Anomaly Detection & Time Series | Assignment

Question 1: What is Anomaly Detection? Explain its types (point, contextual, and collective anomalies) with examples.
Ans:

Anomaly Detection is the process of identifying data points or patterns that significantly differ from normal behavior.

Types:

1. **Point Anomaly –** A single data point is abnormal.
   Example: A credit card transaction of ₹5,00,000 when usual spending is ₹5,000.

2. **Contextual Anomaly –** A data point is abnormal in a specific context (time, location, etc.).
   Example: 30°C is normal in summer but abnormal in winter.

3. **Collective Anomaly –** A group of data points together is abnormal, even if individual points seem normal.
   Example: Multiple small unusual login attempts indicating a cyberattack.

Question 2: Compare Isolation Forest, DBSCAN, and Local Outlier Factor in terms of their approach and suitable use cases.
Answer:

- **Isolation Forest –** Uses random trees to isolate anomalies; outliers are easier to separate.
  Best for: Large, high-dimensional datasets where anomalies are rare and global.

- **DBSCAN –** Density-based clustering; points in low-density regions are marked as outliers.
  Best for: Spatial/geographical data or when clusters have arbitrary shapes.

- **Local Outlier Factor (LOF) –** Compares local density of a point with its neighbors; lower local density = outlier.
  Best for: Detecting local anomalies in datasets with varying densities.

Question 3: What are the key components of a Time Series? Explain each with one example.

Answer:

1. **Trend –** The long-term directional movement in data.
   Example: Steady increase in online sales year over year.

2. **Seasonality –** Regular patterns that repeat at fixed intervals.
   Example: Higher e-commerce sales during Diwali or year-end holidays.

3. **Cyclicality –** Fluctuations occurring over longer, irregular business cycles.
   Example: Sales decline during an economic recession.

4. **Irregular (Noise) –** Random, unpredictable variations.
   Example: Sudden drop in sales due to a temporary website outage.

Question 4: Define Stationary in time series. How can you test and transform a non-stationary series into a stationary one?

Answer:

Stationary (in Time Series):
A time series is stationary if its mean, variance, and covariance remain constant over time.

How to test:

- Use Augmented Dickey-Fuller test (ADF test)
- Check rolling mean and variance plots

How to make it stationary:

- Differencing (subtract previous value)
- Log or square-root transformation
- Detrending / removing seasonality

Question 5: Differentiate between AR, MA, ARIMA, SARIMA, and SARIMAX models in terms of structure and application.

Answer:

- **AR (AutoRegressive)** – Predicts current value using its own past values.
- **MA (Moving Average)** – Predicts current value using past error terms (residuals).
- **ARIMA** – Combines AR + MA with differencing (I) to handle non-stationary data.
- **SARIMA** – ARIMA + seasonal components to model repeating seasonal patterns.
- **SARIMAX** – SARIMA + external (exogenous) variables to improve forecasting accuracy.

