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

Anomaly Detection

Anomaly Detection (also called *outlier detection*) is the process of identifying data points, patterns, or observations that deviate significantly from the expected or normal behavior of a dataset. These anomalies often indicate critical events such as errors, fraud, faults, or rare but important phenomena.

An anomaly is typically defined relative to a normal data distribution, which may be statistical, temporal, spatial, or contextual in nature.

Types of Anomalies

Anomalies are commonly classified into three main types:

1. Point Anomalies

Definition:
A **point anomaly** is a single data instance that is significantly different from the rest of the data.

Key characteristic:

- Individual data point is abnormal by itself.

Example:

- In a dataset of daily temperatures (20–30°C), a value of **60°C** on a single day is a point anomaly.
- In banking, a transaction of **₹10,00,000** from an account that usually transacts below ₹10,000.

Use cases:

- Fraud detection

- Sensor fault detection
- Data cleaning

2. Contextual Anomalies (Conditional Anomalies)

Definition:
A **contextual anomaly** is a data point that is anomalous **only in a specific context**, such as time, location, or condition.

Key characteristic:

- Requires context (e.g., time, season, location) to be considered anomalous.

Example:

- **30°C** is normal in summer but anomalous in winter.
- High electricity usage at **3 AM** in a residential household.
- High heart rate during sleep but normal during exercise.

Context dimensions:

- Time (day/night, season)
- Location
- Environmental conditions

Use cases:

- Time-series analysis
- Climate monitoring
- Healthcare monitoring

3. Collective Anomalies

Definition:
A **collective anomaly** occurs when a group of related data points together form an anomalous pattern, even though individual points may appear normal.

Key characteristic:

- The anomaly is in the **sequence or group behavior**, not in individual points.

Example:

- A sequence of normal network packets that together indicate a **DDoS attack**.
- A sudden continuous drop in sensor readings indicating equipment failure.
- Repeated small withdrawals that together indicate money laundering.

Use cases:

- Intrusion detection systems
- Fraud pattern detection
- Time-series and sequence analysis

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

Comparison of Isolation Forest, DBSCAN, and Local Outlier Factor (LOF)

Isolation Forest, DBSCAN, and Local Outlier Factor are widely used techniques for **anomaly detection**, but they differ significantly in **underlying approach, assumptions, and suitable use cases**.

1. Isolation Forest

Approach:
Isolation Forest is a **tree-based, model-driven** method. It isolates observations by randomly selecting a feature and a split value. Anomalies are isolated **faster** because they are fewer and lie far from normal data points, resulting in **shorter path lengths** in the trees.

Key characteristics:

- Does **not** rely on distance or density
- Works well in **high-dimensional** spaces
- Computationally efficient for large datasets

Suitable use cases:

- Large-scale datasets
- High-dimensional data (e.g., log files, transaction data)
- When data distribution is unknown

Example:
Detecting fraudulent credit card transactions in millions of records.

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

Approach:
DBSCAN is a **density-based clustering** algorithm. It groups data points into clusters based on density and labels points that do not belong to any dense region as **noise (anomalies)**.

Key characteristics:

- Identifies anomalies as **low-density points**
- Does **not** require the number of clusters in advance
- Sensitive to parameters: `eps` and `min_samples`

Suitable use cases:

- Spatial or geometric data
- Datasets with **clearly separated dense clusters**
- When anomalies lie in sparse regions

Limitations:

- Struggles with **varying densities**
- Performance degrades in high dimensions

Example:
Detecting abnormal GPS locations or unusual customer behavior patterns.

3. Local Outlier Factor (LOF)

Approach:
LOF is a **local density-based** method. It compares the density of a data point with the density of its neighbors. A point is an anomaly if its local density is significantly **lower than that of surrounding points**.

Key characteristics:

- Detects **local anomalies**
- Sensitive to choice of `k` (number of neighbors)
- Relies on distance calculations

Suitable use cases:

- Data with **varying densities**
- When anomalies are **contextual or local**
- Moderate-sized datasets

Limitations:

- Computationally expensive for large datasets
- Not ideal for very high-dimensional data

Example:
Detecting unusual customer behavior within a specific segment.

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

Answer:

Key Components of a Time Series

A **time series** is a set of observations recorded sequentially over time, usually at equal intervals (daily, monthly, yearly, etc.). To understand and analyze a time series effectively, it is decomposed into key components.

