Sharad Chaurasiya
+91 7489542109
chaurasiyasharad737@gmail.com
Data Science and Generative AI

Assignment Code: DA-AG-018

Anomaly Detection & Time Series | **Assignment |**

Question 1: What is Anomaly Detection? Explain its types (point, contextual, and collective anomalies) with examples.
Answer - Anomaly detection, also called outlier detection, is the process of identifying data points, events, or observations that deviate significantly from the majority of the data and do not conform to a normal pattern. Anomalies often indicate a problem or a rare event, like fraud, a mechanical failure, or a security breach. It's a key technique in many fields, including cybersecurity, finance, and manufacturing.

Types of Anomalies

There are three main types of anomalies, each defined by how a data point is considered "unusual."

Point Anomalies (Global Outliers)

A **point anomaly** is a single data instance that is significantly different from the rest of the dataset. It's an individual data point that's a clear outlier, regardless of its context.

- **Example:** In a dataset of daily credit card transactions, a single transaction for $10,000 might be a point anomaly if the average transaction amount for that user is typically under $100. This single, high-value transaction is suspicious on its own.

Contextual Anomalies

A **contextual anomaly** is a data point that is abnormal within a specific context but might be considered normal in a different one. The "context" can be anything from time and location to a specific user's behavior.

- **Example:** A spike in electricity usage for a residential home is considered normal in the afternoon on a hot summer day (due to air conditioning). However, that same spike in usage

at 3 a.m. would be a contextual anomaly, potentially indicating a malfunction or an unauthorized activity.

Collective Anomalies

A **collective anomaly** is a set of related data points that, when viewed together, show unusual behavior, even if each individual data point in the set is not anomalous on its own. The pattern or sequence of the data is the key to identifying this type of anomaly.

- **Example:** A patient's electrocardiogram (ECG) might show a series of heartbeats that, individually, seem normal. However, the specific sequence and rhythm of these heartbeats, when analyzed as a group, could indicate an arrhythmia, which is a collective anomaly.

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

Approach: Isolation Forest works on the principle that anomalies are "few and different," making them easier to isolate than normal data points. It builds an ensemble of random decision trees (an "Isolation Forest") by recursively partitioning the data space. Anomalies are those points that get isolated in fewer splits, resulting in a shorter average path length in the trees. The algorithm doesn't require a concept of distance or density; it just measures how easy a point is to separate.

Suitable Use Cases:

- **Large datasets with many dimensions:** Isolation Forest is highly scalable and computationally efficient, making it ideal for big data applications.
- **When you expect a small percentage of global outliers:** It's particularly effective at finding point anomalies, like a single fraudulent transaction or a sensor reading that's far outside the normal range.
- **Unlabeled data:** As an unsupervised algorithm, it doesn't need pre-labeled anomalies for training.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Approach: DBSCAN is a density-based clustering algorithm. It groups together data points that are closely packed, marking points in low-density regions as outliers (or "noise"). It classifies each data point as a **core point** (many neighbors), a **border point** (in a cluster but few neighbors), or an **outlier** (neither). Anomalies are the points that are not part of any dense cluster.

Suitable Use Cases:

- **Data with irregularly shaped clusters:** DBSCAN can find anomalies in datasets where clusters aren't spherical, which is a limitation for other algorithms.
- **When you need to detect collective anomalies:** It excels at finding patterns of unusual behavior that, when viewed as a group, are far from dense clusters.
- **Datasets with a clear separation between normal data and noise:** It's excellent for datasets where you can define what "dense" and "sparse" mean.

Local Outlier Factor (LOF)

Approach: LOF is a density-based algorithm that calculates a local outlier score for each data point. It measures the local density of a point and compares it to the local densities of its neighbors. A point is an outlier if its local density is significantly lower than that of its neighbors. This makes it a great choice for detecting contextual anomalies. A point that might be normal in one dense region could be considered an outlier if it's in a less dense area.