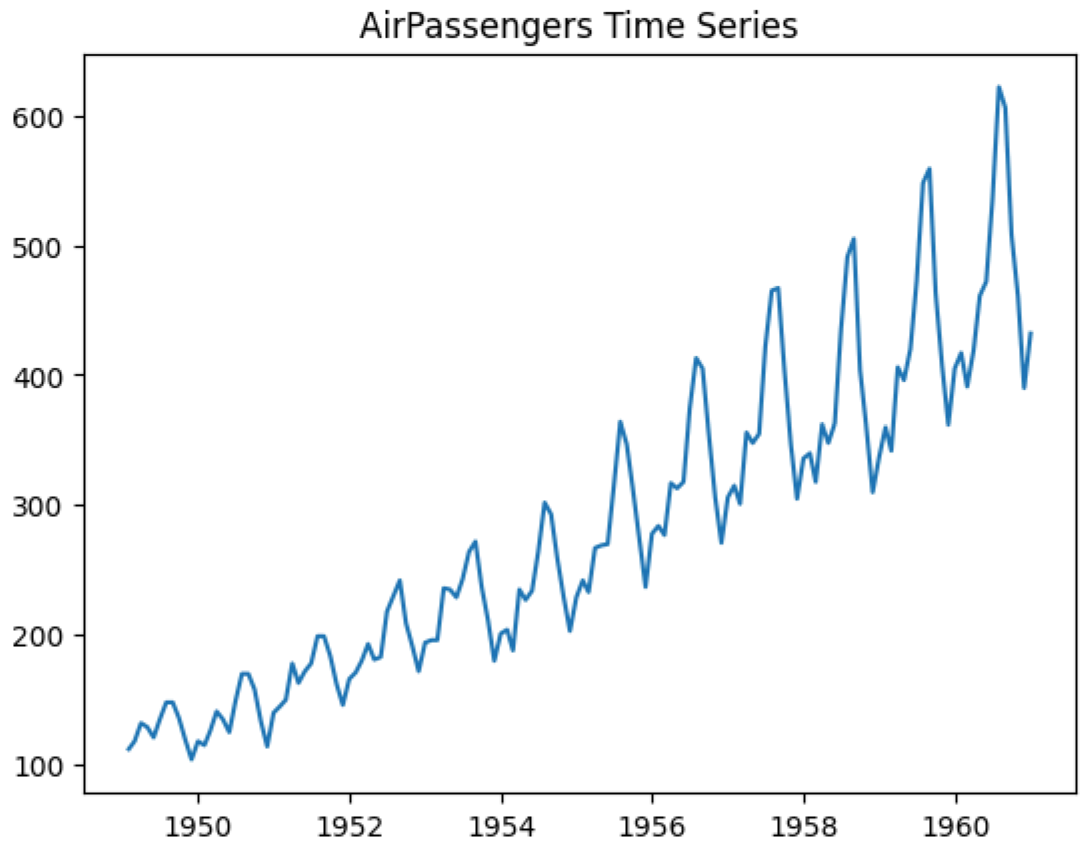
Question 6: Load a time series dataset (e.g., AirPassengers), plot the original series, and decompose it into trend, seasonality, and residual components (Include your Python code and output in the code box below.)