1. Trend (T)

Meaning:
The **trend** shows the long-term direction or overall movement of the data over time.

Example:

- A continuous increase in **mobile phone users** over several years.

2. Seasonality (S)

Meaning:
Seasonality refers to regular and predictable patterns that repeat at fixed intervals.

Example:

- **Electricity consumption** increases every evening and during summer months.

3. Cyclical Component (C)

Meaning:
The **cyclical component** represents fluctuations occurring over long periods, usually due to economic or business cycles. These patterns are not fixed in duration.

Example:

- Periodic rise and fall in **GDP growth** during economic expansions and recessions.

4. Irregular / Random Component (R)

Meaning:
The **irregular component** captures random, unpredictable variations caused by unexpected events.

Example:

- A sudden drop in **tourism** due to a natural disaster or pandemic.

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

Stationarity in Time Series

Stationarity in a time series refers to a property where the statistical characteristics of the series remain constant over time. A stationary time series is easier to model and forecast because its behavior is stable and predictable.

Definition of a Stationary Time Series

A time series is said to be stationary if:

1. Mean remains constant over time
2. Variance remains constant over time
3. Autocovariance / autocorrelation depends only on the lag, not on time

If any of these properties change with time, the series is non-stationary.

Testing for Stationarity

1. Visual Inspection (Informal Test)

- **Time plot:** Look for trends or changing variance
- **Rolling statistics:** Plot rolling mean and variance

Example:
A steadily increasing sales series indicates non-stationarity.

2. Augmented Dickey–Fuller (ADF) Test

- **Null hypothesis (H_0):** The series is non-stationary
- **Alternative hypothesis (H_1):** The series is stationary

Decision rule:

- If **p-value < 0.05**, reject H_0 → series is stationary

- If **p-value ≥ 0.05**, fail to reject H_0 → non-stationary

3. KPSS Test

- **Null hypothesis (H_0):** The series is stationary
- **Alternative hypothesis (H_1):** The series is non-stationary

Using **ADF and KPSS together** gives stronger confirmation.

Transforming a Non-Stationary Series into a Stationary One

1. Differencing

Subtract the previous value from the current value:

$$Y_t' = Y_t - Y_{t-1}$$

- Removes **trend**
- Most common method
- Seasonal differencing can remove seasonality

Example:
Daily sales data with upward trend → first differencing.

2. Transformation (Variance Stabilization)

- **Log transformation**
- **Square root transformation**
- **Box–Cox transformation**

Used when variance increases over time.

Example:
Log of stock prices to stabilize volatility.

3. Detrending

- Fit a trend line and remove it from the data

Example:
Subtracting a linear trend from temperature data.

4. Seasonal Adjustment

- Remove seasonal patterns using seasonal differencing or decomposition

Example:
Monthly retail sales adjusted for festival seasons.

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

Answer:

AR, MA, ARIMA, SARIMA, and SARIMAX are classical **time-series forecasting models**. They differ in **model structure**, **assumptions**, and **application scenarios**.

1. AR (Autoregressive) Model

Structure:

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t$$

Explanation:
The current value depends on its **own past values**.

Application:

- When data is **stationary**
- When past values strongly influence future values

Example:
Forecasting temperature where yesterday's temperature affects today's value.

2. MA (Moving Average) Model

Structure:

$$Y_t = c + \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \cdots + \theta_q\varepsilon_{t-q}$$

Explanation:
The current value depends on **past error terms**.

Application:

- Stationary series
- Useful for modeling **shock effects**

Example:
Modeling sales affected by random promotional errors.

3. ARIMA (Autoregressive Integrated Moving Average)

Structure:
ARIMA(p, d, q)

- **p** → Autoregressive terms
- **d** → Differencing order
- **q** → Moving average terms

Explanation:

Combines AR and MA models with **differencing** to handle non-stationary data.

Application:

- Non-stationary data without seasonality
- Widely used general-purpose forecasting model

Example:
Forecasting monthly sales with trend but no seasonal pattern.

4. SARIMA (Seasonal ARIMA)

Structure:
SARIMA(p, d, q)(P, D, Q)s_ss

- First part → non-seasonal components
- Second part → seasonal components
- **s** → seasonal period (e.g., 12 for monthly data)

Explanation:
Extends ARIMA by explicitly modeling **seasonality**.

Application:

- Data with trend **and seasonal patterns**

Example:
Monthly electricity demand showing yearly seasonality.