Suitable Use Cases:

- **When you need to find local anomalies:** LOF is perfect for datasets where outliers exist within clusters of varying densities. For example, a data point that is a normal value for a dense cluster might be an outlier if it's in a sparse region.
- **High-dimensional data:** LOF is very effective in detecting outliers in datasets with a high number of features.
- **When density is a meaningful measure of normality:** It's suitable for applications where the concept of "being an outlier" is relative to the immediate surroundings, such as in network intrusion detection or manufacturing defect analysis.

Question 3: What are the key components of a Time Series? Explain each with one example.
Answer: A **time series** is a sequence of data points indexed in chronological order. The purpose of analyzing a time series is to understand the underlying patterns and to forecast future values. These patterns, or components, are typically separated into four categories.

---

1. Trend

The **trend** is the long-term upward or downward movement in a time series. It represents the general direction of the data over an extended period, ignoring short-term fluctuations. A trend can be linear (a constant rate of change) or non-linear.

- **Example:** The increasing number of registered users on a social media platform over several years, despite daily or weekly variations.

---

2. Seasonality

Seasonality refers to a predictable and recurring pattern that repeats itself over a fixed period, like a day, a week, a month, or a year. These patterns are often influenced by the calendar and weather. Unlike cyclical variations, the length of a seasonal pattern is always consistent.

- **Example:** A retail store's sales peaking every December due to holiday shopping.

---

3. Cyclicality

Cyclicality describes fluctuations that are not of a fixed period and usually last for more than one year. These variations are often associated with economic or business cycles, such as periods of expansion and recession. The duration and magnitude of a cycle are not as predictable as a seasonal pattern.

- **Example:** The rise and fall of the housing market over a decade, which may not have a fixed, repeating pattern like seasonality.

---

4. Irregularity (Noise)

The **irregular** or **random** component is what's left after the trend, seasonal, and cyclical components have been removed. It represents unpredictable, random fluctuations caused by unusual or unforeseen events that do not have a discernible pattern.

- **Example:** A sudden spike in a stock's price due to an unexpected news event, such as a company merger or a natural disaster.

Question 4: Define Stationary in time series. How can you test and transform a non-stationary series into a stationary one?
Answer: A **stationary** time series is one whose statistical properties—like the mean, variance, and autocorrelation—are constant over time. This means that a time series with a trend (a consistent upward or downward movement), seasonality (predictable, repeating patterns), or a changing

variance is not stationary. Many time series models, such as ARIMA, assume the data is stationary, so transforming a non-stationary series is a crucial preprocessing step.

Tests for Stationarity

You can test for stationarity both visually and statistically.

- **Visual Inspection:** Plot the time series and look for a clear trend (e.g., a series that is consistently increasing or decreasing) or a seasonal pattern. Also, check if the data's variance (the spread of the values) appears to change over time. If any of these are present, the series is likely non-stationary.
- **Statistical Tests:** These are more rigorous and objective.
    - **Augmented Dickey-Fuller (ADF) Test:** This is a popular unit root test.
        - **Null Hypothesis (H0):** The time series is non-stationary (it has a unit root).
        - **Alternative Hypothesis (Ha):** The time series is stationary.
        - **Interpretation:** If the p-value is small (e.g., less than 0.05), you can reject the null hypothesis and conclude the series is stationary.
    - **Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test:** This test has the opposite null hypothesis.
        - **Null Hypothesis (H0):** The time series is stationary.
        - **Alternative Hypothesis (Ha):** The time series is non-stationary.
        - **Interpretation:** If the p-value is small (e.g., less than 0.05), you can reject the null hypothesis and conclude the series is non-stationary.

How to Transform a Non-Stationary Series

The goal is to remove trends and seasonality to stabilize the mean and variance.