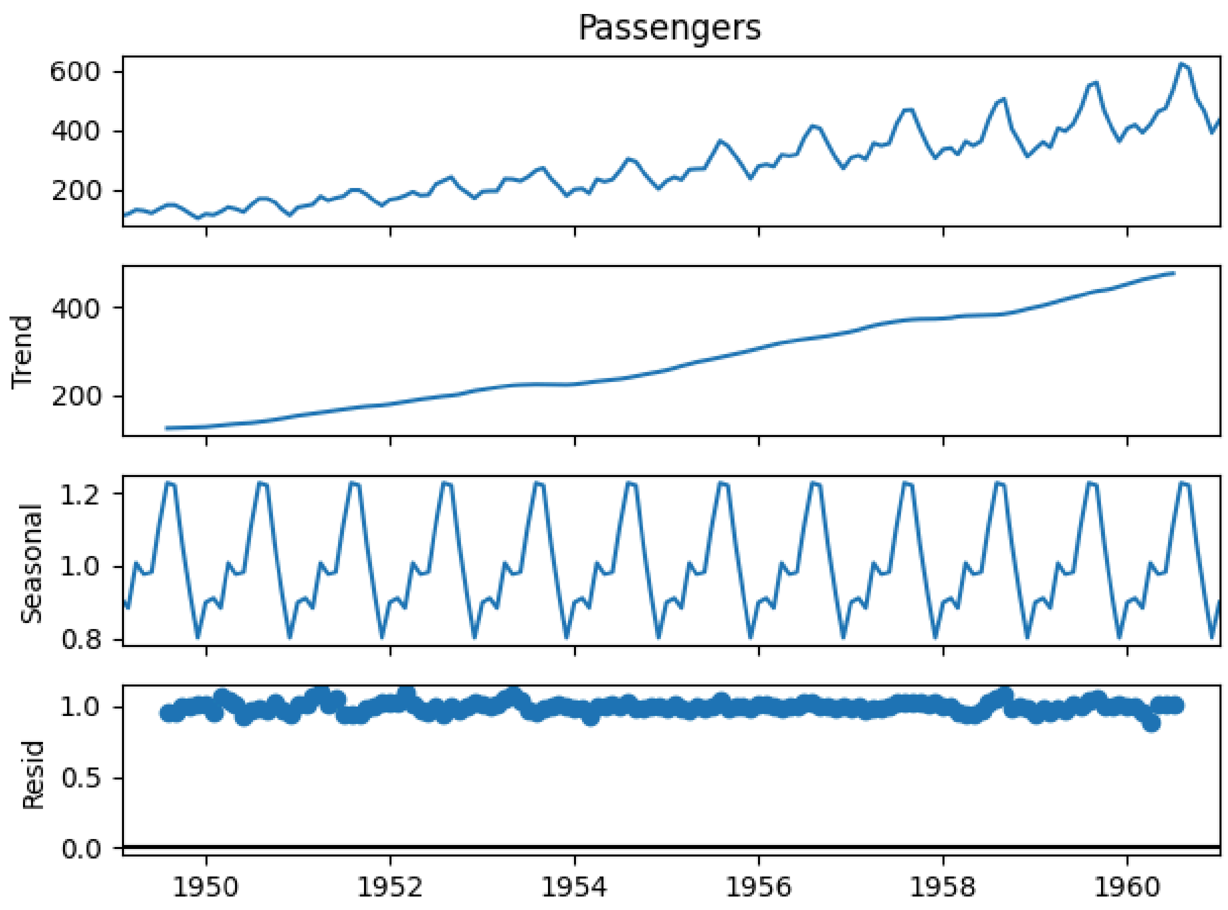
Answer:

```
import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal_decompose

import statsmodels.api as sm



# Load AirPassengers dataset directly

data = sm.datasets.get_rdataset("AirPassengers").data



# Convert to datetime

data['time'] = pd.date_range(start='1949-01', periods=len(data), freq='M')

data.set_index('time', inplace=True)
```

```
# Rename column

data.rename(columns={'value': 'Passengers'}, inplace=True)



# Plot original series

plt.figure()

plt.plot(data['Passengers'])

plt.title("AirPassengers Time Series")

plt.show()



# Decompose

decomposition = seasonal_decompose(data['Passengers'],
model='multiplicative', period=12)

decomposition.plot()

plt.show()
```

O/P :