5. SARIMAX (Seasonal ARIMA with eXogenous Variables)

Structure:
SARIMAX(p, d, q)(P, D, Q)s_ss + exogenous variables

Explanation:
SARIMA model that also incorporates **external (exogenous) variables** that

influence the time series.

Application:

- When external factors affect the target variable
- Improves forecast accuracy

Example:
Predicting sales using seasonality plus advertising spend and discounts.

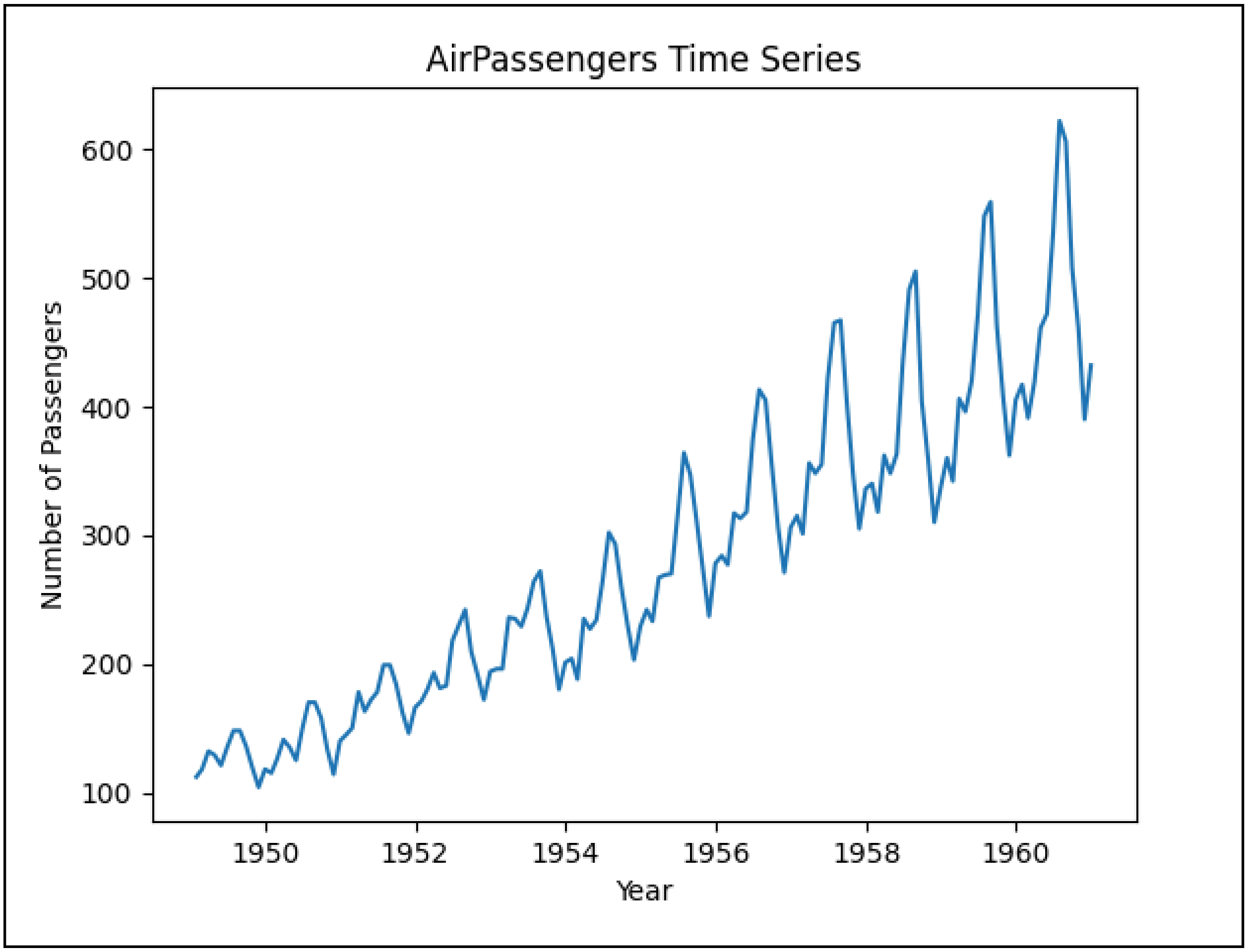
Question 6: Load a time series dataset (e.g., AirPassengers), plot the original series, and decompose it into trend, seasonality, and residual components .