- **Differencing:** This is the most common and effective method for removing a trend. You subtract the previous observation from the current one. If the first difference doesn't achieve stationarity, you can repeat the process (second-order differencing).
    - First difference: $\Delta Y_t = Y_t - Y_{t-1}$
    - Seasonal differencing: If the series has seasonality, you can subtract the observation from the same period in the previous season.
- **Logarithmic Transformation:** This is used to stabilize the variance of a series, especially when the variance increases with the mean. Taking the logarithm of the series can compress the values and make the variance more constant.

- **Decomposition:** You can explicitly decompose a time series into its trend, seasonal, and residual components. By modeling and removing the trend and seasonal parts, the remaining residuals will be a stationary series (the noise).

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

Autoregressive (AR) Models

- **Structure:** An AR(p) model predicts the current value of a time series based on its own **past values**. The "p" denotes the number of lagged observations included in the model.
- **Application:** Suitable for time series data that has a linear correlation with its past values. For example, predicting today's stock price based on the prices of the previous few days.

Moving Average (MA) Models

- **Structure:** An MA(q) model predicts the current value based on past **forecast errors**. The "q" is the number of lagged forecast errors used in the model.
- **Application:** Useful for time series data where the current value depends on past random shocks or white noise. For instance, forecasting a series of temperature changes where random, short-term weather events are a key factor.

ARIMA (AutoRegressive Integrated Moving Average)

- **Structure:** ARIMA is a combination of the AR and MA models, with an additional "Integrated" (I) component. It is denoted as **ARIMA(p, d, q)**.
    - **AR(p):** The autoregressive part.
    - **I(d):** The number of times the data must be **differenced** to make it stationary (remove trends).
    - **MA(q):** The moving average part.
- **Application:** Ideal for **non-stationary** time series data that does not exhibit seasonality. For example, forecasting a company's sales that have a clear upward trend but no repeating seasonal pattern.

SARIMA (Seasonal ARIMA)

- **Structure:** SARIMA extends ARIMA by adding seasonal components. It's denoted as **SARIMA(p, d, q)(P, D, Q)s**.
    - **(p, d, q):** The non-seasonal components (same as ARIMA).
    - **(P, D, Q)s:** The seasonal components, where:
        - **P:** Seasonal autoregressive order.
        - **D:** Seasonal differencing order.
        - **Q:** Seasonal moving average order.
        - **s:** The number of observations in a seasonal cycle (e.g., 12 for monthly data).
- **Application:** The go-to model for **time series with a clear seasonal pattern**. A classic example is forecasting monthly retail sales, which typically peak around the holidays every year.

SARIMAX (Seasonal ARIMA with eXogenous variables)

- **Structure:** SARIMAX is the most comprehensive of these models. It's an extension of SARIMA that includes **exogenous variables** (X). These are external factors that can influence the time series.
- **Application:** Used when you believe external variables affect your forecast. For example, predicting ice cream sales (a seasonal product) using a SARIMAX model that also includes daily temperature as an exogenous variable. The temperature data helps explain fluctuations beyond the standard seasonal pattern.

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 necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
import warnings
warnings.filterwarnings('ignore')

# Load the AirPassengers dataset
# The dataset is available in a standard library for convenience
```

```
df =
pd.read_csv('https://raw.githubusercontent.com/datasets/air-passengers/main/data/air-passengers.
csv')

# Convert 'Month' column to datetime objects
df['Month'] = pd.to_datetime(df['Month'])

# Set 'Month' as the index
df.set_index('Month', inplace=True)

# Plot the original time series
plt.figure(figsize=(12, 6))
plt.plot(df['Passengers'])
plt.title('AirPassengers Time Series Data')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.show()

# Decompose the time series
decomposition = seasonal_decompose(df['Passengers'], model='multiplicative')

# Plot the decomposed components
fig = decomposition.plot()
fig.set_size_inches(12, 8)
plt.show()

# Output the components
# print("\nTrend Component:")
# print(decomposition.trend.head())
# print("\nSeasonality Component:")
# print(decomposition.seasonal.head())
# print("\nResidual Component:")
# print(decomposition.resid.head())
```

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:

```
# Import necessary libraries
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

# Load a sample of the NYC Taxi Fare data
# The dataset has 'value' as the only feature, representing taxi fares
df = pd.read_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/master/nyc-taxi.csv',
index_col=0)

# The Isolation Forest algorithm works best in higher dimensions.
# For a 2D plot, we'll create a synthetic second feature based on 'value'
# A more realistic use case would use multiple features like trip distance, time, etc.
df['value_2'] = df['value'] * np.random.uniform(0.9, 1.1, len(df))

# Isolate the features
X = df[['value', 'value_2']]

# Apply Isolation Forest
# contamination: the expected proportion of outliers in the data
model = IsolationForest(contamination=0.01, random_state=42)
df['outlier_score'] = model.fit_predict(X)

# Filter out the inliers and outliers
outliers = df[df['outlier_score'] == -1]
inliers = df[df['outlier_score'] == 1]

# Visualize the anomalies on a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(inliers['value'], inliers['value_2'], c='blue', label='Inliers')
plt.scatter(outliers['value'], outliers['value_2'], c='red', s=100, label='Anomalies')
plt.title('Isolation Forest Anomaly Detection on NYC Taxi Data')
plt.xlabel('Taxi Fare Value')
plt.ylabel('Synthesized Feature')
plt.legend()
plt.show()

# Print the number of detected anomalies
print(f"Total anomalies detected: {len(outliers)}")
print("\nFirst 5 detected anomalies:")
print(outliers.head())
```

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:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
import warnings
warnings.filterwarnings('ignore')

# Load the AirPassengers dataset
df =
pd.read_csv('https://raw.githubusercontent.com/datasets/air-passengers/main/data/air-passengers.
csv')
df['Month'] = pd.to_datetime(df['Month'])
df.set_index('Month', inplace=True)

# Define the SARIMA model parameters based on data analysis (decomposition)
# Non-seasonal (p, d, q)
p, d, q = 2, 1, 2
# Seasonal (P, D, Q, s) - with a seasonal period of 12 months
P, D, Q, s = 1, 1, 1, 12

# Create and fit the SARIMA model
model = SARIMAX(df['Passengers'],
                order=(p, d, q),
                seasonal_order=(P, D, Q, s),
                enforce_stationarity=False,
                enforce_invertibility=False)

# The `enforce_stationarity=False` and `enforce_invertibility=False`
# are used to bypass checks that can sometimes fail for certain parameter combinations.
# In a rigorous analysis, you'd find the optimal parameters.

results = model.fit(disp=False)

# Get the number of months in the dataset for a clear forecast start
num_months = len(df)

# Forecast the next 12 months
```

```
forecast_steps = 12
forecast = results.get_prediction(start=num_months, end=num_months + forecast_steps - 1)

# Get the confidence intervals
forecast_ci = forecast.conf_int()

# Plot the original data and the forecast
plt.figure(figsize=(15, 7))

# Plot original series
plt.plot(df.index, df['Passengers'], label='Original')

# Plot forecasted values
forecast_index = pd.date_range(start=df.index[-1], periods=forecast_steps, freq='MS')
plt.plot(forecast_index, forecast.predicted_mean, color='red', label='Forecast')

# Plot confidence intervals
plt.fill_between(forecast_index,
                 forecast_ci.iloc[:, 0],
                 forecast_ci.iloc[:, 1],
                 color='pink', alpha=0.3, label='Confidence Interval')

plt.title('SARIMA Model Forecast of AirPassengers')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.show()

# Print the forecast values and confidence intervals
print("Forecasted values for the next 12 months:")
print(forecast.predicted_mean.round(2))
print("\nForecasted confidence intervals:")
print(forecast_ci.round(2))
```