AirPassengers Time Series
100
200
300
400
500
600
1950
1952
1954
1956
1958
1960



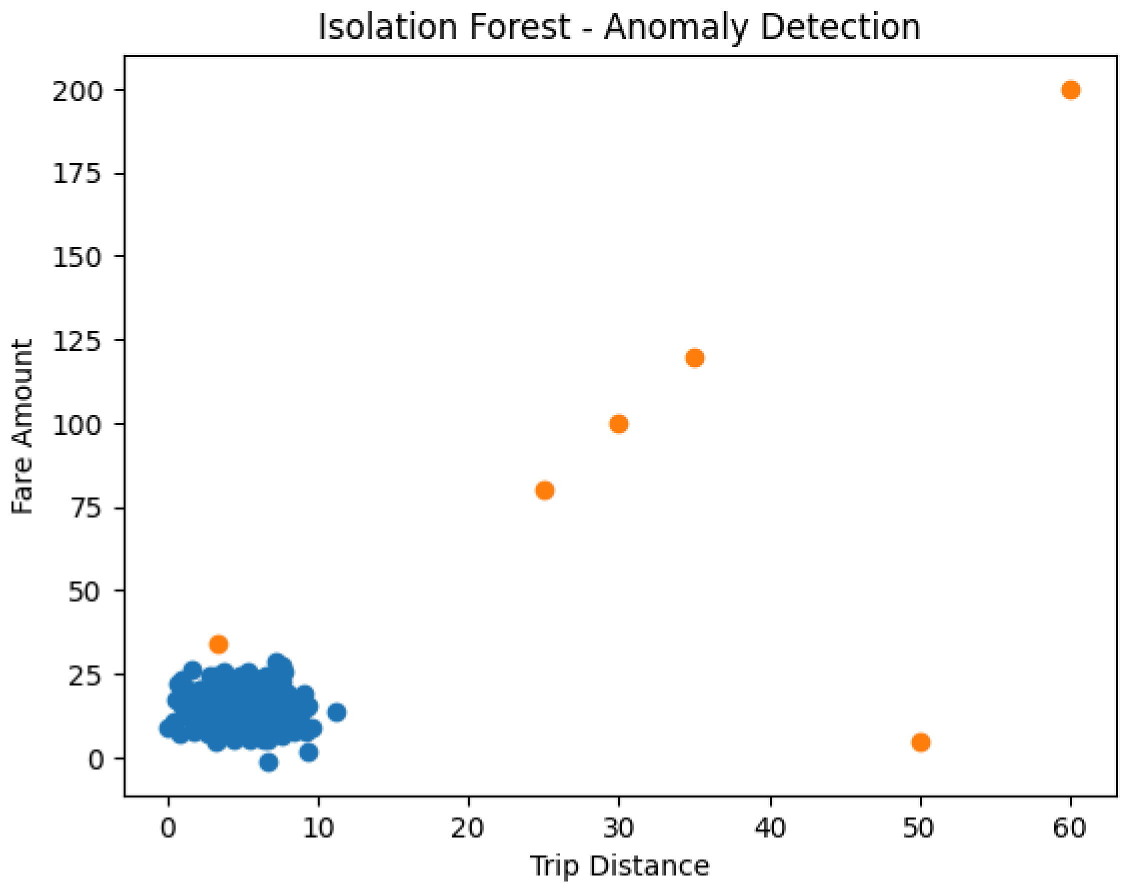
Question 7: Apply Isolation Forest on a numerical dataset (e.g., NYC Taxi Fare) to detect anomalies. Visualize the anomalies on a 2D scatter plot. (Include your Python code and output in the code box below.)

Answer:

```
# Isolation Forest on a numerical dataset (example: NYC Taxi Fare-like
data)



import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest
```

```
# Sample dataset (replace with real NYC taxi data if available)

np.random.seed(42)

n = 300


data = pd.DataFrame({

    "fare_amount": np.random.normal(15, 5, n),

    "trip_distance": np.random.normal(5, 2, n)

})


# Add some artificial anomalies

data.loc[295:] = [[80, 25], [100, 30], [120, 35], [5, 50], [200, 60]]


# Apply Isolation Forest

model = IsolationForest(contamination=0.02, random_state=42)

data["anomaly"] = model.fit_predict(data[["fare_amount",
"trip_distance"]])


# -1 = anomaly, 1 = normal

anomalies = data[data["anomaly"] == -1]

normal = data[data["anomaly"] == 1]


# Visualization (2D scatter plot)

plt.scatter(normal["trip_distance"], normal["fare_amount"])
plt.scatter(anomalies["trip_distance"], anomalies["fare_amount"])

plt.xlabel("Trip Distance")

plt.ylabel("Fare Amount")

plt.title("Isolation Forest - Anomaly Detection")

plt.show()
```

O/P :