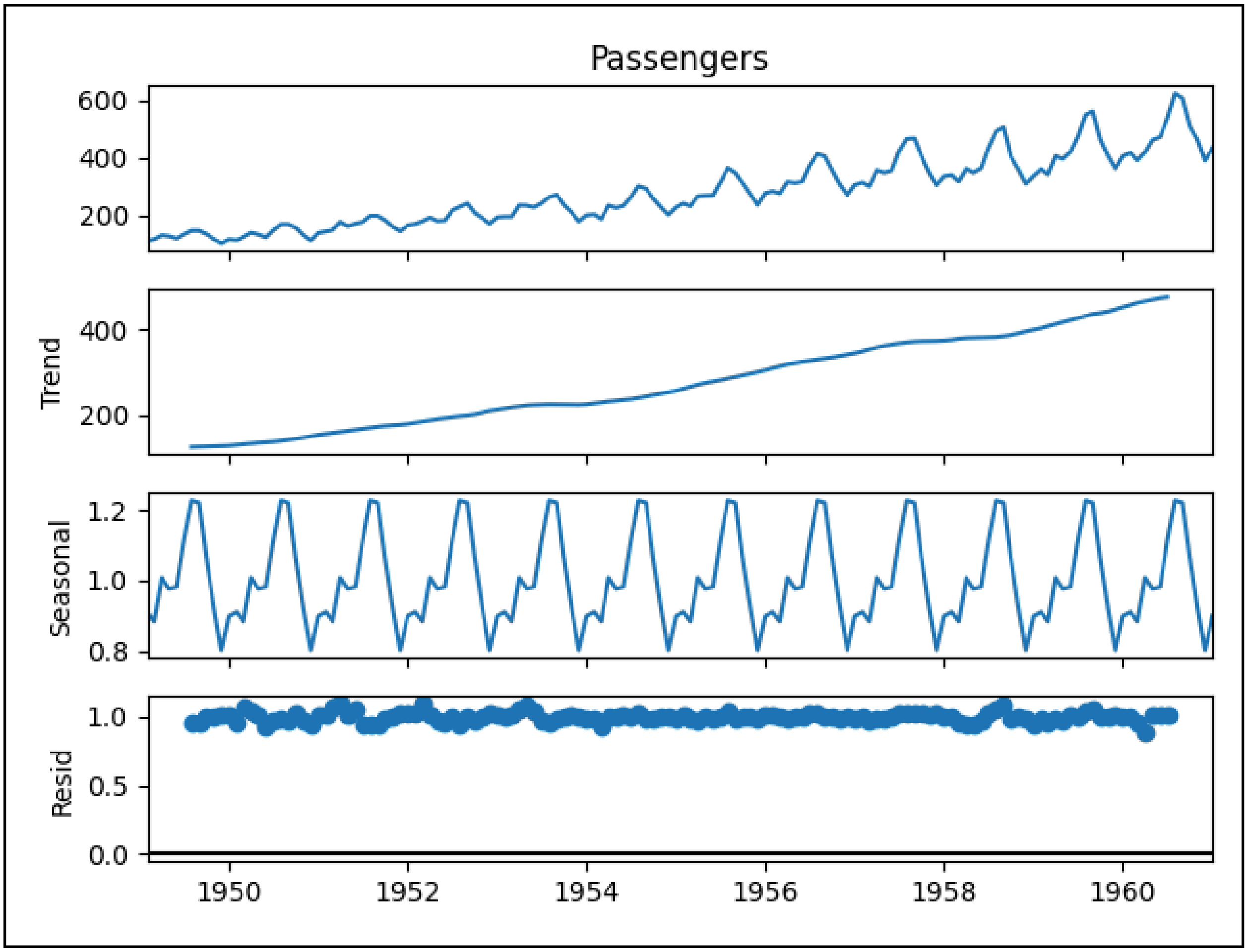
Answer:

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Load AirPassengers dataset
data = {
    "Month": pd.date_range(start="1949-01", periods=144, freq="M"),
    "Passengers": [
        112,118,132,129,121,135,148,148,136,119,104,118,
        115,126,141,135,125,149,170,170,158,133,114,140,
        145,150,178,163,172,178,199,199,184,162,146,166,
        171,180,193,181,183,218,230,242,209,191,172,194,
        196,196,236,235,229,243,264,272,237,211,180,201,
        204,188,235,227,234,264,302,293,259,229,203,229,
        242,233,267,269,270,315,364,347,312,274,237,278,
        284,277,317,313,318,374,413,405,355,306,271,306,
        315,301,356,348,355,422,465,467,404,347,305,336,
        340,318,362,348,363,435,491,505,404,359,310,337,
        360,342,406,396,420,472,548,559,463,407,362,405,
        417,391,419,461,472,535,622,606,508,461,390,432
    ]
}

df = pd.DataFrame(data)
df.set_index("Month", inplace=True)
```

```python
plt.figure()
plt.plot(df["Passengers"])
plt.title("AirPassengers Time Series")
plt.xlabel("Year")
plt.ylabel("Number of Passengers")
plt.show()

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

# Plot decomposition
decomposition.plot()
plt.show()
```

Output:


AirPassengers Time Series
Number of Passengers
100
200
300
400
500
600
1950
1952
1954
1956
1958
1960
Year

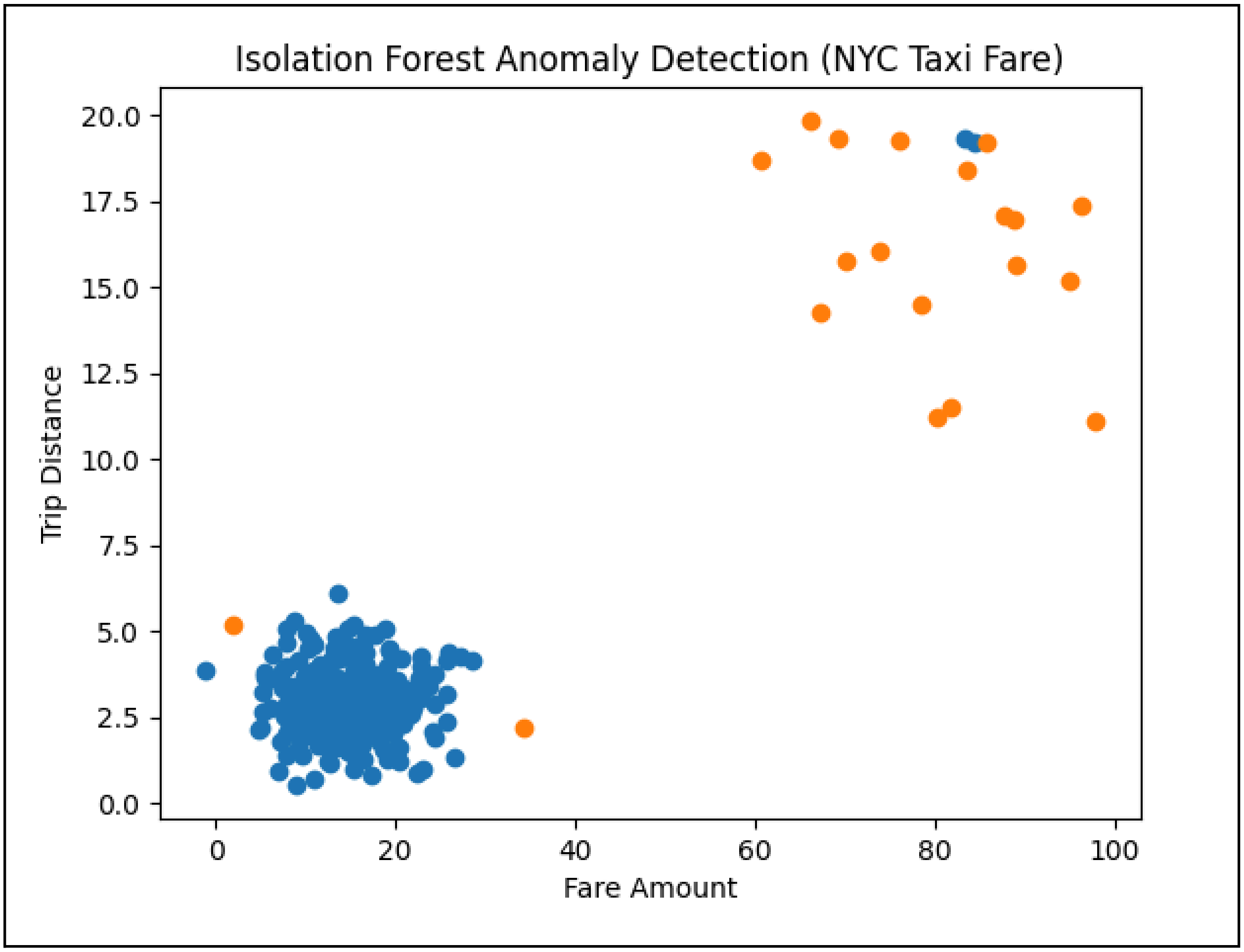


Question 7: Apply Isolation Forest on a numerical dataset (e.g., NYC Taxi Fare) to detect anomalies. Visualize the anomalies on a 2D scatter plot.

Answer:

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

# Create a sample numerical dataset (fare_amount vs trip_distance)
np.random.seed(42)

fare_amount = np.random.normal(loc=15, scale=5, size=300)
trip_distance = np.random.normal(loc=3, scale=1, size=300)

# Add anomalies
fare_outliers = np.random.uniform(60, 100, 20)
distance_outliers = np.random.uniform(10, 20, 20)

fare_amount = np.concatenate([fare_amount, fare_outliers])
trip_distance = np.concatenate([trip_distance, distance_outliers])

df = pd.DataFrame({
    "fare_amount": fare_amount,
    "trip_distance": trip_distance
})

# Apply Isolation Forest
iso_forest = IsolationForest(contamination=0.06, random_state=42)
df["anomaly"] = iso_forest.fit_predict(df)

# Separate normal points and anomalies
normal = df[df["anomaly"] == 1]
anomalies = df[df["anomaly"] == -1]

# Plot results
plt.figure()
plt.scatter(normal["fare_amount"], normal["trip_distance"])
plt.scatter(anomalies["fare_amount"], anomalies["trip_distance"])
plt.xlabel("Fare Amount")
plt.ylabel("Trip Distance")
plt.title("Isolation Forest Anomaly Detection (NYC Taxi Fare)")
plt.show()
```

