Assignment Code: DA-AG-018

Anomaly Detection & Time Series | Assignment

Instructions: Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

Total Marks: 100

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

Answer:

Anomaly detection is the process of identifying unusual data points, patterns, or behaviours that deviate significantly from the norm.
These deviations are often indicators of critical incidents, such as fraud, network intrusion, or system failures.
Types of anomalies include:
1. Point Anomalies – A single data point significantly different from the rest. Example: A sudden temperature spike to 90°C in a sensor normally reading 25°C.
2. Contextual Anomalies – Data points that are abnormal in a specific context. Example: High electricity use at night may be abnormal, but normal during the day.
3. Collective Anomalies – A group of data points that together deviate from expected behaviour. Example: A series of low network activity readings could indicate a system shutdown.

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

Answer:

Isolation Forest: A tree-based algorithm that isolates anomalies by randomly selecting features and split values. Efficient for high-dimensional datasets.
DBSCAN: A density-based clustering method identifying anomalies as points not belonging to any cluster. Works well for spatial or arbitrary-shaped clusters.
Local Outlier Factor (LOF): Measures local density deviation of a data point with respect to its neighbours. Suitable for detecting local outliers in continuous data.
Summary:
- Isolation Forest → Large, high-dimensional datasets
- DBSCAN → Spatial or clustering-based data
- LOF → Local density-based anomaly detection

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

Answer:

Key components:
1. Trend – The long-term direction (upward/downward). Example: Increasing yearly sales.
2. Seasonality – Regular pattern repeating over time. Example: High ice cream sales during summer.
3. Cyclic – Irregular fluctuations over long periods (economic cycles).
4. Irregular/Noise – Random variations caused by unpredictable events.

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

Answer:

A time series is stationary if its statistical properties (mean, variance, autocorrelation) remain constant over time.
To test stationarity, use the Augmented Dickey-Fuller (ADF) test.
To transform non-stationary data: apply differencing, log transformation, or seasonal decomposition to stabilize mean and variance.

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

Answer:

AR (Auto-Regressive): Uses lagged values of the series.
MA (Moving Average): Uses past error terms.
ARIMA: Combines AR and MA on differenced data.
SARIMA: Extends ARIMA with seasonal components.
SARIMAX: SARIMA with exogenous variables (external predictors).

Dataset:

- NYC Taxi Fare Data
- AirPassengers Dataset

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:

Code:

```
# Q6: Load and Decompose AirPassengers dataset
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.datasets import get_rdataset

# Load dataset
data = get_rdataset('AirPassengers').data
data.columns = ['Month', 'Passengers']
data['Month'] = pd.to_datetime(data['Month'])
data.set_index('Month', inplace=True)

# Plot the time series
plt.figure(figsize=(10, 4))
plt.plot(data['Passengers'])
plt.title("AirPassengers Time Series")
plt.xlabel("Year")
```

```
plt.ylabel("Number of Passengers")
plt.show()

# Decompose
result = seasonal_decompose(data['Passengers'], model='multiplicative', period=12)
result.plot()
plt.show()
```

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:

Code:

```
# Q7: Isolation Forest Anomaly Detection
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

# Create synthetic taxi fare dataset
np.random.seed(42)
fare = np.random.normal(15, 5, 200)      # typical fare range
distance = np.random.normal(5, 2, 200)   # trip distance
# Add anomalies
fare[190:] = fare[190:] * 3
distance[190:] = distance[190:] * 2
X = np.column_stack((fare, distance))

# Apply Isolation Forest
clf = IsolationForest(contamination=0.1, random_state=42)
y_pred = clf.fit_predict(X)

# Plot results
plt.figure(figsize=(6, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='coolwarm')
plt.title("Isolation Forest - Anomaly Detection")
plt.xlabel("Fare Amount")
plt.ylabel("Trip Distance")
plt.show()
```

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:

Code:

```
# Q8: SARIMA Forecast for Next 12 Months
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Use same AirPassengers data
series = data['Passengers']

# Fit SARIMA model
model = SARIMAX(series, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
results = model.fit()

# Forecast next 12 months
forecast = results.forecast(steps=12)

# Plot forecast
plt.figure(figsize=(10, 5))
plt.plot(series, label='Original')
plt.plot(forecast, label='Forecast', color='red')
plt.title("SARIMA Forecast - AirPassengers")
plt.legend()
plt.show()
```

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:

Code:

```
# Q9: Local Outlier Factor
from sklearn.neighbors import LocalOutlierFactor

# Synthetic numeric data
X = 0.3 * np.random.randn(200, 2)
```

```
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))
X = np.r_[X, X_outliers]

# Apply LOF
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
y_pred = lof.fit_predict(X)

# Visualize
plt.figure(figsize=(6, 4))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='coolwarm')
plt.title("Local Outlier Factor (LOF) - Anomaly Detection")
plt.show()
```

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?

(*Include your Python code and output in the code box below.*)
Answer:

1. Anomaly Detection:
Use Isolation Forest or Local Outlier Factor on streaming data to detect abnormal spikes/drops in energy consumption.
2. Forecasting Model:
Use SARIMAX for short-term forecasting considering external features (e.g., weather).
3. Validation & Monitoring:
Use rolling forecasts, RMSE/MAE for accuracy, and real-time dashboards to track deviations.
4. Business Impact:
Helps optimize energy production, prevent overloads, reduce costs, and maintain grid stability.