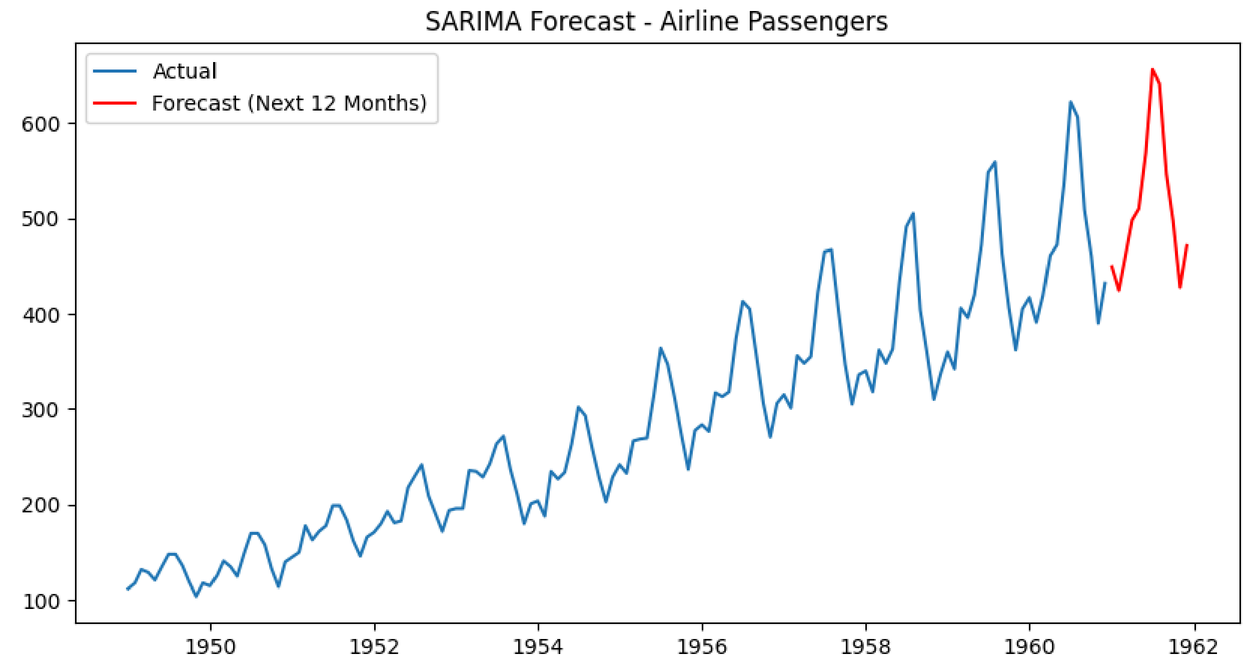
Question 8: Train a SARIMA model on the monthly airline passengers dataset. Forecast the next 12 months and visualize the results. (Include your Python code and output in the code box below.)

Answer:

```
# SARIMA on Monthly Airline Passengers Dataset


import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace.sarimax import SARIMAX


# Load dataset

url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passe
ngers.csv"

df = pd.read_csv(url, parse_dates=['Month'], index_col='Month')


# Train SARIMA model (Seasonal period = 12 for monthly data)

model = SARIMAX(df['Passengers'],

                order=(1,1,1),

                seasonal_order=(1,1,1,12))

results = model.fit()


# Forecast next 12 months

forecast = results.forecast(steps=12)


# Plot results
```

```
plt.figure(figsize=(10,5))

plt.plot(df['Passengers'], label='Actual')

plt.plot(forecast, label='Forecast (Next 12 Months)', color='red')

plt.legend()

plt.title("SARIMA Forecast - Airline Passengers")

plt.show()



# Print forecast values

print(forecast)
```

O/P:



```
961-01-01     449.330169
```

```
1961-02-01    424.386085

1961-03-01    459.031845

1961-04-01    497.864849

1961-05-01    509.862600

1961-06-01    568.258333

1961-07-01    655.810542

1961-08-01    641.190401

1961-09-01    546.391990

1961-10-01    496.800827

1961-11-01    427.673808

1961-12-01    471.235212

Freq: MS, Name: predicted_mean, dtype: float64
```

