```
import os
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
from datetime import datetime

# plotting
import matplotlib.pyplot as plt

# time series model
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error

# logistic regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, roc_curve

# -------------------------
# Parameters
# -------------------------
RAW_URL = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_serie
RAW_SAVE = 'raw_covid_deaths.csv'
MELTED_SAVE = 'covid_deaths_melted.csv'
PROCESSED_SAVE = 'covid_deaths_processed.csv'

# -------------------------
# 1. Download & Load
# -------------------------
print('Downloading dataset from JHU CSSE GitHub...')
df = pd.read_csv(RAW_URL)
print('Shape:', df.shape)

# Save raw copy for reproducibility
df.to_csv(RAW_SAVE, index=False)
print(f'Raw data saved to {RAW_SAVE}')

# -------------------------
# 2. Quick exploration
# -------------------------
print('\n--- Head (first 5 rows) ---')
print(df.head())
print('\n--- Info ---')
print(df.info())
print('\n--- Describe (only numeric) ---')
print(df.describe(include=[np.number]))

# Identify columns
non_ts_cols = ['Province/State', 'Country/Region', 'Lat', 'Long']
all_cols = df.columns.tolist()
# Dates start after the 4th column
date_cols = [c for c in all_cols if c not in non_ts_cols]
for c in non_ts_cols:
    if c in date_cols:
        date_cols.remove(c)

print('\nDetected non-time-series columns:', non_ts_cols)
print('Detected time-series (date) columns count:', len(date_cols))

# -------------------------
# 3. Transform to long (melt)
# -------------------------
print('\nMelting to long format...')
meltdf = df.melt(id_vars=non_ts_cols, value_vars=date_cols, var_name='Date', value_name='Deaths')
print('Melted shape:', meltdf.shape)
meltdf.to_csv(MELTED_SAVE, index=False)
print(f'Melted data saved to {MELTED_SAVE}')

# -------------------------
# 4. Handle missing values
# -------------------------
# For non-time-series: Province/State -> 'N/A'
meltdf['Province/State'] = meltdf['Province/State'].fillna('N/A')
```

```
# Convert Date to datetime
meltdf['Date'] = pd.to_datetime(meltdf['Date'])

# Sort for reliable forward-fill
meltdf = meltdf.sort_values(['Country/Region', 'Province/State', 'Date']).reset_index(drop=True)

# Forward-fill Deaths per region (Province+Country)
meltdf['Deaths'] = meltdf.groupby(['Country/Region', 'Province/State'])['Deaths'].ffill().fillna(0).astype(int)

# Save processed
meltdf.to_csv(PROCESSED_SAVE, index=False)
print(f'Processed data saved to {PROCESSED_SAVE}')

# -------------------------
# 5. Time series model (SARIMAX) example
# -------------------------
# We'll pick a country (or let user change this variable)
COUNTRY = 'India'  # change as needed
print(f"\nPreparing time series for country: {COUNTRY}")

country_ts = meltdf[(meltdf['Country/Region'] == COUNTRY) & (meltdf['Province/State'] == 'N/A')]
if country_ts.empty:
    # some countries have Province entries; aggregate by country instead
    country_ts = meltdf[meltdf['Country/Region'] == COUNTRY].groupby('Date')['Deaths'].sum().reset_index()
else:
    country_ts = country_ts[['Date', 'Deaths']].groupby('Date').sum().reset_index()

country_ts = country_ts.set_index('Date').asfreq('D').fillna(method='ffill')
print(country_ts.tail())

# Train-test split for time series: last 14 days test
TEST_DAYS = 14
train = country_ts.iloc[:-TEST_DAYS]
test = country_ts.iloc[-TEST_DAYS:]

# Fit simple SARIMAX (order and seasonal_order are example choices)
print('\nFitting SARIMAX model (this may take a moment)...')
model = SARIMAX(train['Deaths'], order=(1,1,1), seasonal_order=(1,1,1,7), enforce_stationarity=False, enforce_invertibility=False)
model_fit = model.fit(disp=False)
print(model_fit.summary())

# Forecast
forecast = model_fit.forecast(steps=TEST_DAYS)
rmse = np.sqrt(mean_squared_error(test['Deaths'], forecast))
print(f'Time series RMSE on last {TEST_DAYS} days: {rmse:.2f}')

# Plot actual vs forecast
plt.figure(figsize=(10,5))
plt.plot(train.index[-60:], train['Deaths'][-60:], label='Train (last 60 days)')
plt.plot(test.index, test['Deaths'], label='Actual')
plt.plot(test.index, forecast, label='Forecast')
plt.title(f'{COUNTRY} cumulative deaths - actual vs forecast')
plt.legend()
plt.tight_layout()
plt.show()

# -------------------------
# 6. Logistic Regression on non-time-series (Lat, Long)
# -------------------------
# Create a snapshot: use the latest date per Country/Province combination
latest_date = meltdf['Date'].max()
snapshot = meltdf[meltdf['Date'] == latest_date].copy()

# If multiple entries per country/province exist, they are already per Province/State
print(f"Snapshot date used for classification: {latest_date.date()}")

# Create binary target: High death vs Low death using median threshold
threshold = snapshot['Deaths'].median()
snapshot['HighDeath'] = (snapshot['Deaths'] > threshold).astype(int)

# Features: Lat, Long (drop NaNs)
cls_df = snapshot[['Province/State', 'Country/Region', 'Lat', 'Long', 'Deaths', 'HighDeath']].dropna(subset=['Lat', 'Long'])

X = cls_df[['Lat', 'Long']].values
y = cls_df['HighDeath'].values

# Train-test
```

```
# Train-test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)

scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

clf = LogisticRegression(max_iter=200)
clf.fit(X_train_s, y_train)

y_pred = clf.predict(X_test_s)
y_proba = clf.predict_proba(X_test_s)[:,1]

print('\nLogistic Regression classification report:')
print(classification_report(y_test, y_pred))
print('Confusion matrix:\n', confusion_matrix(y_test, y_pred))
print('ROC AUC:', roc_auc_score(y_test, y_proba))

# Plot ROC
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr)
plt.plot([0,1],[0,1],'--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve - Logistic Regression (Lat,Long -> HighDeath)')
plt.tight_layout()
plt.show()

# Save snapshot with predictions
cls_df['Predicted'] = clf.predict(scaler.transform(cls_df[['Lat','Long']]))
```