Question 1: What is Anomaly Detection? Explain its types (point, contextual, and collective anomalies) with examples.
Answer: Anomaly Detection is the process of identifying data points, events, or patterns that deviate significantly from the expected behavior within a dataset. These anomalies, also known as outliers, can indicate critical issues such as fraud, system failures, or unusual trends. There are three main types of anomalies: **point anomalies**, **contextual anomalies**, and **collective anomalies**. A **point anomaly** occurs when a single data point is significantly different from the rest—for example, a sudden spike in credit card spending when typical transactions are low. **Contextual anomalies** are data points that are only unusual within a specific context; for instance, a temperature of 30°C might be normal in summer but anomalous in winter. Lastly, **collective anomalies** involve a group of data points that, when considered together, show abnormal behavior—such as a series of low sensor readings across multiple devices that may indicate a system malfunction. Understanding these types helps in building effective anomaly detection systems tailored to specific applications

Question 2: Compare Isolation Forest, DBSCAN, and Local Outlier Factor in terms of their approach and suitable use cases.
Answer: Isolation Forest, DBSCAN, and Local Outlier Factor (LOF) are widely used algorithms for anomaly detection, each with distinct approaches and ideal use cases. Isolation Forest works by randomly partitioning data using decision trees, isolating anomalies more quickly due to their distinctiveness. It is highly efficient and well-suited for high-dimensional datasets, making it ideal for applications like fraud detection and industrial monitoring. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) identifies clusters based on data density and labels points in low-density regions as outliers. It performs well on spatial data and datasets with clear cluster structures but may struggle with high-dimensional data. Local Outlier Factor detects anomalies by comparing the local density of a data point to that of its neighbors; points with significantly lower density are considered outliers. LOF is particularly useful in scenarios with varying data densities, such as network traffic analysis or sensor fault detection. Each method offers unique strengths depending on the nature and structure of the data.

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 recorded at regular time intervals, and its analysis helps uncover patterns, trends, and fluctuations over time. The key components of a time series are Trend, Seasonality, Cyclicality, and Irregularity, each revealing different aspects of the data's behavior.

- **Trend** refers to the long-term movement in the data, indicating a general increase, decrease, or stability over time. For example, a steady rise in smartphone sales over several years reflects an upward trend
- **Seasonality** captures regular, repeating patterns that occur at fixed intervals due to external factors like weather or holidays. For instance, electricity consumption tends to increase during summer months due to air conditioning usage.
- **Cyclicality** involves fluctuations that occur over longer, irregular periods, often influenced by economic or business cycles. An example is the rise and fall of unemployment rates during different phases of the economic cycle

- **Irregularity** (or noise) represents random, unpredictable variations in the data that cannot be explained by the other components. For example, a sudden drop in stock prices due to an unexpected geopolitical event is an irregular variation.

Question 4: Define Stationary in time series. How can you test and transform a non-stationary series into a stationary one?
Answer: In time series analysis, stationarity refers to the property of a time series where its statistical characteristics—such as mean, variance, and autocorrelation—remain constant over time. A stationary time series does not exhibit trends or seasonality, making it easier to model and forecast reliably. For example, daily fluctuations in temperature around a stable average without long-term upward or downward movement would be considered stationary.

To determine whether a time series is stationary, analysts commonly use statistical tests like the Augmented Dickey-Fuller (ADF) test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test. The ADF test checks for the presence of a unit root, where a significant p-value indicates non-stationarity. The KPSS test complements this by testing whether the series is stationary around a deterministic trend.

If a time series is found to be **non-stationary**, it can be transformed into a stationary one using several techniques:

- **Differencing**: Subtracting the current value from the previous value to remove trends. This can be applied once or multiple times.
- **Seasonal Differencing**: Subtracting values from the same season in previous cycles (e.g., subtracting July 2024 from July 2023) to remove seasonality.
- **Detrending**: Removing the trend component using regression or moving averages.
- **Logarithmic or Box-Cox Transformation**: Stabilizing variance by compressing large values.

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

Answer: In time series forecasting, several models are used depending on the structure of the data and the presence of trends, seasonality, and external factors. The **AR** (Autoregressive) model predicts future values based on past observations, assuming that previous values influence the current one. For example, predicting today's temperature based on the last few days. The **MA** (Moving Average) model, on the other hand, uses past forecast errors to make predictions, focusing on the noise or shocks in the data.