Output:



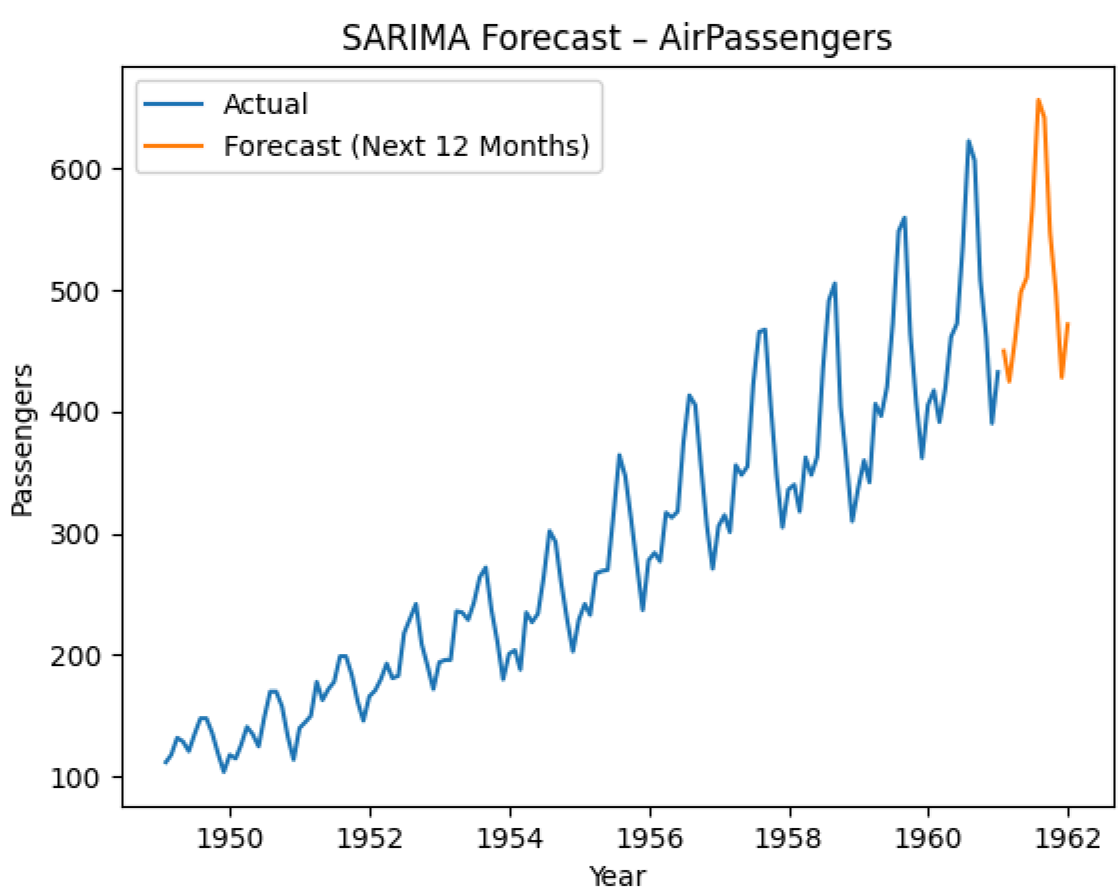
Question 8: Train a SARIMA model on the monthly airline passengers dataset. Forecast the next 12 months and visualize the results.

