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, patterns, or observations that deviate significantly from the normal or expected behavior in a dataset. These unusual observations are called anomalies or outliers and often indicate important events such as fraud, system faults, intrusions, or rare but critical conditions.

Anomaly detection is widely used in areas like fraud detection, network security, healthcare monitoring, and industrial fault detection.

Types of Anomalies

1. Point Anomalies

A point anomaly occurs when a single data instance is significantly different from the rest of the data.

Example:

- In a bank transaction dataset, most transactions are below ₹50,000, but one transaction of ₹10,00,000 occurs suddenly.
- That single transaction is a point anomaly and may indicate fraudulent activity.

2. Contextual Anomalies

A contextual anomaly is an observation that is anomalous only within a specific context (such as time, location, or season).

Example:

- A temperature of 30°C is normal during summer but would be unusual in winter.
- The same value becomes anomalous depending on the context (season).

Contextual anomalies are common in time-series data like weather data, network traffic, or sensor readings.

3. Collective Anomalies

A collective anomaly occurs when a group of related data points is anomalous as a whole, even if individual points may appear normal.

Example:

- A sudden sequence of small network requests that individually look normal but collectively indicate a DDoS attack.
- In ECG data, a sequence of heartbeats may collectively indicate an abnormal heart rhythm.

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 popular techniques for anomaly detection, but they differ in how they identify anomalies and where they are best applied.

1. Isolation Forest

Approach:
Isolation Forest is a tree-based, ensemble method that detects anomalies by isolating data points rather than modeling normal behavior. Anomalies are easier to isolate and thus require fewer splits in decision trees.

Key Idea:

- Anomalies are rare and different → they get isolated quickly.

Suitable Use Cases:

- Large, high-dimensional datasets
- Fraud detection and intrusion detection
- When the dataset has no clear cluster structure

Pros:

- Scales well to large datasets
- Works efficiently with high-dimensional data

Cons:

- Less effective if anomalies are not well separated

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

Approach:
DBSCAN is a density-based clustering algorithm. It identifies anomalies as noise points that do not belong to any dense cluster.

Key Idea:

- Normal data points form dense clusters
- Points in low-density regions are anomalies

Suitable Use Cases:

- Spatial data and datasets with arbitrary-shaped clusters
- When the number of clusters is unknown
- Applications like geographic data analysis and image processing

Pros:

- Automatically detects outliers
- Can find clusters of arbitrary shape

Cons:

- Sensitive to parameter selection (ε and MinPts)
- Struggles with varying densities and high-dimensional data

3. Local Outlier Factor (LOF)

Approach:
LOF is a density-based method that compares the local density of a data point with the

densities of its neighbors. A point is anomalous if it has significantly lower density than its neighbors.

Key Idea:

- Detects local anomalies, not just global ones

Suitable Use Cases:

- Data with varying densities
- Detecting subtle, local outliers
- Applications such as network traffic analysis and fraud detection

Pros:

- Effective for local outliers
- Adapts well to varying density regions

Cons:

- Computationally expensive for large datasets
- Sensitive to the choice of number of neighbors

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

Answer:

A Time Series is a sequence of observations recorded at regular time intervals (such as hourly, daily, monthly, or yearly). Time series analysis helps understand patterns over time and is widely used in forecasting, economics, finance, and weather analysis.

Key Components of a Time Series

1. Trend

The trend represents the long-term movement or direction in the data over an extended period.

Example:

- A company's annual sales steadily increasing over several years due to business growth.

2. Seasonality

Seasonality refers to regular and repeating patterns that occur at fixed intervals due to seasonal factors.

Example:

- Higher electricity consumption during summer months every year because of air conditioner usage.

3. Cyclical Component

The cyclical component represents long-term fluctuations that occur over periods longer than a season, often influenced by economic or business cycles.

Example:

- An economy experiencing periods of expansion and recession over several years.

4. Irregular (Random) Component

The irregular component consists of unpredictable, random variations that do not follow any pattern.

Example:

- A sudden drop in airline bookings due to an unexpected 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 remain constant over time. Stationarity is a key assumption in many time-series models (such as ARIMA) because it makes the series easier to analyze and forecast.

What is Stationarity?

A time series is said to be stationary if:

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

Example:

- Daily fluctuations of stock returns often form a stationary series, while stock prices themselves are usually non-stationary.

Testing for Stationarity

1. Visual Inspection

- Plot the time series
- If the mean or variance changes over time, the series is likely non-stationary

2. Augmented Dickey–Fuller (ADF) Test

- Null hypothesis (H_0): Series is non-stationary
- If p-value < 0.05, reject H_0 → series is stationary

3. KPSS Test

- Null hypothesis (H_0): Series is stationary
- If p-value < 0.05, the series is non-stationary

Using both ADF and KPSS gives a more reliable conclusion.

Transforming a Non-Stationary Series into a Stationary One

1. Differencing

- Subtract the previous value from the current value
- Removes trend and seasonality

2. Log or Power Transformation

- Stabilizes variance when data shows exponential growth

Example:

- Apply log transformation to sales or price data

3. Detrending

- Remove the trend component using regression or smoothing methods

4. Seasonal Differencing

- Remove seasonal effects by subtracting values from the same season.

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

Answer:

Differentiation between AR, MA, ARIMA, SARIMA, and SARIMAX Models (with Dataset Context)

AR, MA, ARIMA, SARIMA, and SARIMAX are classical time series forecasting models that differ in structure and are chosen based on whether the data is stationary, seasonal, or influenced by external factors. Their applicability can be clearly understood using datasets like NYC Taxi Fare Data and the AirPassengers Dataset.