ARIMA (AutoRegressive Integrated Moving Average) combines both AR and MA components and adds differencing to handle non-stationary data. It's suitable for time series with trends but no seasonality—like monthly sales that gradually increase over time. **SARIMA** (Seasonal ARIMA) extends ARIMA by incorporating seasonal components, making it ideal for data with recurring patterns, such as electricity usage peaking every summer. It includes seasonal autoregression, seasonal differencing, and seasonal moving average terms.Finally, **SARIMAX** (Seasonal ARIMA with eXogenous variables) builds on SARIMA by allowing the inclusion of external variables (exogenous regressors) that may influence the time series. For instance, predicting airline ticket sales using both seasonal trends and external factors like fuel prices or holidays. Each model offers increasing complexity and flexibility, allowing analysts to tailor forecasts to the specific characteristics of their data.

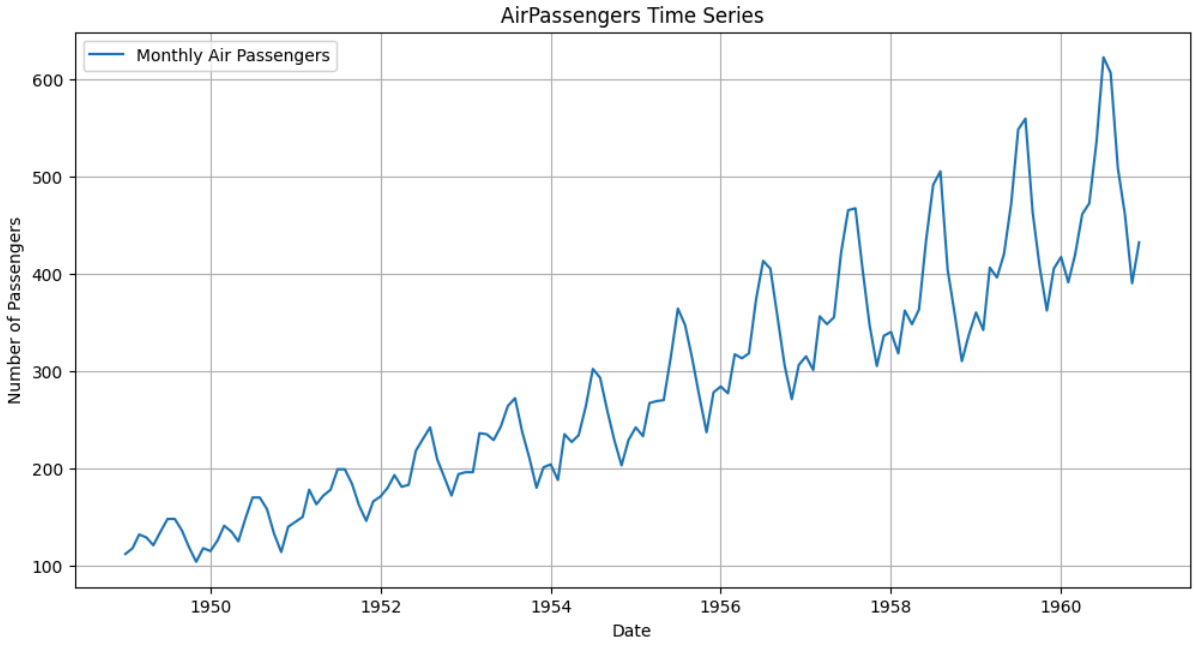
Question 6: Load a time series dataset (e.g., AirPassengers), plot the original series, and decompose it into trend, seasonality, and residual components.