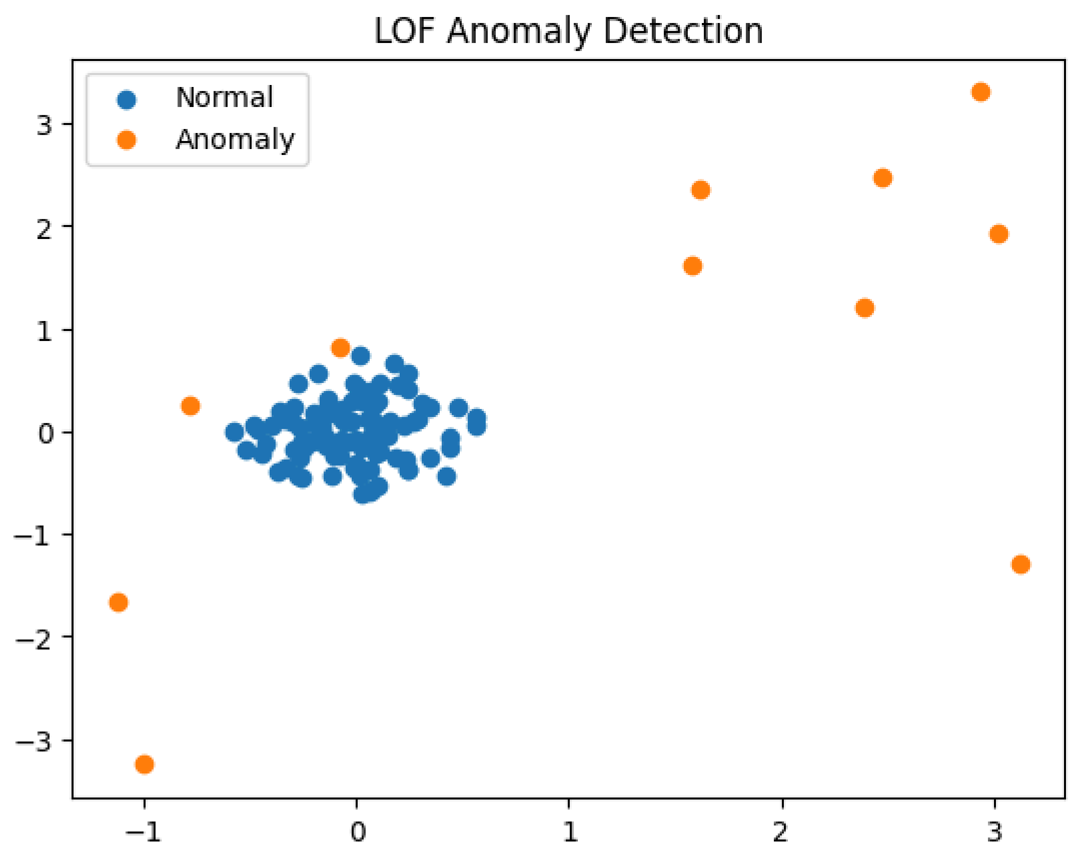
Question 9: Apply Local Outlier Factor (LOF) on any numerical dataset to detect anomalies and visualize them using matplotlib. (Include your Python code and output in the code box below.)

Answer:

```python
# Local Outlier Factor (LOF) Example - Anomaly Detection



import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import LocalOutlierFactor



# Create sample numerical dataset

np.random.seed(42)

normal_data = 0.3 * np.random.randn(100, 2)
```

```
outliers = np.random.uniform(low=-4, high=4, size=(10, 2))

X = np.vstack([normal_data, outliers])



# Apply LOF

lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)

y_pred = lof.fit_predict(X)



# Separate normal and anomaly points

normal_points = X[y_pred == 1]

anomaly_points = X[y_pred == -1]



# Plot

plt.scatter(normal_points[:, 0], normal_points[:, 1], label="Normal")

plt.scatter(anomaly_points[:, 0], anomaly_points[:, 1], label="Anomaly")

plt.legend()

plt.title("LOF Anomaly Detection")

plt.show()
```

O/P:



Question 10: You are working as a data scientist for a power grid monitoring company. Your goal is to forecast energy demand and also detect abnormal spikes or drops in real-time consumption data collected every 15 minutes. The dataset includes features like timestamp, region, weather conditions, and energy usage. Explain your real-time data science workflow: ● How would you detect anomalies in this streaming data (Isolation Forest / LOF / DBSCAN)? ● Which time series model would you use for short-term forecasting (ARIMA / SARIMA / SARIMAX)? ● How would you validate and monitor the performance over time? ● How would this solution help business decisions or operations?