1. AR (Autoregressive) Model

Structure:

- Predicts the current value using its own past values
- Denoted as AR(p)

Application:

- Used for stationary time series without trend or seasonality

Dataset Example:

- NYC Taxi Fare Data: After removing trend and seasonality, short-term fare fluctuations may be modeled using AR if fares depend strongly on recent past values.

2. MA (Moving Average) Model

Structure:

- Uses past error terms (shocks) to predict the current value
- Denoted as MA(q)

Application:

- Suitable for stationary data affected by random shocks

Dataset Example:

- NYC Taxi Fare Data: Sudden fare changes caused by temporary events (e.g., traffic surges or weather disruptions) can be modeled using an MA model.

3. ARIMA (Autoregressive Integrated Moving Average)

Structure:

- Combines AR and MA with differencing to handle non-stationarity
- Written as ARIMA(p, d, q)

Application:

- Used for non-stationary time series with trend but no seasonality

Dataset Example:

- NYC Taxi Fare Data: If taxi fares show a long-term upward trend over years but no strong seasonal pattern, ARIMA is appropriate after differencing.

4. SARIMA (Seasonal ARIMA)

Structure:

- Extends ARIMA by adding seasonal components
- Written as SARIMA(p, d, q)(P, D, Q, s)

Application:

- Suitable for data with both trend and seasonality

Dataset Example:

- AirPassengers Dataset: Shows clear yearly seasonality (12-month cycle) and trend, making SARIMA ideal for modeling passenger counts.

5. SARIMAX (Seasonal ARIMA with Exogenous Variables)

Structure:

- SARIMA + external (exogenous) variables such as weather, holidays, or fuel prices

Application:

- Used when the time series is influenced by external factors

Dataset Example:

- NYC Taxi Fare Data: Fare prediction can be improved using SARIMAX by including exogenous variables like distance traveled, surge pricing, time of day, or weather conditions.

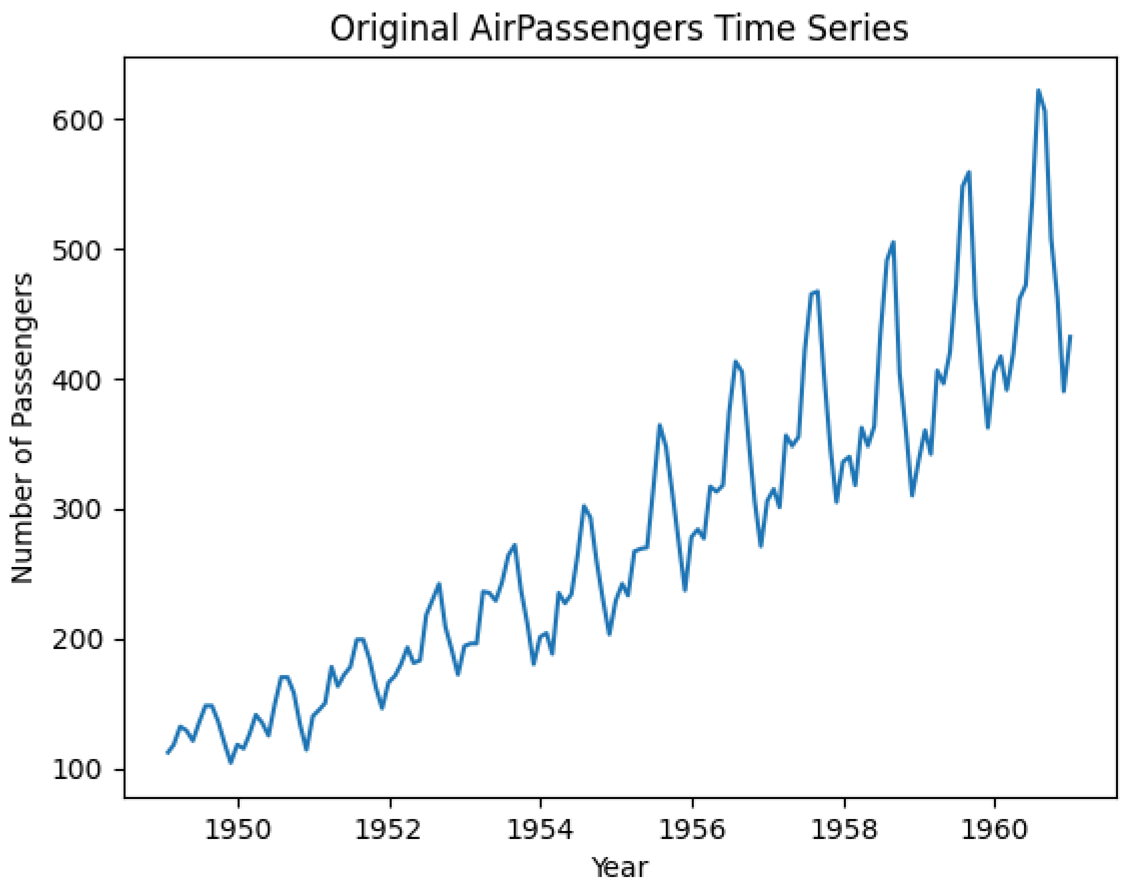
Question 6: Load a time series dataset (e.g., AirPassengers), plot the original series, and decompose it into trend, seasonality, and residual components (Include your Python code and output in the code box below.)