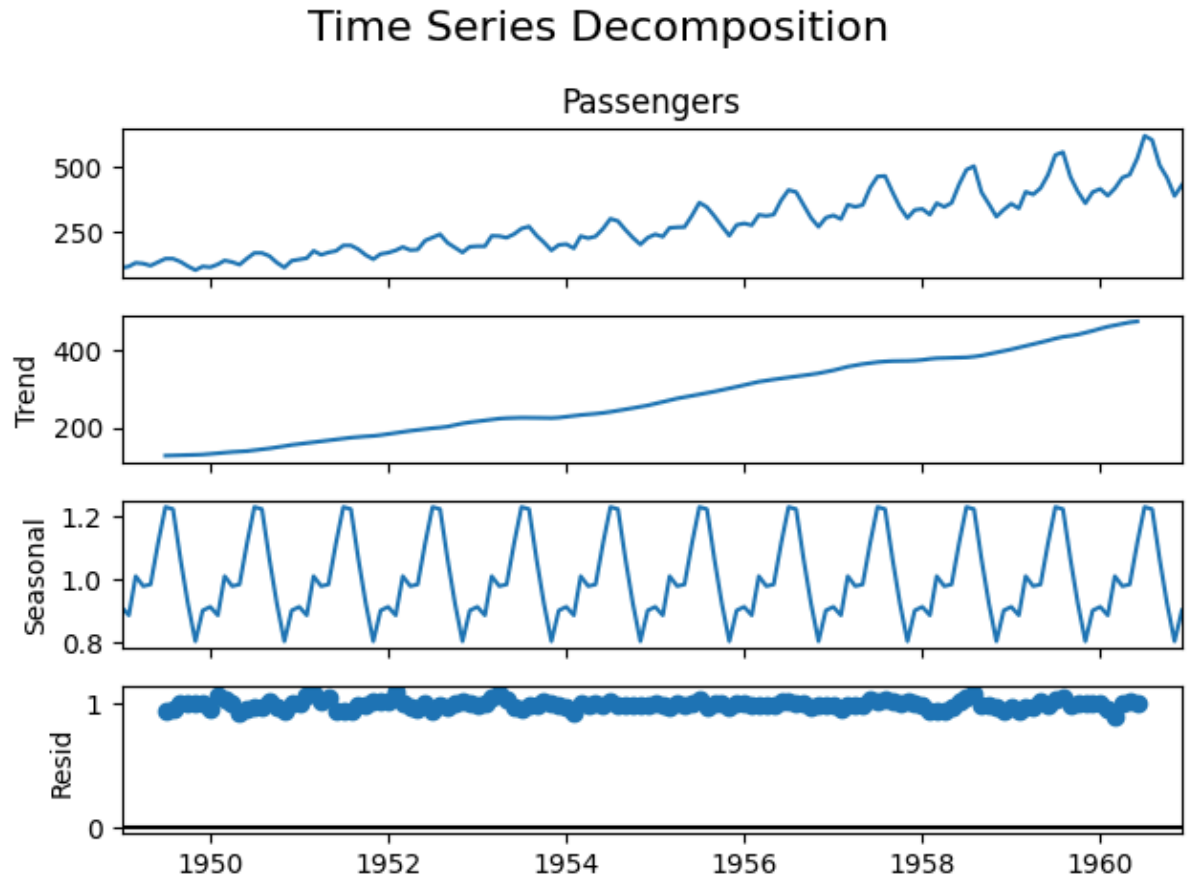
Answer:

```
import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal_decompose

# Load the dataset

df = pd.read_excel("AirPassengers.xlsx")



# Convert 'Month' column to datetime format and set as index

df['Month'] = pd.to_datetime(df['Month'])

df.set_index('Month', inplace=True)



# Rename column for convenience

df.rename(columns={'#Passengers': 'Passengers'}, inplace=True)



# Plot the original time series

plt.figure(figsize=(12, 6))

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

plt.title('AirPassengers Time Series')

plt.xlabel('Date')

plt.ylabel('Number of Passengers')

plt.legend()

plt.grid(True)

plt.show()
```

```
# Decompose the time series

decomposition = seasonal_decompose(df['Passengers'],
model='multiplicative')



# Plot the decomposed components

decomposition.plot()

plt.suptitle('Time Series Decomposition', fontsize=16)

plt.tight_layout()

plt.show()
```

Output:

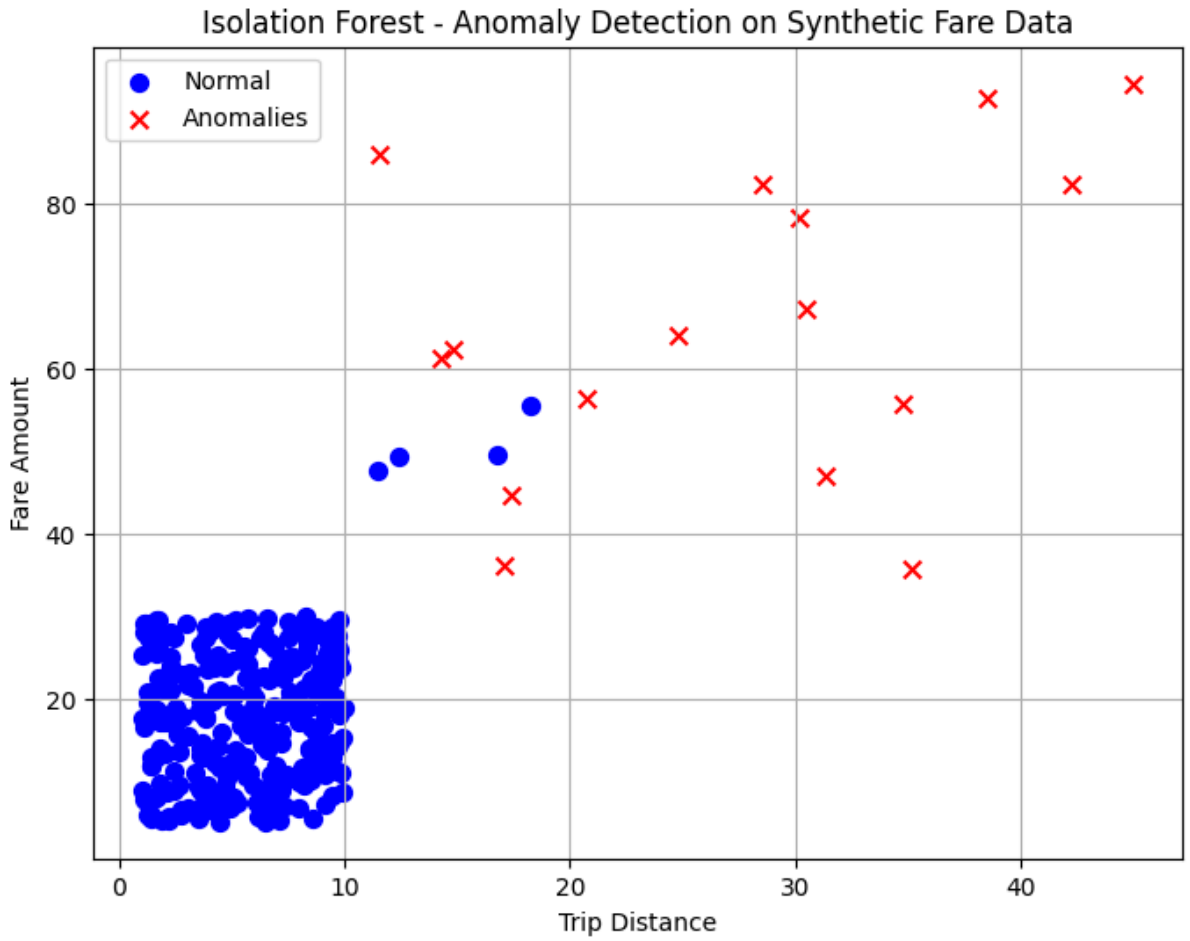

AirPassengers Time Series
Monthly Air Passengers
Number of Passengers
600
500
400
300
200
100
1950
1952
1954
1956
1958
1960
Date


Time Series Decomposition
Passengers
500
250
Trend
400
200
Seasonal
1.2
1.0
0.8
Resid
1
0
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.
Answer:

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

# Step 1: Generate synthetic "NYC Taxi Fare" like data
rng = np.random.RandomState(42)
X_normal = rng.uniform(low=[1, 5], high=[10, 30], size=(300, 2))  # Typical fare range
X_anomalies = rng.uniform(low=[10, 30], high=[50, 100], size=(20, 2))  # Outlier fares
X = np.vstack([X_normal, X_anomalies])

# Step 2: Apply Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42)
labels = iso_forest.fit_predict(X)

# Step 3: Visualize with a 2D scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(X[labels == 1][:, 0], X[labels == 1][:, 1],
            c='blue', label='Normal', s=50)
plt.scatter(X[labels == -1][:, 0], X[labels == -1][:, 1],
            c='red', label='Anomalies', s=50, marker='x')

plt.title('Isolation Forest - Anomaly Detection on Synthetic Fare Data')
plt.xlabel('Trip Distance')
plt.ylabel('Fare Amount')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



