model.joblib')
print("RandomForest model saved as 'rf_gold_model.joblib'")


#  XGBoost Model
```

```
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import numpy as np

# Initialize XGBoost Regressor

xg_reg = xgb.XGBRegressor(
    objective='reg:squarederror',
    n_estimators=300,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=RANDOM_SEED,
    n_jobs=-1
)

# Fit Model (without early_stopping_rounds / verbose)

xg_reg.fit(X_train, y_train)

# Predictions
pred_xgb = xg_reg.predict(X_test)
```

```python
# Metrics Function (Reused)


def print_metrics(y_true, y_pred, name='Model'):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"{name} -> MAE: {mae:.4f}, RMSE: {rmse:.4f}, R2: {r2:.4f}")


# Print XGBoost Metrics
print_metrics(y_test, pred_xgb, 'XGBoost')


# Plot Actual vs Predicted


plt.figure(figsize=(12,5))
plt.plot(test_df['Date'], y_test, label='Actual Price')
plt.plot(test_df['Date'], pred_xgb, label='XGB Predicted Price')
plt.xlabel('Date')
plt.ylabel('Gold Price')
plt.title('XGBoost: Actual vs Predicted Gold Price')
plt.legend()
plt.show()
# Save Model


xg_reg.save_model('xgb_gold_model.json')
print("XGBoost model saved as 'xgb_gold_model.json'")
```

```
# Import necessary libraries

from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as plt

import numpy as np


# Convert Regression to Classification


# Assume y_test is actual price and pred_xgb is predicted price

y_test_class = np.where(np.diff(y_test, prepend=y_test[0]) >= 0, 1, 0)    # 1 = price up, 0 = price
down

pred_xgb_class = np.where(np.diff(pred_xgb, prepend=pred_xgb[0]) >= 0, 1, 0)


# Compute ROC Curve and AUC


fpr, tpr, thresholds = roc_curve(y_test_class, pred_xgb_class)

roc_auc = auc(fpr, tpr)


# Plot ROC Curve

plt.figure(figsize=(7,6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)

plt.plot([0,1], [0,1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')
```

```
plt.title('ROC Curve — XGBoost (Price Up/Down)')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

#LSTM Model
# LSTM sequence setup
SEQ_LEN = 60
mm_scaler = MinMaxScaler()
X_all_scaled = mm_scaler.fit_transform(df[features])
y_all_scaled = mm_scaler.fit_transform(df['Target'].values.reshape(-1,1)).ravel()

def create_sequences(X, y, seq_len):
    Xs, ys = [], []
    for i in range(len(X)-seq_len):
        Xs.append(X[i:i+seq_len])
        ys.append(y[i+seq_len])
    return np.array(Xs), np.array(ys)

X_seq, y_seq = create_sequences(X_all_scaled, y_all_scaled, SEQ_LEN)
seq_split = split_idx - SEQ_LEN
X_train_s, y_train_s = X_seq[:seq_split], y_seq[:seq_split]
X_test_s, y_test_s = X_seq[seq_split:], y_seq[seq_split:]

# Build LSTM
model = Sequential()
```

```
model.add(LSTM(64, return_sequences=True, input_shape=(SEQ_LEN, len(features))))
model.add(Dropout(0.2))
model.add(LSTM(32))
model.add(Dropout(0.2))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.fit(X_train_s, y_train_s, validation_data=(X_test_s, y_test_s), epochs=30, batch_size=32,
shuffle=False)

# LSTM prediction & metrics

# Predict
pred_lstm_scaled = model.predict(X_test_s)
pred_lstm = mm_scaler.inverse_transform(pred_lstm_scaled)
y_test_orig = mm_scaler.inverse_transform(y_test_s.reshape(-1,1))

# Flatten arrays
y_test_orig_flat = y_test_orig.ravel()
pred_lstm_flat = pred_lstm.ravel()

# Correct date slice for LSTM test set
lstm_dates = test_df['Date'].iloc[-len(pred_lstm_flat):].reset_index(drop=True)

# Metrics function
def print_metrics(y_true, y_pred, name='Model'):
```

```
    from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
    import numpy as np
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"{name} -> MAE: {mae:.4f}, RMSE: {rmse:.4f}, R2: {r2:.4f}")

# Print metrics
print_metrics(y_test_orig_flat, pred_lstm_flat, 'LSTM')

# Plot Actual vs Predicted
import matplotlib.pyplot as plt
plt.figure(figsize=(12,5))
plt.plot(lstm_dates, y_test_orig_flat, label='Actual')
plt.plot(lstm_dates, pred_lstm_flat, label='LSTM Predicted')
plt.xlabel('Date')
plt.ylabel('Gold Price')
plt.title('LSTM: Actual vs Predicted Gold Price')
plt.legend()
plt.show()

# Save model
model.save('lstm_gold_model.h5')
print("LSTM model saved as 'lstm_gold_model.h5'")
```

```
#Metrics Summary Table with Accuracy %

import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Function to calculate metrics and accuracy %
def calculate_metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))  # manual RMSE
    r2 = r2_score(y_true, y_pred)
    accuracy_pct = r2 * 100  # approximate accuracy %
    return mae, rmse, r2, accuracy_pct

# RandomForest Metrics
rf_mae, rf_rmse, rf_r2, rf_acc = calculate_metrics(y_test, pred_rf)

# XGBoost Metrics
xgb_mae, xgb_rmse, xgb_r2, xgb_acc = calculate_metrics(y_test, pred_xgb)

# LSTM Metrics (flattened arrays)
y_test_lstm = y_test_orig_flat
pred_lstm_vals = pred_lstm_flat
lstm_mae, lstm_rmse, lstm_r2, lstm_acc = calculate_metrics(y_test_lstm, pred_lstm_vals)

# Create Summary Table
```

```
metrics_summary = pd.DataFrame({
    'Model': ['RandomForest', 'XGBoost', 'LSTM'],
    'MAE': [rf_mae, xgb_mae, lstm_mae],
    'RMSE': [rf_rmse, xgb_rmse, lstm_rmse],
    'R2 Score': [rf_r2, xgb_r2, lstm_r2],
    'Approx Accuracy %': [rf_acc, xgb_acc, lstm_acc]
})

metrics_summary


# Next-Day Prediction Example
latest_row = df.iloc[-1][features].values.reshape(1,-1)
print("Next-day RF:", rf.predict(latest_row)[0])
print("Next-day XGB:", xg_reg.predict(latest_row)[0])

last_seq = X_all_scaled[-SEQ_LEN:].reshape(1, SEQ_LEN, len(features))
pred_next_lstm = mm_scaler.inverse_transform(model.predict(last_seq))[0][0]
print("Next-day LSTM:", pred_next_lstm)
```