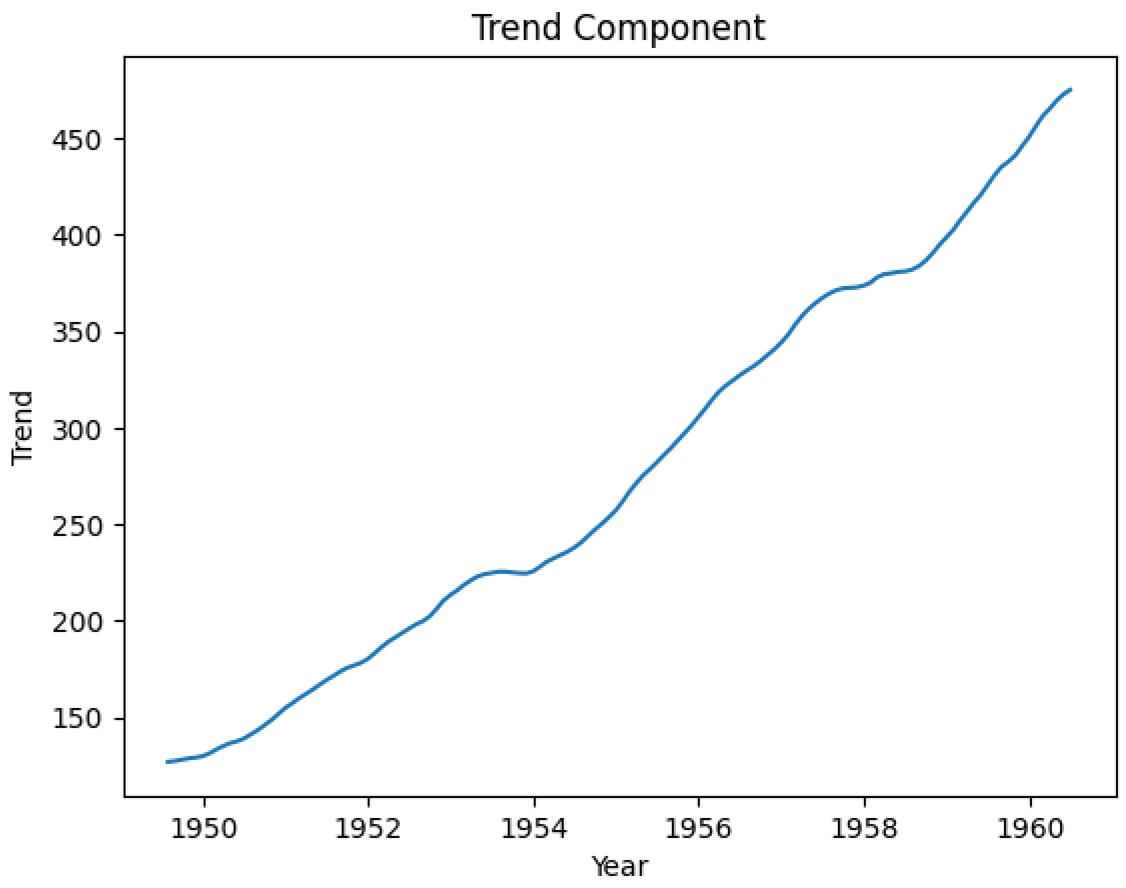
Answer:

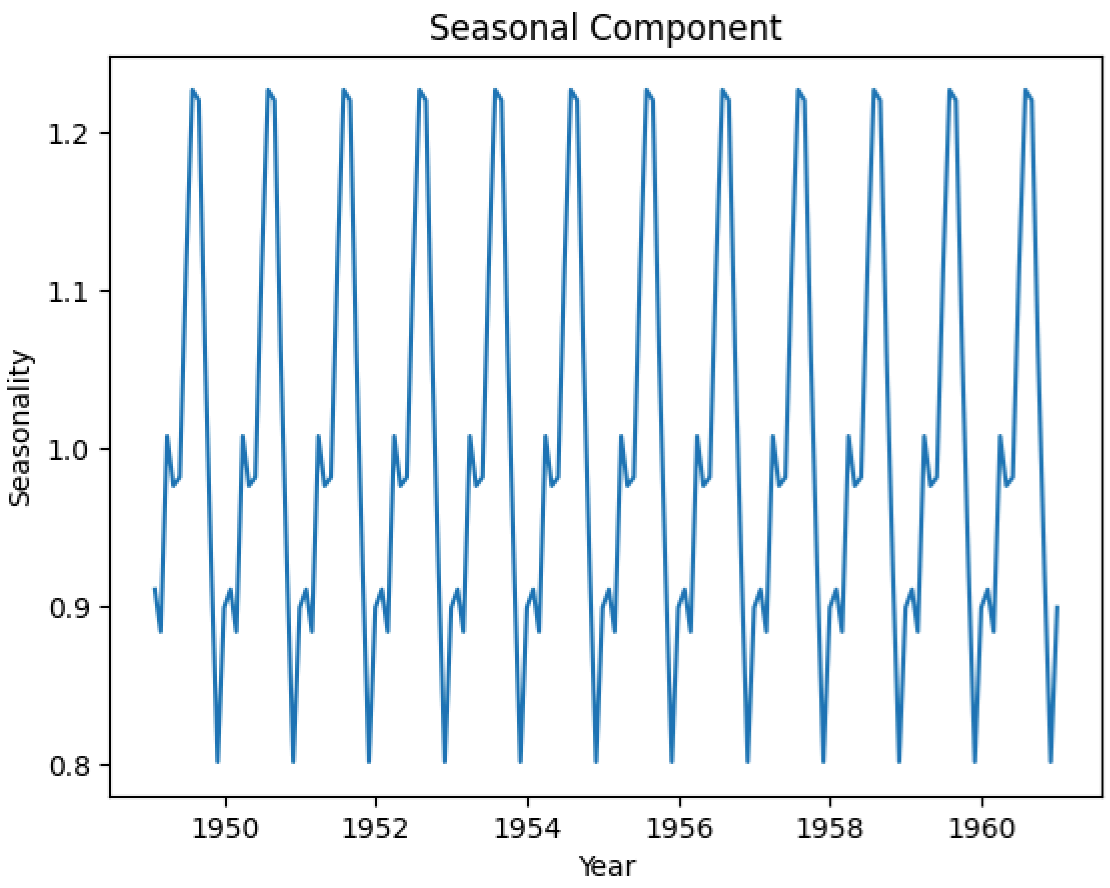
```
# Time Series Decomposition of AirPassengers Dataset

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal_decompose

# AirPassengers monthly data (1949–1960)

data = [

   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

]

# Create time series

dates = pd.date_range(start="1949-01", periods=len(data), freq="M")
```