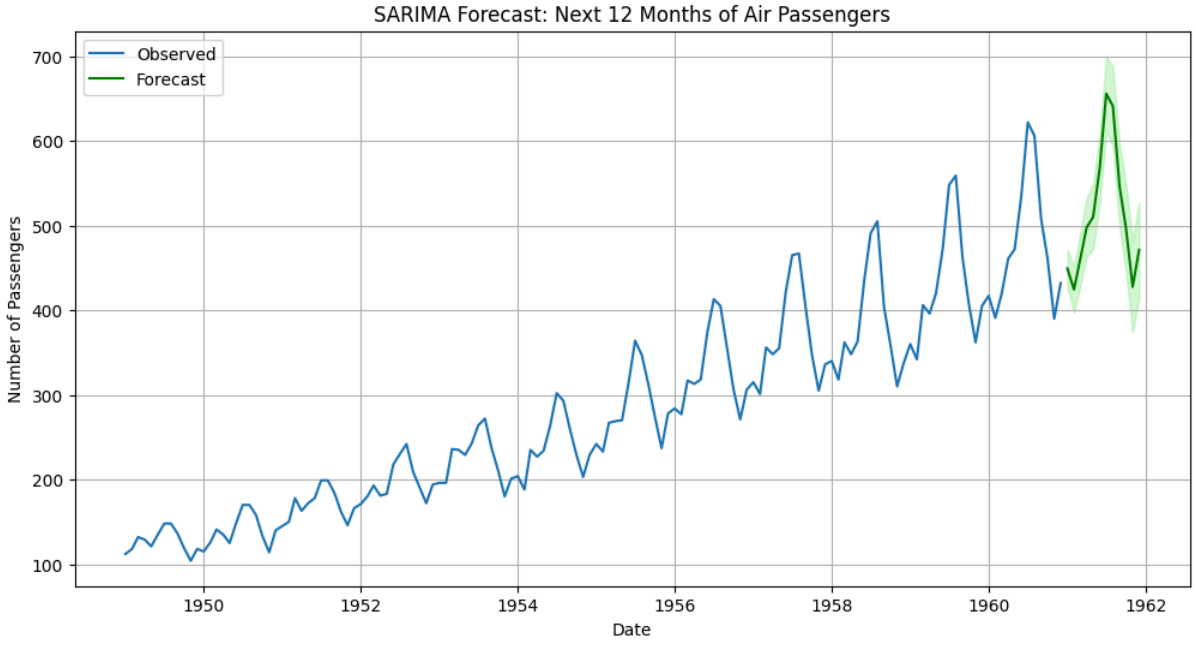
Question 8: Train a SARIMA model on the monthly airline passengers dataset. Forecast the next 12 months and visualize the results.

Answer:

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

# Load the dataset
df = pd.read_excel("AirPassengers.xlsx")
df['Month'] = pd.to_datetime(df['Month'])
df.set_index('Month', inplace=True)
df.rename(columns={'#Passengers': 'Passengers'}, inplace=True)

# Train SARIMA model (parameters can be tuned)
# (p,d,q) = ARIMA terms, (P,D,Q,s) = seasonal terms
model = SARIMAX(df['Passengers'], order=(1,1,1),
seasonal_order=(1,1,1,12))
results = model.fit()
```

```
# Forecast next 12 months
forecast = results.get_forecast(steps=12)
predicted_mean = forecast.predicted_mean
conf_int = forecast.conf_int()

# Plot original data and forecast
plt.figure(figsize=(12, 6))
plt.plot(df['Passengers'], label='Observed')
plt.plot(predicted_mean, label='Forecast', color='green')
plt.fill_between(conf_int.index, conf_int.iloc[:, 0], conf_int.iloc[:,
1], color='lightgreen', alpha=0.4)
plt.title('SARIMA Forecast: Next 12 Months of Air Passengers')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