Answer:

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Load AirPassengers dataset
data = {
    "Month": pd.date_range(start="1949-01", periods=144, freq="M"),
    "Passengers": [
        112,118,132,129,121,135,148,148,136,119,104,118,
        115,126,141,135,125,149,170,170,158,133,114,140,
        145,150,178,163,172,178,199,199,184,162,146,166,
        171,180,193,181,183,218,230,242,209,191,172,194,
        196,196,236,235,229,243,264,272,237,211,180,201,
        204,188,235,227,234,264,302,293,259,229,203,229,
        242,233,267,269,270,315,364,347,312,274,237,278,
        284,277,317,313,318,374,413,405,355,306,271,306,
        315,301,356,348,355,422,465,467,404,347,305,336,
        340,318,362,348,363,435,491,505,404,359,310,337,
        360,342,406,396,420,472,548,559,463,407,362,405,
        417,391,419,461,472,535,622,606,508,461,390,432
    ]
}
df = pd.DataFrame(data)
df.set_index("Month", inplace=True)

# Train SARIMA model
model = SARIMAX(
    df["Passengers"],
    order=(1, 1, 1),
```

```
    seasonal_order=(1, 1, 1, 12)
)
results = model.fit(disp=False)

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

# Plot results
plt.figure()
plt.plot(df["Passengers"], label="Actual")
plt.plot(forecast, label="Forecast (Next 12 Months)")
plt.xlabel("Year")
plt.ylabel("Passengers")
plt.title("SARIMA Forecast – AirPassengers")
plt.legend()
plt.show()
```

Output:



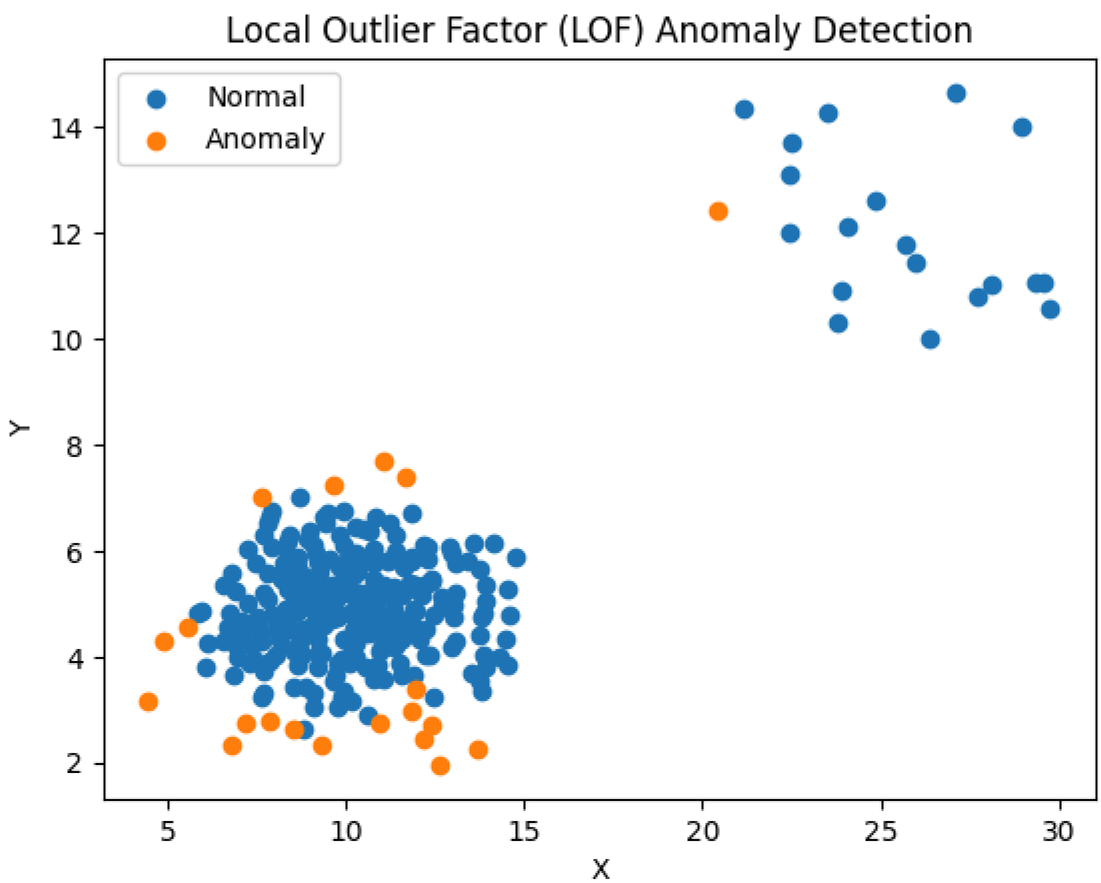
Question 9: Apply Local Outlier Factor (LOF) on any numerical

dataset to detect anomalies and visualize them using matplotlib.
Answer:

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

# Create a sample numerical dataset
np.random.seed(0)

x_normal = np.random.normal(10, 2, 300)
y_normal = np.random.normal(5, 1, 300)

# Add anomalies
x_outliers = np.random.uniform(20, 30, 20)
y_outliers = np.random.uniform(10, 15, 20)

x = np.concatenate([x_normal, x_outliers])
y = np.concatenate([y_normal, y_outliers])

df = pd.DataFrame({"X": x, "Y": y})

# Apply LOF
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.06)
df["anomaly"] = lof.fit_predict(df)

# Separate normal points and anomalies
normal = df[df["anomaly"] == 1]
anomalies = df[df["anomaly"] == -1]

# Plot results
plt.figure()
plt.scatter(normal["X"], normal["Y"], label="Normal")
plt.scatter(anomalies["X"], anomalies["Y"], label="Anomaly")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Local Outlier Factor (LOF) Anomaly Detection")
plt.legend()
```

```
plt.show()
```