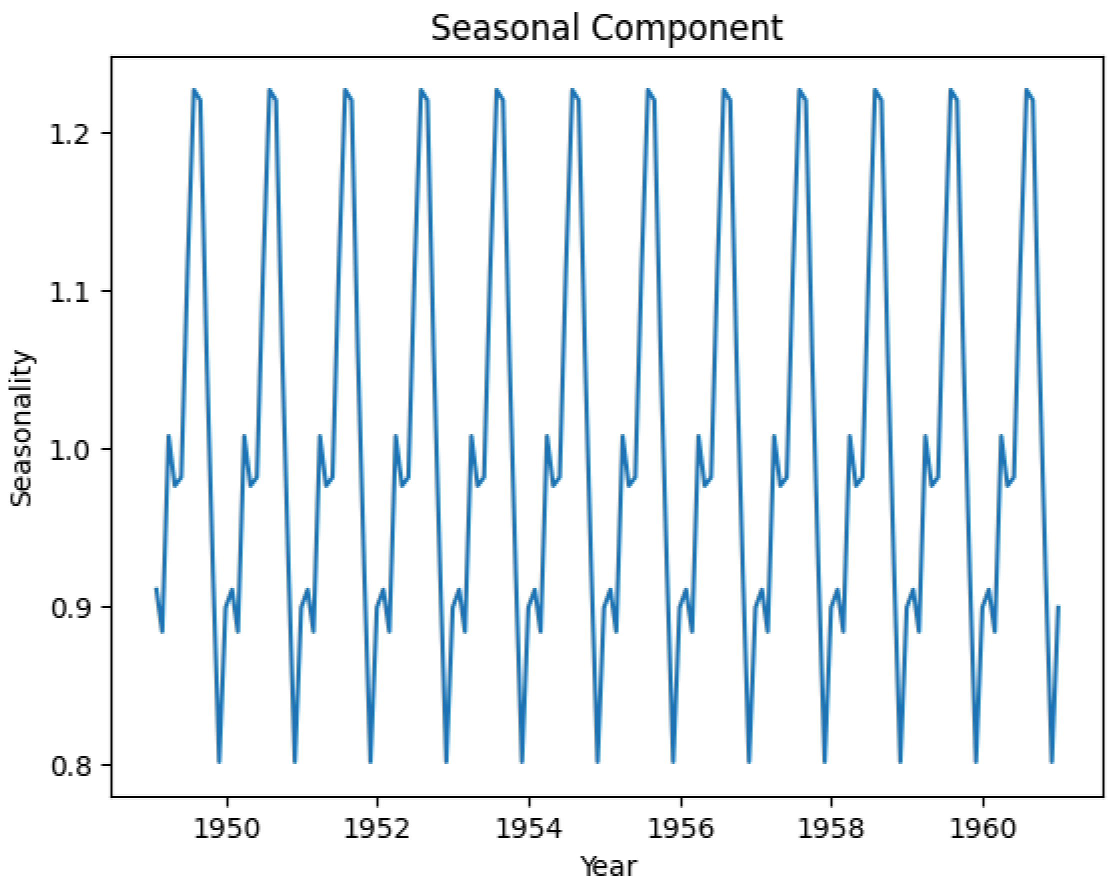
```
ts = pd.Series(data, index=dates)

# Plot original series

plt.figure()

plt.plot(ts)

plt.title("Original AirPassengers Time Series")

plt.xlabel("Year")

plt.ylabel("Number of Passengers")

plt.show()

# Decompose the series

decomposition = seasonal_decompose(ts, model='multiplicative', period=12)

# Plot Trend

plt.figure()

plt.plot(decomposition.trend)

plt.title("Trend Component")

plt.xlabel("Year")

plt.ylabel("Trend")

plt.show()

# Plot Seasonality

plt.figure()

plt.plot(decomposition.seasonal)

plt.title("Seasonal Component")

plt.xlabel("Year")

plt.ylabel("Seasonality")

plt.show()
```

```
# Plot Residuals

plt.figure()

plt.plot(decomposition.resid)

plt.title("Residual Component")

plt.xlabel("Year")

plt.ylabel("Residuals")

plt.show()
```




Trend Component
Trend
Year
150
200
250
300
350
400
450
1950
1952
1954
1956
1958
1960


Seasonal Component
Seasonality
Year
1.2
1.1
1.0
0.9
0.8
1950
1952
1954
1956
1958
1960