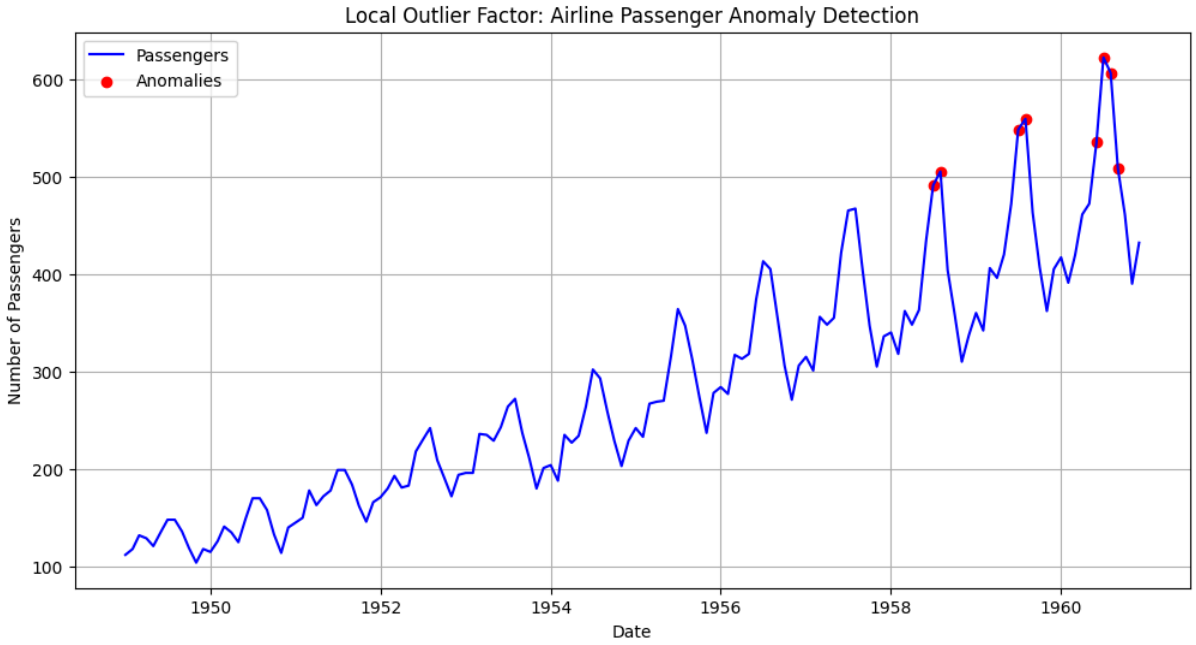
Question 9: Apply Local Outlier Factor (LOF) on any numerical dataset to detect anomalies and visualize them using matplotlib.

Answer:

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

# Load the dataset
df = pd.read_excel("AirPassengers.xlsx")
df['Month'] = pd.to_datetime(df['Month'])
```

```
df.set_index('Month', inplace=True)
df.rename(columns={'#Passengers': 'Passengers'}, inplace=True)

# Prepare data for LOF (reshape required for single feature)
X = df['Passengers'].values.reshape(-1, 1)

# Apply Local Outlier Factor
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.05)
labels = lof.fit_predict(X)

# Add anomaly labels to DataFrame
df['Anomaly'] = labels

# Separate normal and anomalous points
normal = df[df['Anomaly'] == 1]
anomalies = df[df['Anomaly'] == -1]

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Passengers'], label='Passengers', color='blue')
plt.scatter(anomalies.index, anomalies['Passengers'], color='red',
label='Anomalies', marker='o')
plt.title('Local Outlier Factor: Airline Passenger Anomaly Detection')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.grid(True)
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)?
Answer:To detect sudden spikes or drops in consumption:

- Model Choice:
    - Isolation Forest: Efficient for high-dimensional data and adaptable to streaming via mini-batch updates. Good for point anomalies.
    - LOF (Local Outlier Factor): Suitable if the data shows strong local structure. Detects density-based anomalies.
    - DBSCAN: Best if the data forms clusters; more useful in batch mode than streaming.
- Streaming Strategy:
- Use windowing techniques: e.g., sliding or tumbling windows to process data every 15 minutes.
- Normalize features like energy usage, temperature, humidity, and use timestamps in engineered form (hour of day, day of week).
- Deploy models in a framework like Apache Kafka + Spark Streaming or Flask + Celery for real-time scoring.

● Which time series model would you use for short-term forecasting (ARIMA / SARIMA / SARIMAX)?
Answer:

- **Recommended Model**: **SARIMAX (Seasonal AutoRegressive Integrated Moving Average with eXogenous factors)**
    - Handles seasonality (daily, weekly, yearly patterns in energy usage).
    - Includes **exogenous features** like weather and region for richer prediction.

● How would you validate and monitor the performance over time?
Answer:Validation Techniques:

    - **Use TimeSeriesSplit for rolling-origin cross-validation.**
    - **Track Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).**
- **Monitoring & Drift Detection:**
    - **Set up concept drift detection using models like ADWIN or Z-score monitoring for shifting distributions.**
    - **Visual dashboards in Grafana or Power BI for real-time anomaly and forecast monitoring.**

● How would this solution help business decisions or operations?
Answer: Load Balancing:

    - **Accurate forecasts enable power companies to allocate grid resources optimally.**
- **Demand Response Programs:**
    - **Anomaly detection helps initiate consumer-level interventions (e.g., throttling, surge pricing alerts).**
- **Grid Reliability:**
    - **Reduces risk of blackouts or overloads by identifying irregular usage early.**
- **Cost Efficiency:**
    - **Forecasting helps in better procurement and scheduling of power from generators, especially renewable sources.**
- **Policy & Planning:**
- **Seasonal trends empower long-term infrastructure planning and budgeting.**