Ans:

1 Real-Time Workflow (High-Level)

Data Flow:
Streaming data (15-min interval) → Preprocessing (missing values, scaling, feature engineering like hour, day, lag features) →
Parallel Models:

- Anomaly Detection Model
- Forecasting Model
  → Monitoring Dashboard + Alerts

2️⃣ Anomaly Detection (Streaming)

For power consumption spikes/drops:

- **Primary Choice: Isolation Forest**
  - Works well for high-dimensional data
  - Efficient for large datasets
  - Good for real-time scoring

LOF is computationally heavier for streaming.
DBSCAN is better for clustering, not ideal for fast real-time detection.

```
import pandas as pd

import numpy as np

from sklearn.ensemble import IsolationForest

from sklearn.preprocessing import StandardScaler

from statsmodels.tsa.statespace.sarimax import SARIMAX



# -----------------------------

# 1. AUTO-DETECT DATETIME COLUMN

# -----------------------------
# Try to detect datetime column automatically

datetime_col = None


for col in df.columns:

    if 'date' in col.lower() or 'time' in col.lower():

        datetime_col = col

        break


if datetime_col is None:

    raise ValueError("No datetime column found. Please check dataset.")


# Convert to datetime and set index

df[datetime_col] = pd.to_datetime(df[datetime_col])

df = df.sort_values(datetime_col)

df.set_index(datetime_col, inplace=True)


# -----------------------------

# 2. CLEAN COLUMN NAMES

# -----------------------------


# Remove spaces and lowercase everything

df.columns = df.columns.str.strip().str.lower()
```

```
print("Final Columns:", df.columns)


# ------------------------------

# 3. ADJUST FEATURE NAMES HERE

# ------------------------------


# CHANGE these according to your dataset

energy_col = 'energy_usage'

temp_col = 'temperature'

humidity_col = 'humidity'


if energy_col not in df.columns:

    raise ValueError(f"{energy_col} not found. Check actual column name.")


# Drop missing values

df = df.dropna()


# ------------------------------

# 4. ANOMALY DETECTION

# ------------------------------


features = [energy_col]
```

```python
if temp_col in df.columns:

    features.append(temp_col)


if humidity_col in df.columns:

    features.append(humidity_col)


scaler = StandardScaler()

X_scaled = scaler.fit_transform(df[features])


iso_model = IsolationForest(contamination=0.02, random_state=42)

df['anomaly'] = iso_model.fit_predict(X_scaled)


anomalies = df[df['anomaly'] == -1]


print("Total anomalies detected:", len(anomalies))


# -----------------------------

# 5. SARIMAX FORECASTING

# -----------------------------


target = df[energy_col]


# Only use exog if available
```

```
exog = None

exog_cols = []


if temp_col in df.columns:

    exog_cols.append(temp_col)


if humidity_col in df.columns:

    exog_cols.append(humidity_col)


if len(exog_cols) > 0:

    exog = df[exog_cols]


model = SARIMAX(target,

                exog=exog,

                order=(1,1,1),

                seasonal_order=(1,1,1,96),

                enforce_stationarity=False,

                enforce_invertibility=False)


results = model.fit()


# Forecast next 4 steps

if exog is not None:
    future_exog = exog.tail(4)

    forecast = results.forecast(steps=4, exog=future_exog)

else:

    forecast = results.forecast(steps=4)



print("Next 1-hour forecast:")

print(forecast)
```