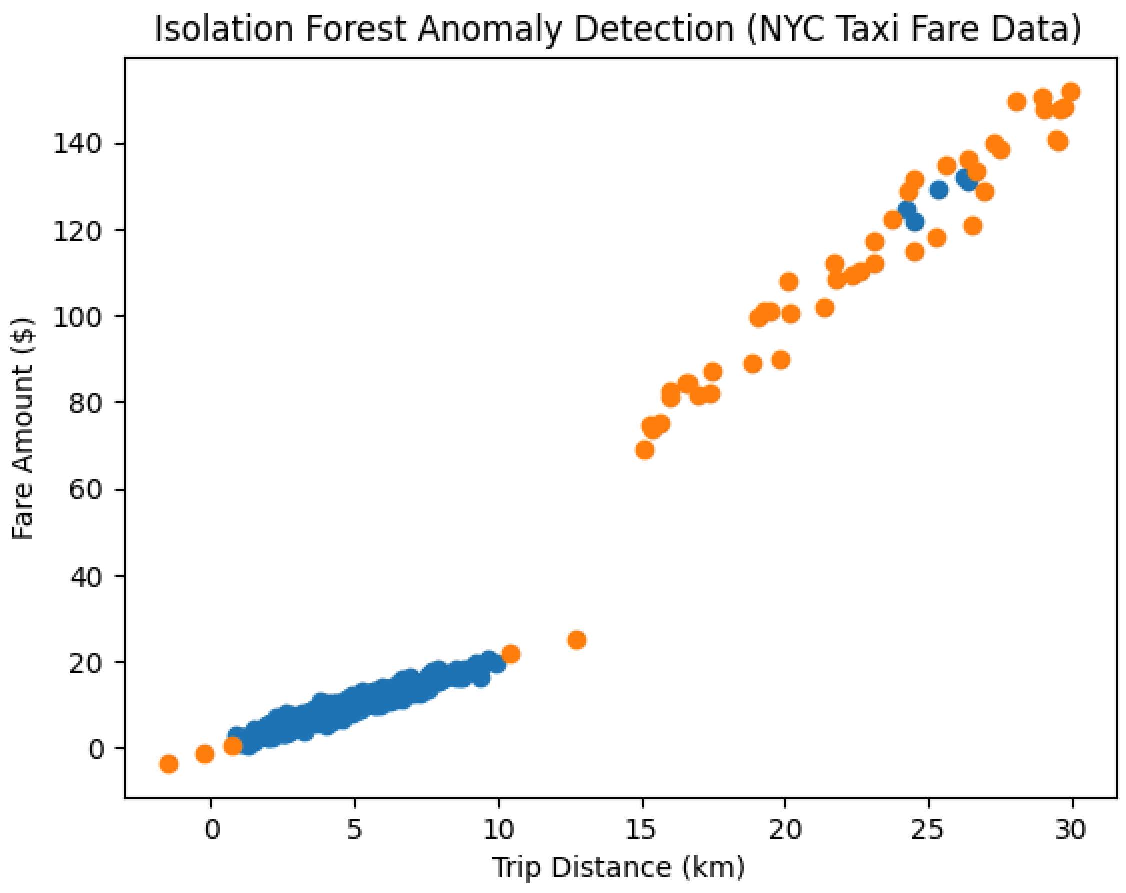


Question 7: Apply Isolation Forest on a numerical dataset (e.g., NYC Taxi Fare) to detect anomalies. Visualize the anomalies on a 2D scatter plot. (Include your Python code and output in the code box below.)

Answer:

```
# Isolation Forest for Anomaly Detection (NYC Taxi Fare Dataset)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest

# Set random seed for reproducibility

np.random.seed(42)
```

```
# Create a synthetic NYC Taxi Fare–like dataset

# Features: trip_distance (km) and fare_amount ($)

normal_distance = np.random.normal(loc=5, scale=2, size=450)

normal_fare = normal_distance * 2 + np.random.normal(0, 1, 450)

# Create anomalies (unusually long trips and high fares)

anomaly_distance = np.random.uniform(15, 30, 50)

anomaly_fare = anomaly_distance * 5 + np.random.normal(0, 5, 50)

# Combine normal data and anomalies

distance = np.concatenate([normal_distance, anomaly_distance])

fare = np.concatenate([normal_fare, anomaly_fare])

df = pd.DataFrame({

   "trip_distance": distance,

   "fare_amount": fare

})

# Apply Isolation Forest

iso_forest = IsolationForest(contamination=0.1, random_state=42)

df["anomaly"] = iso_forest.fit_predict(df[["trip_distance", "fare_amount"]])

# Separate normal points and anomalies

normal = df[df["anomaly"] == 1]

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

# Visualize results

plt.figure()

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

plt.scatter(anomalies["trip_distance"], anomalies["fare_amount"])
plt.xlabel("Trip Distance (km)")

plt.ylabel("Fare Amount ($)")

plt.title("Isolation Forest Anomaly Detection (NYC Taxi Fare Data)")

plt.show()
```