Output:



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?

If I were working as a data scientist for a **power grid monitoring company**, my approach would be **practical, reliable, and business-focused**, because both **forecast accuracy** and **fast anomaly detection** are critical for grid stability.

1. Anomaly Detection in Streaming Data

Since the data arrives **every 15 minutes**, anomalies must be detected **quickly and automatically**.

Method choice:

- **Isolation Forest - Primary choice**
  - Works well on large, continuously growing data
  - Fast and scalable
  - Handles multiple features (usage, temperature, humidity, etc.)
- **LOF** - Used as a **secondary check**
  - Good for detecting **local spikes or drops** in specific regions
- **DBSCAN** - Not ideal for real-time streaming
  - Sensitive to parameters
  - Computationally expensive to re-fit frequently

Final decision:
Use **Isolation Forest for real-time detection**, with periodic LOF analysis for deeper investigation.

2. Time Series Model for Short-Term Forecasting

Energy demand shows:

- Clear **daily and weekly seasonality**
- Strong dependency on **weather conditions**

Best model:

- **SARIMAX**

Why SARIMAX?

- SARIMA handles **seasonality**
- SARIMAX allows **exogenous variables** like:
    - Temperature
    - Humidity
    - Weather conditions

This improves short-term forecasts (next few hours or next day).

3. Validation and Monitoring Over Time

Forecast validation:

- Use rolling-window evaluation
- Metrics:
    - **MAE (Mean Absolute Error)**
    - **RMSE (Root Mean Square Error)**

Anomaly validation:

- Compare detected anomalies with:

    - Historical failure logs
    - Maintenance records
    - Sudden weather events

Continuous monitoring:

- Track error metrics over time
- Retrain models periodically
- Set alert thresholds for:
    - Sudden spikes
    - Sudden drops
    - Model performance degradation

4. Business and Operational Impact

This solution directly helps the business by:

- **Preventing blackouts** through early anomaly alerts
- **Optimizing power generation** based on accurate demand forecasts
- **Reducing operational costs** by avoiding over-production
- **Improving maintenance planning** by detecting abnormal patterns early
- **Enhancing customer satisfaction** with stable power supply