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:

```
# Import necessary libraries
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor

# Create a sample numerical dataset
np.random.seed(42)
X = np.random.randn(200, 2)  # Generate 200 random data points
X[150:155] += 5              # Add a small cluster of outliers
X[160] = [10, 10]            # Add a single, extreme outlier
X[165:170] -= 5              # Add another cluster of outliers

# Apply Local Outlier Factor (LOF)
# n_neighbors: number of neighbors to consider for local density calculation
# contamination: proportion of outliers in the data
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
y_pred = lof.fit_predict(X)
outlier_scores = lof.negative_outlier_factor_

# Filter the data to separate inliers and outliers
outliers = X[y_pred == -1]
inliers = X[y_pred == 1]

# Visualize the anomalies
plt.figure(figsize=(10, 8))
plt.title("Local Outlier Factor (LOF) Anomaly Detection")

# Plot the inliers
plt.scatter(inliers[:, 0], inliers[:, 1], c='blue', label='Inliers')

# Plot the outliers
plt.scatter(outliers[:, 0], outliers[:, 1], c='red', label='Anomalies', s=100, edgecolors='black')

# You can also use the outlier scores to show density on the plot
# plt.scatter(X[:, 0], X[:, 1], c=outlier_scores, cmap='coolwarm', s=50, label='LOF Score')
# plt.colorbar(label='Negative LOF Score')

plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

```
# Print the number of detected anomalies
print(f"Total anomalies detected: {len(outliers)}")
```

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?
Answer: here's the step by step solution : -

This response does not require a code block as it is a conceptual explanation of a data science workflow.

---

1. Anomaly Detection in Streaming Data

For real-time anomaly detection, I'd choose the **Isolation Forest** model. Its primary advantage is its **computational efficiency** and **scalability**, which are crucial for processing data streamed every 15 minutes. The model works by isolating anomalies with fewer random partitions in a decision tree, making it faster than density-based methods like LOF or DBSCAN. It's also well-suited for high-dimensional data, as the dataset includes multiple features like `timestamp`, `region`, and `weather conditions`.

---

2. Time Series Model for Short-Term Forecasting

For short-term energy demand forecasting, the **SARIMAX** model is the most suitable choice. It is an extension of the SARIMA model that includes **exogenous variables**. Energy demand is highly seasonal (daily, weekly, and yearly patterns), which SARIMA's seasonal components (`P, D, Q, s`) can capture perfectly. The SARIMAX model allows the inclusion of external factors like `weather conditions` (e.g., temperature and humidity), which are known to have a significant impact on energy usage, especially for heating and cooling. This allows the model to produce more accurate forecasts than a simple SARIMA model.

---

3. Validation and Monitoring

Validation and monitoring are continuous processes in a real-time system.

- **Initial Validation:** The models would be first validated on historical data. For forecasting, I'd use metrics like **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**. For anomaly detection, **precision** and **recall** would be used.
- **Ongoing Monitoring:** A **real-time dashboard** would display key performance indicators (KPIs). For the forecasting model, this would include a comparison of **forecasted vs. actual values** and a running error calculation. For the anomaly detection model, it would show the number of anomalies detected over time and their context.
- **Model Retraining:** The models would be periodically retrained on a rolling window of recent data to adapt to long-term shifts in usage patterns, such as a changing climate or new industrial consumers.

---

4. Business Decisions and Operations

This solution would have a direct and significant impact on business and operational decisions.

- **Real-time Anomaly Detection:** Operators would be immediately alerted to abnormal spikes or drops. This could indicate a major equipment failure, a security breach, or a sudden, unexpected change in demand, allowing for a **proactive response** to prevent larger issues and ensure grid stability.
- **Short-Term Forecasting:** Accurate energy demand forecasts are crucial for optimizing power generation and distribution. This enables the company to:
    - **Resource Management:** Plan power generation schedules and balance the grid to avoid costly over-generation or service disruptions.
    - **Financial Planning:** Improve budgeting and financial risk management by better understanding future energy requirements.
    - **Proactive Maintenance:** An anomaly in a specific region could trigger a targeted investigation, helping to identify and fix issues before they escalate.