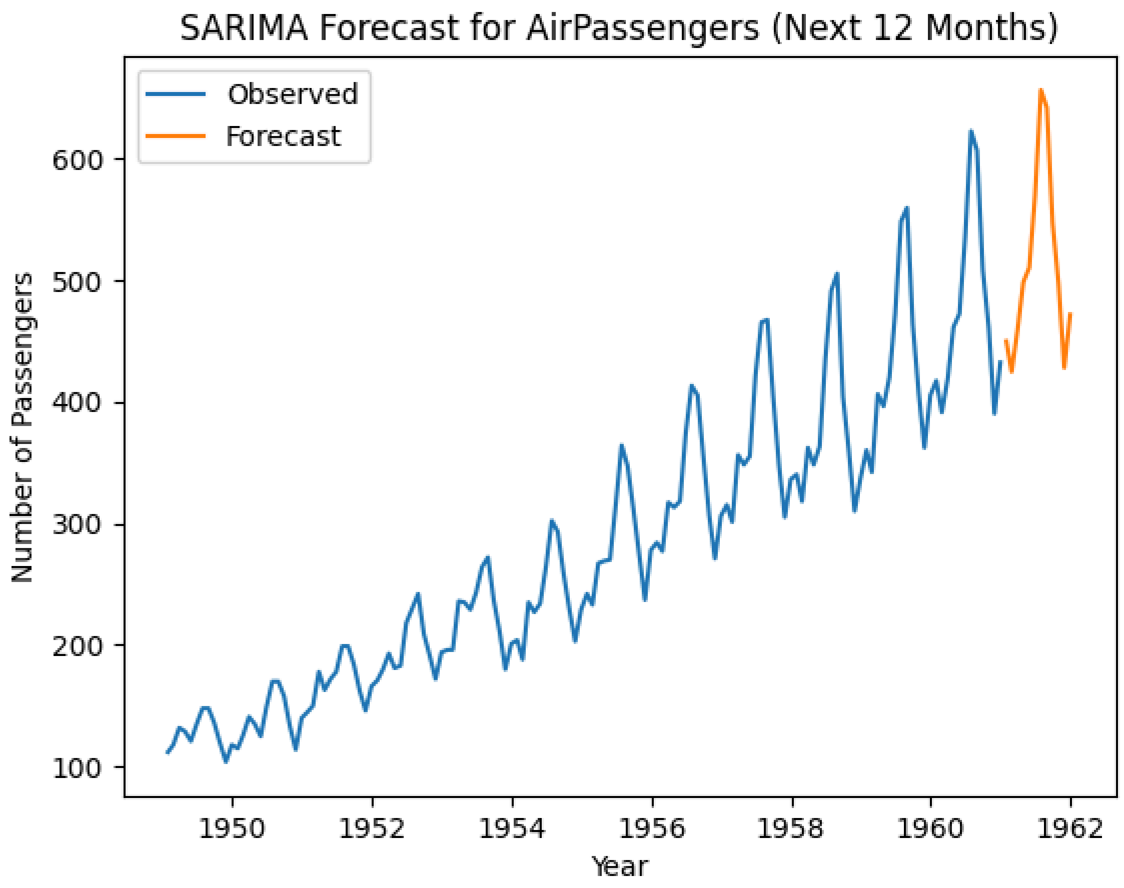
Question 8: Train a SARIMA model on the monthly airline passengers dataset. Forecast the next 12 months and visualize the results. (Include your Python code and output in the code box below.)

Answer:

```
# SARIMA model on AirPassengers dataset with 12-month forecast

import pandas as pd

import matplotlib.pyplot as plt
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# AirPassengers monthly data (1949–1960)

data = [

    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

]

# Create time series index

dates = pd.date_range(start="1949-01", periods=len(data), freq="M")

ts = pd.Series(data, index=dates)

# Train SARIMA model

# (p,d,q)(P,D,Q,s)

model = SARIMAX(ts, order=(1,1,1), seasonal_order=(1,1,1,12))

results = model.fit(disp=False)

# Forecast next 12 months
```

```
forecast = results.forecast(steps=12)

forecast_index = pd.date_range(

    start=ts.index[-1] + pd.offsets.MonthEnd(1),

    periods=12,

    freq="M"

)

# Plot observed data and forecast

plt.figure()

plt.plot(ts, label="Observed")

plt.plot(forecast_index, forecast, label="Forecast")

plt.xlabel("Year")

plt.ylabel("Number of Passengers")

plt.title("SARIMA Forecast for AirPassengers (Next 12 Months)")

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:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.neighbors import LocalOutlierFactor

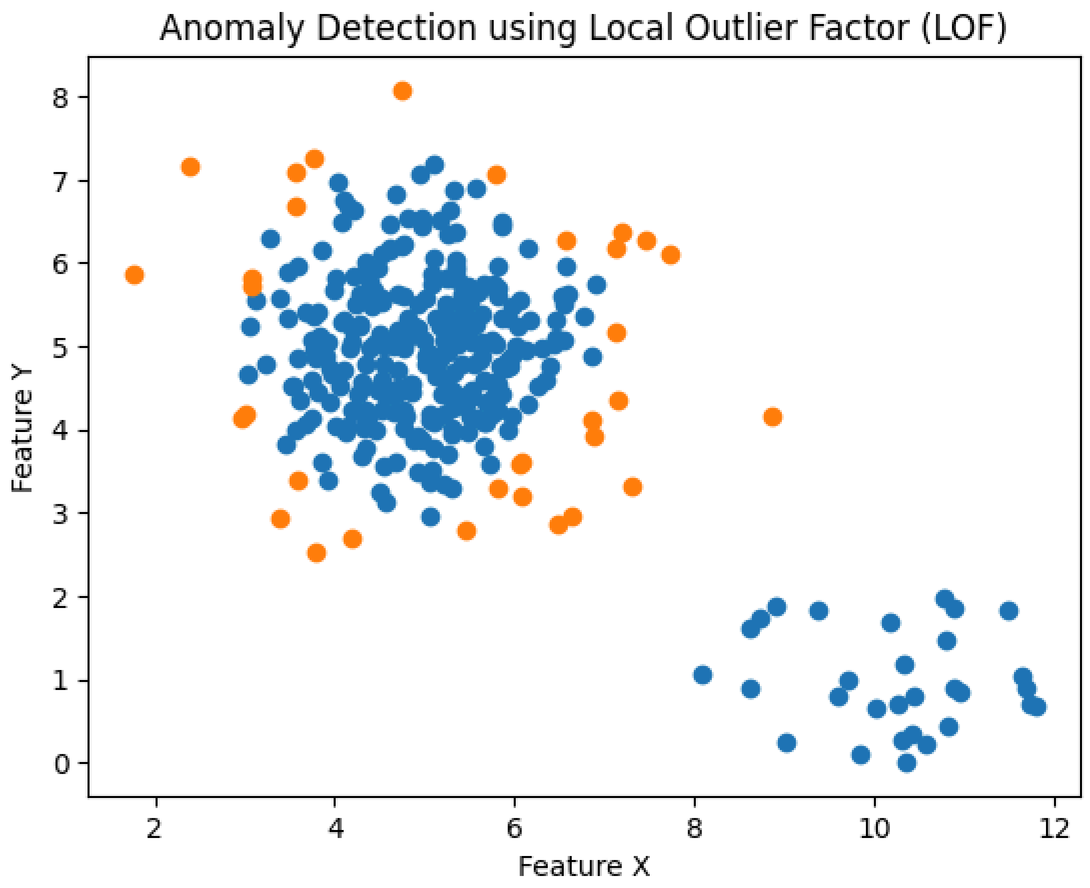
# Set random seed for reproducibility

np.random.seed(42)

# Generate normal data

normal_x = np.random.normal(loc=5, scale=1, size=300)

```
normal_y = np.random.normal(loc=5, scale=1, size=300)

# Generate anomalies

anomaly_x = np.random.uniform(8, 12, 30)

anomaly_y = np.random.uniform(0, 2, 30)

# Combine into one dataset

x = np.concatenate([normal_x, anomaly_x])

y = np.concatenate([normal_y, anomaly_y])

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

# Apply Local Outlier Factor

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

df["label"] = lof.fit_predict(df[["x", "y"]])

# Separate normal points and anomalies

normal = df[df["label"] == 1]

outliers = df[df["label"] == -1]

# Plot the results

plt.figure()

plt.scatter(normal["x"], normal["y"])

plt.scatter(outliers["x"], outliers["y"])

plt.xlabel("Feature X")

plt.ylabel("Feature Y")

plt.title("Anomaly Detection using Local Outlier Factor (LOF)")

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:

Real-Time Data Science Workflow for Power Grid Monitoring

1. Anomaly Detection in Streaming Data

To detect abnormal spikes or drops in energy consumption every 15 minutes, consider these approaches:

Algorithm	Suitability	Notes
Isolation Forest	Good for high-dimensional data	Fast, works well with streaming; isolates anomalies by randomly partitioning data

Recommended: Use Isolation Forest with a sliding window approach (e.g., last 24 hours of data) to continuously retrain and detect anomalies.

2. Short-Term Forecasting Model

For forecasting energy demand in the next few intervals:

Model	Use Case	Notes
ARIMA	Basic time series	Assumes stationarity; not ideal with seasonality or exogenous variables
SARIMA	Seasonal patterns	Handles seasonality well but lacks external features
SARIMAX	Seasonality + exogenous variables	Best fit here due to weather and region features

Recommended: Use SARIMAX to incorporate weather conditions and region as exogenous variables for more accurate short-term forecasting.

3. Validation and Monitoring

To ensure reliability over time:

- Validation:
    - Use rolling cross-validation on historical data.
    - Evaluate with metrics like RMSE, MAE, and MAPE.
    - Monitor anomaly detection precision/recall.
- Monitoring:
    - Deploy real-time dashboards (e.g., Power BI, Grafana).
    - Set up alerting systems for anomaly flags.
    - Track model drift and retrain periodically.

4. Business Impact

This solution supports operations and decision-making by:

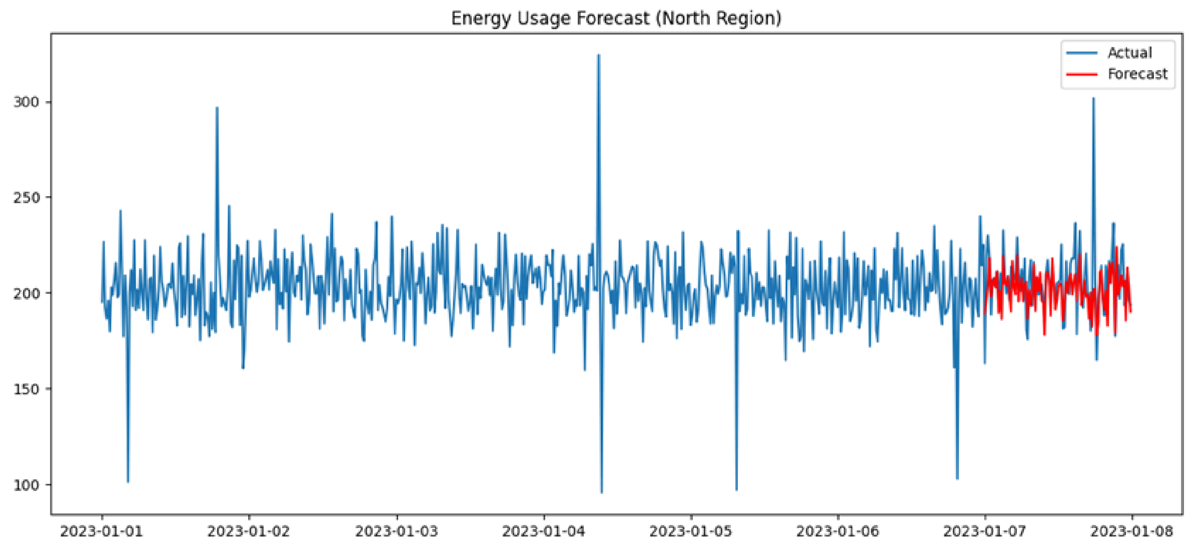
- Preventing outages: Early anomaly detection helps preempt grid failures.
- Optimizing load balancing: Accurate forecasts allow better distribution of energy resources.
- Reducing costs: Efficient energy allocation minimizes waste and peak load penalties.
- Improving customer satisfaction: Stable grid performance enhances reliability for end users.

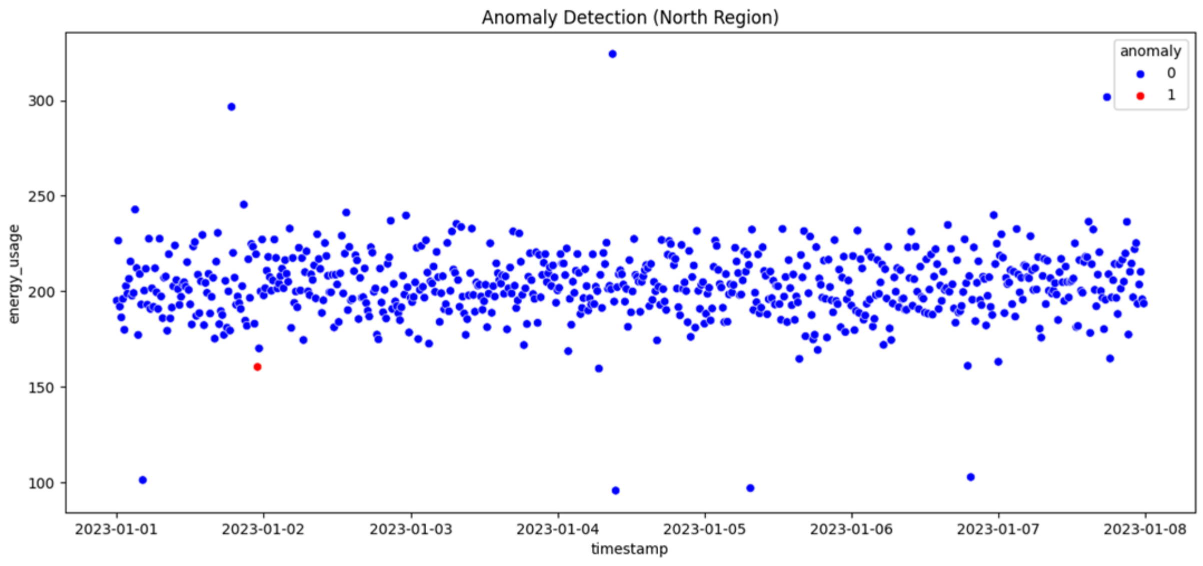
CODE:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest

from statsmodels.tsa.statespace.sarimax import SARIMAX

import seaborn as sns

# 1. Generate synthetic data

np.random.seed(42)

timestamps = pd.date_range(start='2023-01-01', periods=7*24*4, freq='15T')

regions = ['North', 'South', 'East']

data = []



for region in regions:

    temp_base = np.random.uniform(15, 30)

    for ts in timestamps:

        temp = temp_base + np.random.normal(0, 2)

        usage = 100 + temp * 5 + np.random.normal(0, 10)
        # Inject anomalies

        if np.random.rand() < 0.01:

            usage *= np.random.choice([0.5, 1.5])

        data.append([ts, region, temp, usage])

df = pd.DataFrame(data, columns=['timestamp', 'region', 'temperature', 'energy_usage'])

# 2. Anomaly Detection using Isolation Forest

df['hour'] = df['timestamp'].dt.hour

df['day'] = df['timestamp'].dt.dayofweek

features = ['temperature', 'energy_usage', 'hour', 'day']

model = IsolationForest(contamination=0.01)

df['anomaly'] = model.fit_predict(df[features])

df['anomaly'] = df['anomaly'].map({1: 0, -1: 1})

# 3. Forecasting with SARIMAX for one region

region_df = df[df['region'] == 'North'].set_index('timestamp')

series = region_df['energy_usage'].resample('15T').mean()


# Train-test split

train = series[:-96]  # last day for test

test = series[-96:]

model = SARIMAX(train, order=(1,1,1), seasonal_order=(1,1,1,96),
exog=region_df['temperature'][:-96])

results = model.fit(disp=False)

forecast = results.predict(start=len(train), end=len(train)+95, exog=region_df['temperature'][-96:])

# 4. Visualization
```

```
plt.figure(figsize=(14,6))

plt.plot(series.index, series, label='Actual')

plt.plot(test.index, forecast, label='Forecast', color='red')

plt.title('Energy Usage Forecast (North Region)')

plt.legend()

plt.show()

# Anomaly plot

plt.figure(figsize=(14,6))

sns.scatterplot(data=df[df['region']=='North'], x='timestamp', y='energy_usage', hue='anomaly',
palette={0:'blue', 1:'red'})

plt.title('Anomaly Detection (North Region)')

plt.show()
```

OUTPUT:





Anomaly Detection (North Region)
anomaly
0
1
energy_usage
300
250
200
150
100
2023-01-01
2023-01-02
2023-01-03
2023-01-04
2023-01-05
2023-01-06
2023-01-07
2023-01-08
timestamp